Lecture Notes in Computer Science 4125

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Gianluca Moro Sonia Bergamaschi
Sam Joseph Jean-Henry Morin
Aris M. Ouksel (Eds.)

Databases, Information Systems, and Peer-to-Peer Computing

International Workshops, DBISP2P 2005/2006
Trondheim, Norway, August 28-29, 2005
Seoul, Korea, September 11, 2006
Revised Selected Papers

Springer

Volume Editors

Gianluca Moro
University of Bologna, Dept. of Electronics, Computer Science and Systems (DEIS)
Via Venezia, 52, 47023 Cesena (FC), Italy
E-mail: gmoro@deis.unibo.it

Sonia Bergamaschi
Università di Modena e Reggio Emilia, Dip. di Ingegneria dell'Informazione
Via Vignolese 905, 41100 Modena, Italy
E-mail: bergamaschi.sonia@unimo.it

Sam Joseph
University of Hawaii, Dept. of Information and Computer Science
1680 East-West Road, POST 309, Honolulu, HI 96822, USA
E-mail: srjoseph@hawaii.edu

Jean-Henry Morin
Korea University Business School
Anam-Dong, Seongbuk-Gu, Seoul 136-701, Korea
E-mail: jhmorin@korea.ac.kr

Aris M. Ouksel
The University of Illinois at Chicage, Dept. of Information and Decision Sciences
2411 University Hall, Chicago, IL, USA
E-mail: aris@uic.edu

Library of Congress Control Number: 2007924345

CR Subject Classification (1998): H.2, H.3, H.4, C.2, I.2.11, D.2.12, D.4.3, E.1

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-540-71660-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-71660-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12042593 06/3180 5 4 3 2 1 0

Preface

The aim of the International Workshop on Databases, Information Systems and P2P Computing was to explore the promise of P2P to offer exciting new possibilities in distributed information processing and database technologies. The realization of this promise lies fundamentally in the availability of enhanced services such as structured ways for classifying and registering shared information, verification and certification of information, content distributed schemes and quality of content, security features, information discovery and accessibility, interoperation and composition of active information services, and finally market-based mechanisms to allow cooperative and noncooperative information exchanges.

The P2P paradigm lends itself to constructing large-scale, complex, adaptive, autonomous and heterogeneous database and information systems, endowed with clearly specified and differential capabilities to negotiate, bargain, coordinate and self-organize the information exchanges in large-scale networks. This vision will have a radical impact on the structure of complex organizations (business, scientific or otherwise) and on the emergence and the formation of social communities, and on how the information is organized and processed. The P2P information paradigm naturally encompasses static and wireless connectivity and static and mobile architectures. Wireless connectivity combined with the increasingly small and powerful mobile devices and sensors poses new challenges as well as opportunities to the database community. Information becomes ubiquitous, highly distributed and accessible anywhere and at any time over highly dynamic, unstable networks with very severe constraints on the information management and processing capabilities. Which techniques and data models may be appropriate for this environment, and yet guarantee or approach the performance, versatility and capability that users and developers come to enjoy in a traditional static, centralized and distributed database environment? Is there a need to define new notions of consistency and durability, and completeness, for example?

The workshop concentrated on exploring the synergies between current database research and P2P computing. It is our belief that database research has much to contribute to the P2P grand challenge through its wealth of techniques for sophisticated semantics-based data models, new indexing algorithms and efficient data placement, query processing techniques and transaction processing. Database technologies in the new information age form the crucial components of the first generation of complex adaptive P2P information systems, which are characterized by their ability to continuously self-organize, adapt to new circumstances, promote emergence as an inherent property, optimize locally but not necessarily globally, deal with approximation and incompleteness. This workshop also concentrated on the impact of complex adaptive information

systems on current database technologies and their relation to emerging industrial technologies such as IBM's autonomic computing initiative.

The workshop brought together key researchers from all over the world working on databases and P2P computing with the intention of strengthening this connection. In particular, the workshop series emphasizes discussions about methodologies, models, algorithms and technologies related to data management and P2P systems. Researchers from other related areas such as distributed systems, networks, multi-agent systems, artificial intelligence and complex systems are also invited. We seek high-quality and original contributions on the following non-exhaustive list of topics:

- Data models and query languages for P2P systems
- Data placement and query answering in P2P systems
- Indexing, caching and replication techniques for P2P systems
- Transaction management for P2P systems
- Metadata management in P2P systems
- Dynamic schema integration, interoperation
- Emergent semantics and evolution of ontologies in P2P systems
- P2P systems and the Semantic Web
- Wireless and mobile data dissemination, delivery, replication and synchronization in P2P systems
- Scalability, coordination, robustness and adaptability in P2P systems
- Information usage in P2P mobile ad-hoc networks
- Self-organization and emergent behavior in P2P data management systems
- E-commerce and P2P computing
- Participation and contract incentive mechanisms in P2P Systems
- Computational models of trust and reputation
- Community of interest building and regulation, and behavioral norms
- Intellectual property rights in P2P systems
- Resource allocation in P2P systems
- Scalable data structures for P2P systems
- Scalable infrastructure for discovery and composition of P2P services, service definition language and filtering
- Market-based mechanisms for the exchange of information services and resources allocation in P2P systems
- Knowledge discovery and P2P data mining
- P2P-oriented information systems
- Complex adaptive information systems
- Information ecosystems and P2P systems
- Security and privacy in ubiquitous P2P systems
- Grid computing infrastructure based on the P2P paradigm
- Multidisciplinary approaches to P2P systems
- Legal issues in P2P networks

This volume is the post-proceedings of DBISP2P 2005 and 2006, the 3rd and 4th International Workshop on Databases, Information Systems and P2P

Computing,[1]. Both editions were held in conjunction with the International Conference on Very Large DataBases (VLDB), the 2005 edition was in Trondheim, Norway (August 28–29, 2005), while the last one was in Seoul, South Korea (September 11, 2006). The volume contains the papers presented at the workshop, fully revised to incorporate reviewers' comments and discussions; moreover, it includes an invited paper.

The volume is organized according to the following sessions held in the two editions:

Third Edition

- Knowledge Discovery and Emergent Semantics
- Query Answering and Overlay Communities
- Indexing, Caching and Replication Techniques
- Complex Query Processing and Routing
- Semantic Overlay Networks
- Services, Agents and Communities of Interest

Fourth Edition

- Data Placement and Searching
- Semantic Search
- Querying Processing and Workload Balancing
- Continuous Queries and P2P Computing

We would like to thank the invited speakers of the third and fourth edition, respectively, Karl Aberer, full professor at EPFL-IC-IIF-LSIR, for the talk on "Managing trust in distributed environments" and Vana Kalogeraki, assistant professor at the University of California, for the presentation of the work on "Middleware for reliable real-time sensor data management".

We express our deepest appreciation to the invited panelists of the third edition, Karl Aberer, Sonia Bergamaschi, Witold Litwin and Pavel Zezula, who conducted the panel on the theme "Semantic search: is it any new issue in P2P systems ?"

As far as the number of papers is concerned, in the third edition we received 39 contributions, out of which 12 were accepted as full papers and 11 as short works. In the fourth edition, which lasted only one day, among the 29 submissions, 11 works were selected as full papers and 4 as short contributions. All submissions were reviewed for scope, quality, originality and applicability by the Program Committee, to which we express our gratitude for preparing high-quality reviews in a short time. Finally, we would like to acknowledge the Steering Committee for its guidance and encouragement.

This workshop followed the successful first and second editions, which were both held in conjunction with VLDB, in Berlin (Germany) and in Toronto (Canada), in 2003 and 2004, respectively. In recognition of the interdisciplinary

[1] http://dbisp2p.ingce.unibo.it/

nature of P2P computing, a sister event called the International Workshop on Agents and Peer-to-Peer Computing (AP2PC)[2] was held in New York, USA and in Utrecht, The Netherlands, respectively in 2004 and 2005, both in conjunction with the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS).

October 2006

Sonia Bergamaschi
Sam Joseph
Jean-Henry Morin
Gianluca Moro
Aris M. Ouksel

[2] http://p2p.ingce.unibo.it/

Executive Committees

Organizers of the Third Edition

Program Co-chairs	Gianluca Moro (main contact) Dept. of Electronics, Computer Science and Systems University of Bologna, Italy
	Sonia Bergamaschi Dept. of Science Engineering University of Modena and Reggio-Emilia, Italy
	Aris M. Ouksel Dept. of Information and Decision Science University of Illinois at Chicago, USA
Invited Panelists	Karl Aberer EPFL, Lausanne, Switzerland
	Sonia Bergamaschi Dept. of Science Engineering University of Modena and Reggio-Emilia, Modena, Italy
	Witold Litwin Centre d'Études et de Recherches en Informatique Appliquée Université Paris Dauphine, Paris, France
	Pavel Zezula Dept. of Computer Systems and Communications Masaryk University, Brno, Czech Republic
Web Review System	Francesco Guerra and Mirko Orsini University of Modena and Reggio Emilia, Italy

Organizers of the Fourth Edition

Program Co-chairs	Gianluca Moro (main contact) Dept. of Electronics, Computer Science and Systems University of Bologna, Italy

	Sonia Bergamaschi Dept. of Science Engineering University of Modena and Reggio-Emilia, Italy
	Sam Joseph Dept. of Information and Computer Science University of Hawaii at Manoa, USA
	Jean-Henry Morin Korea University Business School, Seoul, Korea
Web Review System	Francesco Guerra and Mirko Orsini University of Modena and Reggio-Emilia, Italy

Steering Committee

Karl Aberer, EPFL, Lausanne, Switzerland

Sonia Bergamaschi, Dept. of Science Engineering
University of Modena and Reggio-Emilia, Italy

Manolis Koubarakis, Department of Informatics and Telecommunications
National and Kapodistrian University of Athens, Greece

Paul Marrow, Intelligent Systems Laboratory,
BTexact Technologies, UK

Gianluca Moro, Dept. of Electronics, Computer Science and Systems
University of Bologna, Italy

Aris M. Ouksel, Dept. of Information and Decision Science
University of Illinois at Chicago, USA
Claudio Sartori, IEIIT-BO-CNR, University of Bologna, Italy

Munindar P. Singh, Dept. of Computer Science
North Carolina State University, USA

Program Committee

Karl Aberer, EPFL, Switzerland
Alessandro Agostini, ITC-IRST Trento, Italy
Peter A. Boncz, CWI, The Netherlands
Silvana Castano, University of Milan, Italy
Isabel Cruz, University of Illinois, USA
Bin Cui, Singapore-MIT Alliance, Singapore
Alex Delis, Polytechnic University, NY, USA
Asuman Dogac, Middle East Technical University, Turkey
Fausto Giunchiglia, University of Trento, Italy

Francesco Guerra, University of Modena and Reggio-Emilia, Italy
Mohand-Said Hacid, Lyon, France
Manfred Hauswirth, EPFL, Switzerland
Vana Kalogeraki, University of California, Riverside, USA
Achilles D. Kameas, Computer Technology Institute, Greece
Anastasios Kementsietsidis, University of Edinburgh, UK
Manolis Koubarakis, University of Athens, Greece
Matthias Klusch, DFKI, Saarbrücken, Germany
Tan Kian Lee, National University of Singapore, Singapore
Maurizio Lenzerini, University of Rome "La Sapienza", Italy
Witold Litwin, University Paris 9 Dauphine, France
Pericles Loucopoulos, UMIST, Manchester, UK
Jayant Madhavan, University of Washington, USA
Alberto Montresor, University of Bologna, Italy
Jean-Henry Morin, University of Geneve, Switzerland
Gianluca Moro, University of Bologna, Italy
Enrico Nardelli, University of Rome Tor Vergata, Italy
Wolfgang Nejdl, Learning Lab Lower Saxony, Germany
Wee Siong Ng, Singapore-MIT Alliance, Singapore
Marìa S. Pèrez-Hernandez, Universidad Politècnica de Madrid, Spain
Jean Marc Pierson, INSA de Lyon, France
Evaggelia Pitoura, University of Ioannina, Greece
Dimitris Plexousakis, Institute of Computer Science FORTH, Greece
Rachel Pottinger, The University of British Columbia, Canada
Wolf Siberski, University of Hannover, Germany
Steffen Staab, University of Koblenz-Landau, Germany
Peter Triantafillou, RA Computer Technology Institute and University of Patras, Greece
Ouri Wolfson, University of Illinois, Chicago USA
Martin Wolpers, Learning Lab Lower Saxony, Germany
PInar Yolum, Bogazici University, Turkey
Pavel Zezula, University of Brno, Czech Republic

Additional Reviewers and Helpers

Christos Goumopoulos
Riad Mokadem
Gabriele Monti
Mirko Orsini
Sahri Soror

Preceding Editions of DBISP2P

The references to the preceding editions of DBISP2P, including the volumes of revised and invited papers, are:

- DBISP2P 2004 was held in Toronto, Canada, August 29–30, 2004. The Web site can be found at http://dbisp2p.ingce.unibo.it/2004/; the proceedings were published by Springer as LNCS volume no. 3367 and are available online at: http://springerlink.metapress.com/content/hhe7htl85kw7/
- DBISP2P 2003 was held in Berlin, Germany, September 7–8, 2003. The Web site can be found at http://dbisp2p.ingce.unibo.it/2003/; the proceedings were published by Springer as LNCS volume no. 2944 and are available online at: http://springerlink.metapress.com/content/v9fpfwe6c2t9/

Table of Contents

Third Edition

Knowledge Discovery and Emergent Semantics

Query Answering and Overlay Communities

Indexing, Caching and Replication Techniques

Complex Query Processing and Routing

Semantic Overlay Networks

Services, Agents and Communities of Interest

Fourth Edition

Invited Talk

Data Placement and Searching

Semantic Search

Query Processing and Workload Balancing

Continuous Queries and P2P Computing

Galois Connections, T-CUBES, and P2P Data Mining

Witold Litwin

CERIA, Université Paris Dauphine, Pl. du Mal. de Lattre, 75016 Paris, France
Witold.Litwin@dauphine.fr

Abstract. Galois connections are bread and butter of the formal concept analysis. They concern objects with properties, represented typically as a single-valued (.true or null) binary attributes. The closed sets and Galois lattices are the most studied connections. We generalize them to the relational database universe with the multi-valued domains. We show interesting queries that appear from, hard or impossible with SQL at present. As remedy, we generalize CUBE to a new operator we call θ-CUBE, writing T-CUBE It calculates the groups according to all the values of the θ operator popular with the relational joins. We show also the utility of new aggregate functions LIST and T-GROUP. In this context We finally discuss scalable distributed algorithms in P2P or grid environment for T-CUBE queries Our proposals should benefit to both: the data mining and the concept analysis over many objects.

1 Introduction

The formal concept analysis studies the relationship between objects and the properties in a space of objects and properties. The Galois connection in this universe is a relationship among some objects and some properties. Whether an object has a property is typically indicated by a binary attribute of the object. Probably the most studied Galois connection is among a set **O** of all the objects sharing some set **P** of properties such that there is no property beyond **P** that would be also shared by all the objects in **O**. One qualifies (**O, P**) as *closed set*. The closed sets over the subsets of a set of objects sharing a set of properties can be ordered by inclusion over **P** or **O.** A popular result is a *Galois lattice*. Finding a closed set let us conclude about the maximal common set of properties. The set may be then abstracted into a concept. The lattice calculus let us see possible abstractions among the concepts.

For instance, the objects may be the students for some diploma. A property may be the final "pass" grade at a course, Fig. 1. A closed set would be any couple (*S*, *P*) such that *S* contains all the students who passed all the courses in *P*, and, for any other course, at least one of the students in *S* failed. It could be for example students 1,3,5 who were the only and all to pass courses *a,b,d,e,g*. The Galois lattice would show the inclusion connections. It could show for instance for our closed set that students 1,3 form also a close set over more courses as they share also *d*. The set of students passing all the exams (one extremity of the lattice) may or may not be empty. Likewise, there may or may not be the course that all students pass (the other extremity). The dean may obviously be interested in mining the resulting (Galois) connections, i.e., searching for some closed sets or selected parts of the lattice. Obviously if a close set

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 1–12, 2007.

includes many more students and much fewer courses than any other, these courses are perhaps a little too easy.

There were interesting applications of the concept analysis to databases, e.g., to mine for ISA relationship, [12]. Our example shows however also that the theory of Galois connections over binary attributes is intrinsically of limited value for that field. After all, courses usually have multi-valued grades, e.g., 0 to 20 in Dauphine. It is a truism to say that relational databases deal almost exclusively with multi-valued attributes. If the concept analysis should apply to databases at a larger scale, there is a need for the related generalization of its universe of discourse. The need may obviously concern applications to other domains. The latter triggered already two generalization attempts, [4], [11], and a trend called *scaling* that instead attempt to map the multi-valued attributes into binary ones. All these proposals were not specific to databases and, as we show later, were in fact too limited for our goal. For instance, the scaling would typically require a change to the database schema under consideration. The proposals in [4] would be limited to attributes with totally ordered domain, and, even in this case, would present other important limitations. Like [11],although for other reasons.

We therefore propose a different generalization, tailored specially for the database needs. As the result, we may mine for the Galois connections using SQL. We mine for the closed sets especially, through a perhaps surprising relationship to the concept of a grouping in SQL queries through the popular GROUP BY, CUBE etc. operators. As the result, we propose a generalization of these operators and a new aggregate function. The database mining gains then new kinds of queries inherited from the work on the concept analysis, which are hard or impossible to achieve with SQL at present. In turn, the concept analysis inherits the database techniques for mining large collections of data. These should help that domain, whose algorithms are basically limited in practice at present to relatively small collections only, i.e., dozens of objects usually and hundreds at most.

To give an idea of our proposal, observe that the notion of sharing a multi-valued property may be given various meanings. Observe further that the basic operation in the relational database universe of discourse, that is an equi-join, may be meant as a property sharing. The meaning is that the values of the join attributes are equal through the "=" comparison operator. More generally, the sharing may concern any operator in the set $\theta = \{=, \leq, <, <>, \geq, > \}$. This is the idea in the θ-joins. The meaning of "≤" operator for instance is that an object with a property symbolized by a join attribute value v, shares this property with any other object whose value v' of the join attribute is such that $v \leq v'$. Joins seen under this angle are also some Galois connections. Notice in particular, in the light of the comments to [4] above, that the use of operators "=" and "<>" does not require a total order on the domain of the join attributes.

Next, observe that the popular GROUP BY operator, also explores the '=' Galois connection among the objects (tuples) that it groups over the given set of (grouping) attributes. More generally, the popular CUBE operator groups over any subset of the set of the grouping attributes, but also along the same '=' connection, [8]. A group T over some of the attributes under the CUBE, together with every other column forming a group over some other attributes of the tuples in T, if there are any such

columns, form a closed set in this sense. Hence, CUBE is the basis for some closed sets computation and analysis. Subsequently, one may order the closed sets by inclusion, getting a generalized Galois lattice.

The generalization of the grouping operators we propose below follows this approach. In short, the new operators group for the other θ-operators as well. We may then calculate the closed sets also over the other θ values. The idea is somehow similar to that in θ-joins with respect to the '='-join (equi-join). The new operator generalizing the CUBE for instance, groups as CUBE, but over the other θ values as well, even different ones combined in a single grouping operation. We term it T-CUBE and read θ-CUBE. It. We act similarly with respect to GROUP BY, ROLLUP and GROUPING SETS, proposing the T-GROUP BY etc. The Galois lattices can be further formed by inclusion over the closed sets.

Like CUBE, T-CUBE does not show explicitly the tuples it has grouped to form a closed set. We combine for this purpose the operator with the LIST aggregate function [14]. To show the attributes of the closed set, we propose a new aggregate function we call T-GROUP. The function renders an aggregated value at an attribute iff given tuples form a group at that attribute over given θ. Otherwise, the result of T-GROUP is null. As typically for the databases, we can further combine the mining of a closed set with that of the attributes not in the set. We can apply any SQL aggregate function to these attributes. Likewise, we can apply aggregate functions other than T-GROUP to the attributes forming the closed sets.

One knows well that CUBE operator is hard to evaluate. The evaluation of T-CUBE may be obviously be even harder. A variety of algorithms for CUBE have been proposed and could serve for '=' closed sets calculus. We propose below algorithms applying also to the other θ 's. They use the scalable distributed calculus of P2P or grid type that appears the most practical. One algorithm uses a distributed hash scheme valid for all θ's. Another scheme we termed SD-ELL is specific to θ = '≥'.

Below we recall the formal definitions of the closed sets and of Galois lattices. Next we define the syntax and semantics of T-CUBE. Finally, we discuss the computation of our closed sets. We conclude with the direction for the future work.

2 Galois Connections

The Galois connections are the mathematical framework for extracting concepts and rules from other concepts. The closed sets and the Galois lattices are the most studied connections. To recall the basics, a *formal context* is a triplet (O,A,I), where O is a set of *objects*, A is a set of *attributes*, and I is a binary relation between A and O ($I \subseteq O \times A$). The notation $o\ I\ a$, $o \in O$, $a \in A$, indicates that $(o,a) \in I$. For $O_1 \subseteq O$, let O_1' be the set of the common attributes to all these objects, i.e., $O_1' = \{a \in A \mid o\ I\ a$ for each $o \in O_1\}$. Vice versa, for $A_1 \subseteq A$, let A_1' be all the objects verifying all the attributes of A_1, i.e., $A_1' = \{o \in O \mid o\ I\ a$ for each $a \in A_1\}$. A *Galois connection* (GC) is the pair of mappings that we denote (f, g), over P(O) and P(A), where: f: $O_1 \rightarrow O_1'$ and g: $A_1 \rightarrow A_1'$. The couple (O_1, A_1) where $O_1 = A_1'$ and $A_1 = O_1'$ constitutes a *closed set*, (CS), also called *concept*. The set O_1 (vs. A_1) is called *extent* (vs. *intent*) of (O_1, A_1).

O \ A	a	b	c	d	e	f	g	h
1	1	1	1	1	1	1	1	
2	1	1	1	1	1	1		1
3	1	1	1	1	1		1	1
4	1	1	1	1		1		
5	1	1		1	1		1	
6	1	1	1		1			1
7	1		1			1		

Fig. 1. A formal context C and its Galois lattice

We usually represent an object *o* as a tuple with an OID attribute and the binary attributes *a* that evaluate to *.true* or '1' iff (*o I a*). Thus *a* = 1 means that *o* possesses the property represented by *a*. Fig.1 is an example of a formal context represented in this way, with *O* = (1,2,3,4,5,6,7) , e.g. students and *A* = (a,b,c,d,e,f,g,h), e.g., exams.

The (double) inclusion relation we denote as "≤ " defines furthermore a partial order of all the concepts over a formal context:

$(O_1,A_1) \leq (O_2,A_2) \Leftrightarrow O_1 \subseteq O_2$ and $A_1 \supseteq A_2 \Leftrightarrow (O_1, A_1)$ is a *subconcept* of $(O_2,A_2) \Leftrightarrow (O_2, A_2)$ is a *superconcept* of (O_1,A_1).

A sub-concept corresponds thus to a concept specialization, with possibly less objects, but with possibly more .true attributes in common (common properties). A super-concept corresponds to the opposite, i.e., realizes an abstraction of its sub-concepts. A *Galois lattice* (GL) is finally the set of all the concepts, ordered by the relation ≤. Fig.2 shows the GL over the formal context at Fig. 1. Notice that we represent the sub-concepts above the super-concepts. Several algorithms determine the CSs, or a GL over a formal context [7] [10]. In practice, these algorithms work only for contexts of a few hundreds of objects at most, i.e., small from the database mining perspective.

3 Multi-valued Galois Connections

A relational database contains tuples whose attribute domains are typically multi-valued. This makes the notion of a GC and the related apparatus of the concept analysis theory basically inappropriate for the database management. The starting point of any application of these notions to the databases has to be some generalization to the multi-valued domain. We proceed towards this goal as follows. First, our objects of interest are the (relational) *tuples* We further define property sharing as follows. As we mentioned, the concept of a θ-join already provides some meaning of such sharing for the join attributes. We consider further a *grouping* operator θ, $\theta \in \{=, \leq, <, <>, \geq, >\}$. Let $a\ (t)$ be an attribute value of tuple t. Then, given θ, a set T of tuples t share the property defined by one or more following attribute values c:

$$\theta = \text{'='} \Rightarrow T = \{t : a\ (t) = c\},\ \theta = \text{'}\leq\text{'} \Rightarrow T = \{t : a\ (t) \leq c\}...$$

Accordingly, consider that we choose c and θ at some attribute x. We say that T forms a *θ-group* for c at x, if every t in T shares c value according to the above rules. In the vocabulary of θ- joins, a tuple t belongs to a group formed at x iff $t.x\ \theta\ c$ =.true. We then call c a *common property* of group T. As usual, one can define the grouping at several attributes, as the intersection of the groups at each attribute chosen.

Given for each x a choice of θ and of some c, present in some tuple, we say that T and some set A' of its attributes form a *θ-closed set* (T, A') if T contains all and only tuples forming the groups for each attribute in A', and there is no group involving all the tuples of T on an attribute not in A'. We abbreviate the term generalized closed set to θ-CS. The notions of a *multi-valued θ-Galois connection* (θ-GC) and of a *multi-valued θ-Galois lattice* (θ-GL) define accordingly.

The choice of c for defining a θ-group and a θ-CS can be the usual one for θ = '='. For the other θ values, we generalize the usual rule. Let some tuple s, termed *seed*, be in the table subject to the grouping. Let $s.x$ be the value of attribute x of s. Let T be the group formed using some θ values at some attributes of s. Let $\theta(x)$ denote θ used at attribute x. Then, if $s.x$ defines a group for $\theta(x)$ over tuples in T, we set $c = s.x$. We form A' from all the groups formed in this way over s.

We represent (T, A') for a relational calculus basically as a tuple $t = (T, v(A))$, where for each attribute $x \in A'$, $t.x = c$ otherwise $t.x$ is null what we denote as $t.x$ = '_'. We qualify this representation of a θ-CS as *basic*. Observe that our rule for $v(A)$ defines in fact a specific aggregate function, in the common, relational database, sense. We consider therefore also the *aggregate* representations of a θ-CS, where one forms $t.x$ using a different aggregate function.

Example 1. We now illustrate our notions of θ- groups and of θ-CS, and their interest for the database mining. Hence, consider Table 1, tabulating relation S of students at Dauphine with key S#, and with the grades per course *a,b…h*. We wish to find any students with the grades of Student 1 for course *a*. If so, we wish to mine for the courses where these students also share the respective grade student 1.

The θ = '=' at all the attributes of S answers our query. Students {1, 2} form then the group at a sharing c = 12. They form also the groups at f, for c = 13. The couple ({1, 2}, {a = 12, f = 13}) forms '='-CS, responding to our query. We aggregate this θ-CS to the tuple ({1, 2},12,_ ,_,_,_,12,_,_}) compatible with the original table and being the basic representation of the CS.

It should be seen that no current SQL dialect expresses the discussed query. Notice also that the calculus would apply to the single-valued attributes, e.g., to pas an exam for a student. It would lead to the usual CSs.

The similar query, but for any student and any course produces several θ-CSs :

({1,2,4,6}, {f = 13}), ({1,3,4}, {g = 12}), ({4, 6}, {b = 14, f = 13}), ({1, 4}, {d = 10, f = 13, g=12})…

We now mine for every student at least as good as student 1 for course a, and perhaps for other courses that we wish to mine therefore as well. We apply θ = '≥' at all the columns, perform the grouping on column a, for a = 12, and find the groupings at other columns. We get the following θ-CS:

({1, 2,4,5},12,_ ,6,10,_,_,_,6})

Table 1. Table S

S#	A	b	c	d	e	f	g	h
1	12	16	6	10	12	13	12	6
2	12	12	8	12	11	13	11	8
3	10	13	16	14	11	8	12	10
4	13	14	11	10	13	13	12	14
5	17	10	10	14	13	10	14	12
6	0	14	3	8	4	13	11	10

Next, we wish the least grade of these students for every course, i.e., the minimal competence level they share. We use at each column the aggregate function $c = \min_T(x)$, where T denotes the students in the group of student 1 over a. Our result is now:

$$(\{1, 2,4,5\},12,10 ,6,10,11,10,11,6\}).$$

The result meets the definition of a CS over multi-valued attributes proposed in [4], termed a *generalized* CS. The example illustrates that the definition in [4] is a specific case of ours, limited to a '≥'-CS, in our vocabulary.

Yet alternatively, for our group {1,2,4,5}, and column b for instance, one could mine for avg (b). The result $b = 13$ would give a different measure of the difference between the grade of student 1 and of the other students for this course. Likewise, we may ask for students who are at least as good as any other student for any set of courses. The set of all '≥'-CSs will be the answer. In this example its cardinal is in fact 39. The calculus for the actual course, with about 20 students, showed over 6.000 θ-CSs. This hints about the difficulties for the efficient processing.

Next, we may have interest in the students at most as good as student 1 on a and possibly at some others courses, or better etc., [15]. We would use θ = '≤' or θ = '>'at all columns getting, in the latter case, CS= ({4,5},12,_, _,_,12, _,_,_}). Again, no current SQL dialect would express the discussed queries.

4 T-CUBES

The notion of grouping is popular with the database management since the introduction of GROUP BY operator. It calculates a single grouping over some columns. It uses Θ = '=' in our vocabulary. Operators for multiple groupings followed. Among them, CUBE is the most general at present. We recall that CUBE ($a_1 \ldots a_n$) calculates all the groups over all the subsets of $\{a_1 \ldots a_n\}$ for Θ = '=' in our vocabulary. The CUBE appears as the natural basis for the calculus of θ-GCs in general, and of θ-CSs specifically. Notice that the notion of θ-CS appears as a natural 2-d extension of that of a group. It operates indeed not only on the tuples common to the group, but also on the common properties.

For our purpose, one has to first generalize CUBE operator so to accept the other values of θ as the grouping criteria. Next, one has to accompany it with aggregate

functions that show the θ-CS composition, with respect to both the tuples and the attribute values. Notice that CUBE by itself does not provide the group composition at present. Our proposal is as follows.

We consider a new operator we call T-CUBE. The operator allows for any θ at any of the columns it should operate upon. It performs the multiple groupings accordingly, e.g., those in *Example 1*. For θ = '=', T-CUBE reduces to CUBE. Over the groups provided by T-CUBE, we use basically two following aggregate functions to form the θ-CS tuple:

1. The LIST (I) function, [14], where I indicates the attribute(s) chosen to identify each tuple composing the θ-CS.
2. The T-GROUP (θ, x) function. This function tests whether column x forms a group, given a seed. If so, it renders c as the result, otherwise it evaluates to '_'. In the latter case, it may invoke another aggregate on the column.

For convenience, we consider T-GROUP (x) = T-GROUP (=, x). We also consider T-GROUP implicit for every attribute without a value expression named in the Select clause and T-CUBE. T-GROUP inherits then for each attribute its θ under T-CUBE. The function is also implicit for attributes without a value expression and not in the T-CUBE, provided T-CUBE uses only one θ value for all the grouping attributes.

We further consider that T-CUBE allows for a restriction on attributes X that is enforced only when X are processed as the grouping attributes. For instance, in our motivating example, one may indicate that, for query Q, only the values of a above 0 are of interest for θ = '≥' groups to form at this attribute. When T-CUBE evaluation in Q leads in turn to the grouping on attribute b only, again just for instance, then Q concerns potentially all the values of a. Hence, we may a get a θ-CS involving student 6 with 0 grade.

Finally, as we allow for $\theta \neq$ '=' in T-CUBE, we implicitly generalize the GROUP BY, ROLLUP… to, respectively, T-GROUP BY, T- ROLLUP... clauses.

Example 2. T-CUBE (a,b,c) means CUBE (a,b,c). Next, T-CUBE ($\geq$,a,b,c) means the groupings using θ = '≥' at all three attributes. To the contrary, T-CUBE ($\geq a, =b, >c$) means the groupings using the different θ values at each attribute. Finally, T-CUBE (=,a : a <> 0,b,c) limits the interest to groups formed at a only for a not 0.

Next, the following query produces mining (1) requested in *Example 1*. The function T-CUBE is implicit at attributes a…h.

```
Select List (s#),a,b,c,d,e,f,g,h
from S
T-CUBE (a)
having a = (select a from S where s# = 1)
```

Here the functions T-GROUP (=, a)… are implicit. We could write the query alternatively in many ways, [15]:

The next easy query also follows up on *Example 1*. It gets the generalized CS of [4], adds to it the average values for b at each group, and restricts the mining to CSs formed from at least 2 members.

```
Select List (s#), min (a), min (b), min (c), min (d), min (e), min (f), min (g),
min (h), avg(b)
from S
T-CUBE (≥, a,b,c,d,e,f,g,h)
having count(*) > 1;
```

See [15] for more motivating examples.

5 T-CUBE Evaluation

The CUBE possibly generates many and big groups [8]. T-CUBE may obviously generate even larger groups. Efficient algorithms for T-CUBE queries are therefore crucial. Besides, with respect to our aggregate functions, the single-attribute LIST function, limited in this way with respect to its semantics in [14], but sufficient here, is standard at SQL Anywhere Studio. This DBMS does not however offer CUBE. Tentative implementations of single-attribute LIST as user-defined functions were also attempted under Oracle that has CUBE. In turn, we are not aware of any implementation of T-GROUP function as yet. Fortunately, one may expect the task of creating T-GROUP as a user-defined function rather simple. These are already in the commercial word, e.g., Oracle and DB2. SQL Server should get user-defined functions in the next release. DB2 and SQL Server have CUBE

With respect to T-CUBE evaluation a scalable distributed architecture for grid or P2P environment seems the best basis at present, [15]. We consider that the *dispatcher* node (peer) coordinates the query evaluation. It calculates any restrictions, and distributes, possibly uniformly, the remaining calculus of θ-CSs to the P2P *application* nodes. Only one node possibly determines a θ-CS. The number of nodes involved is chosen so that each one has a reasonably small and about the same number of θ-CSs to find. A Scalable Distributed Data Structure (SDDS) , e.g. an LH* file, [13], stores the (distributed) result, possibly in the distributed RAM, for orders of magnitude faster access than to disks. Each θ-CS has a key, unique regardless of, perhaps, its duplicate production by different nodes. A duplicated θ-CSs if any, is disregarded in this way when the node calls its local SDDS client component with. Each node reports to the dispatcher once it terminates. The dispatcher performs any post-processing or returns the control to the user.

To design the algorithms, one may start either from the database heritage or from the formal concept analysis heritage. The former approach implies the reuse of GROUP BY and CUBE algorithms, of some of a general relational query evaluation, and of some for an SDDS. The was a large body of work in these domains, [9], [8], [16]. The latter direction implies the algorithms already proposed for computing binary CSs, [10]. More specifically, it involves the algorithms for generalized CSs, [4], [5]. We now examine both directions

5.1 Database Heritage

We target a general algorithm, applying to every θ. Algorithms valid for some θ only, θ = '=' especially, are thus out of scope here. Notice however, as we mentioned, that

the major DBMSs have already the CUBE operator, making the implementation of '='-CS calculus potentially simpler. As for θ-joins, we consider some nested-loop approach. The grouping calculus (i) visits thus sequentially all the tuples concerned, and (ii) for each tuple, it examines all the possible subsets of the attribute values. For each choice, termed the current *seed,* it visits other tuples in *T*, perhaps all, to determine the existing groups.

For instance, if we start with student 1, the successive seeds could be (12,_ ,_,_,_,_,_,_}), (_,16 ,_,_,_,_,_,_}), (12,16 ,_,_,_,_,_,_}), (_,_ ,6 ,_,_,_,_,_}), (12,_,6 ,_,_,_,_,_})… The 1st seed means the grouping calculus over $a = 12$. Next seed leads to the grouping over $a = 12$ and $b = 16$ etc. Different seeds may produce different θ-CSs. They may alternatively produce a duplicate. The latter case results for instance from (12,_ ,_,_,_,_,_,_}) and of (_,_ ,_,_,_,13,_,_).

Our SD calculus replicates *T* on $N > 1$ nodes, numbered 0,1…*N*-1. The choice of *N* should let each node to compute θ-CSs only for a fraction of all the seeds. The subsequent determination of *N* may be *centralized* or *distributed*. The former defines *N* upfront, given the estimate of the number of seeds to process and perhaps of θ-CSs wished to calculate per node. The latter lets the nodes to determine *N* dynamically.

We hint here about the centralized distribution only, see [15] for more. We form the seed as a large integer *s* concatenating the grouping attributes. Each application node generates the seeds or gets them pre-computed in a file, to avoid the duplicates. Then, all nodes use the same hash function *h*, mapping *s* on some number $a = 0,1…N$-1, where *N* is a parameter of *h*. Node *a* calculates then the θ-CS only for *s* such that $h(s) = a$. The dispatcher chooses *N* so to divide enough the interval [0, max(*s*)], where max(*s*) is the actual or a maximal possible *s*, e.g., with all the bits equal to 1. Each node attempts to store each θ-CS computed, with its key and all the aggregated values into SDDS, through its client. The key is the value of θ-CS also seen as integer. This eliminates the duplicates, if any, as we have indicated.

5.2 Concept Analysis Heritage

Several algorithms for CS and GL calculus for binary attributes exist, e.g. the Ganter's classic, [7]. As we mentioned, the universe of discourse of these algorithms makes them unfit for our database mining purpose. Most calculate a GL that is the issue we do not address here (yet). Whether some algorithms can be generalized to our goal remains an open issue.

The concept analysis community has of course noticed the multi-valued attributes. However the main direction to formally deal with, called *scaling*, was apparently simply to conceptually map such attributes into the binary ones, [11]. The scaling also seems unfit for our purpose. An alternative and exclusively formal approach to multi-valued GLs was suggested in [11]. It starts with a generic concept of *description*. This one assigns to an object set-valued attribute values, in the way somehow similar to the way proposed by the symbolic objects approach, [3]. It then orders the objects into a GL according to the inclusion of the descriptions (called *precision* of the descriptions) Our definition of GCs on the basis of the values of the θ-operator is "orthogonal" to the formalism of [11]. That formalism will be perhaps of more use for the databases if the set-oriented attributes become more popular.

Another generalization attempt, we already somehow discussed, apparently unknown to [11], is that in [4], together with the subsequent algorithms in [5]. In our approach, these proposals are all limited to the '≥'-CSs. In contrast, [4] mentions generalization directions we do not deal with, e.g., towards the fuzzy sets. Next, the proposed algorithms are not designed for the groupings with restrictions. They generate the full sets of '≥'-CSs. Already for quite a few tuples and attributes, these sets are in millions of CSs, [15].

Among these algorithms, one termed ELL, generating the '≥'-CSs only, i.e., without their GL, appears the fastest, [2]. ELL has two formulations, termed respectively *iterative* and *recursive*. A scalable distributed version of the iterative ELL was proposed in [1]. The idea was to partition the set of object or of attributes over the application sites. Unfortunately, as we have found, the calculus needed then the inter-application node communication, close to gossiping for every CS generated. The messaging cost of the algorithm was consequently prohibitive, already for even dozens of objects only.

In the wake of the study to overcome this limitation, the recursive ELL appeared an alternate basis for the scalable distributed computation. The "key to the success" was to replicate T[1]. This, - to offset the trouble in [1] that appeared largely due to the partitioning of T instead. We have termed the new algorithm SD-ELL. Like the centralized recursive ELL, SD-ELL recursively splits the computation C of the set of the CSs over T into more and smaller computations of subsets. It then distributes the computations with the copies of T on the application nodes. In the nut-shell, the SD-ELL dispatcher first picks up any tuple, let us say Student 1 in our Table S. It then splits C into C_1 that calculates all the CSs that contain 1, and $C_{\sim 1}$ that does the opposite. C_1 involves in particular the calculus of the CS, let it be A_1, generated by 1, i.e., containing all and only tuples sharing all the values in 1. Technically, A_1 contains thus all and only the tuples with the attribute values greater or equal to those in 1. The dispatcher calculates A_1 by itself. In our example, we have simply $A_1 = \{1\}$.

If only two nodes are available, i.e., $N = 2$, then the dispatcher sends the replicated T and the requests to perform the respective computations to these nodes. Each node uses then locally a slightly modified version of the interactive ELL. The nodes collect the calculated CSs in a (common) SDDS file. With luck, the CPU load of each node is about ½ of a single node. If it is too much or there are more volunteering nodes, the dispatcher decomposes the tasks further. To decompose C_1, the dispatcher picks up any element in the set T/A_1. This set contains all the tuples that can lead to a CS containing 1. Those in A_1 other than 1 obviously cannot.

The dispatcher could pick up thus Student 2. C_1 partitions then recursively into (i) task $C_{1,2}$ computing all the CSs that contain both 1 and 2, and (ii) task $C_{1,\sim 2}$ that contains CSs with 1 alone. Each task goes to an application node, again along with the replicated T. Now, with the uniform distribution luck, both task may end up calculating about ½ of the CSs devoted to C_1. There is also again the CS, let us call it $A_{1,2}$ that the dispatcher may calculate by itself. $A_{1,2}$ contains all the tuples sharing the properties common to both 1 and 2. Actually, $A_{1,2} = \{1,2,4\}$.

On the $C_{\sim 1}$ task side, the dispatcher can first determine the actual set of tuples that cannot generate a CS with 1. This set, let it be R_1, has to have all the tuples with the attribute values that are at most equal to the respective ones in 1. Actually we have

[1] Appeared during brainstorming on the alternatives to [1] with G. Levy & F. Baklouti.

$R_1 = \{1\}$. The dispatcher may further decompose $C_{\sim 1}$ as above, except that it uses R_1 instead of A_1. That is, it first picks up any element in the set T/R_1, let us say 2 again. Then it defines the tasks $C_{\sim 1,2}$ calculating all the CSs that contain 2, but not 1, and its "sibling" $C_{\sim 1,\sim 2}$ that contains neither tuple. The tasks can be sent to two nodes, again along with T.

The replication of T results from a subtlety of the above dichotomies. The calculus of CSs may lead to a kind of collisions for SD-ELL, although not the same as for the hashing calculus. The collision may concern a task that should not include a tuple, e.g., $C_{1,\sim 2}$ with its sibling, $C_{1,2}$ here. A set generated at the former, intended as a CS, may happen to be in fact only a strict subset of a CS at the latter, hence not a CS at all in fact. The difference may be a tuple to avoid at the former, student 2 here. Any CS involving this tuple should be, and effectively is, generated by the sibling, by its definition itself. The task generating the subset should therefore be able to recognize the collision locally. It can, by testing, for each generated set, each tuple to avoid specified in the task definition, e.g., 2 in our case. The test should show that the attribute values of the tuple do not share the common properties of the set. We recall that these are the minimal value of each attribute over the set. Having a copy of T, let every node to know locally any such values. Iff the test fails, the node recognizes the collision and drops the set.

In our example for instance, $C_{1,\sim 2}$ generates the set {1,3,4,5,6}. The properties shared by all the elements of the set are (0,10,3,8,4,8,11,6). The result is not a CS. Student 2 indeed also shares these properties. Hence, 2 must belong to the resulting CS, involving thus the set {1,2,3,4,5,6}. However, task $C_{1,2}$, by its definition, already calculates every CS involving 2. Hence, set {1,3,4,5,6} at the node with $C_{1,\sim 2}$ is a collision. The node of $C_{1,\sim 2}$ identifies the fact by the local test of the attribute values of 2, using its copy of T. As the result, it drops the set {1,3,4,5,6}.

All together, provided again the uniform distribution of the values, the 2nd level of decomposition of both C_1 and $C_{\sim 1}$ on four nodes, reduces the calculus time to about ¼ of that on a single-node. The dispatcher can continue to decompose each task recursively. Each decomposition reduces the remaining set of tuples, after chopping off the next A or R. Hence, the dispatcher can decompose and possibly reduce the overall execution time, as long as (i) the remaining set of tuples is not empty, and/or (ii) there are spare P2P nodes. One can possibly run into thousands of nodes. *A P2P or grid net, seems in this way the only way towards the practical response time for a larger T, characteristic of the database mining.*

The SD application of ELL (SD-ELL) appears a potential basis for '≥'-CSs calculus, with respect to both the distribution and the calculus schemes. Provided however that one can re-engineer it for the restrictions. See [15] for more on SD-ELL.

6 Conclusion

T-CUBE operator, coupled with the LIST and T-GROUP aggregate functions, appears useful for data mining. through the multi-valued Galois connection analysis in larger collections. The P2P and grid environment appears the most appropriate implementation support. Known algorithms for the CUBE processing may apply to T-CUBE for specific θ values.

Future work should concern deeper analysis and implementation of the described T-CUBE evaluation algorithms. One should also implement the multi-attribute LIST and T-GROUP functions The processing of the (multi-valued) θ-GLs remains to be studied. It seems related to lattices in [6]. Finally, the operators T-GROUP BY, T-ROLL UP, T-GROUPING SETS should be studied on their own.

Acknowledgments. A grant from Microsoft Research partly supported this research. The European Commission project ICONS no. IST-2001-32429 helped with early work towards SD- ELL.

References

1. Baklouti. F, Lévy. G. Parallel algorithms for general Galois lattices building. WDAS 2003, Carleton Scientific (publ.).
2. Baklouti. F, Lévy. G. A fast and general algorithm for Galois lattices building. Submited to the Journal of Symbolic Data Analysis. April 2005, 1st revision.
3. Diday, E. Knowledge discovery from symbolic data and the SODAS software. The 4th Europ. Conf. on Principles and Practice of Knowledge Discovery in Databases, PPKDD-2000. Springer (publ.).
4. Diday. E, Emilion. R. Treillis de Galois maximaux et capacités de Choquet. Cahier de Recherche de l'Acadèmie des Sciences. Paris, t.325, Série I, p.261-266, 1997.
5. Emilion. R., Lambert. G, Lévy. G, Algorithmes pour les treillis de Galois. Indo-French Workshop., University Paris IX-Dauphine. 1997.
6. Fagin, R. & al. Multi-Structural Databases. Intl. ACM Conf. on Principles of Database Systems. ACM-PODS 2005.
7. Ganter. B. Two basic algorithms in concept analysis. Preprint 831, Technische Hochschule Darmstadt 1984.
8. Gray, J & al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Data Mining and Knowledge Discovery J.,1 , 1, 97, 29 – 53.
9. Garcia-Molina, H., Ullman, J., Widom, J. Database Systems: The Complete Book. Prentice Hall, 2002.
10. Ganter, B. Wille, R., Franzke, C. Formal Concept Analysis: Mathematical Foundations. Springer-Verlag New York, 1999, 294.
11. Gugisch, R. Many-valued Context Analysis using Descriptions. H. Delugach and G. Stumme (Eds.): ICCS 2001, LNAI 2120, 157-168, 2001, Springer-Verlag.
12. Hainaut, J-L. Recovering ISA Structures. Introduction to Database Reverse Engineering. Chapter 14. (book in preparation), 2005.
13. Litwin, W., Neimat, M.-A., Schneider, D. LH* : A Scalable Distributed Data Structure. ACM Transactions on Database Systems (ACM-TODS), 12,.96.
14. Litwin, W. Explicit and Implicit List Aggregate Function for Relational Databases. IASTED Intl. Conf. On Databases and Applications, 2003.
15. Litwin, W. Galois Connections, T-CUBES, & P2P Database Mining. 3rd Intl. Workshop on Databases, Information Systems and Peer-to-Peer Computing. VLDB 2005. Also CERIA Res. Rep. 2005-05-15, May 2005
16. Litwin, W. Scalable Distributed Data Structures. ACM 13th Conf. On Inf. and Knowledge Mangment (CIKM-2004), Washington DC. 3h Tutorial.

Querying a Super-Peer in a Schema-Based Super-Peer Network*

Domenico Beneventano, Sonia Bergamaschi, Francesco Guerra, and Maurizio Vincini

Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
Via Vignolese 905, 41100 Modena, Italy
bergamaschi.sonia@unimore.it

Abstract. We propose a novel approach for defining and querying a super-peer within a schema-based super-peer network organized into a two-level architecture: the low level, called the peer level (which contains a mediator node), the second one, called super-peer level (which integrates mediators peers with similar content).

We focus on a single super-peer and propose a method to define and solve a query, fully implemented in the SEWASIE project prototype.

The problem we faced is relevant as a super-peer is a two-level data integrated system, then we are going beyond traditional setting in data integration. We have two different levels of *Global as View* mappings: the first mapping is at the super-peer level and maps several *Global Virtual Views* (GVVs) of peers into the GVV of the super-peer; the second mapping is within a peer and maps the data sources into the GVV of the peer. Moreover, we propose an approach where the integration designer, supported by a graphical interface, can implicitly define mappings by using *Resolution Functions* to solve data conflicts, and the *Full Disjunction* operator that has been recognized as providing a natural semantics for data merging queries.

1 Introduction

Current peer-to-peer (P2P) networks support only limited meta-data sets such as simple filenames. Recently a new class of P2P networks, so called schema based P2P networks have emerged (see [1,2,3,4]), combining approaches from P2P as well as from the data integration and semantic web research areas. Such networks build upon peers that use metadata (ontologies) to describe their contents and semantic mappings among concepts of different peers' ontologies. In particular, in Peer Data Management Systems (PDMS) [2] each node can be a data source, a mediator system, or both; a mediator node performs the semantic integration of a set of information sources to derive a global schema of the acquired information.

* This research has been partially funded by the UE-IST SEWASIE project and the italian MIUR PRIN WISDOM project.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 13–25, 2007.

As stated in a recent survey [5], the topic of semantic grouping and organization of content and information within P2P networks has attracted considerable research attention lately (see, for example, [6,7]). In super-peer networks [8], metadata for a small group of peers is centralized onto a single super-peer; a super-peer is a node that acts as a centralized server to a subset of clients. Clients submit queries to their super-peer and receive results from it; moreover, super-peers are also connected to each other, routing messages over this overlay network, and submitting and answering queries on behalf of their clients and themselves. The *semantic overlay clustering* approach, based on partially-centralized (super-peer) networks [9] aims at creating logical layers above the physical network topology, by matching semantic information provided by peers to clusters of nodes based on super-peers.

In this paper we propose an approach which combines the schema-based and super-peer network approaches, that is a *schema-based super-peer* network (called SEWASIE network from the UE IST project where it was developed - www.sewasie.org) organized into a two-level architecture: the low level, called the peer level (which contains a mediator node), the second one, called super-peer level, (which integrates mediators peers with similar content). More precisely,

- a *peer* contains a data integration system, which integrates heterogeneous data sources into an *ontology* composed of: an annotated *Global Virtual View* (GVV) and *Mappings* to the data source schemas.
- a *super-peer* contains a data integration system, which integrates the GVV of its peers into an *ontology* composed of a GVV of the peers GVVs and Mappings to the GVVs of its peers.

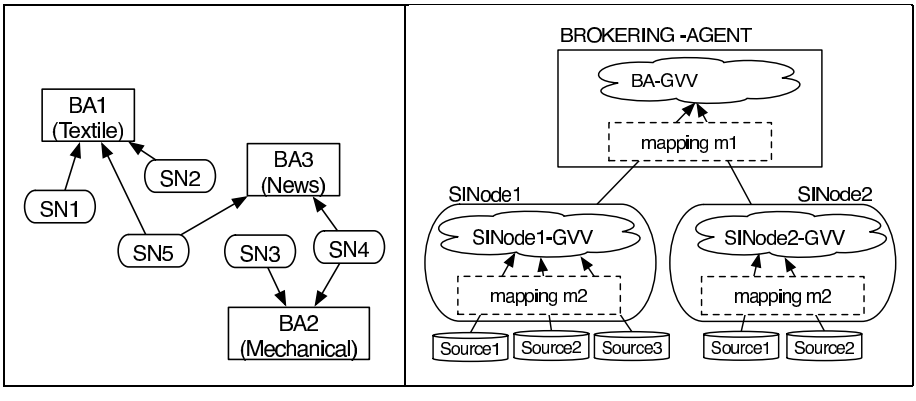


Fig. 1. (a) The SEWASIE network; **(b)** The Brokering Agent/SINodes architecture

A typical scenario of the SEWASIE network is shown in Figure 1.a, where many data peers, called `SINodes` (SN1 to SN5) are linked to different super-peers, called *Brokering Agents* (BA1 to BA3), according to their semantic content. For example, SN1, SN2, SN5 contain semistructured data sources related to the

textile domain (textile enterprises, news, categories, ...) data sources and are clustered in the same BA (BA1). The same for BA2, that refers to *mechanical domain* which contains links to SN3 and SN4. Furthermore, peer nodes may belong to more than a BA, for example SN4 and SN5 belong to BA2 and BA1 respectively and to the "news" super-peer BA3.

In this paper we propose a novel approach for querying a super-peer within a schema-based super-peer network. We focus on querying a single BA (super-peer) (for querying the SEWASIE network for more than one BA see [10,11]) and propose a method fully implemented in the SEWASIE project prototype.

The problem we faced is relevant as a BA is a two-level data integrated system then we are going beyond traditional setting in data integration.

We have two different levels of mappings (figure 1.b): The first mapping (*m1*) is at the BA level and maps several GVVs of SINodes to the GVV of the BA; the second mapping (*m2*) is done within an SINode and maps the data sources into the GVV of an SINode.

Halevy et al [12] showed that, in general, the mapping from the data sources to the BA Ontology is not simply the composition of *m1* and *m2*; Fagin et al [13] showed that second order logic is needed to express composition.

In [14,11] is proved that if *m1* and *m2* are GAV (Global as View) mappings, like in SEWASIE, the mapping is indeed the composition of *m1* and *m2*; this implies that query answering can be carried out in terms of two reformulation steps

1. **Reformulation w.r.t. the BA ontology** (mapping *m1*): this step reformulates the query in terms of the SINodes known by the BA;
2. **Reformulation w.r.t. the SINode ontology** (mapping *m2*): this step reformulates each SINode query obtained in the first step in terms of the data sources known by the SINode;.

This is the algorithm proved to be sound and complete for a two-level data integration system [14].

The paper is organized as follows. Section 1.1 gives an overview of the architecture of the SEWASIE system. In section 2, we introduce a two-level data integration system and in section 3, we define the query reformulation process for this system. In section 3.2 the agent-based prototype for Query Processing in the SEWASIE system is briefly presented. For more detailed description see [15,11].

1.1 SEWASIE Architecture

The SEWASIE network is an agent-based network developed within the UE IST SEWASIE project, and the overall architecture is shown in figure 2.

A user is able to access the system through a central user interface where (s)he is provided with tools for query composition, for visualizing and monitoring query results, and for communicating with other business partners about search results, e.g. in electronic negotiations. Within a SINode, wrappers are used to extract the data and metadata (local schemas) from the sources. The *Ontology*

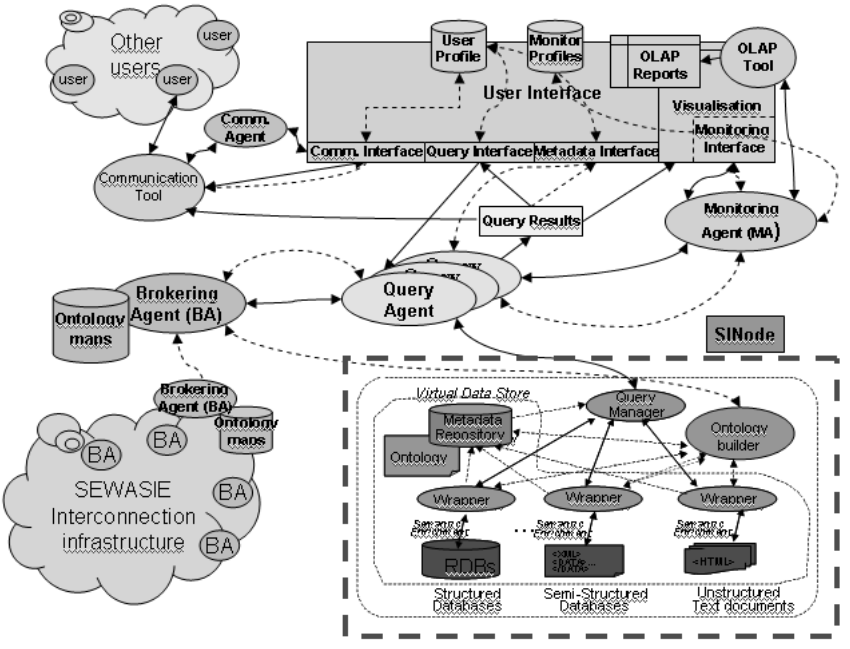


Fig. 2. SEWASIE Architecture

Builder - based on the MOMIS framework [16,17], is a semi-automatic tool to create a domain ontology as a Global Virtual View (GVV) which is annotated w.r.t. a lexical ontology (Wordnet [18], Multiwordnet).

Brokering Agents integrate several GVVs from different SINodes into BA Ontology, that is of central importance to the SEWASIE system. On the one hand, the user formulates the queries using this ontology. On the other hand, it is used to guide the Query Agents to the SINodes providing data for a query.

The SEWASIE network can have multiple brokering agents, each one representing a collection of SINodes for a specific domain. Mappings between different brokering agents may be established. A Query Agent receives the queries (expressed in terms of a specific BA ontology) from the user interface, rewrites the query in terms of the GVVs of the SINodes (in cooperation with the brokering agent) and sends the queries to the SINodes. The result is integrated and stored in a result repository, so that it can be used by the various end-user components.

For example, *Monitoring Agents* can be used to store a query result in a permanent repository. The monitoring agent will then execute the query repeatedly, and compare the new results with previous results. The user will be notified if a document has changed that fits her monitoring profile. Furthermore, the monitoring agent can link multidimensional OLAP reports with ontology-based information by maintaining a mapping between OLAP models and ontologies. Finally, the *Communication Tool* provides the means for ontology-based negotiations. It uses query results, the ontologies of the Brokering Agents, and specific

negotiation ontologies as the basis for a negotiation about a business contract. In addition, it uses several agents to support the negotiators in their decision process (e.g. by filter and ranking offers of potential business partners, or by monitoring the available resources of a company).

The SEWASIE consortium is constituted by the University of Modena and Reggio Emilia, the coordinator, which developed the Ontology Builder, the Query Agent and the agent architecture in collaboration with the University of Roma La Sapienza and University of Bolzano respectively. The user interface is a join effort of University of Roma La Sapienza and University of Bolzano.

2 The SEWASIE System

In this section, we describe the two-level data integration system.

An Integration System $IS = \langle GVV, \mathcal{N}, \mathcal{M} \rangle$ is constituted by:

- A Global Virtual View (GVV), which is a schema expressed in ODL_{I^3} [16], a modified version of the *Object Definition Language*[1]. In particular, in the GVV we have *is-a* relationships and both `key` and `foreign key` constraints.
- A set $\mathcal{N}$ of *local sources*; each local source has a *schema* also expressed in ODL_{I^3}.
- A set $\mathcal{M}$ of GAV mapping assertions between GVV and $\mathcal{N}$, where each assertion associates to an element g in GVV a query $q_{\mathcal{N}}$ over the schemas of a set of local sources in $\mathcal{N}$.
  More precisely, for each global class $C \in GVV$ we define:
  1. a (possibly empty) set of local classes, denoted by $L(C)$, belonging to the local sources in $\mathcal{N}$.
  2. a conjunctive query $q_{\mathcal{N}}$ over $L(C)$.

Intuitively, the GVV is the intensional representation of the information provided by the Integration System, whereas the mapping specifies how such an intensional representation relates to the local sources managed by the Integration System in an SINode.

A **SEWASIE system** is constituted by:

- A set of *SINodes* $\mathcal{SN} = \{SN_1, SN_2, \ldots, SN_n\}$, where each SINode is a Integration System $SN = \langle GVV, \mathcal{N}, \mathcal{M} \rangle$ such that $\mathcal{N}$ is a set of data sources.
- A Brokering Agent BA, which is an Integration System $BA = \langle GVV, \mathcal{N}, \mathcal{M} \rangle$ where $\mathcal{N} = \mathcal{SN}$, i.e., the *local* sources of BA are the SINodes.

The semantics of an Integration System, and then of the SEWASIE system, is defined in [19,11].

In many papers (see [16,17]) we described the MOMIS/SEWASIE approach for the semi-automatic building of the GVV starting from a set of local sources and giving rise to a Mapping Table (MT) for each global class C of GVV , whose columns represent the local classes $L(C)$ belonging to C and whose rows

[1] www.service-architecture.com/database/articles/odmg_3_0.html

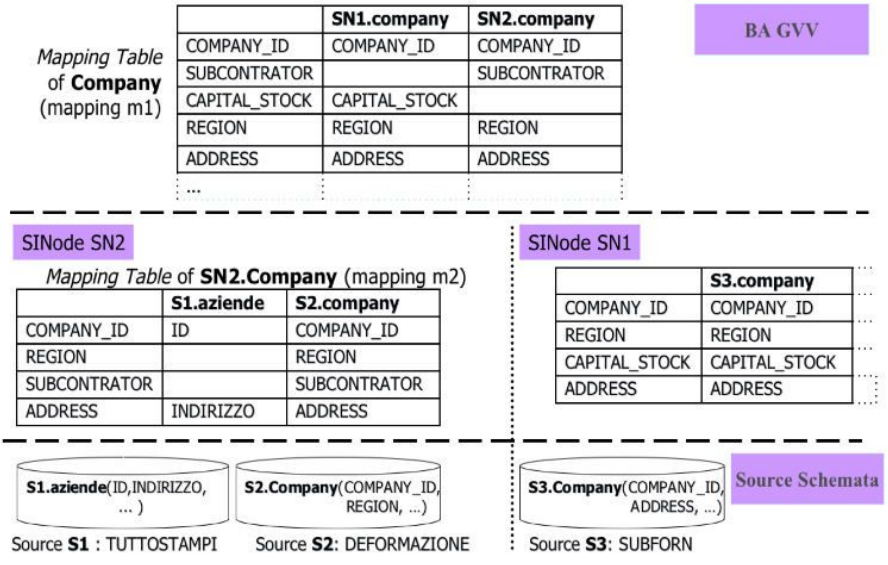


Fig. 3. Example of Mapping in the Mechanical domain

represent the global attributes of C. An element $MT[GA][LC]$ represents the set of local attributes of LC which are mapped onto the global attribute GA. As an example, figure 3 shows part of the Mapping Table of the global class `Company` (of a BA-GVV) that groups the local class `Company` of SINode1 and the local class `Company` of SINode2. At the level of a SINode, we have that (we consider SINode2), the global class `SN2.company` is mapped into the local classes `S1.aziende` and `class S2.company` (where SI and S2 are data sources).

In this paper we face and solve a new problem, that is how to define the conjunctive query $q_{\mathcal{N}}$ associated to a global class C. Our approach is the following: starting from the Mapping Table of C, the integration designer, supported by the Ontology Builder graphical interface [20], can implicitly define $q_{\mathcal{N}}$ by:

1. using and extending the Mapping Table with
   - *Data Conversion Functions* from local to global attributes
   - *Join Conditions* among pairs of local classes belonging to C
   - *Resolution Functions* for global attributes to solve data conflicts of local attribute values.
2. using and extending the *Full Disjunction* operator [21], that has been recognized as providing a natural semantics for data merging queries [22].

Data Conversion Functions

The designer can define how local attributes are mapped onto the global attribute GA by means of *Data Conversion Functions*: for each not null element $MT[GA][L]$ we define a *Data Conversion Function*, denoted by $MTF[GA][L]$,

which represents how the local attributes of L are mapped into the global attribute GA. $MTF[GA][L]$ is a function that mut be *executable/supported* by the local source of the class L. For example, for relational sources, $MTF[GA][L]$ is an SQL value expression; the following defaults hold: if $MT[GA][L] = LA$ then $MTF[GA][L] = LA$ and, if $MT[GA][L]$ contains more than one string attribute, then $MTF[GA][L]$ is the string concatenation.

$T(L)$ denotes L transformed by the Data Conversion Function; the schema of $T(L)$ is composed of the global attributes GA such that $MT[GA][L]$ is not null.

Join Conditions
Merging data from different sources requires different instantiations of the same real world object to be identified; this process is called *object identification* [23]. The topic of *object identification* is currently a very active research area with significant contributions both from the artificial intelligence [24] and database [25,26] communities.

To identify instances of the same object and fuse them we introduce *Join Conditions* among pairs of local classes belonging to the same global class. Given two local classes $L1$ and $L2$ belonging to C, a Join Condition between $L1$ and $L2$, denoted with $JC(L1, L2)$, is an expression over $L1.A_i$ and $L2.A_j$ where A_i (A_j) are global attributes with a not null mapping in $L1$ ($L2$). As an example, for `BA-GVV.Company` the designer can define JC(SN1.Company,SN1.Company) :

SN1.Company.COMPANY_ID = SN2.Company.COMPANY_ID.

Resolution Functions
The fusion of data coming from different sources taking into account the problem of inconsistent information among sources is a hot research topic [27,28,29,23,30]. In the context of MOMIS/SEWASIE we adopt the *Resolution Function* proposed in [23]. A Resolution Function for solving data conflits may be defined for each global attribute mapping onto local attributes coming from more than one local source.

Homogeneous Attributes: If the designer knows that there are no data conflicts for a global attribute mapped onto more than one source (that is, the instances of the same real object in different local classes have the same value for this common attribute), he can define this attribute as an *Homogeneous Attribute*; this is the default in our system. Of course, for homogeneous attributes resolution functions are not necessary. A global attribute mapped onto only one source is a particular case of an homogeneous attribute.

As an example, in `BA-GVV.Company` we define all the global attributes as Homogeneous Attributes except for `Address` where we used a precedence function: `SN1.Company.ADDRESS` has a higher precedence than `SN2.Company.ADDRESS`.

Full Disjunction
We want to define $q_{\mathcal{N}}$ in such a way that it contains a unique tuple resulting from the merge of all the different tuples representing the same real world object. This problem is related to that of computing the natural outer-join of many

relations in a way that preserves all possible connections among facts [22]. Such a computation has been termed as *Full Disjunction* (*FD*) by Galindo Legaria [21].

In our context: given a global class C composed of $L1, L2, ..., Ln$, we consider $FD(T(L1), T(L2), \ldots, T(Ln))$, computed on the basis of the *Join Conditions*.

The problem is how to compute FD. With two classes, *FD* corresponds to the full (outer) join: $FD(T(L1), T(L2)) = T(L1)$ `full join` $T(L2)$ `on` $(JC(L1, L2))$.

In [22] was demonstrated that there is a natural outer-join sequence producing *FD* if and only if the set of relation schemes forms a connected, acyclic hypergraph. In our context, a Global Class C with more than 2 local classes is a cyclic hypergraph, then we cannot use the algorithms proposed in [22]; the computation of *FD* is performed as follows. We assume that: (1) each L contains a key, (2) all the *join conditions* are on key attributes, and (3) all the join attributes are mapped into the same set of global attribute, say K. Then, it can be demonstrated that: (1) K is a key of $\mathcal{C}$, and (2) *FD* can be computed by means of the following expression (called *FDExpr*):

```
(T(L1) full join T(L2) on JC(L1,L2))
         full join T(L3) on (JC(L1,L3) OR JC(L2,L3))
          ...
         full join T(Ln)  on (JC(L1,Ln) OR JC(L2,Ln) OR ... OR JC(Ln-1,Ln))
```

Finally, $q_{\mathcal{N}}$ is obtained by applying Resolution Functions to the attributes resulting from *FDExpr*: for a global attribute GA we apply the related Resolution Function to $T(L1).GA, T(L2).GA, \ldots, T(Lk).GA$; This query $q_{\mathcal{N}}$ is called *FDQuery*.

3 Query Reformulation in the SEWASIE System

The query reformulation takes into account two different levels of mappings (figure 1.b): in [14,11] is proved that if *m1* and *m2* are GAV mappings, the mapping is indeed the composition of *m1* and *m2*; this implies that query answering can be carried out in terms of two reformulation steps: **1. Reformulation w.r.t. the BA ontology** and **2. Reformulation w.r.t. the SINode ontology**. These reformulation steps are similar and are defined by considering the reformulation process for an Integration System $IS = \langle GVV, \mathcal{N}, \mathcal{M} \rangle$, that is constituted by:

1. **Query expansion:** the query posed in terms of the GVV is expanded to take into account the explicit and implicit constraints in the GVV: all constraints in the GVV are compiled in the expansion, so that the expanded query can be processed by ignoring constraints. Then, the atoms in the expanded query are extracted from the expanded query.
2. **Query unfolding:** the atoms in the expanded query are unfolded by taking into account the mappings $\mathcal{M}$ between the GVV and the local sources in $\mathcal{N}$.

The algorithm for Query expansion is reported in [14,11]; its output is the expanded query (called *EXPQuery*) and its atoms (called *EXPAtoms*); *EXPQuery*

is an union of conjunctive queries on GVV; an *EXPAtom* is a *Single Class Query* on a Global Class of the GVV.

In the following we will discuss the unfolding process of an *EXPAtom* by taking into account the new approach to define $q_{\mathcal{N}}$ of the previous section.

3.1 Query Unfolding

We explain the method by considering the BA level, i.e. the BA ontology. Given a global class C related to the local classes $L1, L2, \ldots Ln$, we consider a Single Global Query Q over C:

```
Q = select <Q_select-list> from C where <Q_condition>
```

`<Q_condition>` is a Boolean expression of positive atomic constraints: (GA1 op value) or (GA1 op GA2), where GA1 and GA2 are attributes of C.

As an example, we consider the following query (denoted by `expatom`):

```
expatom: SELECT NAME,CAPITAL_STOCK,REGION,ADDRESS,SUBCONTRACTOR
         FROM company
         WHERE CAPITAL_STOCK>50 AND REGION LIKE 'VENETO' AND SUBCONTRACTOR LIKE 'yes'
```

The output of the query unfolding process is

1. a set of Single Class Queries over the SINodes GVVs (FDAtoms):

   ```
   FDAtom = select <select-list> from SINode.C where <condition>
   ```

   where `C` is a Global Class of the `SINode-GVV`.
2. the *FDQuery* which computes the Full Disjunction of the FDAtoms
3. the resolution functions of the attributes in `<select-list>`

The query unfolding process is made up of the following steps:

1. Atomic constraint mapping: In this step, each atomic constraint of Q is rewritten into one that can be supported by the local class.

The atomic constraint mapping is performed on the basis of the mapping functions defined in the Mapping Table. Moreover, the atomic constraint mapping depends on the definition of the Resolution Functions for global attributes; for example, if the numerical global attribute GA is mapped onto L1 and L2, and we define AVG function as resolution function, the constraint (GA = value) cannot be pushed at the local sources, because of the AVG function has to be calculated at a global level, the constraint may be globally true but locally false. In this case, the constraint is mapped as true in both the local sources. On the other hand, if GA is an homogeneous attribute the constraint can be pushed at the local sources. For example, an atomic constraint (GA *op* `value`) is mapped onto the local class L as follows:

$$(MTF[GA][L] \; op \; \texttt{value}) \text{ if } MT[GA][L] \text{ is not null and the } op \text{ operator is supported into } L$$
$$true \text{ otherwise}$$

2. Select-list computation: The select-list of a FDAtom over the local class L is computed by considering the union of

1. the attributes in `<Q_select-list>` with a not null mapping in L
2. the set of attributes used to express the join conditions for L
3. the global attributes in `<Q_condition>` with a not null mapping in L

The set of global attributes is transformed in the corresponding set of local attributes on the basis of the Mapping Table.

As an example, the set of **FDAtoms** for `expatom` is :

```
FDATOM1 =  SELECT COMPANY_ID, NAME, REGION, ADDRESS, CAPITAL_STOCK
 FROM SN1.company
 WHERE ((CAPITAL_STOCK) > (50) and (REGION) like ('VENETO'))

FDATOM2 = SELECT  COMPANY_ID, NAME, REGION, ADDRESS, SUBCONTRACTOR
 FROM SN2.company
 WHERE ((REGION) like ('VENETO') and (SUBCONTRACTOR) like ('yes'))
```

The *FDExpr* which computes the FD of `FDAtom1` and `FDAtom2` is:

```
FDATOM1 full join FDATOM2 on (FDATOM1.COMPANY_ID=FDATOM2.COMPANY_ID)
```

The unfolded query is then obtained by applying to each query attribute of *FDExpr*, the related Resolution Function:

- for Homogeneous Attributes (e.g. `REGION`) we can take one of the related values (indifferently `FDATOM1.REGION` or `FDATOM2.REGION`);
- for non Homogeneous Attributes (e.g. `ADDRESS`) we apply the related Resolution Function (in this case the precedence function).

After the query reformulation process, we need to consider query processing techniques to evaluate queries over our two-level data integration system. This was not a focus of our present investigation and of the SEWASIE project; at present, we have just implemented a "naive approach" in an agent-based prototype, that will be described in the next section. Techniques for adaptive query processing [31] are well suited for our context.

3.2 An Agent-Based Prototype for Query Processing

Figure 4 shows the functional architecture of the system prototype for Query Management. The coordination of query processing is performed by the **Query Agent**, which accepts the query from the Query Tool Interface, interacts with a BA and its underlying SINode Agents, and returns the result as a materialized view in the SEWASIE_DB.

Playmaker: performs the reformulation of the query w.r.t. the BA ontology. It has two components: the **Expander**, which performs the Query expansion, and the **Unfolder**, which performs the query unfolding. Once the execution of the PlayMaker is completed, the output of the Play Maker computation is sent from the BA to the QA with a single message.

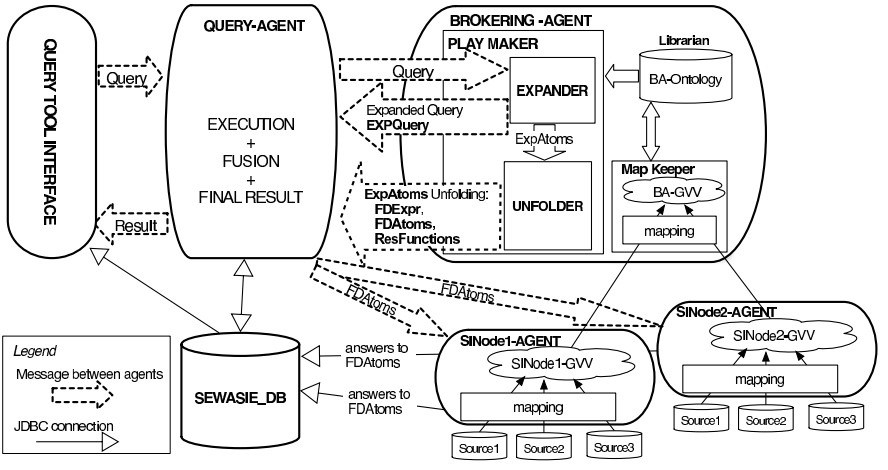


Fig. 4. Functional Architecture

Query Agent: it performs the following 3 steps:

1. **Execution:** for each FDAtom (Parallel Execution)
   - **INPUT:** FDAtom
   - **MESSAGES:** from QA to an SINode Agent
   - **OUTPUT:** a table storing the FDAtom result in the SEWASIE_DB
2. **Fusion:** For each EXPAtom (Parallel Execution):
   - **INPUT:** FDAtoms, FDExpr, Resolution Functions
     (a) Execution of FDExpr (Full Disjunction of the FDAtoms)
     (b) Application of the Resolution Functions on the result of (a)
   - **OUTPUT:** a view storing the EXPAtom result in the SEWASIE_DB
3. **Final result**
   - **INPUT:** Output of the FUSION step
     (a) Execution of the Expanded Query
   - **OUTPUT:** Final Query result view stored in the SEWASIE_DB

A this point, the Query Agent sends a message to the Query Tool Interface with the name of the Final Query result.

SINode Agent: One of the modules of the SINode Agent, the **SINode Query Manager**, executes queries on the SINode GVV, with a query processing similar to the one explained at the BA level.

4 Conclusion and Future Work

Future work will be devoted to investigate efficient query processing techniques to evaluate queries over two-level data integration systems. Furthermore we will investigate efficient query techniques for querying the super-peer network.

The above issues will be the goal of our research group within the running Italian MIUR founded project WISDOM (http://dbgroup.unimo.it/wisdom-unimo), which is coordinated by our group.

References

1. Aberer, K., Cudré-Mauroux, P., Hauswirth, M.: The chatty web: emergent semantics through gossiping. In: WWW. (2003) 197–206
2. Halevy, A.Y., Ives, Z.G., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I.: The piazza peer data management system. IEEE Trans. Knowl. Data Eng. **16** (2004) 787–798
3. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing : A vision. In: WebDB. (2002) 89–94
4. Löser, A., Siberski, W., Wolpers, M., Nejdl, W.: Information integration in schema-based peer-to-peer networks. In Eder, J., Missikoff, M., eds.: CAiSE. Volume 2681 of Lecture Notes in Computer Science., Springer (2003) 258–272
5. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Comput. Surv. **36** (2004) 335–371
6. Broekstra, J.e.a.: A metadata model for semantics based peer-to-peer systems. In: Proc. of the 1st WWW Int. Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID 2003), Budapest, Hungary (2003)
7. Castano, S., Ferrara, A., Montanelli, S., Pagani, E., Rossi, G.: Ontology-addressable contents in p2p networks. In: Proc. of the 1st WWW Int. Workshop on Semantics in Peer-to-Peer and Grid Computing (SemPGRID 2003), Budapest, Hungary (2003) http://www.isi.edu/ stefan/SemPGRID/proceedings/proceedings.pdf.
8. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In Dayal, U., Ramamritham, K., Vijayaraman, T.M., eds.: ICDE, IEEE Computer Society (2003) 49–
9. Löser, A., Naumann, F., Siberski, W., Nejdl, W., Thaden, U.: Semantic overlay clusters within super-peer networks. In Aberer, K., Kalogeraki, V., Koubarakis, M., eds.: DBISP2P. Volume 2944 of Lecture Notes in Computer Science., Springer (2003) 33–47
10. Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Logical foundations of peer-to-peer data integration. [32] 241–251
11. Beneventano, D., Lenzerini, M.: Final release of the system prototype for query management. Sewasie, Deliverable D.3.5, Final Version (2005) http://www.dbgroup.unimo.it/pubs.html.
12. Madhavan, J., Halevy, A.Y.: Composing mappings among data sources. In: VLDB. (2003) 572–583
13. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: Second-order dependencies to the rescue. [32] 83–94
14. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: What to ask to a peer: Ontolgoy-based query reformulation. In Dubois, D., Welty, C.A., Williams, M.A., eds.: KR, AAAI Press (2004) 469–478
15. Bergamaschi, S., P. Fillottrani, G.G.: The sewasie multi-agent system. In: Proc. of the 3rd Int. Workshop on Agents and Peer-to-Peer Computing (AP2PC 2004), New York City, USA July 19-20, 2004. (2004)

16. Bergamaschi, S., Castano, S., Vincini, M., Beneventano, D.: Semantic integration of heterogeneous information sources. Data Knowl. Eng. **36** (2001) 215–249
17. Beneventano, D., Bergamaschi, S., Guerra, F., Vincini, M.: Synthesizing an integrated ontology. IEEE Internet Computing **7** (2003) 42–51
18. Miller, A.: WordNet: A Lexical Database for English. Communications of the ACM **38** (1995) 39–41
19. Cal, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: Data integration under integrity constraints. Information Systems **29** (2004) 147–163
20. Beneventano, D., Bergamaschi, S., Guerra, F., Vincini, M.: Building an integrated ontology within sewasie system. In Cruz, I.F., Kashyap, V., Decker, S., Eckstein, R., eds.: SWDB. (2003) 91–107
21. Galindo-Legaria, C.A.: Outerjoins as disjunctions. In Snodgrass, R.T., Winslett, M., eds.: SIGMOD Conference, ACM Press (1994) 348–358
22. Rajaraman, A., Ullman, J.D.: Integrating information by outerjoins and full disjunctions. In: PODS, ACM Press (1996) 238–248
23. Naumann, F., Häussler, M.: Declarative data merging with conflict resolution. In Fisher, C., Davidson, B.N., eds.: IQ, MIT (2002) 212–224
24. Tejada, S., Knoblock, C.A., Minton, S.: Learning object identification rules for information integration. Inf. Syst. **26** (2001) 607–633
25. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: VLDB. (2002) 586–597
26. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In Halevy, A.Y., Ives, Z.G., Doan, A., eds.: SIGMOD Conference, ACM (2003) 313–324
27. Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tackling inconsistencies in data integration through source preferences. In Naumann, F., Scannapieco, M., eds.: IQIS, ACM (2004) 27–34
28. Bertossi, L.E., Chomicki, J.: Query answering in inconsistent databases. In Chomicki, J., van der Meyden, R., Saake, G., eds.: Logics for Emerging Applications of Databases, Springer (2003) 43–83
29. Greco, G., Greco, S., Zumpano, E.: A logical framework for querying and repairing inconsistent databases. IEEE Trans. Knowl. Data Eng. **15** (2003) 1389–1408
30. Lin, J., Mendelzon, A.O.: Merging databases under constraints. Int. J. Cooperative Inf. Syst. **7** (1998) 55–76
31. Ives, Z.G., Florescu, D., Friedman, M., Levy, A.Y., Weld, D.S.: An adaptive query execution system for data integration. In Delis, A., Faloutsos, C., Ghandeharizadeh, S., eds.: SIGMOD Conference, ACM Press (1999) 299–310
32. Deutsch, A., ed.: Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 14-16, 2004, Paris, France. In Deutsch, A., ed.: PODS, ACM (2004)

Database Selection and Result Merging in P2P Web Search

Sergey Chernov[1], Pavel Serdyukov[2], Matthias Bender[3], Sebastian Michel[3], Gerhard Weikum[3], and Christian Zimmer[3]

[1] L3S Research Center, University of Hannover, Expo Plaza 1, 30539, Hannover, Germany
chernov@l3s.de
[2] Database Group, University of Twente, PO Box 217, 7500 AE Enschede, Netherlands
serdyukovpv@cs.utwente.nl
[3] Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123, Saarbrücken, Germany
{mbender,smichel,weikum,czimmer}@mpi-inf.mpg.de

Abstract. Intelligent Web search engines are extremely popular now. Currently, only commercial centralized search engines like Google can process terabytes of Web data. Alternative search engines fulfilling collaborative Web search on a voluntary basis are usually based on a blooming Peer-to-Peer (P2P) technology. In this paper, we investigate the effectiveness of different database selection and result merging methods in the scope of P2P Web search engine Minerva. We adapt existing measures for database selection and results merging, all directly derived from popular document ranking measures, to address the specific issues of P2P Web search. We propose a general approach to both tasks based on the combination of pseudo-relevance feedback methods. From experiments with TREC Web data, we observe that pseudo-relevance feedback improves quality of distributed information retrieval.

1 Introduction

Current Web search technologies encounter a number of serious obstacles. The first one is the size of indexable Web, which can be hardly covered entirely enough due to limited network bandwidth and finite computational resources of search engines. Consequently, pages are only periodically updated and the outcome of most search engines is out of date. Another issue is a "Deep Web" problem, when search engines cannot get access to the information stored by commercial information providers or to the resources not linked by any page. There is also the noticeable social perspective. The most powerful player ever, Google, monopolizes Web search market. It controls a major part of Web search requests and can establish its own censorship. We think that a search engine having resort to P2P technology could be able to overcome the described limitations. Collaborative crawling can span a larger portion of the Web, if each peer

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 26–37, 2007.

would contribute its own focused crawl into the system. In addition, we disclose various opportunities of topic-oriented search by using intellectual inputs from users. These considerations induced us to launch the Minerva project [2], a P2P Web search engine.

In Minerva, each peer has its own collection of crawled or personal documents. For efficient query routing, all peers collectively maintain a global directory, which contains peer-summary information about which peer has documents for which index terms. This information is organized in peer lists constructed for each term occurring in the system. For example, each peer list contains *df* (document frequency) and a sum of tf (term frequency) for respective term at each peer plus total number of documents (with sum of their lengths) stored at this peer. To make these peer lists accessible to any peer, to share network load and to secure this information from loss, Minerva disseminates all peer lists using Chord distributed hash table (DHT) protocol [22]. It hashes terms and peer network addresses to know which peer is responsible for managing which peer list.

At many points, Minerva resembles a large-scale highly dynamic metasearch engine. We can leverage and adopt existing solutions from metasearch field, see Fig. 1 [8]. A query q is posed on the set of peers P. A database selection problem occurs when we select a subset of peers P' that most probably contain relevant documents. Then system sends q to every peer in P' and obtains a set of document rankings R' from the local search engines. A result merging problem occurs when all rankings in R' are merged into one ranking R_m and the top-k results from it are presented to the user.

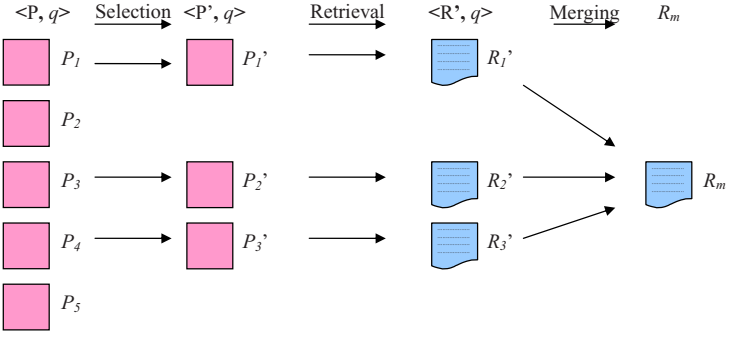


Fig. 1. A query processing scheme in the distributed search system

Looking for an appropriate scheme for the database selection and result merging for Minerva we evaluated several database selection and result merging methods in our prototype. We also proposed new methods based on pseudo-relevance feedback obtained from the best peer in a rough initial database ranking.

An overview of metasearch and recent work on P2P information retrieval (IR) is introduced in Sect. 2. We present details of our approach for database selection

and result merging in Sect. 3. The experimental setup, evaluation methodology, and our results are presented in Sect. 4 and . In Sect. 5 we make some topical conclusions.

2 Related Work

2.1 P2P Search Platforms

ODISSEA [23] uses two-layered search engine architecture and global index structure distributed over the nodes of the system. The distributed version of Fagin's threshold algorithm [11] is used for result aggregation over the inverted lists. In PlanetP [5] each node maintains an index of its content and summarizes the set of terms in its index using a Bloom filter. This approach is effective for several thousands peers, but it is hardly scalable to long queries.

The issue of efficient score aggregation in P2P IR environment with a structured topology was addressed in [3]. An algorithmic framework KLEE for distributed top-k queries was presented in [16].

2.2 Database Selection

All database selection methods fall in two classes: ad-hoc and language model based.

The ad-hoc database selection algorithms suggested in [27,4] use the document frequency of a term in a database as the most important evidence for the database usefulness. The most popular representative of ad-hoc family is CORI [4]. It adapts classic $tf \times idf$ formula, which proved its effectiveness for document retrieval, to the problem of database selection. In that case tf (term frequency) turns into a document frequency, while idf (inverted document frequency) becomes analogous icf (inverted collection frequency).

Another group of ad-hoc approaches considers word counts and merely utilizes the summarized similarity of documents as a usefulness measure [10]. In that case, similarity is just a sum of $tf \times idf$ weights for each query term in each document. The most sophisticated among such methods, GlOSS, was presented in [12].

Several works [26,21] contain quite successful attempts to apply language modeling framework for database selection task. They, again, simply work with a database as a "virtual document" by concatenating all its documents and simulate document retrieval. However, experimental results show that the performance of the language model based selection is not inferior to any of ad-hoc selection methods.

2.3 Result Merging

For consistent merging all search engines must produce a document's relevance score using same retrieval algorithm and global statistics. However, this requirement is not realistic. For example, under $tf \times idf$ scoring scheme the tf component is document-dependent and fair across all databases. In contrast the idf component is collection-dependent and we should globally normalize it.

When environment is cooperative, i.e. scores and documents statistics are provided by peers, usually Kirsch's algorithm [13] is used. At query time it collects local statistics from the selected databases and normalizes document scores. The semi-supervised method for merging results in a hierarchical P2P network [15] uses a centralized database of collection samples. In our setup we cannot afford learning-based approaches, since system is highly dynamic.

A series of publications [7,4,14] describes a CORI merging strategy, which heuristically combines both resource and document scores. In [7] it was also suggested that the result merging based on raw tf values is a worthwhile approach when involved databases are homogeneous. Result merging techniques for topically organized collections were studied in [14]. Their experiments showed that global idf normalization is the best method, but they did not consider real Web pages and overlap between collections.

The approach in [Bau99] is designed for the cooperative environment. A probabilistic ranking algorithm is based on the exported statistics from the search engines. The language modeling based merging from [21] is developed for uncooperative setup, the probability of generating a query from the language model of a document is approximated.

3 Pseudo-relevance Feedback for Distributed IR

Our analysis of the distributed IR state of art showed that there is a lack of original task-oriented approaches. In particular, the authoritative database selection methods are mostly based on the well-known query-document similarity measures applied with minor changes to the database selection task. The main assumption is that the estimated collections can be represented by a concatenation of their documents. At the same time, being merged together text sources lose their individuality and become very heterogeneous and multi-topically oriented. Consequently, the ambiguity of short Web queries becomes extremely high with respect to these enormously long virtual documents.

There are several ad-hoc query expansion and language model based query modeling methods operating on the top-k ranked documents. At the same time, nobody applied these methods in the scope of distributed IR. Moreover, they have never been used simultaneously for IR, what appears to be an omission in our opinion.

The merging task is even more sensitive to the ambiguity of short queries and lack of context. Obviously, we cannot rely on learning methods, since information is outdated quickly in P2P system. Thereupon, pseudo-relevance feedback can be useful. The language model based retrieval algorithms seem the most promising nowadays, so we want to investigate their pseudo-relevance feedback opportunities for merging. We intend not to merely normalize the similarity scores, but also to improve the retrieval algorithm itself.

In the Minerva system, Web pages are crawled with respect to the user's bookmarks and assumed to reflect specific interests. We can exploit this fact using a pseudo-relevance feedback for finding the "preference" language model

from the most relevant database. We assume that it was crawled by the user with query-related interests and contains some pages on the relevant topic. For the pseudo-relevance based model estimation we suggest to use the top-k ranked documents, obtained after query execution on the most highly ranked database for the current query. We increase the number of attributes that can be used to compare databases (***query expansion***) and weight different attributes w.r.t. their level of importance for the relevance evaluation (***query modeling***).

Unfortunately, there is no comparative analysis of pseudo-relevant feedback techniques. We consider two methods to be good representatives of ad-hoc expansion methods, based on the popular IR heuristics Robertson's method [19] and on the Language Model based approach to IR, Ponte's method [17]. Both of them assume that a term is more significant if it appears more often in the top-k documents than in any document of the collection in average. Two query modeling methods have been proposed in [28] by Zhai and Lafferty and in [24] by Tao and Zhai. These methods suppose that any term in the pseudo-relevant documents is generated from two sources: the pseudo-relevance model of a user PR, since they were chosen by the users query, and the background language model GE, which can be approximated by the collection language model well enough. The latter method takes into account that the probability of a term being generated by the relevance model is decreasing in parallel with the document score. In both approaches the estimation of $p(t_k|PR)$ for each term t_k is done using Expectation-Maximization (EM) algorithm [9].

3.1 Database Selection

We propose a two-step database selection method. At first, from the set of all available peers P and a query q we build a peer ranking P' by the database selection method described in [21]. For the Global English language model GE we use an approximation from peer lists corresponding to a specific query. Thus, peers are scored by:

$$Score(q, P_i) = -\sum_{k=1}^{|q|} \log p(t_k|P_i) \tag{1}$$

$$p(t_k|P_i) = \lambda \cdot \frac{ctf_{t_k}}{|P_i|} + (1-\lambda) \cdot p(t_k|GE) \tag{2}$$

$$p(t_k|GE) = \frac{\sum_{i=1}^{|\mathrm{P}|} ctf_{t_k}^{P_i}}{\sum_{k=1}^{|\mathrm{T}|} \sum_{i=1}^{|\mathrm{P}|} ctf_{t_k}^{P_i}}. \tag{3}$$

Here, $p(t_k|P_i)$ is the generation probability of term t_k of language model for collection P_i; λ is a empirically set smoothing parameter between 0 and 1; ctf_{t_k} is the collection term frequency, the number of term occurrences in the database; T is a system vocabulary, the full set of distinct terms on all peers; $p(t_k|GE)$ is the generation probability of term t_k of Global English language model.

At the next step, the query q is executed on the best database in the ranking P'$_1$. At first, the top-k ranked documents are used by ad-hoc query expansion techniques [19,17], to add new terms to the query. Then, this top-k is utilized by query modeling techniques [28,24] to estimate generation probability $p(t_k|PR)$ for each query term from the pseudo-relevance language model PR. As a result, we build new database ranking P" using expanded query and term generation probabilities. We apply cross-entropy, an information-theoretic measure of distance between two distributions:

$$H(q, P_i) = -\sum_{k=1}^{|q|} p(t_k|PR) \cdot \log p(t_k|P_i). \tag{4}$$

Apparently, the lower cross-entropy of a database language model w.r.t. the pseudo-relevance based language model, the higher similarity of these models. It also turns out that in our formula we combine two values expressing term importance: query-specific $p(t_k|PR)$ and global $p(t_k|GE)$.

3.2 Result Merging

Our merging approach also exploits pseudo-relevance feedback adapted for the distributed setup. Executing the query on the peer, which won the highest rank from the database selection algorithm, we obtain the top-k best results for our pseudo-relevance based model estimation. This language model is then used for adjusting merging results, one user with a highly specified collection of documents implicitly helps another user to refine the final document ranking.

As in the database selection approach above, we estimate the probability $p(q|PR)$ from the pseudo-relevance based cluster of documents PR. The probability $p(q|GE)$ is again approximated using the peer lists information. After the probabilities $p(q|GE)$, $p(q|PR)$, and the query q were sent to every peer in ranking P', we compute cross-entropy between the pseudo-relevance based language model PR and the document language models for every document D_{ij}:

$$H(t_k, D_{ij}) = -p(t_k|PR) \cdot \log p(t_k|D_{ij}). \tag{5}$$

At second step we compute the ordinary language modeling similarity score, smoothed by General English language model:

$$p(t_k|D_{ij}, GE) = \log(\lambda \cdot p(t_k|D_{ij}) + (1 - \lambda) \cdot p(t_k|GE)). \tag{6}$$

Finally, we combine both scores in a heuristic manner, the empirically set parameter β lies in interval from zero to 1:

$$s = \sum_{k=1}^{|q|} \beta \cdot p(t_k|D_{ij}, GE) + (1 - \beta) \cdot H(t_k, D_{ij}). \tag{7}$$

The search results are sorted in descending order of similarity scores s and the best top-k URLs are presented to the user.

4 Experiments

4.1 Experimental Setup

The Minerva system is implemented in Java and document databases associated with peers are managed by Oracle DBMS. We conducted experiments with 50 databases created from the TREC-2002, 2003 and 2004 Web Track datasets from the ".GOV" domain. For these three volumes, four topics were selected. The relevant documents from each topic were taken as a training set for the support vector machine classification algorithm and 50 collections were created. The non-classified documents were randomly distributed among all databases. Each classified document was assigned to two collections from the same topic. For example, for the topic "American music" we had the subset of 15 small collections and all classified documents were replicated twice in it. The topics with the numbers of corresponding collections are summarized in the Tab. 1, each collection was managed by one dedicated peer. Assuming that the search in our

Table 1. Topic-oriented experimental collections

N	Topic	Number of collections
1	Health and medicine	15
2	Nature and ecology	10
3	Historic preservation	10
4	American music	15

system should be topic-oriented as well as crawling, we selected the set of the 25 out of 100 title queries from the topic distillation task for the TREC 2002 and 2003 Web Track. Queries are selected with respect to two requirements: at least 10 relevant documents exist and query is related to the "Health and Medicine" or "Nature and Ecology" topics. The set of selected queries is presented in [6], relevance judgments are available on the NIST site (http://trec.nist.gov).

4.2 Database Selection Experiments

The methodology for evaluation of database selection performance is not standardized. The most popular approach is to measure a quality of selection by *cumulative recall*. It shows which method accumulates relevant documents faster by selecting databases from the top of its ranking. Let $|D_i^r|$ be a number of relevant documents on peer P_i; N is the number of collections selected from the top of the database ranking P'. Cumulative recall is a fraction, where term of fraction is a sum of all relevant documents on first N peers from ranking P' and denominator is a total number of relevant documents on all peers in P':

$$Recall = \frac{\sum_{i=1}^{N} |D_i^r|}{\sum_{i=1}^{|P|} |D_i^r|}. \tag{8}$$

Resulting evaluation is based on macro averaging of cumulative recall over all test queries. Usually, it is calculated for every N up to sensible system-dependent number, so we consider selection of at most 20 databases.

We evaluate Robertson's [19] and Ponte's expansion [17] methods and find the best combination of the following parameters for each method: 5, 10 and 20 expansion terms, and 5, 10, 15, 20, 25 pseudo-relevant documents. Boundaries for the latter parameter are derived from the experiments with document retrieval in original papers, where the size of analyzed top-k did not exceed 20. To estimate the best k for the usage of top-k ranked documents in query modeling methods, we take 7 values from top-10 to top-70 pseudo-relevant documents, it conforms with original papers [28,24]. In addition, we study the possibility to expand a query by terms having the greatest $p(t_k|PR)$ assigned by the respective modeling methods. Earlier, they have been applied for massive expansion (with full vocabulary) coupled with modeling itself what is completely unfeasible in distributed IR.

It is interesting to observe that both ad-hoc expansion methods spoil the selection being applied without consequent query modeling. Their best parameter setup allows only to approach the performance of non-expanded queries. This observation proves that expansion methods are sensitive to the number of pseudo-relevant documents used, retrieval quality depends on the fraction of indeed relevant documents in pseudo-relevant subset. Moreover, if we use query model as a source for expansion terms, the performance decreases dramatically.

Both query modeling methods improve the retrieval quality, when used in addition to query expansion methods. The modeling method of Tao and Zhai [24] shows marginally better result. We infer that it is important to apply ad-hoc expansion and query modeling simultaneously.

To get the baseline for our experiments, we measured 4 most popular existing selection methods: two language model based methods from [26,21], CORI [4] and GlOSS [12]. Our results showed that Language model based methods are constantly more effective and method by Si, Callan and others [21] is the best. GlOSS appeared to be the worst. We can observe the performance of these methods and our approach, which uses the combination of Robertsons expansion with Tao and Zhais query modeling, in Table 2. The improvement of our approach upon the performance of baseline method is comparable to the improvement of baseline method over the worst method. Our approach reaches its maximum selection performance with use of only 5 expansion terms what allows its integration into P2P Web search without a significant loss of scalability.

4.3 Result Merging Experiments

For the merging methods evaluation in the Minerva system we used the following score normalization techniques: TF is a merging by raw tf values; TFIDF is merging by local $tf \times idf$ score; TFGIDF uses $tf \times idf$ score with globally normalized idf; CORI is a merging method from [4]; LM is a language modeling retrieval algorithm. More detailes about methods can be found in [6].

Table 2. Cumulative recall at different levels of database rankings

	1	2	3	4	5	10	15	20
Our approach	**0,128**	**0,229**	**0,304**	**0,375**	**0,406**	**0,619**	**0,759**	**0,829**
LM of Si & Callan	0,092	0,187	0,27	0,352	0,399	0,606	0,733	0,81
LM of Xu & Croft	0,091	0,187	0,27	0,354	0,41	0,603	0,735	0,811
CORI	0,089	0,195	0,244	0,324	0,37	0,588	0,73	0,809
GlOSS	0,102	0,179	0,249	0,322	0,376	0,596	0,715	0,781

For the evaluation, we utilized the framework from [21]. For all tested algorithms, the average precision measure is computed over the 25 queries at the level of the top-5, 10, 15, 20, 25, and 30 documents. The parameter λ in LM method is empirically adjusted, different approaches vary it from 0.4 to 0.7. After preliminary experiments we set λ to 0.4, as it produced the most robust and effective results.

We also measured the retrieval accuracy for non-distributed case on the *single database*, which contains all the documents from the 50 peers. Two non-distributed retrieval algorithms were used: $tf \times idf$ is coded as SingleTFIDF, and language modeling retrieval is coded as SingleLM.

Only 10 selected databases participate in a query execution and, therefore, the effectiveness of the query routing algorithm influences the quality of result. We assessed the result merging methods with several database rankings. Due to space limit, we present results only for manually created IDEAL ranking, where the collections are sorted in a descending order of the number of relevant documents.

In Tab. 3 we summarize the results from the result merging experiments with six merging methods and two non-distributed retrieval algorithms. The best results in every category are shown in bold. Here are the main observations:

- Retrieval effectiveness of all result merging methods is similar;
- The LM method shows the best performance, it is robust under every ranking;
- Surprisingly, the TFIDF method is more effective than the TFGIDF technique, It might be the case that GIDF values, which are averaged over all databases, are more influenced by noise, while local IDF values are more topic-specific;
- An effective database ranking allows to outperform single database baseline;

In the second series of experiments, we evaluated our LMPR technique. Ranking PR is purely based on the cross-entropy between pseudo-relevance based and document language models, see Eq. 5; LM and SingleLM methods remain the same; LMPR is a heuristic combination of LM and PR rankings, as described in Eq. 7.

At first, we conducted experiments for the separate PR ranking in order to find the optimum n for estimating our pseudo-relevance based model, for IDEAL ranking the best choice was $n = 10$. After we fixed the n parameter, we conducted

Table 3. The macro-average precision for evaluated merging methods with the database ranking IDEAL

	SingleTFIDF	SingleLM	TFIDF	TFGIDF	CORI	TF	LM
top-5	0,208	0,224	0,248	0,240	0,240	**0,264**	**0,264**
top-10	0,176	0,200	0,204	0,204	0,204	0,184	**0,220**
top-15	0,160	0,181	**0,192**	0,163	0,189	0,155	0,184
top-20	0,158	0,158	**0,180**	0,164	0,178	0,142	0,168
top-25	0,142	0,149	**0,165**	0,150	0,163	0,125	0,157
top-30	0,135	0,141	0,141	0,141	0,144	0,120	**0,145**

experiments with different values of the β parameter. We carried out experiments for $\beta = \{0.1, \ldots, 0.9\}$ and obtained the best combination with $\beta = 0.6$. In Tab. 4 we present our combined LMPR method and show the separate performance of each methods for comparison.

The single PR ranking, which is purely based on the pseudo-relevance feedback, is poor with the IDEAL ranking. The average precision of the LMPR is the same or slightly better in comparison with LM. We conclude that the LMPB combination of the cross-entropy ranking PB with the LM language model with $\beta = 0.6$ is more effective than the single LM method.

Table 4. The macro-average precision with the database ranking IDEAL, top-10 documents for the pseudo-relevance based language model estimation, $\beta = 0.6$

	PR	LMPR	SingleLM	LM
Top-5	0,248	**0,272**	0,224	0,264
Top-10	0,192	**0,220**	0,200	**0,220**
Top-15	0,165	**0,187**	0,181	0,184
Top-20	0,146	**0,170**	0,158	0,168
Top-25	0,130	**0,157**	0,149	**0,157**
Top-30	0,119	0,144	0,141	**0,145**

5 Conclusion and Future Work

In this paper, we evaluated existing database selection and result merging methods. We also proposed and evaluated our approach. Its novelty is in that peers ranking and final documents ranking are refined with use of the pseudo-relevance feedback from the best peer in the preliminary peers ranking. In most cases our methods are more effective than existing ones. We come to the conclusion that pseudo-relevance feedback information from topically organized collections allows to improve a quality of distributed IR. The presented result indicates that in future we can think about methods using pseudo-relevance models from several databases considering different levels of their query expertise.

Acknowledgment

We would like to thank Wolfgang Nejdl and Paul-Alexandru Chirita for helpful comments on a paper draft.

References

1. Christoph Baumgarten. A probabilistic solution to the selection and fusion problem in distributed information retrieval. In *Proceedings of the 22th Annual International Conference on Research and Development in Information Retrieval ACM SIGIR '99*, pages 246–253. ACM Press, 1999.
2. M. Bender, S. Michel, G. Weikum, and C. Zimmer. Bookmark-driven query routing in peer-to-peer web search. In: Callan, J., Fuhr, N., and Nejdl, W., *Workshop Proceedings of the 27th Annual International Conference on Research and Development in Information Retrieval ACM SIGIR '04*, pages 46–57, 2004.
3. W.-T. Balke, W. Nejdl, W. Siberski, and U. Thaden. Progressive Distributed Top-k Retrieval in Peer-to-Peer Networks. *Proceedings of the 21th International Conference on Data Engineering ICDE '05*, Tokyo, Japan, pages 174–185, 2005.
4. J. Callan, *W.B. Croft, editor, Advances in information retrieval*, Chapter Distributed Information Retrieval, Kluwer Academic Publishers, pages 127–150, 2000.
5. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using gossiping to build content addressable peer-to-peer information sharing communities. *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, pages 236–249, 2003.
6. S. Chernov: Result Merging in a Peer-to-Peer Web Search Engine, Saarland University, Master thesis, 2005.
7. J. P. Callan, Z. Lu, and W. B. Croft. Searching Distributed Collections with Inference Networks. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *Proceedings of the 18th Annual International Conference on Research and Development in Information Retrieval ACM SIGIR '95*, pages 21–28, Seattle, Washington, ACM Press, 1995.
8. N. E. Craswell: Methods for Distributed Information Retrieval, PhD thesis, 2001.
9. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society, Series B, Vol. 39(1), pages 1-38, 1977.
10. N. Fuhr. A decision-theoretic approach to database selection in networked IR. ACM Transactions on Information Systems, Vol. 17(3), pages 229-249, 1999.
11. R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. ACM Symp. on Principles of Database Systems, Santa Barbara, USA, pages 102–113, 2001.
12. L. Gravano, H. Garcia-Molina, and A. Tomasic. GlOSS: text-source discovery over the Internet. *ACM Transactions on Database Systems*, Vol. 24(2), pages 229–264, 1999.
13. S. T. Kirsch. Distributed search patent. u.s. patent 5,659,732, 1997.
14. L. S. Larkey, M. E. Connell, and J. P. Callan. Collection selection and results merging with topically organized u.s. patents and trec data. In *Proceedings of the Tenth International Conference on Information and Knowledge Management CIKM '00*, ACM Press, pages 282–289, 2000.

15. J. Lu and J. Callan. Merging retrieval results in hierarchical peer-to-peer networks. In *Proceedings of the 27th Annual International Conference on Research and Development in Information Retrieval ACM SIGIR '04*, pages 472–473, 2004.
16. S. Michel, P. Triantafillou, and G. Weikum. KLEE: A Framework for Distributed Top-K Query Algorithms. In *Proceedings of the 31st International Conference on Very Large Data Bases VLDB '05*, pages 637–648, 2005.
17. J. M. Ponte. A Language Modeling Approach to Information Retrieval. PhD thesis, University of Massachusetts Amherst, 1998.
18. J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the 21th Annual International Conference on Research and Development in Information Retrieval ACM SIGIR '98*, pages 275–281, 1998.
19. S. E. Robertson and S. Walker. Okapi/keenbow at trec-8. In *Proceedings of the 8th Text REtrieval Conference (TREC '99)*, pages 151–162, 1999.
20. P. Serdyukov. Query routing in a peer-to-peer web search engine, Saarland University, Master thesis, 2005.
21. L. Si, R. Jin, J. P. Callan, and P. Ogilvie. A language modeling framework for resource selection and results merging. In *Proceedings of the 10th International Conference on Information and Knowledge Management CIKM '02*, pages 391–397, ACM Press, 2002.
22. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
23. T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. In *International Workshop on the Web and Databases (WebDB '03)*, pages 67–72, 2003.
24. T. Tao and C. Zhai. A mixture clustering model for pseudo feedback in information retrieval. In *Proceedings of the Meeting of the International Federation of Classification Societies*, 2004.
25. G. G. Towell, E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. Learning collection fusion strategies for information retrieval. In *International Conference on Machine Learning*, pages 540–548, 1995.
26. J. Xu and B. W. Croft. Cluster-based language models for distributed retrieval. In *Proceedings of the 22th Annual International Conference on Research and Development in Information Retrieval ACM SIGIR '99*, Berkeley, CA, USA, pages 254–261, 1999.
27. B. Yuwono, D. L. Lee, R. W. Topor, and K. Tanaka. Server ranking for distributed text retrieval systems on the internet. In *Proceedings of the 5th International Conference on Database Systems for Advanced Applications DASFAA '97*, Melbourne, Australia, pages 41-50, 1997.
28. C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the 10th International Conference on Information and Knowledge Management CIKM '01*, ACM Press, pages 403–41, 2001.

Multiple Dynamic Overlay Communities and Inter-space Routing

Pedro Furtado

Department of Computer Engineering, University of Coimbra
Polo II - DEI - FCTUC
P-Coimbra 3030-290, Portugal
pnf@dei.uc.pt

Abstract. In a broad sense, overlays can be setup as spaces "communities" for user, data or resource access, publishing and sharing, forming domains within a larger universe. Ad-hoc creation and linking of spaces can replace single linear address spaces or hierarchically structured ones. Our focus is on how can such completely independent and autonomous spaces be created and linked dynamically and information routed between them: we propose dynamic space management and inter-space linking and routing alternatives. We compare alternative gateway strategies and gateway hotspot avoidance. We also study the efficiency of the proposed schemes analytically.

1 Introduction

An overlay network is formed by a subset of the underlying physical network nodes. The connections between overlay nodes are provided by overlay links (IP-layer paths), each of which is usually composed of one or more physical paths. Most peer-to-peer overlay networks have some relevant characteristics that include flexibility, ease of scalability, self-organization, load-balancing, adaptation, and fault tolerance. Nodes can be added or removed flexibly and the system adapts to such changes automatically. Replication and alternative routing paths can be used to circumvent routing and node failures. One of the most important principles in P2P is totally decentralized control: a node should be able to keep only information on a small set of neighbors and operations such as joining or leaving the network should carry reasonably low overheads. One of the major issues in early unstructured P2P systems was the poor scalability and unnecessary overhead associated with message flooding that was necessary to route messages to nodes. Structured P2P systems avoid flooding by using more elaborate addressing and routing strategies. Efficient content-based access is implemented by hashing objects and lookup queries into corresponding nodes. In the future, maybe overlay networks and P2P can assume a much broader role than only basic file sharing on a single peer-community. P2P is more like a generic flexible grid of cooperating nodes for some common aim. Naturally, there can be many communities or domains as in the internet. Multiple communities can emerge as needed, change and be managed dynamically, each one

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 38–49, 2007.

being an overlay - we call these overlays spaces. These characteristics could not be met satisfactorily with only a partitioned keyspace over a single overlay (e.g. Skipnet in [1]). Multiple spaces are also explored in hierarchical DHTs. Hierarchical DHTs are systems in which peers are organized into groups, and each group has its autonomous intra-group overlay network and lookup service, and the most reliable peers in the groups are designated as super-peers in a top level DHT. They have been explored as a way to significantly improve the performance and resilience to failures over flat DHT designs. Our proposal of completely autonomous spaces subsumes hierarchical DHTs, but is more generic. While hierarchical DHTs are a single, two-level address space over which objects are distributed, they do not represent completely independent and autonomous communities of peers. Our proposals more generically provide dynamic ad-hoc creation and linking of any number of independent communities of users/peers to access and manage resources. In such a scenario, inter-space linking efficiency becomes a relevant issue. In typical DHT overlay networks, inter-node links are setup so that routing into any node takes few hops (e.g. $\log N$ hops in Chord), even though nodes record only a small set of links. There is no such strategy defined for efficient interconnection and routing between autonomous spaces with ad-hoc linking. There should be no centralized control over the interconnection, no global routing tables and no pre-defined routers in these dynamic application-level overlay networks. We discuss and evaluate alternative strategies for solving this issue and also define basic space management primitives. We base most of our work on the Chord system in [2], but it can be implemented in any P2P overlay network framework. In the analysis section we compare the alternatives envisioned in this paper. The paper is organized as follows: section 2 discusses related work. Section 3 discusses space management primitives for space federations and inter-space linking. Section 4 proposes linking and routing alternatives. Section 5 contains a comparative analysis and section 6 concludes the paper.

2 Related Work

There has been a large amount of work in the area of overlay networks. Proposals for alternative organizations of structured overlay networks include CAN [3], Chord [2], Pastry [4], Tapestry [5], Skipnet [1]. None considers multiple spaces (although CAN mentions multiple DHTs) and none considers interconnection issues between spaces. Skipnet does consider hierarchical key spaces, but is still a constrained single space. Work on hierarchical DHT systems in [6] does consider multiple spaces and most reliable nodes in each space are elected as super-peers in a top-level overlay connecting the groups. However, they still represent in fact a single two-level addressing/lookup space. Our design is more generic and subsumes hierarchical DHTs. In our case there is no requirement that there be specific super-peer nodes and that these be part of a top-level overlay. The

world of overlay applications has been evolving from a limited horizon - for file sharing - into a more generic infrastructure with complex schemas and querying functionalities. There are many works on representing schemas and processing more complex queries over P2P, which include MAAN [7], RDFPeers [8], PIER [9] and PeerDB [10] among others. While MAAN and RDFPeers emphasize relatively simple querying over simple (RDF) schemas, PIER and PeerDB evolve more complete data management functionality into a P2P setting.

3 Multiple DHT-P2P Spaces

In this section we discuss how multiple dynamic spaces could be handled. Individual spaces and interconnects between them should maintain the desirable properties of DHT-P2P overlay networks. As in a single overlay network, a node should be able to keep only information on a small set of neighbors and operations related to inter-connection and routing should carry reasonably low overheads. There should be flexibility, scalability, self-organization, load-balancing, adaptation, fault tolerance and decentralized control.

Definition. *We define a space as a P2P overlay network, with its set of nodes and state; A federation is any set of spaces. Disconnected spaces are spaces that do not share nodes; Connected spaces are spaces that share at least one node; A subspace is a space whose nodes are completely contained (shared) within the parent space.*

Figure 1 illustrates a Chord network (S1) with subspaces S2 and S3 and a disconnected space S4. Spaces S1 and S4 can be inter-connected for data exchange. In the rest of this section we discuss the creation primitives for multiple spaces and defer inter-space linking to the next section. Efficient inter-space routing is discussed later on.

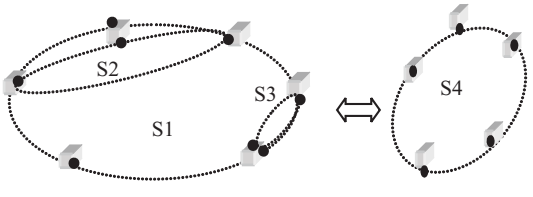


Fig. 1. Chord Spaces

3.1 Creating Multiple Spaces

Considering multiple spaces, we should allow individual space composition to be determined based on different goals (e.g. for multicasting or broadcasting within a group; to harbour a specific schema; for computation; for a common domain or with latency-location properties). In order to allow multiple spaces to be created dynamically for those varied objectives, there must be a primitive

for a user, administrator or an application to create an overlay network (space), possibly with a single node initially:

CREATE Space Type=Chord, bits=20, Nodes=N1, Name=S1;
This example creates an overlay network (space) S1 for up to 2^{20} nodes. The join operation works as described in [2] for Chord. This involves initializing the node state (fingers) and updating the state of existing nodes, as well as copying the keys for which the new node has become a successor. The creator of the space can specify multiple nodes in the creation syntax. There must also be access control and security information, whereby the owner has administrative privileges to grant or revoke access to S1 to nodes, users or entire communities (spaces). Assuming an application requires a smaller subspace S2 with up to 27 nodes within S1, with an initial set of nodes, the following space can be created:

CREATE Space Type=Chord, bits=7, Nodes=N1.1,. . . ,Nn.1, Name=S1.2, Parent=S1;
Therefore, spaces can be contained, overlap by some nodes or be completely separate. As another example of space creation options, consider the creation of topologically-oriented spaces. Internet domain name prefixes (DNS) or the binning strategy of [11] can be used to choose the set of nodes to include in a space. The binning strategy uses landmarks to estimate latencies [12] and clusters nodes into bins according to distance order to the landmark set. The following examples create a topological-oriented S1 subspace with 50 nodes and a domain-oriented space:

CREATE Space Type=Chord, bits=7, Nodes=N1 , Node N=50, Name=S1.3, Parent=S1;
CREATE Space Type=Chord, bits=7, Domain= com.antra, NodeN = 50, Name = ANTRA, Parent=S1;

3.2 Linking Multiple Spaces

We define a gateway as a node through which the spaces can interconnect, and an inter-space link - link for short - as a link between nodes from two different spaces. The gateway is either a node with state from both spaces or a node that contains state from a single space plus an inter-space link. For inter-space connection, there must be at least one "gateway" node(s). Nodes that belong to both spaces can be "natural" gateway(s) for interconnection. In subspaces, every node is a "natural gateway". In non-enclosed spaces, natural gateways may exist if some nodes belong to both spaces. In totally disconnected spaces, there is no "natural" gateway. In either case, it may be necessary to add (more) interconnections between spaces. We distinguish the following interconnections:

Natural and Observer Gateways: A node from space S1 can become a "natural" gateway for interconnection to space S2 by making it join S2 as well. This way it will have fingers to route efficiently within both spaces. However, we do

not assume that forcing some node(s) to join both spaces is desirable, as this depends on applications. Alternatively, it can be an "observer" gateway - contain fingers to S2 as if it were an S2 node. These fingers should be bi-directional (target nodes also contain a finger to the observer node), so that changes in S2 gateways also update the observer. However, observer has no hash-key value associated with it for the other space (S2).

Links: A third alternative is to create bi-directional links (a finger in each node pointing at the other one) between pairs of nodes in S1 and S2 (gateways), as shown in Figure 2a, in which some nodes have links to the other space. The main difference between using observers and links is that the observer has the complete state information as an S2 node (e.g. $\log N$ fingers in Chord), while links have a single finger to an S2 node, requiring an extra hop.

The layout of multiple links or observer fingers is also relevant. Consider a large number of requests being posed into a space. If observers or links are evenly distributed, heavy inter-space traffic will be balanced among gateways (at the overlay network level). Otherwise, the gateways and links become hotspots, potentially raising congestion and delay concerns. For this reason, multiple gateways or links should be evenly distributed.

There are important efficiency issues related to space inter-connection. Within a single space, a node is able to route efficiently in at most x hops ($x = \log N$ in Chord) due to its fingers. This is not guaranteed between independent spaces, where it needs to reach a gateway in each space traversed. In the next section we propose efficient solutions for inter-space routing and avoiding gateway hotspots.

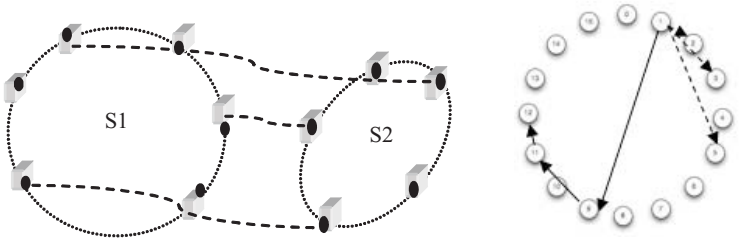


Fig. 2. (a) Interconnection with Multiple Links (b)Chord Routing from 1 to 12

4 Inter-space Routing Policies

Our objective in this section is to evaluate alternative approaches to deal with inter-space routing efficiently, while ensuring decentralization and avoidance of hotspots when inter-space communication becomes prevalent. We use Chord as our basic design framework, but the principles can be applied to any overlay network. Before discussing inter-space connections, we review intra-space routing and broadcasting in Chord, as fundamental principles for what follows.

4.1 Chord Routing and Broadcasting

Chord [2] organizes nodes in an identifier circle modulo 2^m , called the Chord ring. The basic Chord ring requires each node to maintain information on neighbours. Finger tables $(2^k, k = 0, , log_2 N - 1)$ allow each node to maintain the state of only $O(logN)$ neighbours. Chord hashes both the key of a data object and a peer's IP address into an m-bit identifier. The keys' identifiers map to peers' identifiers using consistent hashing. For a lookup query, the object key is hashed and the resulting succ(id) node is sought. The node chooses a finger to hop nearer the succ(id) (e.g. the hop that approximates most to succ(id) node). This process is repeated by the next nodes until the succ(id) node is reached and returns the object to the requester in at most $\log N$ hops (see 2b route from 1 to 12). The authors of [13] proposed a broadcasting strategy for Chord without flooding or sending more than one message into the same node. A user entity initiates the broadcast by submitting broadcast(info) to a node Q. This node acts as a root to a (virtual) broadcast spanning tree. The spanning tree is built as follows: node Q forwards the message into neighbours with a limit parameter that restricts the forwarding space of a receiving node: broadcast(info,limit). The limit parameter for a forward into Finger[i] is set to Finger[i+1], meaning that each neighbour Finger[i] will forward only within the interval [Finger[i], Finger[i+1]. This forwarding is now applied recursively, forming the spanning tree. As all nodes are contacted once, broadcasting has a cost $N - 1$ in number of messages and $\log N$ in number of hops (because nodes have a logarithmically decreasing forwarding range). In comparison, unicasting (lookup query) had a cost $\log N$ in number of messages and $\log N$ in hops.

In the next sections we propose and discuss inter-space routing alternatives (using Chord), based on efficiency (measured in number of messages and number of hops) and state and management overhead. More state overhead can buy efficiency, but the challenge is to keep the state overhead low while being reasonably efficient.

4.2 Strategies with No State Overhead

Consider a query or request to another space. If there is no way to locate the required gateway directly, the query can be flooded or broadcast within the space. In flooding, every node sends the request to all its fingers. Broadcasting avoids some of this overhead using the strategy we described in section 4.1. In either case, eventually, one of the gateway nodes (or subspace nodes) is reached and handles the request (and stops forwarding the message):

Space/Parent Flood (SF, PF): Assuming that, upon receipt of the message to route, every link forwards the request to each of its $\log N$ neighbours, there will be $N \log N$ messages. Flooding is not required in DHT, but we include it for comparison with what an unstructured system would require. We also consider that nodes from the target space that receive the request do not flood and instead use intra-space routing using their finger tables). If the space has

N nodes (typically very large) and the space has S nodes, the overhead of this alternative is:

- Number of messages: $N \log N + \log S$ (single node lookup) or $N \log N + (S-1)$ (broadcast within subspace);
- Number of hops: $\log N + \log S$;
- State overhead: 0 or gateway links or fingers.

Space/Parent Broadcast (SB, PB): If the message is broadcast instead of flooded, we use the strategy in [13] to avoid flooding. The overheads are:

- Number of messages: $\leq (N-1) + \log S$ (single node lookup) or $(N-1)+(S-1)$ (broadcast within subspace). Note: the broadcast worst case is $N-1$ (if the gateway(s) nodes are leafs of the broadcast spanning tree) and the actual value is between 0, if the requester is also a gateway, and $N-1$. As soon as a gateway node is found, the routing starts within the other space;
- Number of hops: $\log N + \log S$;
- State overhead: 0 or gateway links or fingers.

These approaches may pose scalability issues, as they entail a large, unnecessary overhead by distributing so many messages.

4.3 Strategies with Extensive State Overhead

The opposite alternative to having no state overhead is to make each node become a gateway. This way inter-space routing will be fastest, but the state storage and maintenance overhead for a space with N nodes will be $\mathcal{O}(N)$, where N can be very large. Such an overhead is unnecessary.

Subspace Finger (SF) or Tightly Coupled Spaces: If every node has a neighbour from the other space, routing into it is a single hop away from every node. However, in order to do this, each node from space S1 needs an extra finger to link to the other space S2 and, in order to avoid hotspot routes, those fingers should point at S2 nodes in evenly distributed fashion. The fingers should be bi-directional in order to guarantee that a change in one node of one of the spaces also updates the state of the node from the other space. The increase in state overhead for this strategy is very large: at most one finger per space node and the extra management overhead with those fingers. The message and hop cost is $1 + \log S$ (S messages for broadcast in S).

Subspace State (SS) or Mixed Spaces: The best performances could be obtained if every node would maintain state information (fingers) as if it were part of the other space as well ($\log S$ fingers on Chord) -observer status. It would be able to route directly into any other node in the other space. The message and hop cost is $\log S$ ($S-1$ for broadcast in S).

4.4 Gateway-Based Strategies

DHT-Gateway (DG, DiG): This strategy uses the DHT content-based routing capability. The lookup value is the name of the other space. In Chord

this means that the DG is the successor of the hash value of the space name: succ(hash(SS)) for space SS. The DG node is the gateway - besides its typical state, it must either join the other space or create a bi-directional link to it. Requests are routed to the gateway node based on the destination space name in at most $\log N$ messages and hops. This strategy assumes that the node with identity succ(hash(SS)) and the gateway to the other space must be the same one, which is limitative. Alternatively, an indirect approach (DiG) allows the gateway (DG) (node connecting to the other space) and "gateway pointer (DG-pointer)" (node with identity succ(hash(SS))) to be separate entities, by creating a finger in the DG-pointer to the DG node. Figure 3a shows a requester node R of space S routing into subspace SS through the DG-pointer node, which is the node with identity succ(hash(SS)), and from there into the gateway DG, which then routes within SS using normal intra-space routing mechanism of Chord.

The DG is a potential hotspot for heavy inter-space traffic and a single-point of failure. These problems are solved using multiple gateways, as we propose next.

Multiple Gateways (mDG): multiple gateways (and pointers) can be used to avoid single-point-of-failure and gateway hotspots. DG nodes are multiplied (DG-pointers as well, if used). We propose that they be spread evenly by the space. If there are n DG (or DG-pointer) nodes in a space with N nodes, they are chosen as multiples of N/n offsets from the succ(hash(SS)) DG or DG-pointer node. Requesting nodes compute succ(hash(SS)) and choose their nearest gateway (from the N/n offsets). Figure 3b illustrates the routing of request R into subspace SS with mGD using the nearest mDG-pointer. We note that the number of gateway nodes can be made to adjust dynamically depending on traffic, but we leave this issue for future work. Assuming that m gateways are evenly distributed:

- Number of messages: $\leq \log N/m + \log S$ (single node lookup) or $1 + \log N/m + (S-1)$ (broadcast within subspace).
- Number of hops: $1 + \log N/m + logS$;

Space Broadcast with Multiple Gateways (SB-mDG): The use of multiple gateways can also be useful to decrease the overhead of space broadcast. The broadcast then occurs only within a fraction of the space. The overheads are:

- Number of messages: $\leq (N/m - 1) + \log S$ (single node lookup) or $(N/m - 1) + (S-1)$ (broadcast within subspace).
- Number of hops: $\log N/m + \log S$;

The next proposals aim at achieving top performance without large state overhead.

4.5 Caching-Based Strategies

SB- or mDG- with Cached Subspace Fingers (CSF): A balance can be obtained between the efficiency of the SF strategy and the small state achievement

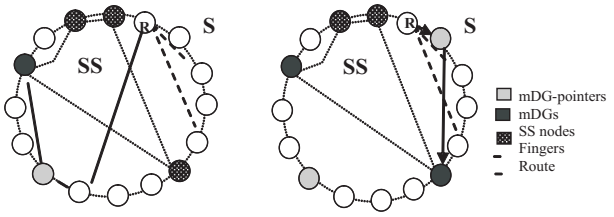


Fig. 3. Routing from R to Subspace: (a) DiG (b) mDG

of SB or mDG, by caching fingers upon use instead of creating and maintaining them force-fully. When a request is made, the node verifies if it already has a finger to the target space, in which case it is used to route directly into it. Otherwise, it uses the mDG or SB strategies to answer the request. However, the gateway node is instructed to return a direct finger into the other space to the requester, which then caches it, optionally with a TTL and LRU policy for caducity. This way, the next time the node makes a request into the other space, the cached finger can be used to access it directly. On-demand lazy updates minimize update overheads when something changes in the other space. If an outdated finger is used, the supposed gateway intercepts it and responds invalidating the finger. If the gateway is not there any more, a timeout invalidates the finger. The requester then resorts to mDG or SB again, after which it gets a valid finger once more.

Gateway Publishing (GP): When a gateway is created (based on some policy, possibly QoS-based), it broadcasts a pair (gateway address, target space) to all other nodes in the space. When the gateway changes its status (e.g. when it ceases to be a gateway) it also broadcasts the state update. This way a node can cache gateway addresses to be used when needed. Multiple gateways can be used and nodes can choose the most appropriate one, based on some policy. This strategy has a specific extra overhead related to the need to broadcast into all nodes the advertisements and the space necessary in each node to hold gateway information. From the perspective of routing overheads and response time, this alternative is similar to CSF after the finger to the gateway is cached, therefore we do not analyze it separately.

5 Comparative Analysis

Figure 4 shows a comparative summary of the overhead of alternatives for a single request. We show the maximum number of messages (#msgs) and number of hops (#hops) that are necessary to route within the requester' space. The last column also shows the state overhead in total number of extra fingers that must be kept in nodes plus gateway (we assume there may be 1 or $f = \log S$ fingers per gateway). We can see that SF, SS, CSF and GP are by far the most efficient from a message routing perspective but involve a considerable total

state overhead (assuming that N is large). On the other hand, although SF and SB have no state overhead and a number of hops equivalent to most other strategies, the number of messages generated is extremely large for SF and large for SB, which has important consequences for the traffic in the whole overlay network. The strategies based on gateways or SB with m gateways have more balanced overheads, although their efficiency is not as good as the best routing strategies. Caching strategies behave like mDG/SB-CSF for the first request (or when the finger is outdated) and like SF for other requests. The next figures use the formulas in Figure 4 to show how the values vary as the number of nodes increases. Figure 5a (log plot) shows the maximum number of messages generated for a single request as the number of nodes N increases from $1k$ to $8M$. In the figure the strategies are roughly the same as in Figure 4 and SBx is "space broadcast" with x gateways. From here we can see that it is important to use routing strategies with state instead of broadcast (or flooding) to minimize the number of messages that are necessary, as the number of messages generated by SB (or SF) can be very large. SB with multiple gateways decreases the overhead significantly, but it is still much above the best strategies. SF, CSF and GP require only one message (direct link) and SS requires 0 messages (it fingers directly within the subspace). The "latency" of a request - in number of hops - is shown in Figure 5b. In no state (SF, SB) or small-state strategies (DG, mDG) multiple gateways are required to drive the number of hops down and the number of hops for space broadcast (SBx) or gateways (mDGx) is similar. The more tightly coupled alternatives SF, CSF, GP and SS are fastest because they provide direct fingers.

In Figures 6 we show the state overheads in #fingers ($L = \log N$ defines a variable number of gateways). Both SS and SF, CSF, GP have overhead that is $\mathcal{O}(N)$, many orders of magnitude larger than the others. Least state overhead (0) is achieved for DG, SB, SF (strategies with larger latencies and #msgs). These results show the tradeoff between state space that is required and traffic that is generated during routing of requests: small or no state alternatives generate mush more traffic as they need to search for the gateways and are slower than strategies with more state, that is, direct fingers into gateways or nodes from the other space to be reached. The SS and SF strategies clearly involve too much state overhead, as each node must always have fingers to nodes in the other space (and in

	#msgs (<=)	**#hops (<=)**	**state**
Space Flood (SF)	N log N	log N	0
Space Broadcast (SB)	N-1	log N	0
DHT-Gateway (DG)	log N	log N	*f*
DG + DGpointer	1+log N	1+log N	1 or *f*
Multiple Gateways (mDG)	1+log N/m	1+log N/m	mx(1or *f*)
SB+mDG	N/m-1	log N/m	mx(1or *f*)
Subspace Finger (SF)	1	1	N
Subspace State (SS)	0	0	N log S
Cached Fingers (CSF)	1	1	N
Gateway Publish (GP)	1	1	N

Fig. 4. Overhead Formulas

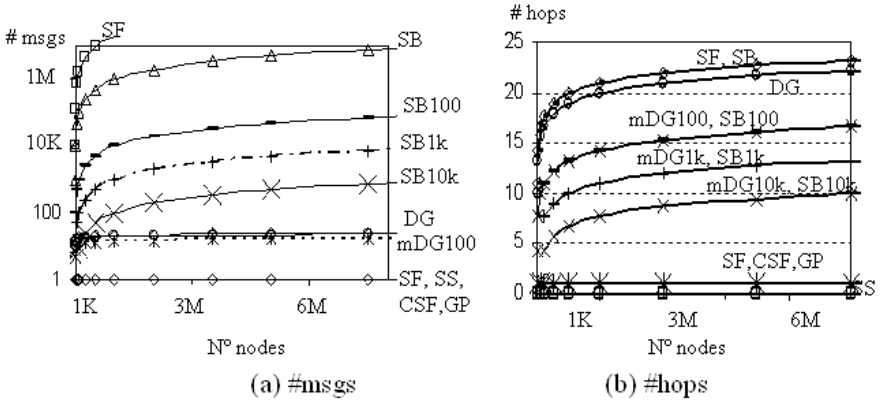


Fig. 5. Overheads: #msgs and #hops

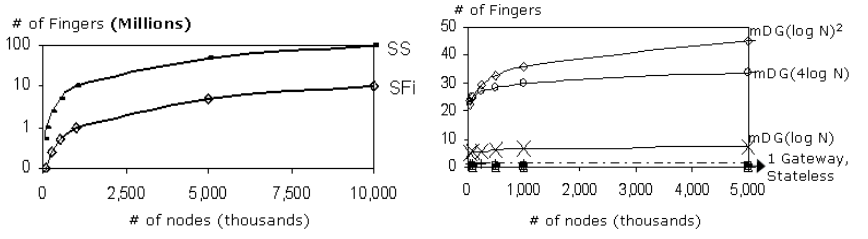


Fig. 6. State Overhead: (a) SS and SF (b) Other Strategies

SS each node must have $\log N$ fingers). This is even more problematic in highly dynamic environments due to state updates. The alternative no-state strategies (SB, SF) are also inconvenient for generating too much traffic overhead. The approaches we devised based on multiple gateways (mDG or SB-mDG) strike some balance between traffic and response time on one side and state overhead on the other side. By assigning the responsibility of broadcasting their gateway status and modifications to it to gateways (GP) we have shown how SF can be implemented easily. Finally, by caching only needed fingers (CSF) on demand, we can have top performance while avoiding unnecessary state overhead of (SS, SF).

6 Conclusions and Future Work

In this paper we motivated and proposed multiple overlay spaces and multiple-space routing in DHT-based overlay networks. We have compared stateful and stateless alternatives and their tradeoffs. We have shown analytically that flood or broadcast-based stateless strategies (SB or SF) can generate a lot of undesirable traffic, while stateful alternatives - space fingers and especially full space

state (SF, SS) - have large state overheads. We also proposed gateway strategies with and without indirection. In order to minimize the gateway bottleneck issue, we proposed alternative multiple gateways (mDG) and finger caching (CSF) strategies. We also proposed gateway publishing (GP), which is associated with CSF. We compared the alternatives analytically concerning the amount of traffic and n of hops, showing the tradeoffs and concluding that CSF may be a good option considering all metrics. In future work we will consider routing strategies and structures in a global universe of overlay spaces using "Space Name Servers".

References

1. Harvey, N., *et al.*: Skipnet: A scalable overlay network with practical locality properties. In: Proc. USENIX Symposium on Internet Technologies and Systems. (2003)
2. Stoica, *et al.*: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. International Conference SIGCOMM. (2001)
3. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proc. International Conference SIGCOMM. (2001)
4. Rowstron, Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proc. International Conference on Middleware. (2001)
5. Zhao, J., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In: Tech. Report CSD-01-1141, UC Berkeley. (2001)
6. Garcez-Erice, L., Biersack, E., Ross, K., P.Felber, Urvoy-Keller, G.: Hierarchical p2p systems. In: Proc. International Conference on Parallel and Distributed Computing. (2003)
7. Cai, M., Frank, M., Chen, J., , Szekely, P.: Maan: A multi-attribute addressable network for grid information services. In: Proc. 4th International Workshop on Grid Computing. (2003)
8. Cai, M., Frank, M.: A scalable distributed rdf repository based on a structured peer-to-peer network. In: Proc. 13th International Conference on WWW. (2004)
9. Huebsch, *et al.*: Querying the internet with pier. In: Proc. International Conference on Very Large Databases. (2003)
10. Ng, W., Ooi, B., Tan, K., Zhou, A.: Peerdb: A p2p-based system for distributed data sharing. In: Proc. 19th International Conference on Data Engineering. (2003)
11. Ratnasamy, S., *et al.*: Topologically-aware overlay construction and server selection. In: Proc. 21th International Conference INFOCOMM. (2002)
12. Ng, W., Zang, H.: Predicting internet network distance with coordinates-based approach. In:Proc. 21th International Conference INFOCOMM. (2002)
13. El-Ansary, Alima, L.O., Brand, P., Haridi, S.: Efficient broadcast in structured p2p networks. In: Proc. International Conference on Peer-to-Peer Systems. (2003)

Benefit and Cost of Query Answering in PDMS

Armin Roth and Felix Naumann

Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
{aroth,naumann}@informatik.hu-berlin.de

Abstract. Peer data management systems (PDMS) are a natural extension to integrated information systems. They consist of a dynamic set of autonomous peers, each of which can mediate between heterogenous schemas of other peers. A new data source joins a PDMS by defining a semantic mapping to one or more other peers, thus forming a network of peers. Queries submitted to a peer are answered with data residing at that peer and by data that is reached along paths of mappings through the network of peers. However, without optimization methods query reformulation in PDMS is very inefficient due to redundancy in mapping paths.

We present a decentral strategy that guides peers in their decision along which further mappings the query should be sent. The strategy uses statistics of the peers own data and statistics of mappings to neighboring peers to predict whether it is worthwhile to send the query to that neighbor—or whether the query plan should be pruned at this point. These decisions are guided by a benefit and cost model, trading off the amount of data a neighbor will pass back, and the execution cost of that step. Thus, we allow a high scale-up of PDMS in the number of participating peers.

1 PDMS and Data Quality

Integrating semantically relevant information is a pressing problem. In practice, it can be observed that a decentralized P2P fashion of data sharing is preferred over centralized data integration systems. Users desire to pose queries to their own schema, and let the queries be transferred via schema mappings to similar peers in the neighborhood. Such requirements are addressed by *peer data management systems* (PDMS) [1,2,3]. Peers serve both as data sources and as mediators and queries are translated and transferred using semantic relationships between peers, so-called mappings, as shown in Fig. 1.

Example application areas include partnerships between companies for developing complex technical products, cooperations of scientific institutions, and ad hoc crisis management [2]. PDMS can also serve as a decentralized infrastructure for mediation between ontologies in the semantic web. Like any information system integrating data from autonomous sources, PDMS are vulnerable to poor data quality in the sources, poor mappings to the sources, and thus poor data quality of query results. Compared to conventional integrated information systems, this problem is particularly large in PDMS. The data is passed through

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 50–61, 2007.

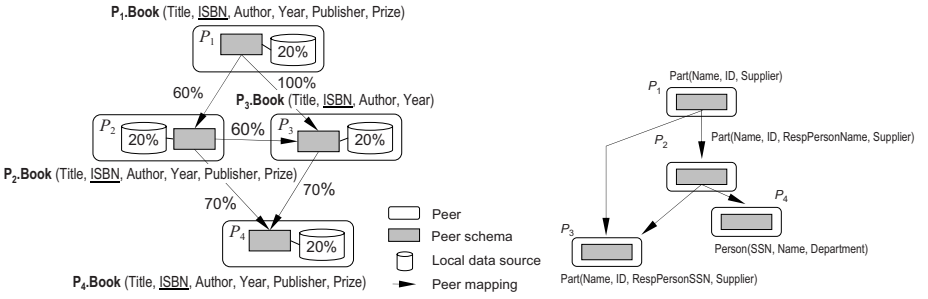


Fig. 1. Two example PDMS with annotated source coverages, mapping selectivities, and schema

numerous mappings, each of which can decrease data quality by cumulative projections and selections.

In general, information quality (IQ) is an important discriminator of data sets. Here, content-related IQ-criteria are of major interest, including accuracy, relevancy, and completeness. Due to their autonomy, local sources at peers are likely to show quite different information quality. For this reason, consideration of information quality in query answering for PDMS consisting of a large number of peers is an important problem. In this paper we concentrate on the IQ criteria *coverage* and *density*, which together form an overall *completeness* measure for data sources and query results [4]. Current query planning algorithms, such as [2], find *all certain answers* to a query. This is not feasible in a web-scale PDMS: The search space becomes enormous and query plans become very long, increasing chance of information loss along the paths. In this paper we propose methods to free PDMS from this restriction and relax the notion of completeness turning it into an optimization goal instead.

Example 1. Consider the simple PDMS depicted on the left in Fig. 1. Its peers share a single relation Book. However, peer P_3 only exports four attributes, whereas the others provide two additional attributes, which are projected out by mappings. Furthermore, the mappings between the peers are annotated with selectivity scores, representing selection operations at the mappings. They are a measure for the fraction of data provided by the peer at the mapping head that is expected to be transferred via the mapping, e.g., the mapping from P_2 to P_3 removes all but 60% of the tuples provided at P_3.

To achieve our goal of efficient and effective query answering in large PDMS, this paper contributes a comprehensive completeness model for peers and mappings within a PDMS. We show how to calculate completeness scores for PDMS query plans. This can be used to improve the search for query plans providing high completeness by pruning mappings which show considerable loss of information. Furthermore, our completeness-aware query planning algorithm needs no central catalog, thus leaves maximum autonomy to the peers.

The remainder of the paper is organized as follows. The completeness model for PDMS is introduced in Sec. 2. Then, we provide a brief overview on query

planning in PDMS in Sec. 3. Our approach to prune subplans is explained in Sec. 4 and experimentally studied in Sec. 5. Related work is discussed in Sec. 6 and we conclude in Sec. 7.

2 PDMS and Completeness

2.1 Data Sources and Mappings in PDMS

We formalize a PDMS as a set $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ of peers, each of which comprises local data sources and mappings both to the local sources and to other peers.

Peers. In general, a single peer can be perceived as a data integration system consisting of a *peer schema* S and local data sources. The peer schema describes data that the peer provides to users, applications, and to other peers. Local data sources are specified by a set $\mathcal{L}$ of *local schemas*. They are connected to the peer schema by a set $\mathcal{M}_L$ of *local mappings*. Other peers are related to a peer schema by *peer mappings*, which form the set $\mathcal{M}_P$. In all, each peer is represented by the four-tuple $P = (S, \mathcal{L}, \mathcal{M}_L, \mathcal{M}_P)$. We use Datalog notation to express the relational data model used for schemata, queries, and mappings.

Mappings. Our approach is based on Global-Local-as-View mappings, or GLaV mappings. Local mappings are of the form $Q_L(\mathcal{L}) \subseteq Q_S(S)$, where Q_L and Q_S are *conjunctive queries*. Similarly, a peer mapping $Q_1(\mathcal{P}_1) \subseteq Q_2(\mathcal{P}_2)$ establishes a relationship between the relations of peer schemas of the two *sets* of peers $\mathcal{P}_1$ and $\mathcal{P}_2$. Intuitively, the peer mapping means that Q_1 always returns a subset of the resulting set of tuples of Q_2. GLaV mappings in their general form can be transformed to a combination of a GaV and a LaV mapping with a fresh relation symbol [2].

Selections play an important role when regarding the completeness of query results. Selection predicates in mappings express implicit knowledge about peer schemas. For instance, in writing a mapping to a peer of a company Ford and its relation Product, one can express that the peer models products of this company *only*. A way to express this selection in a GLaV mapping from a peer of a company AllParts, which offers knowledge about producers of parts, is to insert a selection predicate: Ford.Product(Name, ID, Supplier) $\subseteq$ AllParts.Part(Name, ID, Supplier), Supplier = "Ford". Also, there may be projections in local and peer mappings that can affect the completeness of query results. Consider a mapping from the Part relation of a company Bosch, which includes contact information, to the peer Ford. Because the Product relation of Ford does not offer this information, we must use a projection in the mapping: Ford.Product(Name, ID, Supplier) $\subseteq$ Bosch.Part(Name, ID, -, Supplier).

2.2 Completeness of Data Sets

In many scenarios, users of large integrated information systems are not interested in *all* certain answers, because they are not able to examine them all in detail. Another constraint to large PDMS is the limitation of resources for query

evaluation and transmission of query results. Facing these restrictions, a user may be satisfied with a small number of answers of highest quality.

Coverage. Extensional completeness, also *coverage*, describes the proportion of the size of a tuple set to the number of *all* tuples stored within a PDMS. The measure applies both to the data set a peer actually stores and to a query result.

To calculate coverage we make the closed world assumption for the whole system. Users perceive a PDMS as a single database described by the schema of the peer. Thus, the size of the world $|w_Q|$ referred to by a query Q against this schema is the number of tuples matching the query that can be reached using the network of mappings. In practice, however, knowing this number precisely is not necessary, because it plays only the role of a normalizing factor.

Definition 1 (Coverage). *Let $\mathcal{D}_\mathrm{Q}$ be a set of tuples answering a query Q. The coverage of $\mathcal{D}_\mathrm{Q}$ with respect to a world w_Q is $c(\mathcal{D}_\mathrm{Q}) := |\mathcal{D}_\mathrm{Q}|/|w_Q|$.*

Density. The intension queried by the user is the set of attributes $\mathcal{A}_Q$ asked for in the query. Intensional completeness of data sets, also *density*, first suffers from `null` values in data sources. Secondly, attributes that are mentioned in the query may not be available at certain data sources in the PDMS. The user may be nevertheless interested in having tuples in the query result despite their missing attributes. Values of missing attributes are filled with `null` values, thus creating incomplete tuples. Attribute density is used as a measure for this kind of completeness. The query-dependent density is the arithmetic mean over all attributes occurring in a query.

Definition 2 (Attribute density). *Let a_R be an attribute of a relation R. A projection of a tuple t of this relation to a_R is denoted by $t[a_R]$. With $\perp$ denoting* `null`*, the attribute density of a tuple set $\mathcal{D}$ for R is defined as $d(a_R) := |\{t \in \mathcal{D} \,|\, t[a_R] \neq \perp\}|/|\mathcal{D}|$.*

Completeness. Intuitively, overall completeness can be regarded as an aggregated measure for the ratio of the amount of data in a certain data set to the amount of data in the world w_Q. It is a combination of coverage and density, which we aim to maximize. In [4] it is shown that the completeness score of a data set $\mathcal{D}$ can be calculated as $C(\mathcal{D}) = c(\mathcal{D}) \cdot d(\mathcal{D})$ and $0 \leq C \leq 1$.

3 Query Planning and Completeness

In this section we review a query planning procedure for PDMS and show how to value peer mappings according to their completeness. The following Section 4 uses this measure to prune poor plans or subplans.

3.1 PDMS Query Planning

To translate a query, the subgoals are reformulated and passed on along the mappings to other peers, which in turn recursively send the query to their neighboring peers, etc. Reformulation terminates when all branches of recursion have reached

local sources, where the queries can be evaluated on actual data. Clearly, this process can be performed fully decentrally. To show how the query reformulation actually uses the mappings between the peers and between the local sources and the peers, we briefly review the reformulation algorithm of [2].

Creation of the reformulation tree. Consider a query Q posed to some peer. We aim at a set of query plans, which only contain subgoals representing relations from *local* schemas. The answer to Q is the "union" of the results of all these query plans. The reformulation algorithm published in [2] constructs a so-called rule-goal tree (Fig. 2 on the left). The goal nodes are formed by (reformulated)

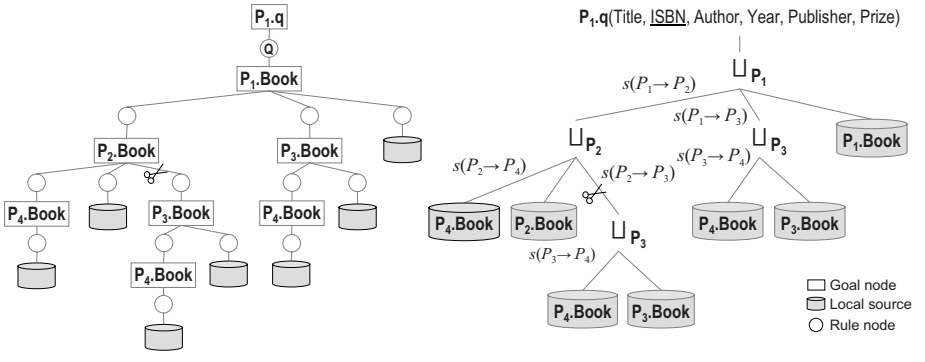


Fig. 2. Rule-goal tree (left) and query plan (right) of our example from Sec. 1. Selections are represented by the annotated mapping selectivities s.

subgoals to be answered, whereas the rule nodes represent the mappings. The algorithm continues by expanding leaf nodes using either peer or local mappings. Depending on the form of the mapping at hand, either a GaV- or a LaV-style reformulation is performed. In the former case new goal leaves are obtained by unfolding the view forming the mapping. If the mapping represents a view *from* any peer or local source on the schema to which the leaf goal node belongs, the MiniCon algorithm for answering queries using views is employed [5]. Please note that this algorithm may be performed fully decentrally by the peers.

Determining query plans. Two approaches are possible to create a query plan. To achieve first query answers quickly, several query plans may be determined sequentially in the way shown in [2]. In contrast, if we want to calculate the overall completeness of the query result, it is more useful to derive a single overall query plan from the rule-goal tree (Fig. 2). To obtain this single query plan, we recursively traverse the rule-goal tree. Several outgoing mappings at a certain goal node lead to a branch in the rule-goal tree. In our query plan all those subtrees are combined by a union operator. Branching rule nodes are created by a GaV expansion containing a join. Such a situation is reflected by a join operation in the query plan. Due to space limitations, we refer to [2] for transforming LaV expansion into the query plan.

3.2 Completeness of Query Plans

Intuitively, query reformulation for PDMS is a search problem. During exploration of the search space, we lack information about the completeness contribution of local data sources not reached yet. As a consequence, to intelligently explore the search space collecting high quality query results, we must decide which mappings promise to be useful. This leads to the question how mappings (and the data sources "behind" them) contribute to the overall completeness of the query result.

In this work, we assume the mappings to include only select-project-join (SPJ) queries. In the following, we show how to calculate the influence of S, P, and J operations used in the mappings on the coverage and density of query results. Additionally, query plans contain union-type operators, which collect results returned by alternative mapping paths starting from a certain peer.

Influence of selections and projections. Applying a selection σ to a tuple set of a relation R reduces the set of tuples by a selectivity factor s. Hence, we can calculate coverage of the selection result as $c(\sigma(R)) = s \cdot c(R)$. Assuming that `null`-values are distributed equally over all tuples, density is not affected by a selection $d(\sigma(R)) = d(R)$. If no statistics about s are available, sampling techniques may be employed to assess it (Sec. 7). Note that this selectivity is applied to the data of the target of a mapping but also on all other sources reachable through that mapping. This observation is the foundation of our heuristic for reducing the reformulation effort (Sec. 4).

Without concessions to the completeness of query answers, projection of query attributes would not be allowed in mappings. Attributes projected out have to be padded with `null`-values. Since a projection $R[\mathcal{A}_P]$, which reduces the attribute set $\mathcal{A}_R$ of R to the attribute set $\mathcal{A}_P$, leaves the number of tuples of R unchanged, the extensional completeness of the result is not affected: $c(R[\mathcal{A}_P]) = c(R)$. In contrast, the *query-dependent* density value is recalculated subtracting the density of the set of attributes projected out: $d_Q(R[\mathcal{A}_P]) = d_Q(R) - d_Q(R[\mathcal{A}_R \setminus \mathcal{A}_P])$. For simplicity, we assume here that projections do not reduce coverage, i.e., duplicates generated by projection are not eliminated. We use this observation in our heuristic to decide which mappings suffer from loss of information (Sec. 4).

Example 2. Suppose we are given the following mapping between two relations at different peers with the attribute densities listed thereafter: P_4.Book(Title, ISBN, Author, Year, Publisher, Price) $\subseteq$ P_3.Book(Title, ISBN, Author, Year) (Fig. 1).

	Title	ISBN	Author	Year	Publisher	Price
P_4	90%	100%	80%	60%	40%	70%
P_3	90%	100%	80%	60%	0	0

The attribute densities and the query-dependent density of $P_4.Book$ as it is exported by peer P_4 amounts to $d_Q(P_4) = 73\%$. Due to the two projections in the mapping (attributes Publisher and Price) the density of the *same* data set is perceived at P_3 with a value of $d_Q(P_3) = 55\%$, assuming the query asks for all attributes. As can be seen, projecting out a third of the attributes reduces the query-dependent density by about a third in this example. It is important

to note that we do not need any statistics to compare mappings wrt. loss of completeness due to projections.

Influence of joins. Suppose we are given the tuple sets of the two relations R_1 and R_2 together with their respective coverage and density values. We aim to calculate completeness for the result of the join $R_1 \bowtie R_2$. If we assume *independence* of the representation of objects, which means that there is no knowledge about extensional overlap, we can draw the following formulas for the expected coverage and density from [4], where $\mathcal{A}$ denotes the union of the attribute sets of R_1 and R_2:

$$c(R_1 \bowtie R_2) = c(R_1) \cdot c(R_2) \tag{1}$$

$$d(R_1 \bowtie R_2) = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} (d_{R_1}(a) + d_{R_2}(a) - d_{R_1}(a) \cdot d_{R_2}(a)) \tag{2}$$

Influence of unions. In general, the contributions to a goal node's answer are not union-compatible. We use the full outerjoin-merge operator $\sqcup$ from [4], which is similar to the "outer union" but allows all common attributes but the key attribute to have conflicting data values. The following equations provide the expected coverage and density of a full outerjoin-merge for the case of independent data sets:

$$c(R_1 \sqcup R_2) = c(R_1) + c(R_2) - c(R_1) \cdot c(R_2) \tag{3}$$

$$d_{R_1 \sqcup R_2}(a) = \frac{d_{R_1}(a) \cdot c(R_1)}{c(R_1 \sqcup R_2)} + \frac{d_{R_2}(a) \cdot c(R_2)}{c(R_1 \sqcup R_2)} - \frac{d_{R_1}(a) \cdot d_{R_2}(a) \cdot c(R_1 \bowtie R_2)}{c(R_1 \sqcup R_2)} \tag{4}$$

Using this, the density of $R_1 \sqcup R_2$, which comprises the attribute set $\mathcal{A}$ is the arithmetic mean of the attribute densities: $d(R_1 \sqcup R_2) = 1/|\mathcal{A}| \sum_{a \in \mathcal{A}} d_{R_1 \sqcup R_2}(a)$. In [4], associativity is shown for the coverage criterion for $\bowtie$ and $\sqcup$; density is proven to be associative only for independent data sets.

Calculating query plan completeness. Using the means presented in the previous section, we now can calculate the *expected* completeness of query plans. In this way we may compare different strategies for completeness-driven PDMS query reformulation. Please note that such calculations cannot be performed by a single peer in a real PDMS, because it is based on global information. Rather, calculations are accumulated along peers as the reformulated query is passed on.

We pass along between the peers information about the aggregated mapping selectivity for every distinguished variable and the accumulated projections. Clearly, several selections based on the same variable may be aggregated along a mapping path by multiplying their selectivities, which we assume to be the result of statistical assessment methods (Sec. 7). If we further assume the variables of the user query being independent of each other, their respective mapping selectivities can also be aggregated by multiplication. The main idea of our algorithm to calculate the query plan completeness is to traverse the rule-goal tree recursively in the same way as for single query plan creation (Sec. 3.1), and to combine the completeness scores on the way back starting with the given values

at the leaf nodes, which represent only local sources. Combinations of coverage and density scores of subtrees are required at the occasions highlighted below. Coverage and density scores of data originating from the local sources are decreased by selectivities and projections in the mappings on the path to a certain peer. If every goal node returns its coverage and density based on the user query, we can calculate the coverage and density the receiving peer perceives using that peer's aggregated mapping selectivity and projections for the path from the peer to the root of the rule-goal tree.

Branching goal nodes. In this case we use the results from the rule node children, which are coverage scores referring to the goal node's peer. As in query plan creation we must perform a union and thus may use Equation (3) to calculate the resulting coverage. Attribute densities are calculated using Equation (4).

Branching rule nodes. To calculate the coverage scores of rule nodes, we use the results from the underlying goal nodes. As a branching rule node is created by a join in a mapping, Equation (1) can be employed to determine the resulting coverage. In the last step we multiply with the mapping selectivity $s(m)$ to return the coverage score of the data the above peer actually receives. With x denoting a distinguished variable of the user query occurring in the mapping m, the mapping selectivity is $s(m) = \prod_{x \in m} s_x(m)$. We use Equation (2) to propagate the attribute densities.

Our algorithm captures LaV expansions as well, but for brevity we omit this description. When our recursive algorithm reaches the root of the rule-goal tree we can calculate the query dependent density and, finally, the completeness of the query. We summarize the set of statistics we assume a peer should maintain in our PDMS setting: (i) Coverage and attribute densities for all local data sources, (ii)Selectivities for all selection predicates in outgoing peer mappings, and join selectivities for outgoing mappings that contain joins over different peers.

4 Pruning Subplans

The rule-goal tree typically becomes very large and shows a high branching factor even for relatively small PDMS with tens of peers [6]. Therefore, handling queries in web-scale PDMS using the algorithm from Sec. 3.1 is not feasible. To keep query reformulation in PDMS tractable for large PDMS, e.g., for semantic web applications [7], it is crucial to optimize both query planning and evaluation to reduce latency and determine first answers quickly.

To meet these problems, we exploit the influence of mappings on the query results to decide which mapping paths are not worth following or may be deferred. Our approach tries to identify mapping paths that preserve potential completeness of the intermediate query results "behind" these mappings. In general, during the query reformulation phase a peer has no knowledge about the completeness of data it will receive during query evaluation. In the rule-goal tree this fact is expressed by having subgoals of local sources only at the leaves of the tree.

If we want to avoid coordination between peers about completeness of intermediate query results, only the "completeness" of peer mappings may be

exploited to prune the search space. We build on the results from Sec. 3.2 to characterize the influence of mappings on the completeness of query results in the PDMS context. In particular, we introduce a heuristic to either prune the search space or defer expansion of certain mappings to determine the first answers showing high completeness quickly. Because query reformulation is done recursively, every mapping that needs not be used potentially saves consideration of many more mappings later. Additionally, in a highly interconnected PDMS the probability that for a certain mapping path there are alternatives with less loss of information is considerable. In Sec. 5 we present experiments showing this effect.

Using selections and projections. Selections not overlapping with the user query may significantly reduce the amount of data transported by a mapping. Based on this observation we propose the following strategy for completeness-based query reformulation in PDMS: If a goal node is related to several mappings that can be used for expansion, we order them by their selectivity, favoring less selective mappings. To break ties, we additionally regard projections, favoring mappings with less projections. In a simple strategy, we only use a threshold for a normalized measure combining selectivity and the number of variables projected out to decide which mappings to follow and which to prune (Fig. 2). This strategy is fully decentral, i.e., no coordination between peers is needed at all. However, we lose answers. That is why this approach requires a trade-off between cost and benefit of the answer, which conforms to the concession to the completeness of query answers in large PDMS described above.

Using joins. Assessing the impact of joins on the coverage of (intermediate) query results is more subtle. There may be situations where very small join hit rates between the relations to be joined lead to considerable loss of information. In such cases it is desirable to have alternative mapping paths that may help to exploit more data from the join partners. The following example illustrates such a situation.

Example 3. Regard the PDMS on the right of Fig. 1. The relation Part models parts of technical products having attributes, such as name, identifier, social security number and name of the responsible person, and the supplier. Person is a simple list of persons that are referenced by the foreign key RespPersonSSN. Assume that P_1 is queried on parts of a certain supplier.

We consider a situation where the overlap between P_3.Part and P_4.Person is very small compared to the size of both relations. As a consequence, the join P_3.Part $\bowtie$ P_4.Person filters out all parts from P_3 where the corresponding author is not included in P_4.Person. It follows that P_2.Part may offer only a small fraction of parts stored at P_3. However, our query posed to P_1 does not ask for responsible persons. Hence, with respect to the query, the join performed at P_2 loses most of the "coverage" available at P_3. In recognizing this situation query reformulation could decide not to expand the goal node representing P_2.Part. Here, this would not affect coverage of the final query result, because there is an alternative mapping from P_1.Part to P_3.Part, which may help to retrieve all parts at P_3.

As a conclusion from this example we propose a look-ahead strategy to handle *joins between peers*: (1) Check if the query to be answered requires to perform the join. (2) If so, try to assess the information overlap between the join relations. (3) If the *loss* of coverage is above a certain threshold, prune or defer the expansion performing the join. (4) If the joined peer that contributes tuples to the query result (P_3 in the example) can be reached on an alternative mapping path, finalize the decision of the last step. Observe, that this approach requires coordination between a small set of peers.

5 Experiments

Due to the complex structure of PDMS the effect of most algorithms and strategies can be validated only experimentally. Our PDMS implements the pruning strategy exploiting selections and projections in the peer mappings. The query reformulation is simulated on a single computer. However, we note that this does not imply any restrictions compared to a fully decentral query reformulation on multiple sites. In our experiments every peer covers only 5% of the size of the world. The following table lists the data sets and their characteristics:

	Peer schema	#Peers	*rank*
$\mathcal{P}_1$	Single relation	10	5.7
$\mathcal{P}_2$	Single relation	30	3.4
$\mathcal{P}_3$	Heterogeneous	50	2.5

Measurements. Alternative paths in highly interconnected PDMS lead to a quite strong convergence of the completeness to the final result, as can be seen in Fig. 3 on the left (solid line). There, the final coverage value is almost reached after about half of the cost for obtaining all answers has been spent. This means that in the second half of the query reformulation phase many mapping paths are exploited that are expected to contribute almost nothing to the final result. We use our approach of threshold-based pruning of mapping paths. Regard the results depicted on the left in Fig. 3. They show that pruning of mappings containing selections with medium selectivities (11% of all mappings concerned, dashed line) may still yield the same completeness as without pruning, but at half of the cost compared to the experiment without pruning. Moreover, if we choose an even stronger pruning condition (33% of all mappings concerned), the cost decreases to 10 times less than without any pruning. The results for the dataset $\mathcal{P}_2$ are displayed on the right in Fig. 3. It also shows a cost reduction of more than an order of magnitude along with improved completeness at any time during query reformulation. In this experiment maximally 20% of all mappings fell under the pruning condition. With the heterogenous PDMS $\mathcal{P}_3$ we yield similar results. There, about 10% of all mappings include selections and a fourth projects out some attributes. We pruned all mappings which are expected to lose more than a third of the data of the peer at their head (34 prunings and 1175 reformulations in total) while still achieving 100% completeness. Observe that

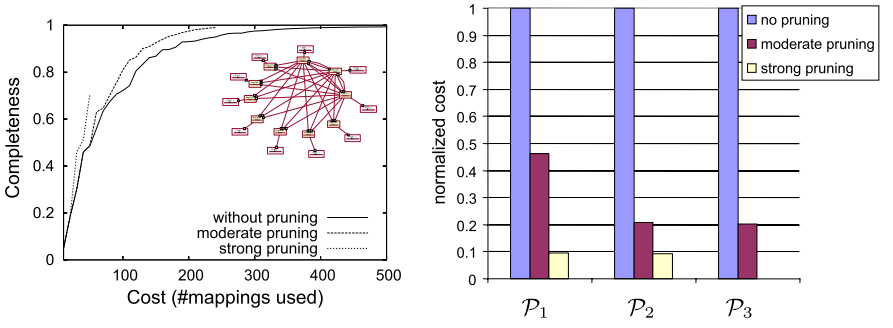


Fig. 3. Results and topology of $\mathcal{P}_1$ under different pruning thresholds and cost reduction. Normalized cost are based on the cost for using all mapping paths. Notice that ending graphs indicate a cost reduction (explicitly depicted on the right).

pruning has to be applied with care: As depicted in the left-most graph in the left diagram in Fig. 3 very strong pruning can lead to a significant reduction of query answer completeness. Of course, choosing suitable pruning thresholds is a matter of experience. We believe that statistics about query answers could help (Sec. 7). In summary, our experiments clearly show effectiveness and considerable efficiency gains by applying a rather simple IQ-based pruning of the search space.

6 Related Work

In this section we review the PDMS literature under the aspects of information quality and efficiency of query answering. The mediation between schemas of a PDMS is the main concern of *Piazza* [2]. Concessions to the completeness of query results are mentioned, but not discussed in detail. New algorithms usable for safe pruning during query reformulation are contributed in [6]. However, they are independent from information quality, which according to the authors remains an open challenge. Additionally, this pruning approach involves non-local coordination between peers, whereas our mapping-based strategy is strictly local to autonomous peers. The *Semantic Gossiping* approach of Aberer et al. uses cycles in mapping networks to examine loss of information [1]. That is, instead of explicitly modeling completeness as in our approach, the authors use instance sampling to assess information quality criteria. The authors use a simple data model and mappings only between attributes and without selection queries. Calvanese et al. [3] propose new semantics for PDMS based on epistemic logic, which leads to general decidability. In this semantics only consistent facts are exported by a peer. However, the "weaker" logic loses some of the answers compared with our first order logic approach. In *Edutella*, semantic overlay networks consist of clusters of semantically "similar" peers [8]. This approach does not utilize arbitrary mappings between peer schemas. According to [9], inaccuracies and uncertainties in mappings are an important research perspective, which we have not adopted yet.

7 Conclusions

Peer data management systems offer a decentralized and dynamic infrastructure to share heterogeneous data between autonomous peers. To scale PDMS to a large number of peers it is crucial to optimally trade-off between the cost of query execution and the benefit of the query answers. We presented a solution for PDMS query reformulation that exploits completeness characteristics of mappings between peers. First, we described the influence of GLaV mappings on the completeness of query answers. Next, we introduced a fully local strategy to prune those mappings that have a high expected information loss based on statistics. Using experiments, we highlighted the the minimal completeness loss during query reformulation and showed the feasibility of our approach. In summary, quality based exploitation of mappings may yield efficiency gains of up to an order of magnitude. Gathering statistics in the PDMS context, relaxing the assumptions of independent variables and equal distribution of `null` values, and a detailed cost model containing network transfers are major challenges for future work. Moreover, we aim to refine our search strategy for query reformulation in presence of limited resources.

Acknowledgments. We want to thank Stefan Winkler for helpful discussions. This research was supported in part by the German Research Society (DFG grant no. NA 432).

References

1. Aberer, K., Cudré-Mauroux, P., Hauswirth, M.: The Chatty Web: Emergent semantics through gossiping. In: World Wide Web Conf. (WWW). (2003)
2. Halevy, A.Y., Ives, Z., Suciu, D., Tatarinov, I.: Schema mediation in peer data management systems. In: Conf. on Data Engineering (ICDE). (2003)
3. Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Logical foundations of peer-to-peer data integration. In: Symposium on Principles of Database Systems (PODS). (2004)
4. Naumann, F., Freytag, J.C., Leser, U.: Completeness of integrated information sources. Information Systems **29** (2004) 583–615
5. Pottinger, R., Levy, A.Y.: A scalable algorithm for answering queries using views. In: Conf. on Very Large Databases (VLDB). (2000)
6. Tatarinov, I., Halevy, A.: Efficient query reformulation in peer data management systems. In: Conf. on Management of Data (SIGMOD). (2004)
7. Heese, R., Herschel, S., Naumann, F., Roth, A.: Self-extending peer data management. In: Conf. Datenbanksysteme in Business, Technologie und Web (BTW), Karlsruhe, Germany (2005)
8. Löser, A., Nejdl, W., Wolpers, M., Siberski, W.: Information integration in schema-based peer-to-peer networks. In: Conf. on Advanced Information Systems Engineering (CAiSE). (2003)
9. Madhavan, J., Bernstein, P.A., Domingos, P., Halevy, A.Y.: Representing and reasoning about mappings between domain models. In: Proc. of the National Conf. on Artificial Intelligence (AAAI). (2002)

Cooperative Prefetching Strategies for Mobile Peers in a Broadcast Environment

Wei Wu[1] and Kian-Lee Tan[1,2]

[1] Singapore-MIT Alliance, 4 Engineering Drive 3, Singapore 117576
wuw@nus.edu.sg
[2] National University of Singapore, Singapore 117584
tankl@comp.nus.edu.sg

Abstract. In a broadcast environment, mobile hosts can build up a mobile peer-to-peer (P2) network via short-range transmissions. Such a network can facilitate cooperation among the peers. In this paper, we study one such cooperation to reduce the access latency of queries - that of collaborative prefetching. We propose a novel scheme called the Announcement-based Cooperative Prefetching (ACP) to enable mobile peers to prefetch and share different objects. Our simulation results show that our cooperative prefetching scheme performs much better than individual prefetching and cooperative caching schemes.

1 Introduction

Data broadcast [1] has been regarded as a very scalable data dissemination model in mobile environments. In a push-based data broadcast environment, a server repetitively broadcasts data objects, and mobile hosts (MHs) get data by listening to the broadcast channel. Before getting its needed data object, a MH may need to wait for a long time if the number of data objects being broadcast is large. To reduce the access latency, the MHs cache their frequently accessed data objects locally, but cache miss still happens when the cache space is not big enough.

Prefetching [2,8] is a technique that improves a MH's response time. In prefetching, a MH continuously listens to the broadcast channel and prefetches important data objects to reduce cache misses. The importance of a data object to a MH may be determined by the MH's access frequency to the object or the waiting time before the next broadcast of the object or some other criterion.

Via short-range wireless technologies such as IEEE802.11 and Bluetooth, the MHs in a data broadcast environment can build up a mobile peer-to-peer (P2P) network. Data management problem in mobile P2P network has begun to gain more research interests [13]. Several cooperative caching schemes for mobile peers in broadcast environment have been proposed to reduce their waiting time [4,6]. In cooperative caching schemes, when a mobile peer encounters a local cache miss, it sends out a request to its neighbors in the mobile P2P network via short-range communication. If its neighbors have the required data object, the

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 62–73, 2007.

object is transmitted to the initiating peer, otherwise the initiating peer tunes in to the broadcast channel and waits as normal.

Since it has been shown that prefetching performs better (in response time) than demand-driven caching (such as LRU) and cooperative caching performs better than individual caching, we predict that cooperative prefetching may further improve the performance of mobile peers. Unfortunately, The inherent characteristics of mobile P2P network such as dynamic and unpredictable network topology impose challenges on cooperative prefetching.

In this paper, we first analyze the challenges for mobile peers to prefetch data cooperatively in broadcast environments. Next, we present our solution – the Announcement-based Cooperative Prefetching (ACP) strategy. Our ACP scheme enables the mobile peers not only to answer queries cooperatively, but also to prefetch data cooperatively, while keeping the mobile peers autonomous.

Caching and prefetching in a mobile broadcast environment have received much attention in the literature. Cache replacement strategies for MHs have been proposed in [10,14], while prefetching schemes have been studied in [2,8].

Several cooperative caching schemes for mobile peers have also been proposed for different environments and applications [4,5,6,16].

2 System Model and Assumptions

2.1 System Model

In the system, a server broadcasts data objects to the many MHs through a broadcast channel. We assume all MHs are in the server's transmission range. There is no uplink from MHs to the server – MHs cannot send requests to the server.

MHs build up a mobile P2P network via short-range wireless transmission. Every mobile peer (hereafter called MP) can communicate with the MPs in its transmission range. The MPs do not forward the messages they received. There are two kinds of communications between MPs: broadcast and P2P transmission. When a MP broadcasts a message, the MPs in its transmission range will all receive the message. When a MP sends a P2P message, the target MP receives the message and other MPs discard the message.

2.2 *PT* heuristic and Assumptions

The PT heuristic was proposed in [2] for individual prefetching. Under the PT heuristic, a MH always prefetches data objects that have a high PT value. The PT value for a data object D at time t is defined as $P * T$, where P is the MH's access probability to D, T is the waiting time from t until D is broadcast. The rationale behind the PT heuristic is to prefetch those data objects whose expectation of waiting time is long. The PT heuristic improves the response time for MHs by reducing the cost of cache misses.

In our cooperative prefetching strategies, we also employ the PT heuristic. Thus, we make the following two assumptions.

- Every MH knows its access probability to all its interested data objects. This is for the value of P. In practice, a MH can estimate its access probability to data objects based on its access history.
- For each data object, the MHs know when it will be broadcast. Thus at any time, every MH knows the remaining time before the broadcast of each data object. This is for the value of T. In practice, it can be calculated from the server's broadcast index.

In this paper, we focus on designing the cooperative prefetching strategy among MPs. For simplicity, we also assume that all data objects are of the same size and data objects are not updated.

3 Announcement-Based Cooperative Prefetching (ACP)

Since the objective of cooperative prefetching is to combine the benefits of both prefetching and cooperation, a simple cooperative prefetching (denoted SCP) strategy can be like this: each MP does prefetching individually and sends request to neighbors when a local cache-miss happens. Note that in SCP, although the MPs answer queries cooperatively, they do prefetching individually. There are, however, two limitations with SCP.

- The MPs may be prefetching the same objects. Since each MP prefetches objects based on its access distribution, MPs in the system may be prefetching the same objects if they have similar access characteristics. In this situation, few queries can be salvaged from neighboring peers' cache.
- The topology of the mobile P2P network is dynamic and unpredictable when the MPs are neither static nor group-based, so it is impossible to find an optimal cooperative prefetching plan in advance.

To overcome the above problems of SCP, we propose the Announcement-based Cooperative Prefetching (ACP) strategy. The aim of ACP is to make MPs not only answer queries cooperatively, but also do prefetching cooperatively, even when the MPs have similar access characteristics and move freely.

When MPs move freely, every MP's neighbors are changing. It seems an immediate effect is that a MP should not rely on its neighbors for caching objects. However, we observe that in prefetching, the biggest penalty for a MP's not prefetching a data object D is the MP has a query for D arising just after D is broadcast. We also observe that it takes some time for all neighbors of a MP to move beyond the communication range. Thus we argue that a MP can have some extent of *reliance* on its neighbors for prefetching some data objects.

The idea of the ACP strategy is: in deciding whether to prefetch an object D, if a MP knows whether its neighbors will prefetch D, it can make a wiser prefetching decision for D. For example, if a MP M knows that several of its neighbors are prefetching D, then M may choose not to prefetch D if D is not very important to M, since M has a chance to get D from its neighbors. The benefits of this are: 1) M saves its cache space for another valuable data object,

2) it avoids the problem that M and its neighbors are prefetching the same data objects, thus 3) the overall data availability is increased, and the MPs can have more queries answered by neighbors. Now what we need is simply a mechanism for a MP to know the prefetching decisions of its neighbors.

A MP can tell its neighbors whether it will prefetch a data object by broadcasting a message. Note that only the information that the neighbors will prefetch the data object will have impact on a MP's prefetching decision. So it is only necessary for the MPs that decide to prefetch the data object to broadcast prefetching decision. Furthermore, each time a MP meets a more valuable data object on the air, it replaces a cached object with the new one. As a result, some objects may be replaced soon after they are prefetched, if they are not important to the MP. Thus telling neighbors that it will prefetch D does not tell how long D will be kept in its cache. Accordingly, simply knowing that some neighbors will prefetch D, a MP does not know the extent of reliance it can put on its neighbors.

Considering these factors, we design our ACP strategy as follows. The server sends out index among its broadcast of data objects so that the MPs know at when the data objects will be broadcast. **Before** a data object D is broadcast, every MP decides whether to prefetch D based on the *PT* heuristic[1], and we call this **the first decision**. If the first decision is "yes" then the MP further predicts how long D will be in cache. If it predicts that D will be in cache for a long time, then the MP broadcasts an **announcement** message to its neighbors. At the same time, each MP whose first decision is "yes" counts the number of announcement messages for D it receives. **When** D is broadcast, every MP makes its **final decision** on whether to prefetch D according to the importance of D to it and the number of announcement messages it received. If the final decision is still "yes", the MP prefetches D; otherwise, the MP does not prefetch it.

To summarize, when a MP decides to prefetch D and believes it will cache D for a quite long time, it sends out an announcement message to its neighbors; a MP's final prefetching decision for D is based on both the importance of D and the number of neighbors who will prefetch D. The objective of the announcement is to affect the neighbors' prefetching decisions. Note that the announcement should be made before the broadcast of D, e.g. 2 seconds before D is broadcast, but the MPs need not make announcement at the same time.

Now, there are two issues to be addressed for ACP: 1) How to predict the time D will be in cache and whether it deserves an announcement? 2) How should the neighbors' announcements for D, if any, affect a MP's final prefetching decision for D?

3.1 Deciding Whether to Send Out Announcement

Each MP makes its first decision for D based on the *PT* heuristic [2]. If there is empty cache space and the access probability for D is not zero, the first decision is "yes", if the cache space is full, the MP checks the cached objects and see

[1] In fact, our ACP scheme is not limited to use the *PT* heuristic; ACP is actually a cooperation protocol and each MP is free to use any individual prefetching scheme.

whether there is a cached object whose PT value is lower than the PT value of D, if so, the first decision is "yes", otherwise, the first decision is "no".

In ACP, every MP records how long a data object was kept in its cache the last time: when a data object is prefetched, the MP records the timestamp, and when the data object is replaced, it calculates the time the object is in the cache and records it. The recorded **keeping time** of D is used to predict how long D will be in cache this time.

If the first decision is "yes", the MP decides whether to send out an announcement by checking the following inequation.

$$\frac{keeping\,time}{Tinterval} \geq \delta \tag{1}$$

Here δ is a parameter greater than zero and it is a threshold, and $Tinterval$ is the time between two consecutive broadcasts of the data object. If the inequation is satisfied, the MH sends out an announcement, otherwise it does not. The intuition is: when an announcement for D is made (the inequation is satisfied), the neighbors know that the MP may keep D longer than $Tinterval * \delta$, so they may have confidence to rely on the MP for D.

Note that δ is a parameter that can be tuned. Also note that a MP whose first decision for D is "yes" should count the number the announcement messages for D it receives.

3.2 Making Final Decision

In ACP, a MP's final prefetching decision for D is determined by the following factors: 1) PTd, the PT value of D; 2) PTc, the PT value of the replace candidate (the cached object with the lowest PT value); 3) γ, the reliance parameter; 4) n, the number of announcement messages for D the MP received.

Here γ is a parameter (greater than 0 and smaller than 1) modeling the extent of reliance a MP can put on **a** neighbor who sent out an announcement for D. In other words, γ models the probability that when the MP has a query for D the neighbor still has D and is within the MP's communication range.

One intuition is that the more the number of neighbors sending out announcement messages for D, the more reliance the MP can put on its neighbors for D, since it is possible for the MH to get D from any of them. Thus in our scheme, the total reliance the MH can put on its neighbors for D is $\gamma * n$.

When D is broadcast, each MP makes its final prefetching decision for D as follows.

- If the first decision is "no", the final decision is "no".
- If the first decision is "yes" and the MP did not send out announcement for D, then check whether the following inequation is satisfied.

$$PT_d(1 - \gamma * n) > PT_c \tag{2}$$

If the inequation is satisfied, the final decision is "yes", otherwise it is "no".

– If the first decision is "yes" and the MP has sent out an announcement for D, then if $\gamma * n$ is smaller than 1, the final decision is "no", else ($\gamma * n$ is greater than 1) the MH generates a random number p between 0 and 1, if p is bigger than $1/(\gamma * n)$, the final decision is "no", else the final decision is "yes".

The intuition of this final decision process is: (a) if the first decision is "no", D is not important for the MP, so the MP should not prefetch D. (b) If the first decision is "yes" but the MP did not sent out an announcement, D is important but not very important to the MP, and the MP's neighbors will not rely on the MP for D (since it did not make announcement), so the MP is free to decide whether to prefetch D based on the total reliance it can put on its neighbors and it checks whether it still needs to prefetch D after deducting the reliance on neighbors from D's PT value. (c) If the MP sent an announcement for D, D is very important for the MP, and its neighbors may rely on it for D, thus it is not free to put reliance on its neighbors and give up prefetching D. However, if it received enough announcements from neighbors ($\gamma * n > 1$), which means it is possible that the MPs share common access interests for D and too many MPs will prefetch D and the neighborhood is wasting cache space, then it has a chance by tossing a coin to decide whether to prefetch D. Even though some MHs having made announcement may choose not to prefetch D after the coin flip, the overall availability of D is guaranteed by the process of coin flips, with a high probability.

The final decision for D is made **when** D is broadcast. That's why the MPs should send out announcement **before** D is broadcast. If the final decision is "yes", the MP prefetches D, otherwise, it does nothing.

Note when a MP relies on its neighbors for D, it means the MP is ready to take the communication overhead for requesting D from neighbors. The MPs who do not intend to take the communication overhead can tune its reliance parameter γ to a small value so that it will not rely too much on neighbors.

3.3 Answering Queries Cooperatively

As in existing cooperative caching schemes, when a local cache-miss happens, the MP broadcasts a request to see whether its neighbors have the needed object. If so, the MP may retrieve the data object from the neighbor who replies first and the query gets answered, otherwise, the MP waits the server's broadcast of the needed object.

One variant of this query answering strategy can be as follows: when a local cache-miss happens, the MP checks the T value of the requested object, if the T value is big, it sends out the request message, otherwise the MP does not send out the request message and waits for the object on broadcast channel. This variant saves some energy for both the initiating MP and its neighbors.

4 Simulation Model

To learn the performance of the ACP strategy, we conducted detailed simulation experiments. In this section, we present the simulation model. It is implemented in Java using the simjava-1.2 package [7].

The simulated mobile environment is an X*Y (m^2) area where there are a broadcast server and *NumClient* MPs. The bandwidth of the server's broadcast channel is *BBWidth* Mbps while the bandwidth among MPs' communication is *P2PBWidth* Mbps. All MPs are in the server's communication range. The transmission range of the MPs is *TransRange* m. A MP only communicates with the MPs in its transmission range. When a MP receives a message from neighbor, it does not forward the message to other MHs. There is no multi-hop communications.

Server Model. The server has *NumDisks* broadcast disks and the total number of data objects is *DBSize*, and the size of each data object is *DataSize* KB. The server broadcasts the data objects repeatedly.

Client Model. Each MP accesses data objects in its access range and the size of access range is *AccessRange*. The data objects in access range are divided into access regions of size *RegionSize*, and the number of access regions for a MP is *AccessRange*/*RegionSize*. The MPs share a certain percentage (*Overlap*) of their access regions as common access regions and the remaining access regions are selected randomly for each MP. We assign a Zipf distribution with a *skewness* parameter θ to the access regions and the data objects within an access region have the same probability to be accessed. When assigning access distributions to the access regions, the overlapped parts always get high access probabilities. This is to avoid scenarios that are biased to our proposed scheme.

Queries. A MP's queries are generated randomly according to its access distributions. The time interval between a MP's two consecutive queries is *ThinkTime* broadcast units (a broadcast unit is the time required to broadcast a data object). Each MP generates *NumQuery* queries. To avoid the cache warm-up effects, each MP first generates *NumWarmUp* warm-up queries after its cache is full.

Cache Space. The size of each MP's cache space is *AccessRange***CacheSize*. For example, if the size of Access Range is 200 and the *CacheSize* is 20%, then the cache space can hold 40 data objects.

Movement Model. We use a variant of "random waypoint" mobility model [3] as the MPs' movement pattern. The MP randomly chooses a destination in the modeled area and randomly chooses a speed which is around *MoveSpeed* m/s. Then the MP moves to the destination with that speed. After arriving the destination, the MP pauses for *PauseTime* seconds. After that, the MP randomly chooses another destination and speed, and repeats this movement pattern. At the beginning, the MPs are randomly scattered in the area.

Default Parameter Settings. The default parameter settings are given in Table 1. To find the optimal values for parameters δ and γ, we conducted detailed experiments to learn the effects of δ and γ on ACP's performance (see [12]). We find that setting δ and γ too high (e.g. 0.9) or too low (e.g. 0.1) both deteriorate ACP's performance, which is consistent with the intuition. ACP performs best

Table 1. Default Parameter Settings

Parameters	Values	Parameters	Values
Area	1000m * 1000m	NumDisk	1
DBSize	1000 data objects	DataSize	1 M
BBWidth	10 Mbps	P2PBWidth	2 Mbps
NumClient	100	TransRange	150 m
AccessRange	200 data objects	Overlap	30% Access Range
RegionSize	10 data objects	CacheSize	20% Access Range
MoveSpeed	2 m/s	PauseTime	30 s
NumQuery	1000	NumWarmUp	300
Skewness θ	0.5	ThinkTime	10 broadcast unit
δ	0.3	γ	0.5

and quite stable when δ and γ are set to values between 0.3 and 0.6. In the experiments presented below, we use 0.3 and 0.5 as optimal values of δ and γ for all MPs. In practice, optimal values for δ and γ can be tuned for different applications and scenarios and then be shared as global knowledge by the clients.

5 Experiments and Results

In the simulated experiments, the performance metric used is the average response time (or the average access latency) measured in broadcast units. For the queries answered by neighboring peers' cache, the latencies caused by message exchange and data object transmission are counted in.

To learn the performance of our Announcement-based Cooperative Prefetching (ACP) strategy, we compare it with the *PT* heuristic [2] Individual Prefetching (IP) scheme, the recently proposed DGCoca [5] Cooperative Caching (CC) scheme, and the Simple Cooperative Prefetching (SCP) scheme.

Due to space limitation, here we only present the representative experiments. For more simulated experiments and results, interested reader can refer to [12].

5.1 Effect of Cache Size

The aim of the first set of experiments is to study the effect of cache size on different cache scheme's performance by varying cache size from 1% to 100%.

Figure 1 shows that with the increasing of cache size, all schemes exhibit better response time, because the more cache space a MP has, the more data objects are cached locally, and more queries are answered by local cache hits. We see that the performance of CC is better than IP, and the performance of SCP is similar with CC's, while ACP outperforms all of them. SCP and CC perform better than IP because in SCP and CC the MPs answer queries cooperatively. The reason that ACP performs better than SCP is ACP solves the problems we described in section 3, thus more data objects are prefetched globally. ACP outperforms CC because ACP drives the MPs to continuously fresh the cache

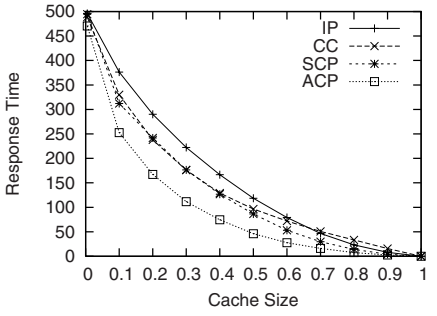


Fig. 1. Effect of Cache Size

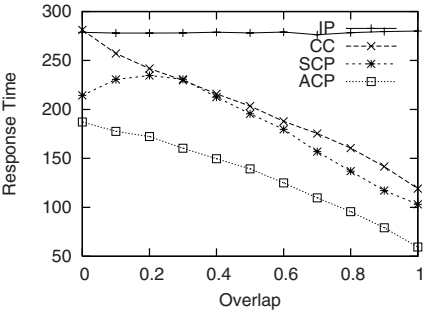


Fig. 2. Effect of Overlap

space with objects the MPs should not miss. Putting another way, ACP is an proactive cooperative caching scheme while CC is a passive one. We even observe that the gain of ACP out of them is considerable and stable.

5.2 Effect of Overlap

The effect of overlap (degree of common interests) on different scheme's performance is studied by varying the overlap percentage from 0 to 100%.

Figure 2 shows both ACP and CC's performance increase when the overlap degree is increased. It is because with more overlapped access regions, more data objects are prefetched (cached) cooperatively and also more queries are answered cooperatively. It is straightforward that overlap has no effect on IP scheme.

We observe that even when the overlap is set to 0, the performance of ACP is still much better than other schemes. At first glance, it may look like a mistake. In fact, it indeed shows the advantage of the ACP scheme. Note that when the overlap is set to 0, the access ranges of the MPs are all selected randomly. When $NumClient * AccessRange > DBSize$, although we set overlap to 0, there are still some "invisible" overlaps. When there is overlap, ACP works.

Figure 2 clearly shows the problem of SCP. When the overlap is increased, the performance declines first then improves. It declines first because the MPs have similar access probabilities to the specified overlapped parts, so the problem of prefetching same objects appears. The performance later improves because when the overlap is big enough, the MPs gain from answering queries cooperatively.

5.3 Effect of Skewness θ

We study the effect of the skewness of a MP's access distribution (Zipf) on different scheme's performance by varying the skewness from 0 to 1.

As shown in figure 3, the performance of all schemes improve with increasing skewness in access pattern. When the access is skew, a MP accesses some data objects more frequently than it does to others, and it will prefetch/cache the

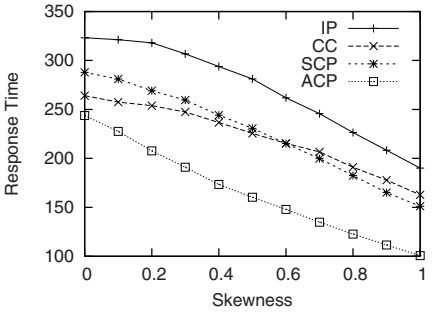


Fig. 3. Effect of Skewness

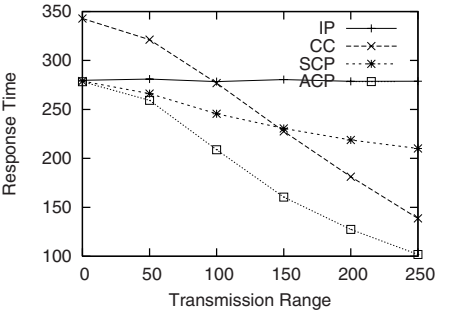


Fig. 4. Effect of Transmission Range

frequently accessed objects so that the MP experiences more local cache hits. We observe that IP, SCP and ACP are more sensitive to the skewness than the CC does. It's because prefetching schemes are more sensible to access skewness than demand-driven caching schemes do.

5.4 Effect of Transmission Range

The effect of MP's transmission range on different scheme's performance is studied by varying MP's transmission range from 0 to 250 m. Figure 4 shows with the increase of transmission range, a MP is able to communicate with more neighbors and the cooperation among neighborhood takes effect.

5.5 Effect of Move Speed

This set of experiments examines the effect of MP's move speed on different scheme's performance by varying *MoveSpeed* from 0 to 20m/s.

From Figure 5, we observe that the performance of CC declines fast with the increase of the MPs' move speed while the performance of ACP is affected little by the MPs' move speed. In CC, when a MP puts reliance on its neighbors for a data object D, it neither caches D nor listens to the broadcast for D, so the cost of a cache miss (global cache miss) is high. When the MPs move at higher speed, a MP's neighbors leave faster, so the chance for cache miss is higher. This is the reason why CC's performance declines tremendously with higher speed. In ACP, when a MP relies on its neighbors for a data object D, it does not prefetch it, however, the MP is still listening to the broadcast, so when D is broadcast again, the MP may prefetch it if this time no many neighbors announce to prefetch it. Another reason that ACP is not affected much by move speed is that in ACP a MP relies on its neighbors for a data object only when the data object is not very important for it or when the data object is a commonly interested one. Remember that in ACP in two situations a MP may rely on its neighbors for a data object: 1) when the data object's PT value is quite low; 2) when it receives

many announcements from neighbors. For the first situation, it is not a big deal when neighbors move away. For the second situation, with a high probability the coming neighbors have the data object, because it is a commonly interested one.

Another observation is when speed is changed from 0 to 1(m/s), the performance of ACP improves, but, when the speed continues increasing, ACP's performance declines. This is because: when the speed is 1, the MPs move slowly. The good things of moving slowly are: first, the MPs are moving, so they have chance to meet "good" neighbors; second, the speed is low so that the neighborhood do not change very frequently. While the speed is fast, the negative effect of changing neighborhood is shown.

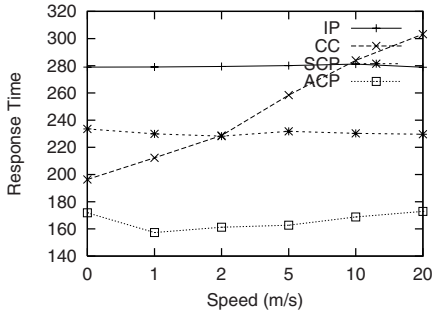


Fig. 5. Effect of Move Speed

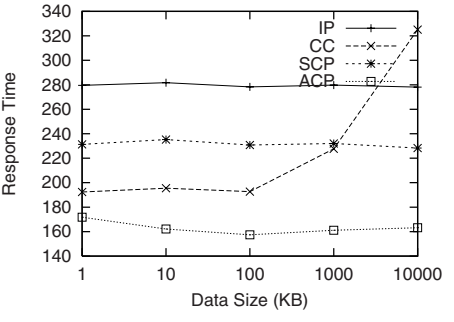


Fig. 6. Effect of Data Size

5.6 Effect of Data Size

The effect of data size on different scheme's performance is studied by varying DataSize from 1 KB to 10 MB. When the broadcast bandwidth is determined, for broadcasting same number of data objects, the broadcast cycle for large objects is longer than that for small objects, and the dynamics of neighbors relative to the broadcast cycle is stronger– more neighbors leave and more neighbors come during one broadcast cycle. So from figure 6, we see that the effect of data size is similar with the effect of move speed shown in figure 5.

6 Conclusion

In this paper, we have studied the problem of cooperative prefetching in a broadcast environment. In such an environment, mobile peers can build up a mobile P2P network via short-range communications. We have proposed an announcement-based prefetching scheme (ACP) to allow peers to prefetch different objects to facilitate more resource sharing. We have conducted an extensive performance study, and our results showed that ACP improves the performance considerably.

References

1. Swarup Acharya, Rafael Alonso, Michael Franklin and Stanley Zdonik. Broadcast disks: data management for asymmetric communication environments. in SIGMOD 1995.
2. Swarup Acharya, Michael J. Franklin and Stanley B. Zdonik. Prefetching from Broadcast Disks. in ICDE 1996.
3. Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. in MobiCom 1998.
4. Chi-Yin Chow, Hong Va Leong and Alvin Chan. Peer-to-Peer Cooperative Caching in Mobile Environments. in ICDCSW 2004.
5. Chi-Yin Chow, Hong Va Leong and Alvin T.S. Chan. Distributed Group-based Cooperative Caching in a Mobile Broadcast Environment. in MDM 2005.
6. Takahiro Hara. Cooperative caching by mobile clients in push-based information systems. in CIKM 2002.
7. Fred Howell and Ross McNab. SimJava. http://www.dcs.ed.ac.uk/home/hase/simjava/.
8. Haibo Hu, Jianliang Xu and Dik Lun Lee. Adaptive Power-Aware Prefetching Schemes for Mobile Broadcast Environments. in MDM 2003.
9. Francoise Sailhan and Valrie Issarny. Cooperative Caching in Ad Hoc Networks. in MDM 2003.
10. Chi-Jiun Su and Leandros Tassiulas. Joint broadcast scheduling and user's cache management for efficient information delivery. in MobiCom. 1998.
11. Ouri Wolfson and Bo Xu. Dissemination of Spatial-Temporal Information in Mobile Networks with Hotspots. in DBISP2P 2004.
12. Wei Wu and Kian-Lee Tan. Cooperative Prefetching Strategies for Mobile Peers in a Broadcast Environment. http://web.mit.edu/~wuwei/www/papers/CoPre.pdf
13. Bo Xu and Ouri Wolfson. Data Management in Mobile Peer-to-Peer Networks. in DBISP2P 2004.
14. Jianliang Xu, Qinglong Hu, Dik Lun Lee and Wang-Chien Lee. SAIU: an efficient cache replacement policy for wireless on-demand broadcasts. in CIKM. 2000.
15. Liangzhong Yin, Guohong Cao, Chita Das and Ajeesh Ashraf. Power-Aware Prefetch in Mobile Environments. in ICDCS 2002.
16. Liangzhong Yin and Guohong Cao. Supporting Cooperative Caching in Ad Hoc Networks. in INFOCOM 2004.

Symmetric Replication for Structured Peer-to-Peer Systems*

Ali Ghodsi[1], Luc Onana Alima[2], and Seif Haridi[1,2]

[1] KTH/Royal Institute of Technology
{aligh,seif}@kth.se
[2] Swedish Institute of Computer Science (SICS)
onana@sics.se

Abstract. Structured peer-to-peer systems rely on replication as a basic means to provide fault-tolerance in presence of high churn. Most select replicas using either multiple hash functions, successor-lists, or leaf-sets. We show that all three alternatives have limitations. We present and provide full algorithmic specification for a generic replication scheme called symmetric replication which only needs $O(1)$ message for every join and leave operation to maintain any replication degree. The scheme is applicable to all existing structured peer-to-peer systems, and can be implemented on-top of any DHT. The scheme has been implemented in our DKS system, and is used to do load-balancing, end-to-end fault-tolerance, and to increase the security by using distributed voting. We outline an extension to the scheme, implemented in DKS, which adds routing proximity to reduce latencies. The scheme is particularly suitable for use with erasure codes, as it can be used to fetch a random subset of the replicas for decoding.

1 Introduction

Research on structured peer-to-peer systems have produced systems which have strong performance in presence of dynamism. To cope with the dynamism, these systems rely on replication as a basic means to achieve robustness and fault-tolerance.

Most existing structured peer-to-peer systems either use *multiple hash functions*, *successor-lists*, or *leaf-sets* for choosing replicas.

Contribution. We analyze using multiple hash functions, successor-lists, and leaf-sets, and point out their disadvantages. Thereafter, we propose a new replication scheme, called *symmetric replication*, which we have implemented and added to the DKS system[1]. We provide full algorithmic specification of our scheme, something which we have not found for any other replication schemes for structured peer-to-peer systems. The advantages of symmetric replication are manifold. First, it is a general end-to-end scheme and can be applied to all

* This work was funded by the European project EVERGROW, the Vinnova project GES3 in Sweden.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 74–85, 2007.

structured peer-to-peer systems. Furthermore, each join and leave operation only requires sending 1 message to maintain the replication degree. Moreover, nodes can make concurrent requests to any specified replica. This opens up a window of opportunities. It is more secure as multiple requests to different replicas do not need to pass through the same node. Hence, distributed voting can be using the compare the results to increase security, without the risk of having the results tampered by one node. It can also be used for load-balancing by sending requests to a random replica. It is particularly useful if used in conjunction with erasure codes, as a random subset of size k of the replicas can be fetched concurrently to reconstruct the original data. Finally, we show an optional extension of symmetric replication, which is used in DKS to achieve proximity neighbor selection.

Outline. Section 2 gives preliminaries. In Section 3, we analyze three major existing replication schemes. We introduce our proposed scheme in Section 4. Section 5 outlines different techniques that can be built on-top of symmetric replication. Finally, the last sections, 7 and 8, discuss related work and conclude.

2 Preliminaries

In this section we present preliminary definitions used in the rest of the paper.

We assume a distributed system modeled by a set of peers communicating by message passing through a communication network that is: (*i*) connected, (*ii*) asynchronous, (*iii*) reliable, and (*iv*) provides FIFO communication.

A distributed algorithm running on a peer in the system is described as a set of rules of the form:

$$R \;::\; \frac{\textbf{receive}(\mathit{Sender},\ \mathit{Receiver},\ \textsc{Message}(\mathit{arg}_1, .., \mathit{arg}_n))}{\text{Action}}$$

The rule R describes the event of receiving MESSAGE from *Sender* at the peer *Receiver* and the *Action* taken to handle that event. A *Sender* of a message executes the statement **send**(*Sender*, *Receiver*, MESSAGE($arg_1, .., arg_n$)) to send a message to *Receiver*.

We now give the definitions used in the rest of the paper.

In most systems, each peer in the system is assumed to receive a logical identifier from an identifier space, denoted $\mathcal{I}$, which is perceived as a ring (modulo the size of the space). We assume for simplicity that the identifier space is discrete and defined as $\mathcal{I} = \{0, \cdots, N-1\}$ for some large constant N ($N \in \mathbb{N}$). The identifier space is a metric space has a distance function $d : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$ satisfying the following criteria: **a**) $d(x, y) \geq 0$, **b**) $d(x, y) = 0$, iff $x = y$, **c**) $d(x, y) = d(y, x)$, **d**) $d(x, z) \leq d(x, y) + d(y, z)$. If requirements **c** and **d** are not satisfied we call it a "pseudo"-metric space.

We now formally define a structured P2P system.

Definition 1. *A structured P2P system is a P2P system with a "pseudo"-metric space where each peer in the system has got an identifier from the "pseudo"-metric space and the choice of the neighbors of a peer are constrained in terms of the distance function of the "pseudo"-metric space.*

On top of a structured P2P system a distributed hash table (DHT) abstraction can be built by deterministically mapping each identifier i in the identifier space to a peer with identifier p. We denote the identifiers of the peers in the system at a certain time $\mathcal{P}$ ($\mathcal{P} \subseteq \mathcal{I}$).

To make the rest of the paper concrete, we will define a distance function, as well as a mapping from identifiers to peers, as commonly used in [2,1,3,4]. Our replication scheme, however, does not assume these definitions and can thus be applied to a variety of structured P2P systems.

We will assume the distance function is defined as:

$$d(x, y) = y \ominus x$$

The operator $\ominus : \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{I}$ is defined as:

$$\ominus(x, y) = x - y \bmod N$$

Similarly $\oplus : \mathcal{I} \times \mathcal{I} \rightarrow \mathcal{I}$ is defined as:

$$\oplus(x, y) = x + y \bmod N$$

We use infix-notation for the binary operators $\ominus$ and $\oplus$ to ease the reading.

For the mapping of identifiers to peers we map each identifier i in the system to its *successor*, which is the first peer met in the identifier space going in clockwise direction starting at i. The function $s_{\mathcal{P}} : \mathcal{I} \rightarrow \mathcal{P}$ is used for this purpose:

$$s_{\mathcal{P}}(i) = i \oplus \min\{d(i, p) : p \in \mathcal{P}\}$$

We call a peer n *responsible* for an item i iff $s_{\mathcal{P}}(i) = n$. Sometimes we will refer to peer n as the *master peer* for item i to distinguish it from other peers replicating item i.

To provide a DHT abstraction, each data item d is mapped to the identifier space using a globally known function H. Hence, a data item d is stored on the peer $s_{\mathcal{P}}(H(d))$.

3 Major Existing Replication Schemes

The use of several hashing functions for replication, which is mentioned in CAN, Tapestry, and other systems[5,6], is closest to our symmetric replication scheme. However, it has several major disadvantages. It requires the inverse of the hashing functions to maintain the replication factor, which is impossible by the definition of hash functions. To see why, assume a replication degree of two, and hence two different hashing functions, H_1 and H_2, which are known by all nodes. Assume a node with identifier 10 is storing any items with identifiers in the range $[5, 10]$. An item with key "course" might be mapped to identifier 7 using H_1, and therefore be stored at node 10. If node 10 fails, this item should be fetched from the other replica and replication to maintain the replication degree 2. To do this,

however, it would be required to find out the key "course" such that the node responsible for H_2("course") can be contacted. Worse, even if the inverse of the hash functions were available, each single item that the failed peer maintained would be dispersed all over the system when using different hash functions, making it necessary to fetch each item from a different peer.

If the replication degree is not restored each time there is a failure, items soon disappear from the system. Assume every node fails with exponential distribution with intensity *lambda*. Then every node fails after an average of $\frac{1}{\lambda}$ time units. Given replication degree f, after an expected $\frac{f}{\lambda}$ time all replicas would be lost.

The successor-list scheme is, however, able to restore the replication degree after failures. The scheme [2] fulfills two purposes. One is to replicate items on the successors such that lookups for items on a failed peer can be resolved by its successor, since the failed peer's items automatically become the responsibility of the successor. The other purpose is to store routing information about f successors, such that as soon as a node's successor, p, is detected as failed, it is quickly replaced with p's successor. The leaf-set scheme[3,7] has the same two uses as the successor-list scheme. But in addition, a lookup request might first arrive at one of the replicas, which then can resolve the lookup.

Our conjecture is that the two mechanisms should be separated. While having routing information about the successors or leafs is useful for routing table correction, replication on the same set has several disadvantages.

The first disadvantage is that both schemes need $\Omega(f)$ messages for every join and leave event to maintain a replication degree of size f. The reason for this is that if a node leaves the system, its f successors (or $\lfloor \frac{f}{2} \rfloor$ predecessors and successors in the leaf-set scheme) will by definition belong to the successor-list (or leaf-set) of a node which they previously were not in. Hence, they need to fetch or release items. Figure 1 illustrates this with an example using the successor-list scheme. The figure shows a system with the peers $1, ..., 7$ as indicated by the circles. For simplicity, the system contains the items $1, ..., 7$. Assuming a replication factor of 3, the figure shows the identifiers of the items each peer is replicating. If peer 4 has failed, peers 5, 6, 7 need to establish connections with other peers and fetch the items 1, 2, 3 respectively to maintain the replication factor.

Furthermore, the re-establishment of the replication degree needs to be coordinated by some node that triggers a replication maintenance algorithm at each of the successors (and predecessors in the leaf-set case). The coordinating peer might however fail or leave the system making it necessary to use a more robust algorithm. Many implementations, such as Bamboo[8], or the previous version of the DKS system[1], used an epidemic algorithm, where each node sends a message to its neighbors whenever it detects a change, leading to $\theta(f^2)$ messages for every event, or time interval if the algorithm is periodic, given a replication degree of size f.

Furthermore, any request to a specific replica, m, must first be routed to a node in the successor-list, or the leaf set, before it can be forwarded to m. The reason behind this is that the requesting node has no information about the logical identifier of the replicas, while the nodes in the successor-list, or the

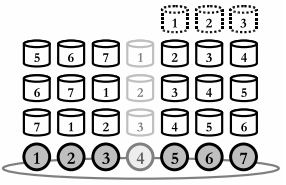


Fig. 1. A system populated with peers $1, ..., 7$, as indicated by the circles in the figure. The figure shows the identifier of the items each peer is storing given that the replication factor is 3. E.g. peer 1 is replicating items 5, 6, 7.

leaf-set, maintain such information. In the successor-list scheme, the first replica routed to will always be the numerically closest replica in the successor-list, while in the leaf-set this can be any of the replicas. In both systems, however, the first replica met is a bottleneck, which can fail, decelerate the whole operation, or in the case of an adversary, launch a malicious attack.

The leaf-set scheme is, however, better in this respect as it naturally balances requests to different replicas, given that $f \geq 2^b$, where f is the replication degree, and 2^b is the arity of the search tree.

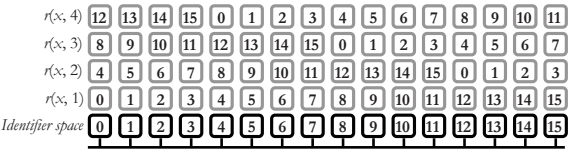


Fig. 2. The identifiers associated with each identifier in the system in a system with identifier space of size $N = 16$ and a replication factor of size $f = 4$

4 The Symmetric Replication Scheme

The main idea behind symmetric replication is that each identifier in the system should be associated with f other identifiers. If identifier i is associated with identifier r, then any item with identifier i should be stored at the peers responsible for identifiers i, and r. Similarly, any item with identifier r should also be stored at the peers responsible for the identifiers i, and r.

Formally, each identifier in the system is associated with a set of f distinct identifiers such that the following always holds: if the identifier i is associated with the set of identifiers $r_1, ..., r_f$, then the identifier r_x, for $1 \leq x \leq f$, is associated with the identifiers $r_1, ..., r_f$ as well.

Put differently, the identifier space is partitioned into $\frac{N}{f}$ equivalence classes such that identifiers in an equivalence class are all associated with each other. Any such partition will work, but we will for simplicity chose the congruence classes modulo f.

```
Subroutine :: JoinReplication
              send(n : succ : RetrieveItems(pred,n))

R1 :: receive(m : n : RetrieveItems(start, end))
      for r :=1 to f do
          items[r] := ∅
          i := start
          while i ≠end do
                i := i ⊕ 1
                items[r][i] := localHashTable[r][i]
          od
      od
      send(n : m : Replicate(items,start, end))

R2 :: receive(m : n : Replicate(items, start, end))
      for r :=1 to f do
          i := start
          while i ≠end do
                i := i ⊕ 1
                localHashTable[r][i] := items[r][i]
          od
      od

Subroutine :: LeaveReplication
              for r :=1 to f do
                  items[r] := ∅
                  i := pred
                  while i ≠n do
                        i := i ⊕ 1
                        items[r][i] := localHashTable[r][i]
                  od
              od
              send(n : succ : Replicate(items, pred, n))
```

Fig. 3. Rules $R1$, and $R2$ show the replication algorithm for joins and leaves

We now explain how each identifier i is associated to f other identifiers to achieve replication degree f. Let $\mathcal{F} = \{1, ..., f\}$, then identifier i is associated to the f different identifiers given by the function $r : \mathcal{I} \times \mathcal{F} \rightarrow \mathcal{I}$ defined as: $r(i, x) = i \oplus (x - 1)\frac{N}{f}$

Figure 2 shows how identifiers are associated in an identifier space of size $N = 16$ and a replication factor $f = 4$. The black boxes illustrate each identifier in the identifier space, and on-top of each black box the identifiers associated with it are shown in light boxes. For example, identifier 0 is associated with the identifiers 0 ($r(0, 1) = 0$), 4 ($r(0, 2) = 4$), 8 ($r(0, 3) = 8$), 12 ($r(0, 4) = 12$).

As we mentioned before, in a system without any replication, each item with identifier i is stored at the responsible peer given by $s_{\mathcal{P}}(i)$, which in our example is the successor of item i. To replicate items in our scheme, the responsible peer of identifier i stores every item with an identifier associated with i. This implies that to find an item with identifier i, a request can be made for any of the identifiers associated with i.

Formally, in a system with the peers $\mathcal{P}$, an item with identifier i is stored on the f peers given by $s_{\mathcal{P}}(r(x, i))$, for all x ($1 \leq x \leq f$).

For example, if the identifier 0 is associated with the identifiers 0, 4, 8, 12, any peer responsible for any of the items 0, 4, 8, or 12 has to store all of the

```
R3 :: receive(m : n : InsertItem(key, value))
      for r :=1 to f do
          replicaKey := key ⊕ (r − 1)N/f
          respNode := FindSuccessor(replicaKey)
          send(n : respNode : AddItem(replicaKey, value, r))
      od

R4 :: receive(m : n : AddItem(key, value, r))
      localHashTable[r][key] := value

Subroutine :: LookupItem(key, r))
              replicaKey := key ⊕(i − 1)N/f
              respNode := FindSuccessor(replicaKey)
              send(n : respNode : GetItem(replicaKey))

R5 :: receive(m : n : GetItem(key))
      send(n : m : GetItemResp(Key, localHashTable[r][key]))
```

Fig. 4. The replication algorithms for inserting and looking up items shown by rules *R*3, *R*4

items 0, 4, 8, and 12. Hence, if we are interested in retrieving item 0, we can ask the peer responsible for any of the items 0, 4, 8, 12.

For the symmetry requirement to always be true, it is required that the replication factor f divides the size of the identifier space N. We find this reasonable as the size of the successor-list, as well as N, are constants in most systems. We have developed an intricate scheme where f can be freely chosen with a deviation of $|1|$, but omit it for space reasons.

Algorithms. We now give a description of all algorithms. The algorithms might need to be slightly modified to fit a system with a different mapping of identifiers to peers.

Each peer in the system has all its items stored in a two-dimensional (f, N)-array denoted *localHashTable*. The first dimension of the array represents the f identifiers associated with the identifier in the second dimension of the array. Hence, *localHashTable*$[i][j]$ represents items with identifiers $r(j, i)$.

Whenever a new peer n joins the system, it makes a call to the sub-routine JoinReplication (Fig.3) which immediately sends a RetrieveItems-message to its successor (denoted *succ*) asking it about all items n should be storing. The message declares which items it is interested in by specifying a range (*pred*,n), where *pred* is its predecessor's identifier and n is its own identifier.

Once the successor peer receives the RetrieveItems-message it initializes an empty two-dimensional (f, N)-array called *items*. Thereafter, each item associated with an identifier in the specified interval is copied from *localHashTable* to *items* and sent back in a Replicate-message to the newly joined peer. Upon receipt of the Replicate-message, the newly joined peer copies *items* to its *localHashTable*. The new peer is now ready to receive requests from other peers in the system.

The leave algorithm (Fig.3) works similarly to the join algorithm. Whenever a peer wants to leave the system it makes a call to the sub-routine called LeaveReplication which copies all items it is responsible for and sends them

```
Subroutine :: FailureReplication(failedId, predId, r)
              start := predId ⊕(r − 1)N/f
              end := failedId ⊕(r − 1)N/f
              respNode := FindSuccessor(start)
              send(n : respNode :
                      RestBcast(start, end, Msg(start, end, n)))

Subroutine :: MsgHandler(start, end, n'))
              for r :=1 to f do
                  items[r] := ∅
                  i := start
                  while i ≠end do
                          i := i ⊕ 1
                          items[r][i] := localHashTable[r][i]
                  od
              od
              send(n : n' : Replicate(items,start, end))
```

Fig. 5. The replication algorithms for failures

in a Replicate-message to its successor. Notice that we do not delete items that are no longer a peer's responsibility. If disk space is limited, such an optimization could be added.

Figure 4 shows the algorithms used to insert or lookup an item. To save space, we have not shown the asynchronous algorithm for finding the responsible of an item. Such an algorithm can commonly be found in most structured P2P systems. We assume that the sub-routine **FindSuccessor** implements a simple synchronous distributed algorithm which finds the peer responsible for a given identifier (See [2] for such an algorithm).

To insert an item, the inserting peer simply makes concurrent insertions to every location where the replica should be stored.

For the lookup algorithm, we only show a sub-routine that takes the two parameters *key* and i ($1 \leq i \leq f$) and finds the responsible peer for the i:th replica of identifier *key*. On top of this abstraction, different kinds of lookup services can be built, such as the ones mentioned in Section 5.

For handling failures, the algorithm shown in Figure 5 is used. The sub-routine **FailureReplication** is called at the successor of the failed peer with parameters specifying the failed peer's identifier, the failed peer's predecessor's identifier, and an integer specifying which of the f replicas to fetch the items from.

For example, assume the system depicted by Figure 2 populated with peers 0, 3, 4, 6, 7, where peer 3 has failed. Peer 4 should ideally fetch items in the range $(1, 3)$ to restore the replication degree. Items $(1, 3)$ are associated with items $(5, 7)$, $(9, 11)$, and $(13, 15)$. Peer 4 chooses to fetch them from the peers responsible for $(5, 7)$ which are peers 6 and 7. Instead of sending the message around the ring in the interval $(5, 7)$ a restricted version of our broadcast algorithm[9] is used which covers the given interval in $O(M)$ messages, where M is the number of peers in the given interval. Assuming a uniform distribution of peer identifiers, the restricted broadcast needs to send one message on average on every failure.

5 Exploiting Symmetric Replication

In this section we discuss simple end-to-end techniques that exploit symmetric replication's ability to do concurrent requests to replicas to enhance the security and performance of the system.

In the DKS system, distributed voting is used to ensure that data items received are not tampered with. This is done by sending requests to all m replicas and deciding which replica to accept based on a majority vote. The probability that an item has been tampered can be calculated and reported to the requesting user or application. If the probability that an item is tampered is p, and m ($2 \leq m \leq f$) concurrent requests are made out of which a majority of g ($0 \leq g \leq m$) answers are identical, the probability of such a configuration is given the Bernoulli trials: $\binom{m}{g} p^g (1-p)^{m-g}$. The system can automatically increase the number of concurrent requests m to achieve a certain degree of certainty in the results.

The advantage of symmetric replication is not only restricted to enhancing the security of the system. Symmetric replication can be used to send out multiple concurrent requests and picking the first response that arrives. The advantages of this are twofold. First, it enhances performance. Second, it provides fault-tolerance in an end-to-end fashion since the failure of a peer along the path of one request does not require repeating the request as it is likely that another one of the concurrent requests succeeds. If such a scheme is not used, outgoing messages have to be buffered at a peer together with timers, and whenever a timeout occurs, the messages need to be sent again with risk of ending up at the same failed node.

Proximity Neighbor Selection. The symmetry property could be used within the routing process to achieve proximity neighbor selection. This is particularly useful in systems such as $\mathcal{DKS}$, Chord, and Koorde, where the legitimate state of the routing information is rigid[10].

The idea is that each peer in the system augments its routing table to contain f entries for each routing entry, one for each replica of a routing entry.

For example in a Chord system with an identifier space of size N, each peer p maintains pointers to the successors of the identifiers $p \oplus 2^i$ for all i ($0 \leq i < log(N)$). To enhance this system, the routing information at each peer is augmented with a pointer to the responsible peer of every identifier associated with the identifier $p \oplus 2^i$. Every entry in the routing table is also tagged with proximity information.

Proximity neighbor selection can then be achieved in the following way. To route a message to the peer responsible for identifier d, each message in the routing process is piggy-backed with a parameter r that specifies which of d's replicas is currently searched for. A peer n in the routing process can then calculate its distance to the r:th replica of d. Peer n now has f peers that it can choose among which each have a shorter distance to each respective replica of d. Naturally, peer n routes to the peer which has the best proximity, and updates r in the outgoing message to reflect the intended replica.

6 Evaluation

We have have simulated the symmetric replication scheme and the successor-list scheme in a stochastic discrete event simulator developed using Mozart[11]. The symmetric replication scheme is implemented using the algorithms described in this paper. The lack of algorithmic specification of the successor-list scheme, or any other scheme for that matter, led us to implement a successor-list scheme in which every join or leave only costs f messages. In reality, however, many schemes use more messages to maintain the replication degree.

The method of independent replications has been used to generate unbiased estimates from independent and identically distributed variables. All the simulations are non-terminating where nodes join and leave with an exponential distribution with parameter λ, and a replication factor of 5.

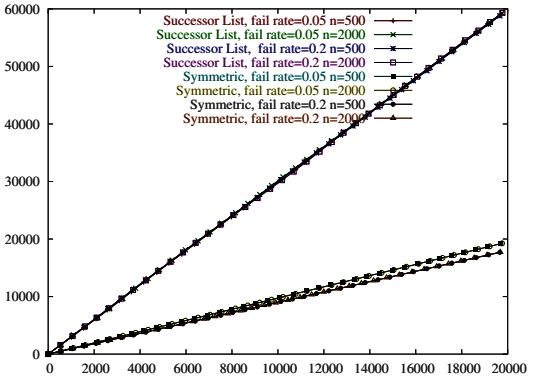


Fig. 6. Symmetric Replication vs. Successor-List Scheme. X-axis shows the simulated time line, while the Y-axis shows the total number of messages consumed to maintain the replication degree.

Figure 6 shows different simulations where the probability of ungraceful failures is 0.05, 0.1, and 0.2. We also vary the initial number of nodes that are in the system before the warm-up period to 500 and 2000.

The figure shows that the successor-list scheme consumes more messages to maintain the replication degree as nodes join and leaves the system, while the symmetric replication scheme maintains the replication degree with less amount of messages.

7 Related Work

Beehive[12] proposes to pro-actively replicate items and achieves $O(1)$ lookup latency. Beehive works well with structured P2P systems based on fixed space division[1], such as Tapestry, Pastry, and P-Grid, it is however not suited for

[1] For more information of fixed and relative space division, please refer to [10].

systems based on relative space division, such as Chord, Koorde, and DKS. In contrast, symmetric replication works with with both type of systems. In addition, Beehive replicates on its leaf set as well. Furthermore, Beehive does not address security issues as the authors acknowledge. Another disadvantage is that adaptive replication schemes are difficult to build transactions on-top of, while constant degree schemes are well-suited for that purpose.

In, [13], the security breaches of the successor-list scheme are identified, no solution is however proposed for the particular problem.

8 Conclusions

We have analyzed the three main approaches used for replication in structured peer-to-peer systems, multiple hash functions, successor-lists, and leaf-sets, and found that they have drawbacks.

The first scheme has the drawback that the replication degree cannot be restored after failures, and hence items will disappear after a while. The other schemes both require at least $\Omega(f)$ messages for each join and leave event to maintain a replication degree of size f. In practice, however, epidemic algorithms are used which use $O(f^2)$ messages in each round.

The second disadvantage is that the requesting peer cannot directly route to a specific replica, but the request is routed to the first replica encountered, which then forwards the request to the desired replica. The possibility of routing directly to a specific replica is useful if an insertion or update is required, or if several replicas is to be looked up concurrently. The first replica encountered is thus a bottleneck, which can fail, decelerate the operation, or launch a security attack. The leaf-set scheme is better, however, as the first replica met can be any of the replicas, while in the successor-list scheme, the same peer is always encountered when searching for a given item.

To rectify the problems in the mentioned other schemes, we proposed a new scheme and provided full algorithmic specifications of it. The scheme is applicable to all structured peer-to-peer systems. In our scheme, every join and leave operation requires $O(1)$ message to maintain the replication degree, independent of the size of the replication factor. Furthermore, requests can be directed to any specific replica directly. As a result, concurrent requests can be made, which can be used to prevent security attacks by using distributed voting. Our simulations verify that the cost of maintaining the replication degree is lower when using symmetric replication.

Finally, we outlined an optional extension to our scheme to achieve proximity neighbor selection.

References

1. Distributed k-ary System: http://dks.sics.se (2005)
2. Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: ACM SIGCOMM 2001, San Deigo, CA (2001) 149–160

3. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Lecture Notes in Computer Science **2218** (2001)
4. Kaashoek, M.F., Karger, D.R.: Koorde: A Simple Degree-optimal Distributed Hash Table. In: The 2nd Interational Workshop on Peer-to-Peer Systems (IPTPS'03). (2003)
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: ACM SIGCOMM 2001. (2001) 161–172
6. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.: Tapestry: A global-scale overlay for rapid service deployment. IEEE Journal on Selected Areas in Communications (Special Issue: Recent Advances In Service Overlay Networks) **22** (2004) 41–53
7. Rowstron, A., Druschel, P.: Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In: Proceedings of the 18th SOSP (SOSP '01), Chateau Lake Louise, Banff, Canada (2001)
8. Bamboo: http://bamboo-dht.org/ (2003)
9. Ghodsi, A., Alima, L.O., El-Ansary, S., Brand, P., Haridi, S.: Self-Correcting Broadcast in Distributed Hash Tables. In: 15th International Conference, Parallel and Distributed Computing and Systems, Marina del Rey, CA, USA (2003)
10. Alima, L.O., Ghodsi, A., Haridi, S.: A Framework for Structured Peer-to-Peer Overlay Networks. In: LNCS post-proceedings of Global Computing, Springer Verlag (2004) 223–250
11. Mozart Consortium: http://www.mozart-oz.org (2003)
12. Ramasubramanian, V., Sirer, E.: Beehive: The Design and Implementation of a Next Generation Name Service for the Internet. In: ACM SIGCOMM 2004. (2004)
13. Sit, E., Morris, R.: Security Considerations for Peer-to-Peer Distributed Hash Tables. In: The 1st Interational Workshop on Peer-to-Peer Systems (IPTPS'02). (2002)

A Gradient Topology for Master-Slave Replication in Peer-to-Peer Environments*

Jan Sacha and Jim Dowling

Distributed Systems Group, Trinity College Dublin, Ireland
{jsacha,jdowling}@cs.tcd.ie

Abstract. Open peer-to-peer architectures offer many possibilities for replicating database content, but designers have to deal with problems such as peer churn rates and inherent uncertainty in decision making. The lack of global knowledge of peer characteristics poses the specific problem of reliable peer discovery for database replica placement. This paper describes a self-organising algorithm for generating a peer-to-peer gradient topology that helps to solve the problem of replica placement through the clustering of peers with similar uptime and performance characteristics. We evaluate the algorithm by simulation, and propose an approach for master-slave replication that exploits the properties of the presented topology.

1 Introduction

The peer-to-peer paradigm for building distributed systems has become extremely popular and it has received increased attention as more and more novel applications are invented and successfully deployed. The main advantage of the paradigm is that it allows the construction of systems with unprecedented size and robustness, mainly due to their inherent decentralisation and redundant structures. However, the P2P paradigm introduces challenges that are often not dealt with properly in many proposed P2P architectures. Massive scale and very high dynamism makes it impossible to capture and maintain a complete picture of the entire P2P network, consequently, a peer or any other entity is only able to maintain a partial or estimated view of the system. Inherent decentralisation, an open environment, lack of trust and unreliable communication introduce distributed decision making problems. In particular, there is a problem in the database field of how to select the most suitable peers for storing data.

Existing P2P systems, such as Distributed Hash Table (DHT) based approaches, usually assume that all peers are similar and have equal capabilities for maintaining data [1]. For example in Chord [2] it is assumed that the distribution of resources among the peers is uniform. However, this was shown not to be the case in real-life systems, where the distribution of peer characteristics,

* This work was supported by the European Union funded "Digital Business Ecosystem" Project IST-507953.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 86–97, 2007.

such as the number of connections, the uptime, available bandwidth or the storage space, usually exhibit the so called scale-free or heavy-tail property [3,4,5,6]. Systems that do not address the heterogeneity of the environment and do not adapt their structures to the environment often suffer poor performance, especially in the face of high churn rates, i.e., high frequency of peers entering and leaving the system [7,8,9].

The main contribution of this paper is a self-organising neighbourhood selection algorithm, that clusters peers with similar performance characteristics and generates a gradient network topology that helps to solve the problem of dynamic replica placement. We briefly describe an approach for master-slave database replication, a heuristic for master election and a heuristic for replica discovery, which all exploit the properties of the gradient topology in order to improve the stability of the replicas and minimise the overhead for replica maintenance. We evaluate the neighbourhood selection algorithm through simulation and performance measurements.

The rest of the paper is organised as follows. In section 2 we discuss the general requirements for data distribution and replica placement. In section 3 and 4 we introduce the concept of gradient topology and we demonstrate a neighbour selection algorithm that generates the proposed topology. In section 5 an approach for master-slave replication based on the gradient topology is presented. Section 6 contains a detailed description of the simulation settings and the experimental results. Finally, in sections 7 and 8 we discuss the related work and our plans for future work.

2 Peer Utility Metrics

When addressing the persistent data requirements for a distributed system, we must decide on where to store the data. There are two extremes; one is to store all data in a centralised server, which introduces scalability problems, and the other one is to partition the data among a set of peers using some indexing scheme, for example a distributed hash table. Many existing P2P systems assume that all peers have identical capabilities and responsibilities, and that the data and load distribution is uniform among all peers [1,2]. However, this approach has a drawback that the use of low bandwidth/stability/trust peers to store data can potentially degrade the performance of the entire network [7].

To solve this problem, many systems only allow data to be stored on the fastest, highest bandwidth, and most reliable, trusted peers, called *superpeers* [8,9]. This approach, however, introduces another problem of superpeer election. It is not obvious how to identify and select the superpeers from the set of peers in the network, mainly due to the lack of a global knowledge about the system. Solutions based on flooding potentially require communication with all N peers in the network. Other solutions include hard-wiring them in the system or configuring them manually. However, this conflicts with the assumptions of self-management, decentralisation, and the lack of a central authority that controls the structure of the system. An adaptive self-organising system is preferable,

where the peers automatically and dynamically elect superpeers, accordingly to the demand, available resources and other runtime constraints. Alternatively, the system may resign from the two-state distinction between superpeers and ordinary peers and it may assign roles for peers relative to their capabilities.

The selection of peers for replica placement could potentially be based on criteria such as peer stability, available bandwidth and latency, storage space, processing performance, an open IP address and willingness to share resources. Peer availability, or uptime, is especially relevant, since every peer entering or leaving the system introduces extra overhead, due in part to required data transfers or routing structure reconfiguration. The system could also employ a peer reputation model and use it as a criteria for replica placement, for example only the most trusted peers might be allowed to host a replica.

We define a peer's *utility* as a function of the above parameters. The utility is a critical factor in the algorithm that generates the P2P gradient topology and is used in the replica placement strategy presented in this paper.

In a closed system, where all peers trust each other, it is sufficient that every peer evaluates its own utility level. Neighbouring peers can exchange the utility information without any verification procedure, since trust is assumed. In an open, untrusted environment, a separate mechanism is needed to assure that the utility claimed by the peers corresponds to their actual status. Managing trust in a decentralised system, however, is beyond the scope of this paper and will not be discussed further.

A key principle in our approach is that *persistent data is stored by the highest utility peers*. This strategy addresses a problem faced by many existing P2P systems, where some data items, especially less popular, are hardly accessible or even lost due to peers' instability or lack of resources such as storage space or bandwidth. In our design, the system tries to maximise data availability, security and the quality of service by placing data replicas on the most reliable, high performance hosts.

3 Gradient Topology

We propose a P2P topology, which we call a *gradient topology*, where the highest utility peers, maintaining persistent data, are highly connected with each other and form a logical *core* of the network, while the network around the core is composed of other peers considered less performant or less reliable (see Figure 1). Assuming the scale-free, heavy-tailed distribution of resources [4,5,6], a great majority of peers belongs to the latter category. The number of core peers is relatively small, but the amount of contributed resources and stability of core peers is significantly higher than the outer peers. The main advantages of grouping high utility peers in a logical core are the following:

– Searching for high utility peers maintaining replicas, or suitable for maintaining replicas, is less expensive in terms of number of messages generated, since it only requires communication with a small subset of peers in the network.

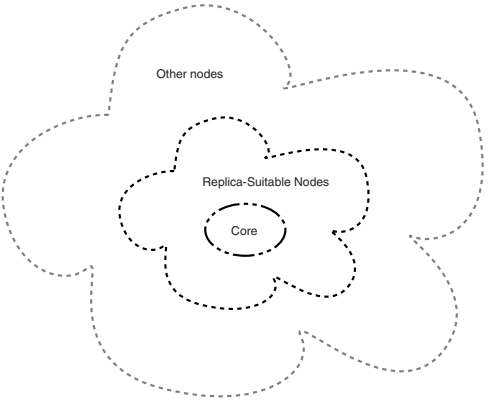


Fig. 1. Gradient topology based on peer utility

– The overhead for replica synchronisation is reduced since peers storing replicas are located close to each other, at least in terms of number of hops, and are connected by stable, good quality routes.

In practice, the core peers act more as servers, while the outer peers act more as clients. The core peers should be well-connected, have high bandwidth and processing power, and should be able to maintain a relatively high number of connections. On the other hand, peers far from the core are not suitable for maintaining replicated data, due to poor reliability, resource constraints or the owner's unwillingness to contribute resources to the system. It is not desirable to place such peers in the core since they would decrease the system's performance.

4 Neighbour Selection Algorithm

In order to create the gradient P2P topology and enable the emergence of a stable core, we initially thought of the following neighbour selection rule: *two peers may become neighbours if they have similar utility status.* Additionally, high utility peers should have more neighbours, since they have more resources available to maintain network connections.

However, our early experiments showed that a greedy selection policy, where peers always select neighbours with the most similar characteristics to their own, leads to high network clustering and in consequence long distances between peers. In some cases the clusters got disconnected and the network became partitioned. Another problem was that the highest utility peers did not always connect to a single core, and sometimes there were multiple clusters of high utility peers separated from each other by a number of lower utility peers.

We improved the algorithm by allowing the peers to connect to other non-similar peers, with the connection probability exponentially decreasing with the difference in peer utility. However, it turned out that a randomised strategy,

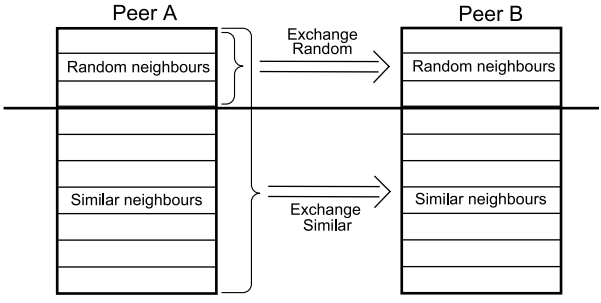


Fig. 2. Neighbourhood set exchange from Peer A to Peer B

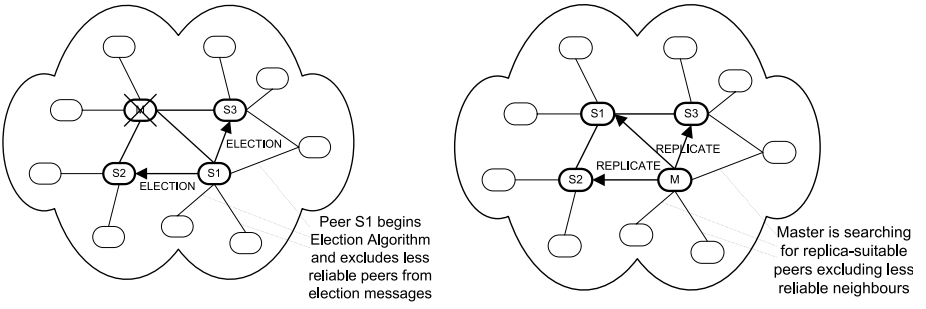
where a percentage of neighbours was always selected at random, gave the best results. Random connections serve several purposes. First of all, they allow the peers to discover potential neighbours with similar utility level, even in remote regions of the network, which in turn enabled the formation of a single cluster containing the highest utility peers. Random connections thus play a similar role to the exploration in traditional multi-agent systems. Second, random links prevented the graph from being disconnected. As shown in [10], even a small number of random connections, for example 20 per peer, is sufficient to make the probability of network partitioning negligibly small in practice. Finally, our randomised algorithm has the advantage that it is quite simple and it does not require tuning parameters specific to the deployment environment, such as the network size or average peer connectivity.

A peer maintains two independent sets of neighbours, randomly-selected connections, and greedily-selected connections to peers with similar utility status (see Figure 2). New connections are discovered by gossiping, i.e., by randomly contacting already existing neighbours and exchanging with them the lists of connections. It is important to note, that the connections inserted to the random sets are always selected from other peers' random sets, which guarantees that the sets remain random. The details of the algorithm implementation and the simulation settings are described in Section 6.

5 Replication Strategy

In this section, we demonstrate how the gradient network topology can be exploited by a master-slave database replication strategy. We present an approach, where the replica placement is based on peer utility, available resources and demand. Due to the information contained in the network structure, the selection of high utility peers suitable for replica placement does not require communication between all peers in the system.

We assume that each peer can potentially create an independent database, and replicate it over some of the peers in the network in order to improve its availability and persistence guarantees. A peer that creates the first copy of a database, which



(a) Peers S1, S2 and S3 compare their utility to elect a new master.

(b) Master compares the utility of peers S1, S2 and S3 to select the best peer for slave replica placement.

Fig. 3. Slave replica searching and master election algorithms exploiting the implicit information about peer utility contained in the network topology

we call the *master replica*, becomes the database owner. Subsequent replicas of the database hosted by other peers are called *slave replicas*. The users issue *queries* to the database that can be resolved by any replica. The owner, and potentially other authorised users, can also *update* or *delete* a database. There is only one master replica of a database and it is responsible for handling and synchronising updates. Slave replicas are created automatically, on demand.

We restrict the set of peers that are allowed to create database replicas to those with utility above a replica-suitable threshold. A *master replica threshold* defines a minimum peer utility to host a master replica, and a *slave replica threshold* defines a minimum utility level to host a slave replica. It is important to note, that the system does not require any form of consensus between peers on the threshold values, since the thresholds can be determined for each database individually.

5.1 Replica Placement

In our approach, a peer that already hosts a replica, in particular the master, requests a new replica placement on one of its neighbours when a certain condition is met, for example when the number of user queries exceeds some critical level and a new replica is needed to handle the load. It is crucial in this approach that a peer initiating the replication must be able to find other peers suitable for hosting new replicas, i.e., peers with utility above the slave replica threshold. This should be satisfied in the gradient topology, since all peers storing replicas are clustered in the highly connected core, and share a similar, high utility status. Search messages do not need to be propagated to lower utility neighbours, since all high utility peers are located in the core of the network (see Figure 3(b)).

Replicas are removed on demand, adaptively. When a peer decides that the cost of a replica's maintenance is higher than the cost of handling queries, for

example the frequency of queries drops below some threshold value, the replica can be deleted. Alternatively, replicas might be selected for removal using a Least Recently Used strategy as in Freenet [11].

5.2 Replica Synchronisation

Database replicas must be synchronised between the master and the slaves after update operations. We add the constraint that an update operation, either a modification, an addition or a removal, must be performed on the master replica, while queries can be handled by any slave. This requirement shouldn't significantly reduce the scalability of the system if the rate of queries is much higher than the rate of updates, which is commonly the case for example in name services or knowledge base systems. If an update is delivered to an ordinary replica, the replica forwards it to the master, and the master propagates the update to all replicas. This guarantees that concurrent updates from different peers are serialised and sent in the same order to all copies of the database, hence, there are no write-write conflicts.

The updates can be propagated either *instantaneously*, or in a *lazy* fashion, for example by *periodic gossiping*, with the latter technique being used to reduce bandwidth consumption and improve scalability at the cost of reduced data consistency. Lazy replica synchronisation can be initiated either by the master or by the slave, for instance after a peer crash or a restart. Thus the system provides eventual consistency of the replicas [12]. The core peers storing replicas should be well-connected, have high bandwidth and processing power, which should enable fast data dissemination and frequent replica synchronisation.

5.3 Master Election

In the P2P gradient topology, peers have relative positions defined by their utility metric. This allows us to develop an efficient election algorithm that doesn't require flooding, as peers can use a heuristic that excludes peers with lower utility, i.e., topologically further from the core, when sending election messages (see Figure 3(a)). This heuristic, however, does not guarantee that the highest utility peer will become a master unless all peers in the core are fully connected. Therefore, we modify our heuristic to provide a directed, gossiping election model. The election initiating peer also sends the election message to a certain number of neighbouring peers with lower utility, but still inside the core, and they can restrict further propagation of the message to only peers with higher utility. Given high enough connectivity between peers in the core, within a certain probability the peer with the highest utility should win the election.

5.4 Replica Discovery

A searching mechanism is needed for peers to discover nearby replicas of a database they request access to. Traditional unstructured systems, such as Gnutella, have used flooding algorithms for finding resources. This approach works

Algorithm 1: Main loop of the simulation

```
for N steps of the simulation do
    increase the number of peers by 1%;
    probabilistically remove peers according to their utility;
    forall peers p in the network do
        p.step();
    end
end
```

Fig. 4. Main loop of the simulation

well for highly replicated data, however, it doesn't scale in principle. A number of techniques have been proposed to improve search in unstructured P2P networks, such as random walk, iterative deepening or routing indices [13].

For the topology presented in this paper, we are designing a gradient routing algorithm that will be based on two main factors: the neighbour utility information (utility gradient), and heuristic values learned by the system (as for instance in Freenet [11]). By increasing the probability of forwarding queries to high utility neighbours, the algorithm can effectively route queries towards the core of the network.

6 Evaluation

We evaluated our approach by modelling a P2P network in RePast, a multi-agent simulation toolkit for large-scale systems. The simulation was implemented in Java. A network was created consisting of over 100,000 peers, which we believe is sufficiently large to model realistic large-scale applications. The experiments ran on a Pentium 4 machine with a 3GHz processor and 3GB main memory.

The simulation is based on a discrete time model and all events are executed in discrete time steps. The actions performed by peers are synchronous, however, our algorithm does not rely on the peer synchrony and hence the results obtained in the experiments can be generalised for asynchronous environments as well. Following the assumptions of decentralisation and a lack of a global view of the system, each peer maintains a limited number of neighbours and performs actions using local knowledge only. We assume also a scale-free distribution of resources with the maximum number of peer connections following the power-law (Pareto) distribution, starting from 10 connections and reaching about 50 neighbours for the most powerful peers. Similarly, the utility distribution follows the power-law. We model the network growth by adding new peers at each step of the simulation, where we start with a network containing only one peer, and at each time step the size is increased by 1 percent. Additionally, at each step a number of peers are removed, which models random failures or peer departures, with the probability of a peer departure being inversely proportional to its utility. Bootstrapping of the system is implemented with a centralised name repository

Algorithm 2: Agent step

```
if number of neighbours = MAX_NEIGHBOURS then
    disconnect random neighbour;
end
if number of similar neighbours < MAX_SIMILAR then
    choose randomly neighbour p from all known neighbours;
    get all neighbours n_1..n_k from p;
    choose peer n with the most similar utility from n_1..n_k;
    connect to n;
end
if number of random neighbours < MAX_RANDOM then
    choose randomly neighbour p from all known neighbours;
    get all random neighbours n_1..n_k of peer p;
    choose randomly peer n from n_1..n_k;
    connect to n;
end
```

Fig. 5. Algorithm performed by an agent at one step of the simulation

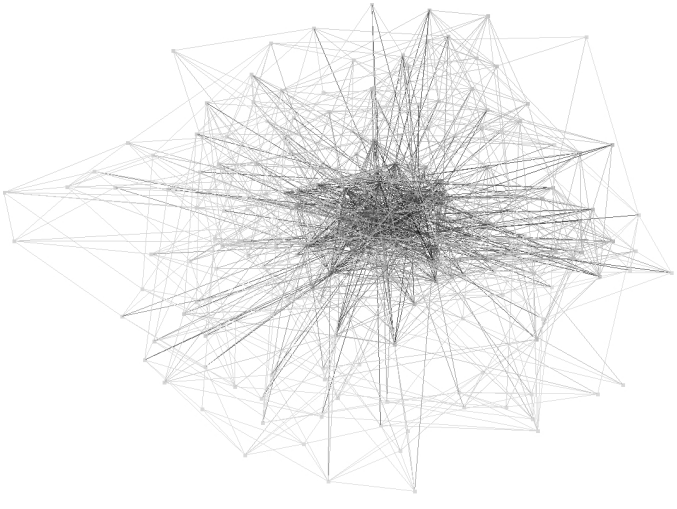
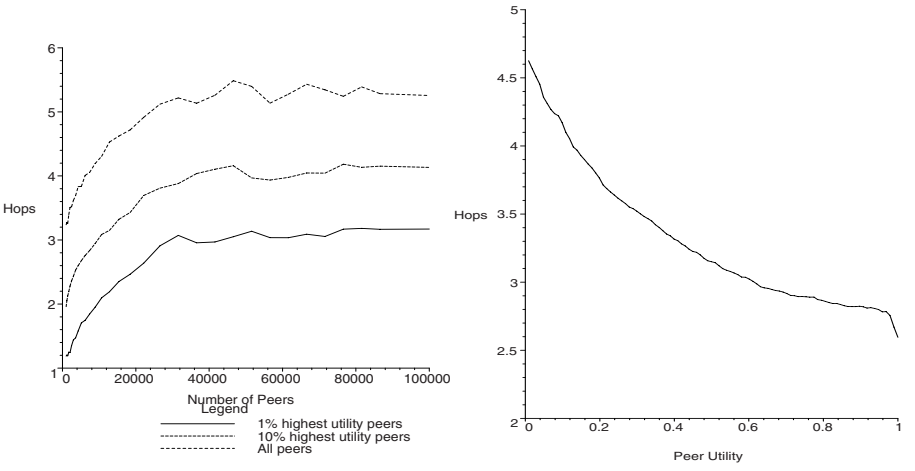


Fig. 6. Visualisation of a 200-node network in our RePast simulation using the Fruchmen-Reingold algorithm. High utility peers are marked with dark colors. Stable core of the network is visible in the center.

containing the 50 most recent peer names, where peers are added and removed in FIFO order. Figure 4 shows the pseudo-code of the simulation.

Figure 5 shows the randomised algorithm performed by each peer at every step of the simulation. A peer maintains two independent sets of neighbours, randomly-selected connections, and greedily-selected connections to peers with similar utility status. The latter set is twice as large as the former. At every step, if the number



(a) Average distance between peers as a function of the network size.

(b) Average distance between peers as a function of peer utility, network size 100,000 peers.

Fig. 7. Results of the neighbourhood selection algorithm

of neighbours of a peer reaches the maximum, the peer drops a random connection, and attempts to establish a new one by gossiping with a neighbour.

Figure 6 presents a visualisation of a sample network consisting of 200 peers, generated by the neighbour selection algorithm, where we can see that the topology evolved to the desired form where the highest utility peers are clustered together and constitute a stable core.

Figure 7 shows the measurement results obtained from our simulation. We can see that the average path length between peers is relatively low (about 5-6 hops for 100,000 peers), and that it varies with peer utility. The average distance between the highest utility peers is lower than the average distance between lower utility peers. This confirms our observation that the highest utility peers are highly connected with each other and form a stable core of the network.

7 Related Work

Most existing P2P systems that exploit the heterogeneity of the environment are based on structured P2P overlays. In Chord [2] it has been noted that a single physical peer can host multiple *virtual peers,* and therefore, the heterogeneity of resources in a DHT network can be addressed by assigning more virtual peers to higher performance hosts. OceanStore [14] proposed to elect a primary tier "consisting of a small number of replicas located in high-bandwidth, high connectivity regions of the network" for the purpose of handling updates, but no specific algorithm for the election of such a tier is presented. Mizrak et. al.[9] proposes a super-peer based approach for the exploitation of resource heterogeneity, however, unlike our architecture, it relies on a DHT overlay.

In the field of unstructured P2P networks, there has been a lot of work devoted to searching (e.g. [13]) and to replication strategies [15], but there has been limited research on network topology generation and peer neighbourhood selection algorithms. Yang and Molina [8] investigate general principles of superpeer-based networks and give practical guidelines for the design of such networks, however, they don't describe any specific algorithms for super-peer election or network structure maintenance. The closest to our approach is the work of Jelasity, in particular his research on decentralised topology management (T-Man [10]), however, he doesn't address the problem of reliable peer discovery and dynamic replica placement in an open P2P system.

8 Conclusions and Future Work

This paper describes the general requirements for data replication in peer-to-peer environments. We have proposed a gradient network topology where the highest utility peers are highly connected with each other and form a logical core suitable for maintaining data replicas. A self-organising neighbourhood selection algorithm has been presented that generates the proposed gradient topology by clustering peers with similar uptime and performance characteristics. The algorithm has been evaluated through simulation, and measurement results have confirmed that the approach is scalable and robust.

The main advantage of the gradient topology is that the network structure contains information about the peer utility, which allows the peers to discover other high utility peers without flooding the entire network with search messages. The gradient topology allows the search space to be limited to a small subset of all peers in the system. This property allows us to address the problem of dynamic replica placement, master replica election, and replica discovery in an open, decentralised environment. The gradient topology should also reduce the cost of replica maintenance, since peers storing data replicas are located close to each other and are connected by stable, high-capacity routes.

Our project is still at an initial stage and requires a lot of further research. We are planning to develop a heuristic routing mechanism based on peer utility gradient, which will allow peers to discover database replicas in their proximity. We are also going to evaluate the proposed replica placement strategies and the master election algorithm. We are building a prototype implementation based on the Berkeley DB middleware.

References

1. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proceedings of the 18th International Conference on Distributed Systems Platforms, Springer (2001) 329–350
2. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM Computer Communication Review. Volume 31., ACM Press (2001) 149–160

3. Barabsi, A.L., Bonabeau, E.: Scale-free networks. In: Scientific American. Volume 288. (2003) 60–69
4. Sen, S., Wong, J.: Analyzing peer-to-peer traffic across large networks. In: Transactions on Networking. Volume 12., ACM Press (2004) 219–232
5. Leibowitz, N., Ripeanu, M., Wierzbicki, A.: Deconstructing the kazaa network. In: Proceedings of the 3rd International Workshop on Internet Applications, IEEE Computer Society (2003) 112–120
6. Pouwelse, J., Garbacki, P., Epema, D., Sips, H.: The bittorrent p2p file-sharing system: Measurements and analysis. In: the 4th International Workshop on Peer-To-Peer Systems, Cornell, USA (2005)
7. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a dht. In: Proceedings of the USENIX 2004 Annual Technical Conference. (2004) 127–140
8. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: Proceedings of the 19th International Conference on Data Engineering, IEEE (2003) 49–60
9. Mizrak, A.T., Cheng, Y., Kumar, V., Savage, S.: Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In: Proceedings of the 3rd IEEE Workshop on Internet Applications. (2003) 104–111
10. Jelasity, M., Babaoglu, O.: T-man: Gossip-based overlay topology management. In: the 3rd International Workshop on Engineering Self-Organising Applications, Utrecht, The Netherlands (2005)
11. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. In: Proceedings of the 1st International Workshop on Designing Privacy Enhancing Technologies, Springer (2000) 46–66
12. Tanenbaum, A., van Steen, M.: Distributed Systems: Principles and Paradigms. Prentice Hall, New York (2002)
13. Yang, B., Garcia-Molina, H.: Improving search in peer-to-peer networks. In: Proceedings of the 22nd International Conference on Distributed Computing Systems, IEEE Computer Society (2002) 5–14
14. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., , Zhao, B.: Oceanstore: An architecture for global-scale persistent storage. In: Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems. (2000) 190–201
15. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: Proceedings of the 16th International Conference on Supercomputing, ACM Press (2002) 84–95

A Content–Addressable Network for Similarity Search in Metric Spaces*

Fabrizio Falchi[1], Claudio Gennaro[1], and Pavel Zezula[2]

[1] ISTI-CNR, via G. Moruzzi 1, 56124 Pisa, Italy
{fabrizio.falchi,gennaro}@isti.cnr.it
[2] Masaryk University, Brno, Czech Republic
zezula@fi.muni.cz

Abstract. In this paper we present a scalable and distributed access structure for similarity search in metric spaces. The approach is based on the Content–addressable Network (CAN) paradigm, which provides a Distributed Hash Table (DHT) abstraction over a Cartesian space. We have extended the CAN structure to support storage and retrieval of generic metric space objects. We use pivots for projecting objects of the metric space in an N-dimensional vector space, and exploit the CAN organization for distributing the objects among the computing nodes of the structure. We obtain a Peer–to–Peer network, called the MCAN, which is able to search metric space objects by means of the similarity range queries. Experiments conducted on our prototype system confirm full scalability of the approach.

1 Introduction

The proliferation of digital contents such as video, images, or text imposes the use of access methods for efficiently storing and retrieving this information. The concept of similarity searching based on relative distances between a query and database objects has become a solution for a number of application areas, e.g. data mining, signal processing, geographic databases, information retrieval, or computational biology. This approach formalizes the problem by the mathematical notion of the *metric space* [1], so the data elements are assumed to be objects from a metric space domain where only pairwise distances between the objects can be determined by respective distance function.

However, the need to deal with larger and larger, possibly distributed, archives requires an access structure to speedup the retrieval. Unfortunately, the use of single–site access structures is becoming prohibitive due to the lack of scalability of such systems, however fast they are. In fact, as the current literature demonstrates, see for example [2], the response time of access structures for metric spaces is linearly increasing with the size of the searched file.

* This work was partially supported VICE project (Virtual Communities for Education), funded by the Italian government, and by DELOS NoE, funded by the European Commission under FP6 (Sixth Framework Programme).

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 98–110, 2007.

The approach proposed in this paper is to use a Peer–to–Peer (P2P) structure composed of a network of nodes whose number can vary on the basis of the size of the data-set. The aim is to maintain the global response time stable as the data-set size grows. In this respect, the P2P paradigm is quickly gaining in popularity due to their scalability and self-organizing nature, forming bases for building large-scale similarity search indexes at low costs. However, most of the numerous P2P search techniques proposed in the recent years have focused on the single-key retrieval [3,4,5].

In particular, we present a distributed storage structure for similarity search in metric spaces that is based on the original idea of the *Content–Addressable Network* (CAN) [4], which is a distributed hash table abstraction over the Cartesian space. Our distributed storage structure, called *MCAN*, is able to index objects of a generic metric space. The advantage of the metric space approach to the data searching is its "extensibility", since in this way, we are able to perform the *exact match*, *range*, and *similarity* queries on any collection of metric objects. More in general, our proposal can be seen as a Scalable and Distributed Data Structure, (SDDS) – original proposal LH* [6] is intended for the primary key retrieval – which uses the P2P paradigm for the communication in a Grid-like computing infrastructure. A fundamental property of this paradigm is that insertion of an object, even if it implies a node split, does not require immediate update propagation to all network nodes.

The rest of the paper is organized as follows. In Section 2, we summarize the necessary background information. Section 3 presents the MCAN distributed structure and its functionality. Section 4 reports the results of performance evaluation experiments. Section 5 concludes the paper and outlines directions for future work.

2 Background

2.1 Content–Addressable Network (CAN)

The CAN is a distributed hash table that uses a function for mapping "keys" onto "values" in order to assign them a position into the table. In the CAN, the table is composed of a finite set of individual network nodes. Each node of the network is dynamically associated with a partition of a d-dimensional Cartesian space. Usually, the Cartesian space is a d-torus (in the sense that the coordinate space wraps), and is targeted to store (K, V) pairs, where K is an identifer of the object and V is a pointer to a copy of the object. The basic operations of the CAN are insertion, lookup and deletion of respective (K, V) pairs. In order to be compatible with the metric space, we generically refer to these pairs as "objects", and we use the notation $X \in \mathcal{S}$, for indicating an object X of an arbitrary space $\mathcal{S}$ of all possible pairs (or objects) $X \equiv (K, V)$.

From the formal point of view, we can define the mapping function of the CAN as:

$$G : \mathcal{S} \rightarrow P^N, \tag{1}$$

where P^N is an hyper–rectangle of $\mathbb{R}^N$ defined as:

$$P^N = [0, D_1] \times [0, D_2] \times \ldots \times [0, D_N], \tag{2}$$

with D_i denoting the $i-th$ side length of the CAN structure.

The principle of the CAN is to divide the hyper-rectangle P^N into a finite number of distinct rectangular zones, each of them associated to one and only one node of the network. The nodes are responsible for storing and searching of objects covered by their zone. Moreover, each node is aware of the nodes that cover adjacent zones, i.e., its neighbors. More precisely, for an N-dimensional space, two zones are neighbors if their sides overlap along $N-1$ dimensions and are adjacent along one dimension.

The basic operation in CAN is the lookup(key) function, which returns the corresponding "value" (the IP address of the node, for instance) for the given "key" (the coordinates of the point). This is useful for insertion, deletion, and retrieval purposes. The search starts from an arbitrary node of the CAN, and proceeds by routing a message towards its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates. In general, if we divide P^N uniformly into n zones, each node maintains $2N$ neighbors. Furthermore, the average routing path length is given by $(N/4)n^{(1/N)}$. In a real scenario, since the objects are not uniformly distributed, the space will be not uniformly divided and these values could vary significantly (see Section 4).

2.2 Metric Spaces

The mathematical metric space is a pair $\mathcal{M} = (\mathcal{D}, d)$, where $\mathcal{D}$ is the *domain* of objects and d is the *distance function* able to compute distances between any pair of objects from $\mathcal{D}$. It is typically assumed that the smaller the distance, the closer or more *similar* the objects are. For any distinct objects $X, Y, Z \in \mathcal{D}$, the distance must satisfy the following properties:

$d(X, X) = 0$	*reflexivity*
$d(X, Y) > 0$	*strict positiveness*
$d(X, Y) = d(Y, X)$	*symmetry*
$d(X, Y) \leq d(X, Z) + d(Z, Y)$	*triangle inequality*

2.3 Pivot-Based Filtering

In general, the pivot-based algorithms can be viewed as a mapping F from the original metric space $\mathcal{M} = (\mathcal{D}, d)$ to a N-dimensional vector space with the L_∞ distance. The mapping assumes a set $T = \{P_1, P_2, \ldots, P_N\}$ of objects from $\mathcal{D}$, called pivots, and for each database object O, the mapping determines its characteristic (feature) vector as $F(O) = (d(O, P_1), d(O, P_2), \ldots, d(O, P_N))$. We obtain a new metric space as $\mathcal{M}_N(\mathbb{R}^N, d^\infty)$. At search time, we compute for a query object Q the query feature vector $F(Q) = (d(Q, P_1), d(Q, P_2), \ldots, d(Q, P_N))$ and discard for the search radius r an object O if

$$d^\infty(F(O), F(Q)) > r \tag{3}$$

In other words, the object O can be discarded if for some pivot P_i,

$$| d(Q, P_i) - d(O, P_i) | > r \tag{4}$$

Due to the triangle inequality, the mapping F is *contractive*, that is all discarded objects do not belong to the result set. However, some not-discarded objects may not be relevant and must be verified through the original distance function $d(\cdot)$. For more details, see for example [7].

3 MCAN

The basic idea of our approach is to extend the CAN architecture in order to manage objects X of a generic metric space $\mathcal{M} = (\mathcal{D}, d)$. However, in metric spaces it is not possible to exploit any knowledge of coordinate information, and only distances between objects can be computed. To cope with this problem, we use the pivots paradigm for mapping the objects of the metric space to an N dimensional vector space. In particular, let $P_1, \ldots, P_N$ be the number of pivots selected from the metric data-set, we map an object $O \in D$, by means of the function $F()$ (introduced in the previous section) defined as:

$$F(O) : \mathcal{D} \rightarrow \mathbb{R}^N = (d(O, P_1), d(O, P_2), \ldots, d(O, P_N)) \tag{5}$$

This virtual coordinate space is used to store the object O into the MCAN structure, specifically into the node that owns the zone where the point $F(O)$ lies. Note that, the coordinate space of the MCAN is not Cartesian since the distance between two objects in MCAN is evaluated by means of the L^∞ distance (instead of the Euclidean distance). Routing in MCAN works in the same manner as for the original CAN structures. An MCAN node maintains a coordinate routing table that holds the IP address and virtual coordinate zones of each of its immediate neighbors in the coordinate space.

3.1 Notation

In this section we provide a number of definitions required to present our results. We use the capital letter for indicating metric space objects $X \in \mathcal{D}$, the overline small letter for denoting the corresponding vector in the coordinate space $\overline{x} \in \mathbb{R}^N$, and x_i for representing the values of its i-th coordinate. Moreover, we denote a node of MCAN by the bold symbol $\mathbf{n}$. Since there is no possibility of confusion, we use the same symbol $d(.)$ for indicating the distance between metric objects and for indicating the L^∞ distance between the corresponding point in the coordinate space, e.g., $d(\overline{x}, \overline{y}) = d^\infty(F(X), F(Y))$, where $\overline{x} = F(X)$ and $\overline{y} = F(Y)$. As we already explained, the MCAN is contractive, therefore $d(\overline{x}, \overline{y}) \leq d(X, Y)$ always holds.

Each node $\mathbf{n}$ maintains its region information referred as $\mathbf{n}.R$. Moreover, since the region $\mathbf{n}.R$ is an hyper–rectangle it can be uniquely identified by its vertex closer to the origin, denoted as $\mathbf{n}.R.\overline{x} = (\mathbf{n}.R.x_1, \mathbf{n}.R.x_2, \ldots, \mathbf{n}.R.x_N)$, and by

the lengths of the relative sides, i.e., $\mathbf{n}.R.l_1, \mathbf{n}.R.l_2, \ldots, \mathbf{n}.R.l_N$. More precisely, the region $\mathbf{n}.R$ is defined as follows

$$\mathbf{n}.R = \{\forall \overline{x} \in \mathbb{R}^N \mid \forall\ i,\ \mathbf{n}.x_i \leq x_i < \mathbf{n}.x_i + \mathbf{n}.l_i\}$$

The node $\mathbf{n}$ also maintains the set of the neighbor nodes' information $\mathbf{n}.M = \{\mathbf{m}_1, \ldots, \mathbf{m}_h\}$.

Given a point $\overline{x} = F(X)$, the predicate $X \in \mathbf{n}$ allows us to check if the corresponding point $\overline{x}$ lies in the zone maintained by the node $\mathbf{n}$. More formally:

$$X \in \mathbf{n} \quad \Leftrightarrow \quad \overline{x} \in \mathbf{n}.R$$

A range query of radius r and centered in the object C is denoted as $\mathcal{Q} = (\overline{c}, r)$. The predicate $\mathcal{Q} \cap \mathbf{n}$ allows to check if the query region $\mathcal{Q}$ intersects the zone associated with $\mathbf{n}$. Note that, the range query in the L^∞ space is given by an hypercube of side $2r$ centered in $\overline{c}$.

We can now introduce the formal definition of an N-dimensional MCAN structure, referred as MCAN^N, which is composed of a set of k ($k > 0$) network nodes $\{\mathbf{n}_1, \ldots, \mathbf{n}_k\}$ such as:

1. $\forall\ i, j \mid i \neq j \quad \mathbf{n}_i.R \cap \mathbf{n}_j.R = \emptyset$
2. $P^N = \bigcup_{i=1}^{k} \mathbf{n}_i.R$
3. $\mathbf{n} \in \mathbf{m}.M \Leftrightarrow$
   $\exists k \mid 1 \leq k \leq N, (\mathbf{n}.R.x_k + \mathbf{n}.R.l_k = \mathbf{m}.R.x_k) \vee (\mathbf{m}.R.x_k + \mathbf{m}.R.l_k = \mathbf{n}.R.x_k),$
   $\forall w \neq k\ [\mathbf{n}.R.x_w, \mathbf{n}.R.x_w + \mathbf{n}.R.l_w[\cap [\mathbf{m}.R.x_w, \mathbf{m}.R.x_w + \mathbf{m}.R.l_w[\neq \emptyset$

In the definition, Point 1. states that the zones covered by the network nodes do not overlap. Point 2. states that the union of the zones cover the whole MCAN^N space P^N (there are no holes). Finally, Point 3. declares the condition for a network node $\mathbf{n}$ to be a neighbor of $\mathbf{m}$ (as explained in Section 2).

3.2 Construction

An important feature of the CAN structure is its capability to dynamically adapt to data-set size changes. As we will see in the experimental evaluation, we are interested in preserving the scalability of the MCAN, which means that we want to maintain stable the response time of the queries. Since the size of the space allocated to store objects in each node is limited, when a node exceeds its limit it splits by sending a subset of its objects to a free node and by assigning its part of original region. Note that, limiting the storage space, and then the number of objects each node can maintain, we also limit the number of distance computations a node have to evaluate during a range query computation.

It is important to observe that in some cases we might want to use all the nodes available in the network. Previous work like [4] have studied this possibility in a generic CAN structure by allowing a node to split even if it does not exceed it storage space. Obviously, such methodology can also be applied in our MCAN. On the other hand, in a P2P environment, we would like to leave the nodes

the possibility to freely join and leave the network, without corrupting it. As explained in [4], this is possible with a CAN, even providing some fault-tolerance capabilities [8].

Since the pivots needed to be determined before the insertion starts, we assume a characteristic subset of the indexed data-set (about 500 objects) is known at the beginning. In the MCAN, we use the Incremental Selection algorithm described in [7]. In principle this algorithm tries to maximize the average L^∞ distance between arbitrary pairs of vectors of the N–dimensional space (i.e. $d^\infty(F(X), F(Y))$).

3.3 Insertion

An insert operation can start from any node of the MCAN. It starts by mapping the inserted object X to the virtual coordinate space using function $F()$, then it checks if $\overline{x} = F(X)$ lies in the zone maintained by the node $\mathbf{n}$ itself (i.e. $X \in \mathbf{n}$). If this is not the case, the node has to forward the insertion request. From this point, the insertion proceeds with the greedy routing algorithm used for standard CAN structures: the inserting node forwards the insertion operation to the neighbor node which is closer to the point $\overline{x}$ by using the L^∞ distance. The objective is to find the node $\mathbf{n}$ for which $X \in \mathbf{n}$, minimizing the number of messages. If $\overline{x}$ lies in the region maintained by the receiving node, the object X is stored there, otherwise a neighbor node is selected with the same technique and the insert operation is forwarded again until the object X is inserted.

The node $\mathbf{m}$ which stores the object X must reply to the node who started the insert operation. If the node $\mathbf{m}$ exceeds its capacity it is split. Eventually, the object X is inserted into $\mathbf{m}$ or into the new allocated node.

3.4 Split

In MCAN, we apply a balanced split, that is the resulting regions contain practically the same amount of data (object occupancy). During this process, the splitting node will just request a node from a free node list to join the network, and one half of the data, in terms of occupancy, is reallocated there.

If we define $\mathbf{n}_1$ as the splitting node, $\mathbf{n}_1.R$ as the old region, $\mathbf{n}_1.R'$ as the new one, and $\mathbf{n}_2$ as the new node, the split regions must satisfy the following equations:

$$\mathbf{n}_1.R' \cup \mathbf{n}_2.R = \mathbf{n}_1.R \ , \quad \mathbf{n}_1.R' \cap \mathbf{n}_2.R = 0$$

Moreover, to respect these constrains, we create the new two regions by dividing the original one along one coordinate of the space. Therefore, the new regions, $\mathbf{n}_1.R'$ and $\mathbf{n}_2.R$, must satisfy the following two equations:

$$\mathbf{n}_1.R'.x_s = \mathbf{n}_1.R.x_s, \ \mathbf{n}_2.R.x_s = \mathbf{n}_1.R.x_s + \mathbf{n}_1.R'.l_s, \ \ \mathbf{n}_2.R.l_s = \mathbf{n}_1.R.l_s - \mathbf{n}_1.R'.l_s$$

Note that that we only have to chose s and $\mathbf{n}_1.R'.l_s$. In order to decide s, for each dimension i we find $\mathbf{n}_1.R'.l_i$ that divide the objects into two halves. To avoid regions with small sides we chose s as the dimension i for which $|\mathbf{n}_1.R'.l_i - \mathbf{n}_1.R.l_i/2|$ is minimum.

After the splitting process, the node $\mathbf{n}_1$ sends a message to all its neighbors informing them about the update of its region. To those neighbors, which are also neighbors of $\mathbf{n}_2$, it sends also information about the new node. The new node is informed by $\mathbf{n}_1$ about its neighbors that are a subset of the $\mathbf{n}_1$ neighbors. At the end, $\mathbf{n}_1$ can discard information about the nodes that no more are its neighbors.

3.5 Range Query

A range query operation can start from any MCAN node. As shown in Figure 1, for a given query object and range radius, there is a certain number of nodes whose regions intersect the query region (which is an hypercube $\mathcal{Q} = (\overline{c}, r)$ as defined in Section 3.1). Obviously, only the intersecting nodes must process the range query operation. The requesting node maps the query object into the virtual coordinate space using the function $F()$. Then it checks if it is involved in the range query operation (i.e., when it intersects $\mathcal{Q}$). If the node is not involved in the query, it forwards the range query operation to the neighbor node that is closest to region $\mathcal{Q}$, using the L^∞ distance. This operation is performed in a similar way as described for the insert operation.

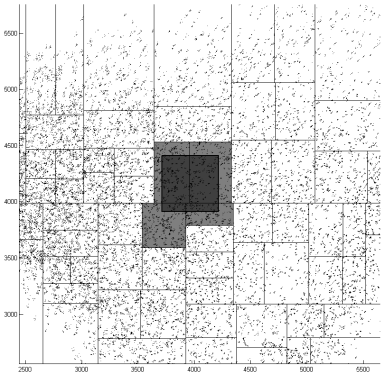


Fig. 1. Example of range query in a two dimensional space. The darker square is the query region, while the brighter rectangles correspond to the involved nodes.

When a node that is involved in the range query is reached by the query request, it forwards it to each neighbor that is also involved and then it starts processing the range query over its local data-set. During the range query execution inside a single node, a local access structure can also be used. In this paper, we used the same pivots chosen to define the MCAN space to reduce the number of distance evaluations performed inside a single node. Using the pivot-based filtering, we are able to significantly reduce the number of distance evaluations inside the nodes. In a more sophisticated implementation of MCAN, each node could have its own local data structure to efficiently search inside a single node.

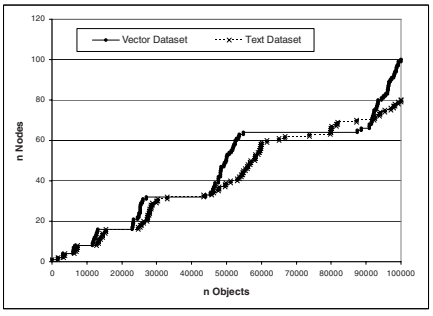


Fig. 2. Number of nodes for increasing data-set size

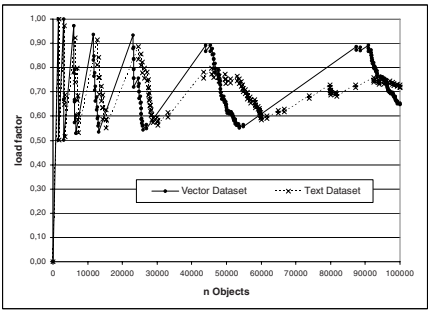


Fig. 3. Average number of objects per node for the MCAN^3 and for increasing data-set size

In order to allow the requesting node to know when all the nodes involved in the query have finished to work, the MCAN proceeds as follows:

- A node involved in the range query:
  - it receives the range query request $\mathcal{Q} = (\overline{c}, r)$ and a (possibly empty) list of the nodes already involved in the query (which we refer to as *INL*),
  - it forwards the query request to its neighbors involved in the query which are not included in *INL* adding them to the list and sending the new *INL** to them,
  - it computes the query over its local data,
  - it replies with a message containing its result set (if any) and the *INL**.
- The requesting node as it receives the reply messages, updates a local list of the involved nodes in the query, and marks the ones that have already answered.
- The requesting node will know that the operation is terminated when all nodes of the local list will have replied (i.e., when all the nodes of the list have been marked). The result set of the query is given by union of the result sets of the replying nodes.

Note that, the first node involved in the query receives an empty *INL*. Another important observation is that with this scheme we do not guarantee that a node does not receive multiple requests for the same query. However, this is not a problem since each distinct query is associated with a unique identifier, so that a node ignores multiple requests.

4 Performance Evaluation

In this section we present a preliminary experimental evaluation of MCAN. The metric data-sets used are: 100,000 of 45-dimensional vectors of color features extracted from images; 100,000 Czech sentences of length between 20 and 300

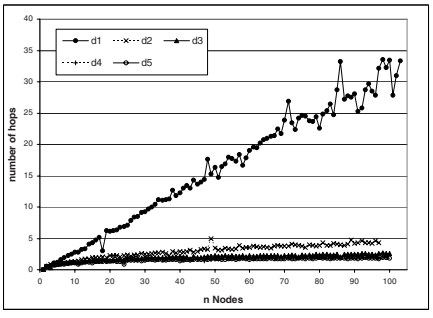


Fig. 4. Average number of hops for different dimensions as the number of nodes grows (vector data-set)

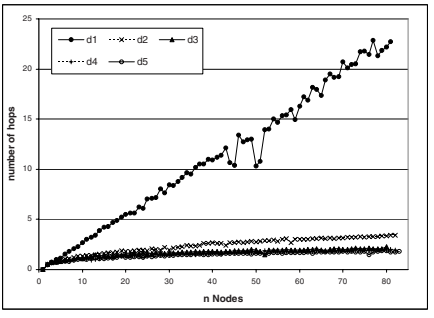


Fig. 5. Average number of hops for different dimensions as the number of nodes grows (text data-set)

characters. Vectors are compared by the Euclidean distance measure while for sentences we use the Edit distance.

We analyze the behavior of the structure in different dimensional spaces: from 1–d (i.e., involving one pivot), to 5–d space (i.e., involving five pivots). As already explained, we use the pivots also to reduce the number of distance computations during the query evaluation on individual nodes. However, independently of the number of dimensions N used by MCAN^N, we always generate 10 pivots in the experiments and we use the first N pivots for creating the MCAN^N zones. Moreover, all 10 pivots are used for filtering during a range query execution internally in nodes.

To study the scalability of the system, we fix the storage space available for each node and then, starting from a single server, we add objects into the system. When a server reaches its storage space limit, it splits. The limit was chosen in a way that after all the 100,000 objects have been inserted, the MCAN^N is composed of around 100 nodes. The node from which an insert operation or a range query starts is randomly selected. Moreover, in order to study the scalability of the system we perform a range query operations every 5,000 insertions.

In Figure 2, we report the number of nodes in the system as the data-set grows, for the MCAN^3 case (the other cases are very similar). Note from these experiments that, the number of nodes exhibit a stepwise behavior. This is due to the fact that the objects are randomly ordered, therefore the nodes are filled uniformly and then they tend to split at the same time. This is particularly evident for the vector data-set, where the objects have a fixed size, while the size of objects of the text data-set (strings) is variable.

In Figure 3, we report the average load factor for both the data-sets. We define the load factor as the total number of objects stored into the MCAN structure divided by the capacity of storage available on all nodes. As can be seen in the figure, the values are always between 0.5 and 1. This is always guaranteed, because when a node is split, half of the objects are migrated to the new node, therefore the node occupation cannot be less than 50%.

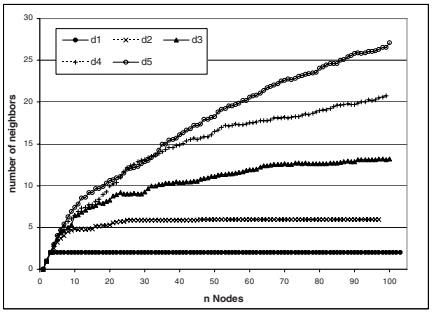


Fig. 6. Average number of neighbors for different dimensions as the number of nodes grows (vector data-set)

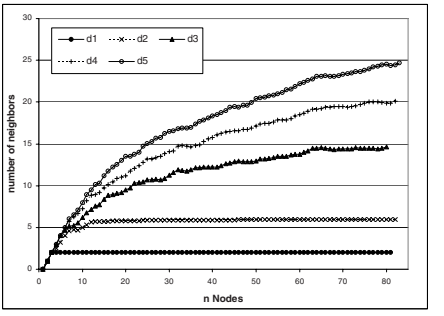


Fig. 7. Average number of neighbors for different dimensions as the number of nodes grows (text data-set)

In Figures 4 and 5, we report the average number of hops an insert operation travels, starting form a random node. For a given number of nodes, the number of hops is strictly correlated to the average number of neighbors each node has: the more the neighbors, the less the hops. In Figures 6 and 7, we report the average number of neighbors as a function of the total number of nodes for different space dimensionality. Comparing the number of hops with the number of neighbors, we can see that a good choice for the space dimensionality could be $N = 3$. In fact, by using more than 3 dimensions we do not reduce significantly the number of hops but significantly increase the number of neighbors and correspondingly the complexity of choosing the next node during the forwarding operations.

4.1 Range Query

For the performance evaluation of range queries, we selected 100 random objects from the data-set and for each of them we performed 8 different range queries every 5,000 insert operations. Due to lack of space, we do not report the average result set size for the different query radii, since they are linear to the data-set size. However, the heaviest range queries return around 3% of the objects for both vector and text data-sets. Note that, these results are independent from the type of access structure but depend on specific characteristics of the given data-sets.

In Figures 8 and 9 we report the average percentage of nodes involved during a range query operation for different radii as the data-set size grows. Observe that the bigger is the radius of the range query, the more the nodes involved in the query evaluation are. In a naive distributed system we could randomly distribute the objects among the nodes but in this case we would always involve all the nodes even for small radii.

For simple operations like the exact match, the standard CAN has been proved to be scalable. MCAN extends CAN by allowing similarity operations

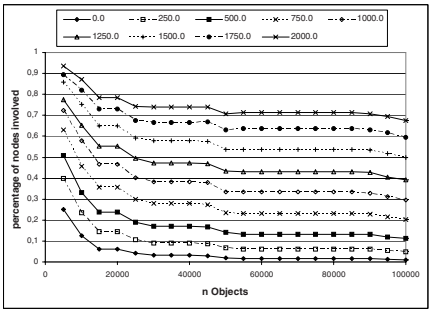


Fig. 8. Percentage of nodes involved in the range query as function of the data-set size for different radii (vector data-set)

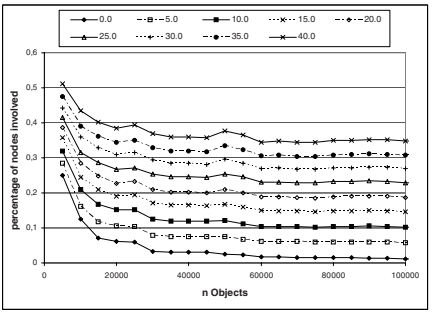


Fig. 9. Percentage of nodes involved in the range query as function of the data-set size for different radii (text data-set)

over generic metric space data-sets. In this scenario, we must be able to perform more complex operations such as similarity range queries. To preserve scalability also for such operations, we need more nodes as the complexity of the query grows. This aspect is evident in the plots of Figures 8 and 9, where the percentage of nodes involved for a small radius is smaller then the ones we obtains for greater radii. Note that, for a given range query, the percentage of nodes involved is almost constant. In fact, for a given range query the number of results is linearly dependent on the number of objects in the data-set and then the number of nodes involved is proportional to the number of results.

To study the complexity of the range queries, we use the number of distance computations. However, for the case of the edit distance (i.e., the Czech-sentences data-set) we must consider the fact that the complexity of a single distance computation is not constant but it is proportional to the string lengths. In this case we decided to use the *equivalent complexity of the edit distance* defined as $L(a)L(b)/\mu(L)^2$, where a, b are two strings evaluated with the edit distance, $L(.)$ is the length of the string, and $\mu(L)$ is the average length of the strings of the data-set.

In Figures 10 and 11, we report the average complexity of the range query operations as function of the number of equivalent distance computations of the most stressed node. This quantity measures in a way the *intraquery parallelism* as the parallel response time of a range query, if we neglect the message latency. In fact, the requesting node will have to wait the answer of all the involved nodes and then the response time of the query will be proportional to the number of distance computations of the most stressed node. Obviously this quantity is upper bounded by the capacity of the nodes of the MCAN. However, our experiments show that for most of the ranges, the intraquery parallelism remains quite lower than this upper bound, which, for example, in the case of the vector data-set is 1,542.

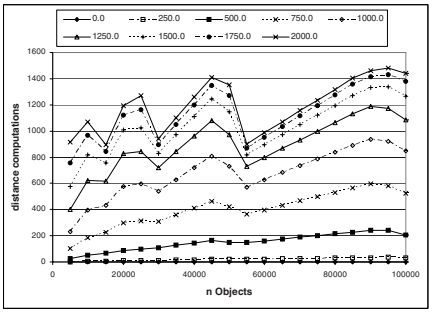


Fig. 10. Average number of distances evaluated by the most stressed node for each query and for different query range (vector data-set)

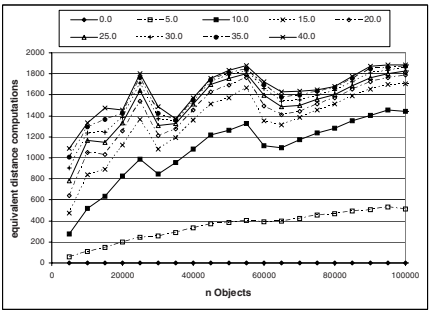


Fig. 11. Average number of equivalent distances evaluated by the most stressed node for each query and for different query range (text data-set)

5 Related Work and Conclusions

There have been several recent attempts to propose distributed structures for multi-dimensional or vector-based data. The MAAN structure [9] uses locality preserving hashing to support multi-attribute and range queries under the Chord protocol. The kd-trees and space-filling curves have been used by Prasanna, Yang and Garcia-Molina in [10] to support multi-dimensional range queries in P2P environments. A P2P system for information retrieval based on the vector space model and the latent semantic indexing together with the CAN P2P protocol has been proposed by Tang, Xu and Dwarkadas [11]. The problem of vector-based similarity search in P2P Data Networks has nicely been formalized by Banaei-Kashani and Shahabi [12] as the family of Small-World based Access Methods, SWAM. So far, the only native metric-based distributed data structure is the GHT* [13,14].

To the best of our knowledge, the MCAN structure is the first attempt to bridge Content–Addressable Networks and the capabilities of metric space indexing. MCAN is based on the concept of choosing pivots to map objects of a generic metric space in a multidimensional vector space of the MCAN. Since the mapping is contractive, 100% recall for queries processed by the MCAN is guaranteed.

The results summarized in Figures 8 and 9 should be considered as the first attempt to also demonstrate the *interquery parallelism* ability of MCAN. In fact, if on the one hand it is important to guarantee fast response time to individual queries, on the other hand the query should not involve the whole network, because other queries can be issued to the network at the same time, and not active nodes can simultaneously start evaluating them. Obviously, queries with large radii need more computational resources than small queries, but typically, there is always sufficient space for other queries to run. Also observe that the computational load on nodes activated by a query is not the same and on some

nodes the load is really minor. We are planning to fully investigate this issue in the near future.

Further future directions include the implementation of the Nearest Neighbor queries, more sophisticated leaving and join policies, and transaction management.

References

1. Chvez, E., Navarro, G., Baeza-Yates, R., Marroqun, J.L.: Searching in metric spaces. ACM Comput. Surv. **33** (2001) 273–321
2. Dohnal, V., Gennaro, C., Savino, P., Zezula, P.: D-index: Distance searching index for metric data sets. Multimedia Tools and Applications **21** (2003) 9–13
3. Devine, R.: Design and implementation of DDH: A distributed dynamic hashing algorithm. In: FODO. (1993) 101–114
4. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proc. of ACM SIGCOMM 2001. (2001) 161–172
5. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for internet applications. In: Proc. of the 2001 ACM SIGCOMM Conference. (2001) 149–160
6. Litwin, W., Neimat, M.A., Schneider, D.A.: LH* — a scalable, distributed data structure. ACM Transactions on Database Systems **21** (1996) 480–525
7. Bustos, B., Navarro, G., Chvez, E.: Pivot selection techniques for proximity searching in metric spaces. In: Proc. of SCCC01. (2001) 33–40
8. Saia, J., Fiat, A., Gribble, S.D., Karlin, A.R., Saroiu, S.: Dynamically fault-tolerant content addressable networks. In: IPTPS. (2002) 270–279
9. Cai, M., Frank, M., Chen, J., Szekely, P.: Maan: A multi-attribute addressable network for grid information services. In: GRID '03: Proceedings of the Fourth International Workshop on Grid Computing, Washington, DC, USA, IEEE Computer Society (2003) 184
10. Ganesan, P., Yang, B., Garcia-Molina, H.: One torus to rule them all: multi-dimensional queries in p2p systems. In: WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, New York, NY, USA, ACM Press (2004) 19–24
11. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks (2002)
12. Banaei-Kashani, F., Shahabi, C.: Swam: a family of access methods for similarity-search in peer-to-peer data networks. In: CIKM '04: Proceedings of the Thirteenth ACM conference on Information and knowledge management, ACM Press (2004) 304–313
13. Batko, M., Gennaro, C., Zezula, P.: A scalable nearest neighbor search in p2p systems. In: Proc. of the the 2nd International Workshop on Databases, Information Systems and Peer-to-Peer Computing. Lecture Notes in Computer Science (2004) To appear.
14. Batko, M., Gennaro, C., Zezula, P.: Scalable similarity search in metric spaces. In: Proc. of the DELOS Workshop on Digital Library Architectures: Peer-to-Peer, Grid, and Service-Orientation. (2004) 213–224

Range Query Optimization Leveraging Peer Heterogeneity in DHT Data Networks

Nikos Ntarmos, Theoni Pitoura, and Peter Triantafillou

R.A. Computer Technology Institute and
Computer Engineering & Informatics Dept.,
University of Patras, Rio, Greece
{ntarmos,pitoura,peter}@ceid.upatras.gr

Abstract. In this work we address the issue of efficient processing of range queries in DHT-based P2P data networks. The novelty of the proposed approach lies on architectures, algorithms, and mechanisms for identifying and appropriately exploiting powerful nodes in such networks. The existence of such nodes has been well documented in the literature and plays a key role in the architecture of most successful real-world P2P applications. However, till now, this heterogeneity has not been taken into account when architecting solutions for complex query processing, especially in DHT networks. With this work we attempt to fill this gap for optimizing the processing of range queries. Significant performance improvements are achieved due to (i) ensuring a much smaller hop count performance for range queries, and (ii) avoiding the dangers and inefficiencies of relying for range query processing on weak nodes, with respect to processing, storage, and communication capacities, and with intermittent connectivity. We present detailed experimental results validating our performance claims.

1 Introduction

Structured P2P systems have provided the P2P community with efficient and combined routing/location primitives. This goal is accomplished by maintaining a structure in the system, emerging by the way that peers define their neighbors. These systems are usually referred to as Distributed Hash Tables (DHTs)[1,2,3]. DHTs have managed to take routing and location of data items in P2P systems to the next level; from the nondeterministic, flood-based techniques used in unstructured P2P overlays, DHTs provide us with strong probabilistic (under node failures and skewed data and access distributions) guarantees on the worst-case number of hops required to route a message from a node to any other node in the system or, equivalently, for a node in the system to locate data items published therein. Unfortunately, traditional DHT overlays were designed to only support exact-match queries. This has led researchers to investigate how they could enhance P2P systems to support more complex queries[4,5,6,7,8,9,10,11,12,13,14,15]. On another axis, one of the main characteristics of widely deployed P2P networks (e.g. Gnutella, Kazaa, etc.) is that participating peers are largely heterogeneous, with regard to their processing power, available main memory and disk storage, network bandwidth, and internet connection uptime. Relevant studies of P2P networks [16,17] have shown that this large heterogeneity is also depicted in the distribution of the query processing

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 111–122, 2007.

chores across the node population; in the Gnutella network, about 70% of nodes share no files at all with the community, while 5% of nodes serve almost 95% of the queries posed.

Recognizing heterogeneity among peers and harnessing it to speed-up complex query processing has not been done before in the structured P2P world, although most workable real-world P2P applications utilize it, by building multi-level hybrid networks. Our philosophy is based on this very observation. We wish to bring this "hybrid" design into the DHT world and utilize it for complex query processing. The question now is how to do this efficiently. Therefore, we believe that harnessing the power of powerful and "altruistic" nodes is the key to providing an efficient way to expedite complex query processing in a P2P setting. In this work we discuss the range query case. Note that we do not propose another range queriable P2P overlay. We leverage the functionality, scalability, and performance of such network overlays, and present a two-layered architecture were powerful nodes are identified and assigned extra tasks. We do so in an efficient and low-overhead way, using the functionality already provided by the underlying structured P2P network, with significant gains in range query processing costs. To our knowledge, this is the first work to look into this issue.

2 Range Queries over DHTs

Traditional DHTs. DHTs use an m-bit circular identifier space for nodes and objects/documents, and modulo-N arithmetic (with $N = 2^m$ being the maximum number of nodes/documents in the system). Both node and document IDs are usually based on some randomizing (usually cryptographic) hashing (e.g. SHA-1) of a node/document specific piece of information. Nodes maintain links to other nodes in the overlay, according to the DHT's geometry and distance function[18]. Using these links, DHTs can route between any two nodes in the overlay in $O(\log N)$ hops, while maintaining $O(\log N)$ links. A document d inserted into a DHT is stored on the node whose ID is closer to the document's ID, according to the DHT's distance function. This node is called the document's "successor" (or "root"). We assume that data stored in the P2P network are structured in a $(k + l)$-attribute relation $R(a_1,..., a_k,b_1,...,b_l)$, where a_i, b_i are the attributes of R, with every tuple t in R being uniquely identified by a primary key $t.key$. This key can be either one of the attributes of the tuple, or can be calculated otherwise (e.g. based on the values of one or more of the attributes of t). Furthermore, attributes a_i are used as single-attribute indices of t, with each a_i being characterized by the domain $t.a_i.\mathcal{D} : \{t.a_i.v_{min}, t.a_i.v_{step}, t.a_i.v_{max}\}$ of its values.

Now suppose that every index tuple is added to the DHT, using an ID generated by (SHA-1) hashing the attribute's value. This would result in tuples being spread across all participating nodes in a uniform manner, but would also lead to tuples with successive index values being stored on completely unrelated nodes. This fact renders traditional DHTs highly inefficient for range query processing; given a range on the domain of the index values, a traditional DHT has to execute queries for each and every value in the range interval! A range query for r consecutive values would require on average r queries to be executed, for a total of $O(r \times log(N))$ hops. In a non-densely populated value domain, most of these queries would return no data items. If all nodes had global knowledge on every value stored in the P2P overlay (we'll call this system the *Enhanced*

DHT), they could then skip queries for non-existing values. Thus, if only r' out of r values exist in the system, it would take on average r' queries, or $O(r' \times log(N))$ hops. Although better than $O(r \times log(N))$, this still is too expensive for a real-world system.

Range-queriable DHTs. After the first wave of DHTs, the peer-to-peer research community started investigating structured overlays that would allow for more complex queries than simple equality, while achieving the same performance and scalability figures of early DHTs. This has lead to the design and implementation of several specially crafted DHTs, capable of efficient complex query processing: SkipNet[4], Skip Graphs[5], OP-Chord[11], PIER[13], Mercury[14], P-Trees[6], as well as the works by Ganesan et al.[7,8], Gupta et al.[9], and Sahin et al.[10], are examples of such systems.

The common idea behind these overlays with regard to range query processing, is that traditional DHTs destroy the locality of content, due to the randomizing (cryptographic) hash functions used to construct document IDs prior to insertion, but locality is a desired property when sequential access is sought, as is the case in range queries. Due to this, these overlays use the actual content (e.g. attribute values in a P2P-based RDBMS environment, file names in a file-sharing system, etc.) rather than the outcome of a (cryptographic) hash function to sort and store documents on the overlay. Thus document locality is preserved and range query processing consists of: (i) locating the node responsible for the start of the range, and (ii) following one-hop "successor" pointers until we reach the node responsible for the end of the range, gathering results in the meantime. If the desired range spans q nodes on the overlay, the above lead to a hop-count complexity of $O(\log N + q)$; $O(\log N)$ hops for phase (i), plus q more hops for phase (ii). We will use the term *LP-DHT ring* to refer to such a locality-preserving, range-queriable overlay in the rest of this paper.

3 The RangeGuard

Our intention is to form a second LP-DHT ring, the RangeGuard ring, above the LP-DHT ring (fig. 1), composed of powerful nodes – the RangeGuards or RGs – burdened with extra functionality chores. Each such node is responsible for storing the index tuples placed in nodes between its predecessor RangeGuard and itself. Thus, if there are M RGs in the system, they partition the normal LP-DHT ring into M continuous and disjoint ranges. Each RG maintains routing information for both the lower-level ring and the RangeGuard ring. Additionally, there is a direct link from each peer to the next RG in the upper-level ring (i.e. to the RG responsible for the peer). Nodes in the lower-level ring probe their RG (e.g. as part of the standard LP-DHT stabilization process), and automatically update the index tuples it stores.

RGs must be (i) powerful enough and willing to withstand the extra (storage, processing, communication) load, and (ii) connected most of the time, to provide hop-count guarantees for range queries and to avoid large transfers due to their joining/leaving. This, in turn, calls for a mechanism to identify and exploit candidate RGs in an efficient and transparent manner[1]. Given the functionality offered by our initial (without RangeGuards) infrastructure, RGs are identified and located as follows.

[1] We assume that peers will not act maliciously or selfishly (leaving countermeasures for such behavior as a possibility for future work), as is the case with most DHT-based research.

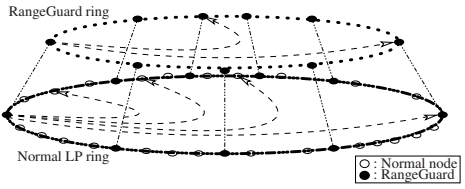


Fig. 1. The RangeGuard architecture. *RG*s form a second LP-DHT ring of their own, taking responsibility (using consistent hashing) for ranges of nodes on the lower-level ring.

3.1 Node Performance Counters and the Node Performance Relation (NPR)

The administrator of each node selects whether she wants her node to be a candidate for RangeGuard membership or not (much like it is done now with super-peers in most unstructured P2P sharing applications). A candidate node n, with id $n.id$, keeps track of the amount $n.\alpha$ of retrieval and/or just routing requests it serves. This information is updated periodically, every $\mathcal{E}$ seconds (also called an *epoch*). Thus, it keeps two *node performance counters* (or NPCs) – $n.\alpha_c$ and $n.\alpha_p$ – of requests served during the current and the previous epoch respectively.

This information is stored in the system as a four-attribute node performance relation $\mathcal{NPR} : \{n.id, n.\alpha_p, \mathcal{U}, status\}$, with primary key $n.id$, and indexed by $n.\alpha_p$. $status$ is a boolean variable, set to $true$ if the node is a member of the RangeGuard. $\mathcal{U}$ is a counter, incremented on every update of the tuple, and left-shifted (assuming a little-endian architecture) (i) on every update or (ii) every $\mathcal{E}$+δ seconds, with the timer being reset on every update. δ is a quantity depending on measurable characteristics (e.g. round-trip / ping time) of the end-to-end connection between node n and the node storing the index tuple with n's metadata. $\mathcal{U}$ encapsulates the amount of time a peer stays connected to the network. We believe that the network uptime of a peer and the number of requests it has served during this time, are enough evidence of a node's power and fitness for the RG ring. More elaborate metrics may be used instead, under the same intuition of storing these tuples on the lower LP-DHT ring; investigation of such metrics is an open issue and a subject of ongoing and future work. If a node wishes to cease being a candidate RG, it suffices to set a low value (e.g. 0) for its α_p and stop updating this information (so that its $\mathcal{U}$ value decays with inactivity). Also note that the $\mathcal{NPR}$ tuple of a node n is stored on this very node, since the primary key of the relation is the $n.id$ attribute.

Cost of Maintaining NPR. Candidate RG nodes must update their $\mathcal{NPR}$ index tuple – remember that $\mathcal{NPR}$ tuples are also indexed by their $n.\alpha_p$ field. This operation requires 2 LP-DHT ring lookups every epoch $\mathcal{E}$; one lookup to delete the index tuple for the old value of $n.\alpha_p$, and one to insert the index tuple for the new value of $n.\alpha_p$. The overall cost, in terms of hops, is $O(log(N))$ (since every lookup needs $O(log(N))$ hops), while the bandwidth consumption is minimal given the very small size of these index tuples. Alternatively, we can either keep a link to the node last seen storing the relevant index tuple and start the lookup from there, or follow a soft-state approach. Moreover, the overall cost is tunable via $\mathcal{E}$, so we can trade-off $\mathcal{NPR}$ index freshness for bandwidth and hops.

3.2 Joining the RangeGuard

A node that is to join the RangeGuard uses its RG as the "bootstrap" node for the RangeGuard ring. The RG is responsible for retrieving the metadata of the candidate node and checking whether it is powerful enough (i.e. has served more requests than a predefined threshold) and has stayed online for long enough (based on the corresponding $\mathcal{U}$ value) to be allowed into the RangeGuard. If all prerequisites[2] are met, the standard LP-DHT join protocol is executed and the candidate node is promoted to the RG ring, otherwise the protocol terminates. After that, it updates the $status$ field in its entry in the $\mathcal{NPR}$ relation on the lower-level ring to reflect its promotion to RG status and notifies nodes in its arc of responsibility of its existence. Alternatively, this step may be left as part of the lower-level ring stabilization/maintenance process. The cost of joining the RangeGuard ring consists of: (i) the cost to contact and send the relevant NPR tuple to the RG responsible for the joining node (1 hop), and (ii) the cost of the standard LP-DHT ring *join* protocol, for the RG ring. Thus, the hop-count cost for joining the RG ring is in $O(log(M))$, while the extra bandwidth consumption is minimal (given the expected small size of the RG ring and the very small size of $\mathcal{NPR}$ tuples).

Admission into the RangeGuard. There are two ways for a node to be admitted into the RangeGuard: either (i) be promoted by a node already in the RangeGuard who wishes to shed some of its load, or (ii) volunteering to take up some region in the address space for which there exists no RG.

i. Promotion. Due to irregularities in the data or access distribution, a RangeGuard may get overloaded with incoming requests. Moreover, it is possible for a region in the RangeGuard to be underpopulated (e.g. imagine the RangeGuard ring in its setup phases). In such cases, a member of the RangeGuard can ask for support from candidate RangeGuards by promoting them to RG status.

With the infrastructure described earlier, when a RG wants to promote a node to RangeGuard status for a region around a point p – i.e. the id of a node in a distance of at most ϵ from a point p in the lower-level ring, with access count greater than a in the previous epoch, and a $\mathcal{U}$ value above u – it merely executes the range query:

select id **from** $\mathcal{NPR}$ **where** $\mathcal{U} > u$ **and** $\alpha_p > a$ **and** $0 \leq id - p < \epsilon$

The result set of this query will contain the IDs of candidate RGs in the region of interest. It is then up to the RG who originated the query to select the best candidate, inform it of its promotion, and initiate the *join* protocol to add it to the RG ring.

ii. Volunteering. Candidate RGs may lie in any region on the lower-level ring. For data/access distribution irregularity reasons similar to the ones urging RangeGuards to ask for support, it is possible for some regions to have such low data/access loads that the RangeGuard responsible for them has never been in need of support. This could result in large arcs/ranges on the lower-level ring being mapped to a single RangeGuard node, located many hops away on the lower-level ring from the first nodes on

[2] These thresholds will probably vary depending on the semantics sought from the RangeGuard. Calculating crisp theoretical thresholds is an orthogonal issue and left as future work.

this arc. Although this is not an issue in the steady state, it may increase the time needed by a node on this arc to find a new RG, should the current RG leave the system abnormally.

We, thus, allow candidate RG nodes on the lower-level ring to volunteer for an RG position; if a candidate RG detects a situation as the one described earlier (i.e. a large distance between itself and its RG), it can contact the latter and ask to be promoted to RG status. The RG is responsible for going through the α_p and $\mathcal{U}$ checks and admitting the candidate to the RangeGuard or not.

3.3 Leaving the RangeGuard

Similarly, a RG may decide to leave the RangeGuard if it finds itself in a situation where its arc of responsibility becomes very small (due to candidate RGs being promoted to RG status in its vicinity), or the load it faces as an RG drops below some predefined threshold (e.g. an estimate of the load it should have, based on uniform data/access load). A RG that wishes to leave the RangeGuard ring, goes through the following steps: (1) it follows the LP-DHT ring "leave" protocol, transferring its RG-related stored data to the appropriate node(s) on the RG ring, (2) it updates the $status$ field in its entry in the $\mathcal{NPR}$ relation on the lower-level ring, to denote that it is no longer a RangeGuard, and (3) optionally, it notifies the nodes that link to it on the RG ring to update their links, or leaves this to be done as a part of the RG ring stabilization/maintenance process.

Note that our approach uses standard LP-DHT ring operations to set up and maintain the RangeGuard ring. With the exception of the second step, the procedure described above is the standard LP-DHT *leave* protocol. Also note that the leaving RG does not need to notify nodes on its arc of responsibility of their new RG, since that will be achieved during the lower-level ring stabilization process. Consequently, the cost for a node to leave the RangeGuard is equal to the cost of executing the standard LP-DHT *leave* protocol for the RG ring, while data transfer is minimal due to the very small size of $\mathcal{NPR}$ tuples and the size of the RG ring. On the other hand, there is the requirement for several RangeGuard peers with enhanced capabilities. This is not unrealistic, since many peers in real-life applications have proved to be more powerful. With the notion and exploitation of RangeGuards we can harness this power heterogeneity to achieve higher efficiency in range query processing.

3.4 Range Query Processing Using RGs

With RangeGuards in the scene, a query $(a_i.v_{low}, a_i.v_{high})$ on attribute a_i will be sent from the requesting node directly (1 hop) to the RG responsible for the requesting node's data. After this point the RangeGuards assume responsibility to gather the requested information, using the LP-DHT algorithm described earlier, except that now all operations take place on the RangeGuard ring (fig. 2). With data placement on the lower ring being reflected on the RangeGuard ring, the requested index tuples will reside between RG_l responsible for $a_i.v_{low}$ and RG_h responsible for $a_i.v_{high}$. This algorithm requires 1 routing hop to reach the RangeGuard ring, another $O(log(M))$ hops in the

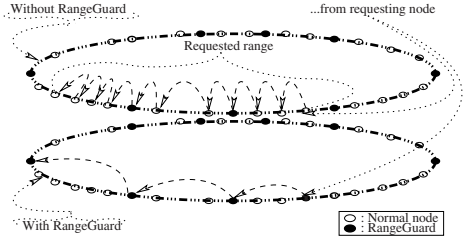


Fig. 2. Range query processing with and without *RG*s

RG ring to reach RG_l, and as many routing hops as there are RangeGuards between RG_l and RG_h. Moreover, the $O(log(M))$ term can be further improved to $O(1)$, by using techniques similar to those presented in [19] or [20].

Since, there will probably be much fewer RGs in the system than there are nodes, and RangeGuards are more powerful (with respect to computing capacities, network bandwidth, and network uptime) than the average node in the system, this architecture is significantly more efficient that the one presented earlier. Specifically, for $5\% \times N$ RGs, although the worst-case hop-count efficiency remains in $O(N)$, it now has a rather significantly lower constant modifier (i.e. a 5% – or 20 times – lower hop-count). Note that we require a mere $5\% \times N$ nodes to be powerful and altruistic in our setting; as relevant research has pointed out [16,17], we can expect an average 5% of the node population in wide-scale peer-to-peer data sharing networks, such as Gnutella and Kazaa, to be powerful, altruistic nodes. Thus, by harnessing the full power of all these nodes, we can achieve even higher performance gains than those outlined above.

3.5 Modifications to the LP-DHT Overlay

We have extended the underlying LP-DHT system in the following fields. First we have provided appropriate protocols to allow nodes to join/leave the RangeGuard ring, while guaranteeing correct operation of the overall system, as well as a method to discover and use candidate RGs (described in detail in sect. 3.2 and 3.3). We have also altered the query processing protocol, to utilize and harness the extra functionality offered by the RangeGuard (described in sect. 3.4). As far as routing state is concerned, we have added 1 more entry to each node to point to the RG responsible for it. For fault-tolerance reasons and faster recovery from failing/leaving RGs, we may choose to maintain links to the next k RGs. Note that routing state size is still in $O(\log N)$.

With the extra information in the nodes' routing tables, we also need to tweak the stabilization process to include the RG entry in the set of links to probe. Much like the standard stabilization process, a node issues a query for its ID on the RG ring. The relevant information is by design stored on the responsible RG; thus, the response to this query will originate from the RG currently responsible for the arc in which the querying node is located. If the RG responsible for the node has changed (e.g. due to more candidate RGs joining the RangeGuard), the node will get back a response from a different RG and will thus update its RG link. Finally, if all of a node's k RG links

have failed simultaneously, then the node can fall-back to querying the lower-level ring for a RangeGuard (i.e. a node whose $status$ field is set to $true$) in its vicinity.

4 Load Distribution on the RG Ring

In order to emulate the 95%-5% observation of [16] (i.e. 5% of all nodes serve 95% of all requests in the system), we have nodes on the LP-DHT ring flip a biased coin and dispatch queries to the RG ring with a 0.95 probability, while processing them solely on the LP-DHT ring with probability 0.05. Apart from relevant provisions by the LP-DHT, the load is further balanced on the RG ring by the load-aware join/leave protocols.

As far as join/leave is concerned, remember that RGs may call for support from candidate RG nodes when overloaded, and may decide to leave the RG ring if their load is too low or their arc of responsibility is too narrow. In the former case, the RG calling for help can choose among several candidate RGs, as returned from the relevant query. Since this node knows which part of its arc of responsibility causes it the more load, it can choose an appropriate candidate RG to shed this very load. The above algorithm is able to provide us with the basics for having a balanced access load. As already mentioned, we expect candidate RangeGuards to be uniformly distributed on the lower-level ring. Thus, RangeGuards calling for help will have a good probability of finding a candidate RangeGuard in their arc of interest.

With respect to data placement, if the popularity of a value does not depend on its position in the attributes domain (e.g. value v is not the most popular for all attributes in the system) then having multiple attributes mapped on the same ring translates to having multiple popular items distributed to all nodes on the system. In the opposite case, a random but easily computable offset value (e.g. the cryptographic hash of the the attribute's name) can be added (mod the maximum document ID) to all values in the attribute's domain. This provides us with enough randomization in the data placement to guarantee similar results, as we shall see in sect. 5.2. Note that, for an N-node system and the worst-case skewed distribution (i.e. one value being selected with probability 1 and the rest with probability 0), then N attributes are required to have a balanced load, under a best-case distribution of load based solely on the above facts. However, since the RG ring is much smaller than the LP-DHT ring, the required number of attributes is much smaller (i.e. $M << N$ attributes for an M-node RG ring).

Furthermore, due to this difference in the sizes of the RG and the LP-DHT rings, every RG node is responsible for the values assigned to multiple nodes on the LP-DHT ring, which leads to an even smoother distribution of the load on the RG ring. Moreover, we can easily apply load balancing techniques developed for the underlying LP-DHT, to further balance the load on both lower-level and RG nodes (e.g. virtual nodes[21], or load-aware node migration[22], when using OP-Chord[11]; rely on the load balancing effects of the overlay itself, when using SkipNets[4] or Skip Graphs[5], etc.).

5 Performance Evaluation

We will be using our home-brewed LP-DHT, OP-Chord[11], as our overlay of choice. OP-Chord is based on Chord, with an order-preserving hash function used instead of

SHA-1 for document ID generation, and the same range query processing principles discussed in sect. 2. We have extended the basic Chord simulator (available through http://www.pdos.lcs.mit.edu/chord/), adding support for index tuples and range queries, and implementing our OP-Chord and RangeGuard architectures. We have chosen to test two aspects of the system: (i) the hop count efficiency of our range query processing algorithm, and (ii) the distribution of storage requirements and accesses on participating RG nodes during range query processing, under realistically skewed distributions.

5.1 Hop Count

The experiments used a single-index-attribute relation, with the index attribute taking $5,000$ integer values, following a Zipf distribution with θ=0.7 over D[17]. Range queries are generated using a separate Zipf distribution over the domain D (again with θ=0.7) for their lower bound, and a uniformly distributed range span S, ranging from 1% to 50% of the attribute domain. We report on a series of $50,000$-queries experiments for a system with N=$1,000$ nodes, M=50 ($\approx 5\% \times N$) range guards, and $50,000$ tuples (the reported results are not sensitive to these values).

Performance Reference Points. We have compared the hop-count efficiency of the RG architecture against (i) plain Chord (PC), as a representative of traditional DHTs, (ii) an imaginary, enhanced Chord (EC), where for each range $\mathcal{R}$ the system knows the IDs of the n' nodes storing values in $\mathcal{R}a$, (iii) our OP-Chord architecture, and (iv) a hybrid system where 95% of queries are processed on the RG ring and the remaining 5% are dealt with on the OP-Chord ring (see sect. 4). Assume we have an integer range query $r = <v_{low}, v_{high}>$. Further assume that the requested index tuples are stored on n' nodes, under Chord's hashing scheme, and on k nodes, managed by k' RangeGuards, under our OPHF scheme. Then, in order to gather all possible results:

- PC: $|r|$ queries, or $O(|r| \log(N))$ hops, are needed.
- EC: $|n'|$ queries, or $O(|n'| \log(N))$ hops, must be executed.
- OP: we must first route to the node holding v_{low} and then follow k-1 successor pointers, for an $O(\log(N)+k)$ overall hop count.
- RG: we must route to the closest RG (1 hop), then to the RG holding v_{low} ($O(\log(M)$ hops) and follow $k' - 1$ successor pointers, for an $O(\log(M) + k')$ overall hop count.
- $OP + RG$: we flip a biased coin and choose among OP or RG processing, with the hop count complexities outlined above.

Fig. 3 summarizes the measured hop-counts per range query. Traditional DHTs were not designed with range queries in mind thus Chord performs poorly. The (unrealistically) enhanced Chord brings the required hop count down to ≈20% of the Chord figure. However, total global knowledge is required to implement this approach. On the other hand, by using the order-preserving hashing scheme and the RangeGuard architecture, the hop count is decreased by a factor of approx. 50 to 500 compared to PC,

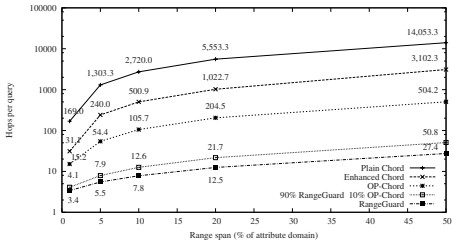


Fig. 3. Hop count per range query (log-plot)

10 to 110 compared to EC, and 5 to 20 compared to OP for different range spans, with the performance of $OP + RG$ following closely behind.

5.2 Load Distribution

The effect of random offsets and of overlapping multiple attributes in the access/storage load balancing is beneficial in our setting. To showcase this claim, we have performed the following experiment: assume we have an OP-Chord ring; we add nodes to the system, at random positions on the OP-Chord ring (simulating the quasi-uniform placement resulting from the use of SHA-1); we let the system stabilize and add $20,000$ multi-attribute tuples in the system. The values of the index attributes are drawn from the $[1,\ 40,000]$ integer interval according to a Zipf distribution with $\theta = 0.7$. If, on the other hand, we assume a uniform value occurrence distribution (as opposed to the above Zipfian distribution), the following results carry on to a Zipf load access distribution. We vary (i) the number of index attributes per tuple, from 1 (the classic single-attribute case of the currently available Chord system) up to 400 attributes, and (ii) the number of nodes in the network from $1,000$ to $5,000$, $10,000$, and $20,000$. Note that, e.g. in the $20,000$-node case, should these nodes be RG nodes, they would be enough to administer a $400,000$-node network, under the 5%-intuition described earlier.

Figure 4 shows the ratio of the highest to the lowest load in the system. Naturally, the optimal load ratio is 1, in which case all nodes in the system will have the same load. With a $\theta = 0.7$ Zipfian value occurrence distribution in an 1,000-node network, the highest-to-lowest single-attribute node access/storage load ratio load is 7.5, dropping to 1.97 for 8, and 1.06 for 400 attributes. We have noted on the figures the load ratio for the single-attribute case (denoted by the "load = " points) and the number of attributes required for this load to drop below 2 (denoted by the "# of attributes" points). With nodes being placed on the lower-level ring using Chord's SHA-1, we can expect RangeGuards to be uniformly distributed on the ring. Thus, the above situation holds for both the lower-level ring and the RangeGuard ring of our architecture. Note, however, the increase in the latter with the number of nodes in the network. As we expect RG nodes to be a small percentage of all network nodes, the above results show that, for the RG ring, load distribution will be within acceptable bounds (even without the other relevant mechanisms discussed in sect. 3). For much larger networks, we shall either need a very large number of attributes to achieve a good load distribution and/or more elaborate load balancing mechanisms[23].

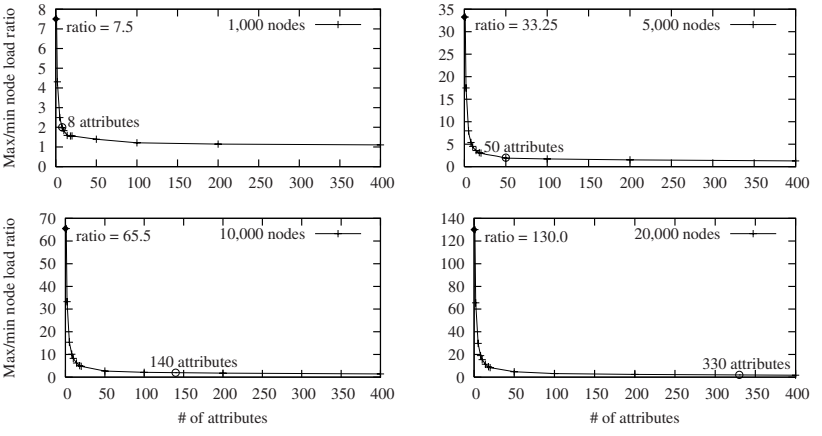


Fig. 4. Highest-to-lowest node load ratio

6 Related Work

All earlier research efforts focusing on complex query processing over DHTs [4,5,6,7,8,9,10,11,12,13,14,15] failed to recognize and exploit the key fact that the appropriate utilization of powerful nodes can speed up query processing significantly. Viewed from a complementary angle, earlier research failed to recognize that in large scale data sharing networks, there exist nodes which are weak, with respect to their processing, storage, and communication capacity, and that there also exist nodes with orders of magnitude more horsepower[17]. Our proposal avoids the pitfall of relying upon weak nodes for query processing. Furthermore, we follow a data management approach to discovering and harnessing powerful nodes: keeping metadata for participating nodes as a relation over the LP-DHT ring allows us to swiftly and efficiently locate such nodes and, by promoting them to RangeGuard status, to use them in the core of routing and query processing.

7 Conclusions

With this work we address the problem of efficient range query processing in structured P2P networks. Our approach leverages existing DHT-based P2P research. Our approach is centered on a new architecture that facilitates the exploitation of powerful nodes, coined RangeGuards, in the network, assigning to them specific tasks for further significant speedups during range query processing. This architecture is based on: (i) a way to efficiently identify and collect RangeGuards, and (ii) mechanisms to utilize them during range query processing. Our performance results have shown that significant savings can be achieved by the proposed architecture. Perhaps most importantly, a key advantage of the proposed architecture is that the dangers and inefficiencies of relying on weak nodes for range query processing, with respect to their processing, storage, and communication capacities, and their intermittent connectivity are avoided.

Acknowledgments

Peter Triantafillou was partly funded by FP6 of the EU through IST DELIS (#001907). Nikos Ntarmos was funded by the PENED 2003 Programme of the EU and the General Secretariat for Research and Technology of the Hellenic State.

References

1. Stoica, et al., I.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. ACM SIGCOMM. (2001)
2. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. ACM SIGCOMM. (2001)
3. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc. Middleware. (2001)
4. Harvey, N., Jones, M., Saroiu, S., Theimer, M., Wolman, A.: Skipnet: A scalable overlay network with practical locality properties. In: Proc. USITS. (2003)
5. Aspnes, J., Shah, G.: Skip Graphs. In: Proc. SODA. (2003)
6. Crainiceanu, A., Linga, P., Gehrke, J., Shanmugasundaram, J.: Querying peer-to-peer networks using P-trees. In: Proc. WebDB. (2004)
7. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: Proc. VLDB. (2004)
8. Ganesan, P., Yang, B., Garcia-Molina, H.: Multi-dimensional indexing in peer-to-peer systems. In: Proc. WebDB. (2004)
9. Gupta, A., Agrawal, D., Abbadi, A.: Approximate range selection queries in peer-to-peer systems. In: Proc. CIDR. (2003)
10. Sahin, O., Gupta, A., Agrawal, D., Abbadi, A.: Query processing over peer-to-peer data sharing systems. Technical Report UCSB/CSD-2002-28, UC Santa Barbara (2002)
11. Triantafillou, P., Pitoura, T.: Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In: Proc. DBISP2P. (2003)
12. Gribble, et al., S.: What Can Peer-to-Peer Do for Databases, and Vice Versa? In: Proc WebDB. (2001)
13. Huebsch, et al., R.: Querying the internet with PIER. In: Proc. VLDB. (2003)
14. Bharambe, A., Agrawal, M., Seshan, S.: Mercury: Supporting scalable multi-attribute range queries. In: Proc. SIGCOMM. (2004)
15. Andrzejak, A., Xu, Z.: Scalable, efficient range queries for grid information services. In: Proc. P2P. (2002)
16. Adar, E., Huberman, B.: Free Riding on Gnutella. First Monday (2000)
17. Saroiu, S., Gummadi, K., Gribble, S.: A measurement study of peer-to-peer file sharing systems. In: Proc. MMCN. (2002)
18. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proc. SIGCOMM. (2003)
19. Gupta, A., Liskov, B., Rodrigues, R.: One hop lookups for peer-to-peer overlays. In: Proc. HotOS IX. (2003)
20. Ntarmos, N., Triantafillou, P.: AESOP: Altruism-Endowed Self-Organizing Peers. In: Proc. DBISP2P. (2004)
21. Rao, et al., A.: Load balancing in structured P2P systems. In: Proc. IPTPS. (2003)
22. Karger, D., Ruhl, M.: New algorithms for load balancing in P2P systems. In: Proc. IPTPS. (2004)
23. Pitoura, T., Ntarmos, N., Triantafillou, P.: HotRod: Range query processing and load balancing in peer-to-peer data networks. Technical Report TR 2004/12/05, R.A. Computer Technology Institute (2004)

Guaranteeing Correctness of Lock-Free Range Queries over P2P Data*

Stacy Patterson, Divyakant Agrawal, and Amr El Abbadi

Department of Computer Science, University of California Santa Barbara
Santa Barbara, CA 93106, USA
{sep,agrawal,amr}@cs.ucsb.edu

Abstract. As P2P systems evolve into a platform for full-fledged distributed database management systems, the need arises for sophisticated query support and guarantees on query correctness. While there has been recent work addressing range queries in P2P systems, the work on query correctness is just beginning. Linga et al.[1] provided the first formal definition of correctness for range queries in P2P systems and described a lock-based range query technique that is provably correct. A natural question that arises is whether it is possible to develop a lock-free protocol that can meet the same guarantee of correctness. In this paper, we demonstrate the feasibility of lock-free correct protocols by first developing a simple, proof-of-concept query protocol and verifying that this protocol meets the correctness conditions. We then describe a more robust extended protocol and prove that for stable systems with only item insertions, item deletions, and item redistributions, this extension insures that every range query can be satisfied correctly.

1 Introduction

P2P systems provide the benefits of fault tolerance, load balancing, and scalability, making them a promising platform for distributed storage systems. Initial work on P2P systems focused on the development of distributed hash tables, or P2P indexes [2,3,4,5]. These P2P indexes allow for the storage of key/value pairs and the ability to do exact match search for an item using the item's key. In order to realize the full potential of P2P systems for distributed data management, these systems must provide support for more sophisticated query predicates. Recent work in the area of P2P range indexes enables efficient single and multi-dimensional range queries [6,7,8,9,10]. However, these systems do not provide any guarantees on the correctness of query results. Specifically, they do not guarantee that the query results include all and exactly those items that satisfy the range query predicate in the presence of item insertions, item deletions, and range redistributions. Such correctness guarantees are a necessary step in the evolution of P2P systems into full-fledged distributed database management systems.

* This research was funded in part by NSF grants IIS 02-23022, CNF 04-23336, and INT 00-95527.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 123–134, 2007.

Linga et al.[1] are the first to address correctness of range queries. This work provides a formal definition of correctness for range queries and describes a technique for range queries that is provably correct. The technique relies on locking to insure that no data items that satisfy the query predicate will be omitted from the result. A natural question that arises is whether it is possible to develop a lock-free technique that meets the same definition of correctness. In this paper, we demonstrate that a lock-free provably correct query protocol is feasible. We develop a simple, lock-free protocol in the context of P-Ring[7], the same P2P range index used in the lock-based approach. Our protocol returns only correct query results and rejects any query that cannot be satisfied correctly under the current system conditions. We also develop an extended protocol that greatly decreases the number of queries that will be rejected by the system. Specifically, in a stable ring where no new peers join and no peers fail or leave, this extension insures that every range query can be processed correctly without being rejected.

The remainder of the paper is organized as follows. In Section 2, we present the system model and provide background on P-Ring. In Section 3, we formalize the notion of query correctness. In Section 4, we describe a simple, correct range query protocol, and in Section 5, we describe a more robust extension to that protocol. Finally, we conclude in Section 6.

2 Background

2.1 System Model

The model we adopt is a generalization of many existing P2P systems. The system consists of a collection of peers, P, where a peer is a single processor that contributes some amount of storage space to be used by the system. Each peer, $p \in P$, has a unique physical identifier, such as an IP address. Each peer also has a unique logical identifier, denoted $p.id$, which is a key chosen from a discrete key space. We assume the existence of an underlying network layer that allows a peer to communicate with another peer using a direct communication channel. All messages are delivered within some known, bounded time-delay and message ordering is preserved within any given channel.

The system allows nodes to join and leave at any time. We assume a fail-stop model for peer departures. The system provides a $lookup(key)$ operation which locates, with high probability, the peer that is the immediate successor of the key. The system also provides operations for item insertion and item deletion. Additionally, the system supports single dimensional range queries of the forms $[lb, ub]$, $(lb, ub]$, $[lb, ub)$, and (lb, ub) where lb is the lower bound and ub is the upper bound of the range predicate.

2.2 P-Ring

We describe our techniques in the context of P-Ring, a P2P index framework that supports both range queries and exact match queries. Our techniques can be generalized to other range indexes that support single dimension range queries [6,8].

The P-Ring architecture is divided into several layers, each encapsulating specific functionality of the system. The foundation of a P-Ring system is a Chord ring [4] which provides the connectivity in the system. The Data Store layer sits on top of the Chord ring and is responsible for data item storage and load balancing operations. P-Ring also provides a Replication Manager that is responsible for maintaining replicas to improve item availability in the presence of peer failures. Finally, the P-Ring Content Router provides a routing structure that enables efficient *lookup*(*key*) operations. We encapsulate the functionality needed for correctness in the lower levels of the system. Our approach is therefore limited to changes in the Chord ring and Data Store layers.

Chord Ring. In the Chord ring, each peer, p, stores the identity (both physical and logical) of its predecessor, denoted $pred(p)$, and its successor, denoted $succ(p)$, in the ring. p also keeps a list of additional successors as a redundancy measure in case $succ(p)$ fails. Each peer is responsible for a portion of the key space which is the range $(pred(p).id, p.id]$. These values are the lower and upper bounds of the peer's range, which we also denote by $p.lb$ and $p.ub$ respectively. In a consistent ring $pred(p).ub = p.lb$ for all peers, p, in the system. We assume that when a new peer joins the ring, its predecessor and successor are notified and update their successor and predecessor pointers to reflect the existence of the new peer as part of the join process.

Data Store. One of the goals of a P2P storage system is to evenly distribute data items among all of the peers in the system. In many systems, load balancing is achieved through the use of consistent hashing [11]. For example, in CFS [12], peer IDs are generated using a hash function that insures with high probability that the IDs are uniformly distributed across the key space. Item IDs are also generated using a hash function, and items are stored at the peer whose ID immediately succeeds the item ID. This approach is effective in insuring that every peer is responsible for roughly the same number of items. However, a hash function does not generate IDs that preserve the order of the items.

In P2P range indexes, the item ID assignment policy must preserve item order so that range queries can be answered efficiently. To maintain item order, P-Ring uses the search key of the item as the item ID, denoted $i.skv$ where i is an item. With this policy, a range query of the form $[lb, ub]$ can be answered by first doing a *lookup*(*lb*) to locate the peer responsible for the lower bound of the query and then traversing along the Chord ring following successor pointers until the peer responsible for *ub* is reached.

Since the P-Ring ID assignment scheme does not guarantee uniform item distribution across peers, it is possible for a peer to become heavily loaded if a particular range contains too many items. It is also possible for a peer to be underloaded if its range is less popular. To address this issue, P-Ring provides three explicit load balancing operations, *split*, *merge*, and *redistribute*. If a peer becomes overloaded, it invites a new peer to join the ring and divides its range and the corresponding data items with the new peer through a *split* operation. If a peer's storage space is underutilized, it informs its successor through a *merge*

operation. The successor, in a *redistribute* operation, either gives part of its range and associated items to its predecessor or gives up its entire range to its predecessor and leaves the ring. With these load-balancing operations, P-Ring can guarantee that the number of items stored at each peer is between *sf* and *2sf* for some storage factor *sf*.

3 Query Correctness

We adopt the definition of *correct query results* given in [1]. This definition depends on the notion of a *history*, which describes the operations of the system and a partial ordering upon them. The partial order, $\leq$, is a "happened before" relationship such that for any two operations o_1 and o_2, we say that o_1 happened before o_2 if o_1 completed before o_2 began. If it is not the case that o_1 happened before o_2, it is possible the operations executed in parallel. The formal definition of a history is as follows.

Definition 1 (History). *History $\mathcal{H}$ is a pair $(O, \leq)$ where O is a set of operations and $\leq$ is a partial order defined on these operations.*

The definition of a correct query result also relies on the definition of a *truncated history*.

Definition 2 (Truncated History). *Given a history $\mathcal{H} = (O_{\mathcal{H}}, \leq_{\mathcal{H}})$ and an operation $o \in O_{\mathcal{H}}$, $\mathcal{H}_o = (O_{\mathcal{H}_o}, \leq_{\mathcal{H}_o})$ is a truncated history if $O_{\mathcal{H}_o} = \{o' \in O_{\mathcal{H}} | o' \leq_{\mathcal{H}} o\}$ and $\forall o_1, o_2 \in O_{\mathcal{H}_o}(o_1 \leq_{\mathcal{H}} o_2 \Rightarrow o_1 \leq_{\mathcal{H}_o} o_2)$.*

In other words, a truncated history is a history that contains all and only those operations that happened before a particular operation.

3.1 Correct Query Results

Intuitively, a correct query result will contain exactly those items in the P2P system that satisfy the range predicate. The dynamic nature of the system complicates the definition of what it means for an item to be "in the system". At a high level, an item is in the system, or *live*, if it has been inserted at some peer and not yet been deleted from any peer. We use the same terminology from [1] but adopt a slightly different definition of a live item. $insertItem(i)$ denotes the successful insertion of item i into the system. $deleteItem(i)$ denotes the successful deletion of the item i from the system. We use $items_{\mathcal{H}}(p)$ to denote the collection of items stored at a peer p.

Definition 3 (Live Item). *An item i is live in a history $\mathcal{H}$, denoted $live_{\mathcal{H}}(i)$, iff $(insertItem(i) \in \mathcal{H}) \wedge (deleteItem(i) \notin \mathcal{H})$.*

Additionally, in P2P systems, we must address the issue of node failures with regards to the items stored at a failed node. When a node fails, its successor in the Chord ring assumes responsibility for the node's range, but only once the successor becomes aware of the failure. We consider this detection of failure to

be an implicit delete of every item stored at the failed peer. So, it is only after the failure is detected that these items are no longer live.

We now state the definition of a correct query result from [1]. In the definition, $satisfies_{\mathcal{Q}}(i)$ denotes whether item i satisfies the range predicate of query $\mathcal{Q}$.

Definition 4 (Correct Query Result). *Given a history $\mathcal{H}$, a set R of items is a correct query result for a query Q initiated with operation o_s and successfully completed with operation o_e iff the following two conditions hold:*

1. $\forall i \in R$ ($satisfies_Q(i) \wedge \exists o \in O_{\mathcal{H}}(o_s \leq_{\mathcal{H}} o \leq_{\mathcal{H}} o_e \wedge live_{\mathcal{H}_o}(i))$)

2. $\forall i$ ($satisfies_Q(i) \wedge \forall o \in O_{\mathcal{H}}(o_s \leq_{\mathcal{H}} o \leq_{\mathcal{H}} o_e \wedge live_{\mathcal{H}_o}(i)) \Rightarrow i \in R$)

The first condition states that if an item i is included in the query result, then it was live at some time during the query. The second condition states that every item that was live for the entire duration of the query is included in the result.

The goal of this paper is to explore lock-free techniques for range queries that produce correct query results according to the above definition.

3.2 Incorrect Query Results: Examples

In [1], the authors present two examples of how a naive approach to range query execution that consists of simply traversing along the ring will give incorrect query results. We summarize these examples here. In Section 4.3 we show how our lock-free implementation disallows these incorrect scenarios.

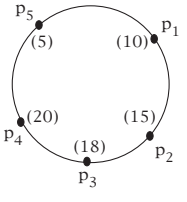


Fig. 1. Example P-Ring

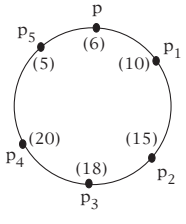


Fig. 2. P-Ring with new node

Inconsistent Successor Pointers. Consider the ring shown in Figure 1 and suppose each peer maintains a successor list of size 2. The successor list for p_4 is $\{p_5, p_1\}$. p_5 is responsible for the range $(20, 5]$ and p_1 is responsible for the range $(5, 10]$. Suppose that p_1 becomes overloaded and splits its range with a new peer p, as shown in Figure 2. When p joins, it becomes responsible for the range $(5, 6]$ and p_1 becomes responsible for the range $(6, 10]$. A query with range predicate $(20, 9]$ arrives at p_4, and p_4 responds to the query with the data items in the range $(20, 5]$. Then, p_5 fails. p_4 tries to forward the query to p_5 and detecting the failure, forwards the query to p_1. Note that p_4 has not yet updated its successor list to reflect the existence of p. p_1 will respond to the query with items in the range $(6, 9]$ and the range $(5, 6]$ will have been omitted from the query result.

Concurrent Redistribution. In the second scenario, we see that it is possible for a query to produce incorrect results even if the successor pointers are completely consistent. Consider again the ring in Figure 1. Suppose a query $(10, 18]$ arrives at p_2. p_2 returns the items in the range $(10, 15]$ and forwards the query to its successor, p_3. Suppose, at the same time, p_3 is in the process of redistributing part of its range with p_2 and has transferred the range $(15, 16]$ to p_2. When the query arrives at p_3, p_3 will return items in the range $(16, 18]$ and the items in $(15, 16]$ will be missing from the query result.

In both cases, the incorrect query results stem from the fact that there is some degree of uncertainty about the range for which a peer is responsible. If we can eliminate this uncertainty by clearly defining the range for which a peer can safely answer queries, then we can use this to produce a correct query protocol.

4 A Simple Protocol

First we examine how to remove any uncertainty in range responsibility that may be introduced through the *split*, *merge*, and *redistribute* operations. Then, we present a correct, lock-free range query protocol.

4.1 Range Ownership

When a peer performs a *split*, *merge*, or *redistribute* operation, its range changes. It is this change that is problematic for range queries. The first step to query correctness is to clarify range responsibility when these operations are performed. To do this, certain steps of each operation must be performed atomically. We outline these atomic steps for the *merge* operation. The atomicity requirements for *split* and *redistribute* are defined similarly.

When peer p experiences underflow, it invokes the *merge* operation to request that its successor relinquish some or all of its range to p. The result is that p increases the range it is responsible for and receives the data items associated with the range addition. The *merge* operation is given in Algorithm 1. In line 2, p alerts its successor of the underflow and waits for the successor's response. In line 4, p updates its set of items to include the items that were given up by its successor. In line 5, p updates its range to include the range relinquished

Algorithm 1. $p.merge()$

1. // send message to successor and wait for result
2. $(newRange, newItemList) = succ(p).initiateMerge(p, |p.range|)$;
3. // execute next two steps atomically
4. $p.list.add(newItemList)$;
5. $p.range.add(newRange)$;

by its successor. By executing steps 4 and 5 atomically, p insures that it will only process operations (item insertions, item deletions, lookups, and range queries) for the new range once it has incorporated all live items in the range into its Data Store.

4.2 Correct Range Queries

The simple query protocol is shown in Algorithm 2 and Algorithm 3. This is a slight modification of the original P-Ring protocol[1] without locking. The query begins with a *lookup*(lb) operation. *rangeQuery* is then invoked at the peer responsible for lb. *processQuery* is invoked at each subsequent peer that participates in the query ending at the peer responsible for ub. We assume for simplicity that *processQuery* and *rangeQuery* are each executed atomically. Rather than passing the same lower bound to its successor when forwarding the query along the ring, each peer p sends $p.ub$ as the lower bound. By doing so, p informs its successor of what part of the range query remains to be answered. The successor accepts the query only if it is able to exactly satisfy this lower bound, thus eliminating the possibility of gaps or duplicates in the query results.

We assume that during any query execution there are only a finite number of new peers entering the system. Therefore, every query terminates at some time.

Theorem 1. *Using the simple protocol, every query that is accepted (terminates with no peer rejecting it) produces a correct query result.*

Proof. Consider a query $Q = [lb, ub]$ that begins with operation o_s and ends successfully with operation o_e. The operations *rangeQuery* and *processQuery* insure that, if the query terminates without rejection, the intervals r satisfied at each of the peers that participate in the query are non-overlapping. The union of these intervals is exactly equal to $[lb, ub]$.

Since the union of the intervals satisfied at each peer equals $[lb, ub]$, $\forall i \in R\ satisfies_Q(i)$ holds. Let i be any item such that $i \in R$ and $satisfies_Q(i)$ holds. It must be the case that i was returned by some peer in line 6 of *processQuery* or *rangeQuery*. If we call this invocation of *processQuery* (or *rangeQuery*) operation o, then we have $insertItem(i) \leq_{\mathcal{H}} o$, and if i was deleted in this history, $o \leq_{\mathcal{H}} deleteItem(i)$. So, there exists an operation o such that, $o_s \leq o \leq o_e$ and also such that $live_{\mathcal{H}_o}(i)$. Therefore, Condition 1 of the the Correct Query Result definition holds.

Consider i such that $satisfies_Q(i)$ is true and $\forall o \in O_{\mathcal{H}}(o_s \leq_{\mathcal{H}} o \leq_{\mathcal{H}} o_e \wedge live_{\mathcal{H}_o}(i))$ holds. We claim that there must be some peer p such that $i \in p.range$ for the operation $p.processQuery$ (or $p.rangeQuery$). Suppose this is not the case. Then, for some peers p_1 and p_2 where p_2 is the successor of p_1, we have $i.skv > p_1.ub$ during the invocation of $p_1.processQuery$ (or $p_1.rangeQuery$) and $i.skv < p_2.lb$ during the invocation of $p_2.processQuery$. In this situation,

p_2 would detect the discontinuity in the range satisfied by p_1 and its own range and would reject the query at line 2 of *processQuery*. Since the query terminates successfully, it must be the case that there exists a peer p such that $i \in p.range$ for the operation $p.processQuery$. Therefore $i \in R$. This proves Condition 2 of the Correct Query Result definition. □

Algorithm 2. $p.rangeQuery(lb, ub, initiator)$

```
1. if lb ∉ p.range then
2.    Reject query
3. else
4.    r := (lb, ub] ∩ p.range [1]
5.    items := items in p.items that are in range r
6.    Send items to initiator
7.    if ub ∉ p.range then
8.       Invoke succ(p).processQuery(p.ub, ub, initiator) asynchronously
9.    end if
10. end if
```

Algorithm 3. $p.processQuery(lb, ub, initiator)$

```
1. if lb ≠ p.lb then
2.    Reject query
3. else
4.    r := (lb, ub] ∩ p.range [1]
5.    items := items in p.items that are in range r
6.    Send items to initiator
7.    if ub ∉ p.range then
8.       Invoke succ(p).processQuery(p.ub, ub, initiator) asynchronously
9.    end if
10. end if
```

4.3 Incorrect Query Results Revisited

For both examples of incorrect range query results given in Section 3, if the simple protocol is used, some peer will reject the query. In the case of the inconsistent successor pointers, p_4 will forward the query with range $[5, 10]$ to p_1. p_1 will then reject the query because the lower bound of the range query, 5, does not equal $p_1.lb$, which is 6. For the case of concurrent range redistribution, p_2 will forward the query [15, 18] to p_3. p_3 will reject the query because the lower bound of the query, 15, does not equal its lower bound, 16.

[1] The items may be returned in parallel with the forwarding of the query to $succ(p)$.

5 Extension to Simple Protocol

One drawback of the scheme described above is that the system cannot satisfy a range query if that range encompasses a portion of the index that is concurrently in transit from one peer to another as a result of a redistribution. This issue exists even if there are no topology changes in the system. In this section, we describe an extension to the simple technique that overcomes this drawback. We show that in a system with no topology changes, the extended protocol can satisfy all queries correctly.

Suppose that peer p_2 relinquishes part of its range to its predecessor p_1 as a result of a *redistribute* operation. Instead of deleting the items that p_2 has given up to p_1, p_2 marks the items as *relinquished* and keeps them for some period of time. What can p_2 safely do with the relinquished part of its range? It cannot accept any inserts or deletes for this range because it is no longer the owner of the range and cannot guarantee that the future owner of the range, p_1, can consistently incorporate these insert and delete operations. p_2 also cannot accept *lookup*(*key*) operations for the relinquished range because it may report that no item exists for a given key even if an insert for this key has been successfully completed at p_1. Similarly, p_2 may return an item in response to a lookup after that item has been successfully deleted from p_1. p_2 can, however, use this relinquished range to satisfy range queries that have been forwarded from p_1. If p_1 forwards a query to p_2 with a lower bound equal to the lower bound of the relinquished range, then p_1 has not yet assumed responsibility for this range, and therefore no new items have been inserted into nor have any items been removed from the range. So, p_2 can return items in the relinquished range with no risk of omitting any live items and no risk of returning any deleted items.

The extended query protocol that uses this approach is given below. As in the simple protocol, the query begins with a *lookup*(*lb*) operation. *rangeQuery* in Algorithm 2 is invoked at the peer responsible for *lb*. The *processQuery* algorithm given in Algorithm 4 is invoked by each subsequent peer that participates in the query. The relinquished range is denoted by $p.range^* = (p.lb^*, p.ub^*]$. The set of items in $p.range^*$ is denoted $p.items^*$.

The question arises as to how long p_2 needs to keep the relinquished range and items. p_2 can delete the relinquished range and items once it knows that p_1 has received them. This confirmation can come in the form of an explicit acknowledgment message from p_1. Additionally, if p_2 receives a query forwarded by p_1 with the lower bound equal to $p_2.lb$ and not $p_2.lb*$, p_2 no longer needs to store the relinquished range. Finally, since p_2 is storing the relinquished range to answer queries on behalf of p_1, if p_2 detects that p_1 has failed or if p_2 is notified that it has a new predecessor, it no longer has any use for the relinquished range. Note that p_2 can delete the relinquished items at any time, and after the deletion, query processing becomes identical to the simple protocol.

We now prove the correctness of the extended protocol.

Algorithm 4. *p.processQuery*$(lb, ub, initiator)$

```
1. if lb ≠ p.lb and lb ≠ p.lb* then
2.    Reject query
3. else
4.    if lb = p.lb* then
5.       r* := (lb, ub] ∩ p.range*
6.       items := items in p.items* that lie in range r*
7.       lb := p.ub*
8.    end if
9.    if ub > p.lb then
10.      r := (lb, ub] ∩ p.range
11.      items := items∪ items in p.items that lie in range r
12.   end if
13.   Send items to initiator[2]
14.   if ub ≥ p.ub then
15.      Invoke succ(p).processQuery(p.ub, ub, initiator) asynchronously
16.   end if
17. end if
```

Theorem 2. *Using the extended protocol, every query that terminates successfully without being rejected produces a correct query result.*

Proof Sketch. The extended protocol is identical to the simple protocol except for the case where a range redistribution between a peer p and its predecessor $pred(p)$ takes place concurrent with the processing of a range query at $pred(p)$ and p.

Consider a query $Q = [lb, ub]$ that begins with operation o_s and ends successfully with operation o_e. As in the simple protocol, the operations *rangeQuery* and *processQuery* insure that, if the query terminates without rejection, the intervals $r \cup r^*$ satisfied at each of the peers that participate in the query are non-overlapping. The union of these intervals is exactly equal to $[lb, ub]$. Therefore, $\forall i \in R\ satisfies_Q(i)$ is true.

Let i be any item returned by peer p (i.e. $i \in R$ and $satisfies_Q(i)$ holds). i was either in $p.range$ or in $p.range^*$ during the execution of $p.processQuery$ (or $p.rangeQuery$). If $i \in p.range$, then Condition 1 for a Correct Query Result holds by the same argument given in the proof of Theorem 1. If $i \in p.range^*$ then $insertItem(i)$ happened before p sent i to $pred(p)$ in a *redistribute* operation. And $pred(p).processQuery$ happened before $pred(p)$ received items from the *redistribute* (otherwise, $pred(p)$ would have forwarded the query with lb equal to $p.lb$). So, $pred(p).processQuery \leq_{\mathcal{H}} redistribute \leq_{\mathcal{H}} p.processQuery$. If i was deleted, then $pred(p)$ must have received p's relinquished range before the $deleteItem(i)$ operation so $insertItem(i) \leq_{\mathcal{H}} redistribute \leq_{\mathcal{H}} deleteItem(i)$.

[2] As in the simple protocol, the items may be returned in parallel with the forwarding of the query to $succ(p)$.

Taking $o' = redistribute$, $live_{\mathcal{H}_{o'}}(i)$ is true, and therefore Condition 1 for a Correct Query Result holds.

The proof of Condition 2 is similar to that given in the proof of the simple protocol. □

This extended protocol greatly increases the system's ability to satisfy queries. In fact, in a system with no topology changes, the extended protocol can satisfy all queries correctly. We prove this by first showing that in a stable ring every *lookup*(*key*) will eventually complete successfully . Therefore, for a range query $Q = [lb, ub]$, it is always possible to locate the peer responsible for lb. We then show that if a peer keeps the relinquished range until it receives confirmation that the range has been assumed by its predecessor, no query will ever be rejected.

Lemma 1. *In a stable ring with consistent successor pointers having only item insertions, item deletions, and range redistributions every lookup*(*key*) *will eventually terminate successfully.*

Proof. Suppose that due to outdated routing information, a *lookup*(*key*) message is forwarded to a node p after p has transferred the range containing *key* to its predecessor. When p receives the lookup request, it detects that it is not the owner of *key* and forwards the request either to its successor or to some other peer in its routing structure. It can be shown that eventually some peer will be permanently responsible for the range containing *key*. At that time, the lookup must terminate successfully. In any redistribution, a node p with range $(p.lb, p.ub]$ transfers some portion of that range, $(lb, new\_lb]$, to its predecessor, and p becomes responsible for the range $(new\_lb, ub]$. Since the key space is discrete, eventually there is some node that is responsible for the range $(ub-\delta, ub]$ with $key \in (ub - \delta]$ such that no further subdivision of the range $(ub - \delta, ub]$ is possible. In this case, the only way for the peer to give up responsibility for this range is to leave the system. This is impossible under the assumption that the system topology is stable. □

Theorem 3. *In a stable ring with consistent successor pointers having only item insertions, item deletions, and range redistributions, every query can be answered correctly using the extended protocol.*

Proof. Theorem 2 shows that any query that is not rejected produces a correct query result. Here we show that in a stable ring, no query will ever be rejected. By Lemma 1, every *lookup*(lb) eventually arrives at the peer responsible for the lb. The first step in a range query with range predicate $[lb, ub]$ is a *lookup*(lb) operation and therefore in *p.rangeQuery* the *Reject* statement at line 2 will never be invoked. A query will be rejected at line 2 in *p.processQuery* if lb is not equal to either the lower bound of *p.range* or the lower bound of $p.range^*$. In the case that there is a concurrent redistribution for which $pred(p)$ has not yet received the new range and items, lb will be equal to $p.lb^*$. In all other cases, any query forwarded by $pred(p)$ will be equal to $p.lb$. So, if a query is rejected at line 2, then it must have been forwarded to p by some peer other than $pred(p)$.

This can only occur if that peer has an inconsistent successor pointer, which is impossible under the assumption of the theorem. □

6 Conclusion

In this paper, we have presented two lock-free techniques for range queries in P2P systems that provably guarantee correct query results, a simple protocol and a more robust extension. Both techniques have the benefit of simplicity in analysis and implementation. Additionally, in a system with no topology changes, the extended technique can satisfy every range query correctly. This work represents an initial step towards better formalization and understanding of P2P systems.

References

1. Linga, P., Crainiceanu, A., Gehrke, J., Shanmugasundaram, J.: Guaranteeing correctness and availability in p2p range indicies. In: SIGMOD. (2005) 323–334
2. Ratnasamy, S., Francis, P., Handley, M., Karp, R.M., Shenker, S.: A scalable content-addressable network. In: SIGCOMM. (2001) 161–172
3. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware. (2001) 329–350
4. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM. (2001) 149–160
5. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications **Vol. 22, No. 1** (2004) 41–53
6. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM. (2004) 353–366
7. Crainiceanu, A., Linga, P., Machanavajjhala, A., Gehrke, J., Shanmugasundaram, J.: P-ring: An index structure for peer-to-peer systems. Cornell University Technical Report (2004)
8. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: VLDB. (2004) 444–455
9. Gupta, A., Agrawal, D., El Abbadi, A.: Approximate range selection queries in peer-to-peer systems. In: CIDR. (2003) 141–151
10. Sahin, O.D., Gupta, A., Agrawal, D., El Abbadi, A.: A peer-to-peer framework for caching range queries. In: ICDE. (2004) 165–176
11. Karger, D., Lehman, E., Leighton, T., Levine, M., Lewin, D., Panigrahy, R.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: ACM Symposium on Theory of Computing. (1997) 654–663
12. Dabek, F., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: SOSP '01. (2001) 202–215

Publish/Subscribe with RDF Data over Large Structured Overlay Networks*

Erietta Liarou, Stratos Idreos, and Manolis Koubarakis

Department of Electronic and Computer Engineering
Technical University of Crete, GR73100 Chania, Greece
{erietta,sidraios,manolis}@intelligence.tuc.gr

Abstract. We study the problem of evaluating RDF queries over structured overlay networks. We consider the publish/subscribe scenario where nodes subscribe with long-standing queries and receive notifications whenever triples matching their queries are inserted in the network. In this paper we focus on conjunctive multi-predicate queries. We demonstrate that these queries are useful in various modern applications e.g., distributed digital libraries or Grid resource discovery. Conjunctive multi-predicate queries are hard to answer since multiple triples are necessary for their evaluation, and these triples will usually be inserted in the network asynchronously. We present and evaluate query processing algorithms that are scalable and distribute the query processing load evenly.

1 Introduction

Evaluating RDF queries in distributed environments is an open research problem. Semantic Web research can gain a lot from recent developments in the area of peer-to-peer (P2P) systems, and especially from results in the area of structured overlay networks. We discuss RDF query processing over a popular kind of such networks, called distributed hash tables (DHTs) [1]. DHT protocols allow nodes holding data items to self-organize and offer data lookup functionality in a provably efficient, scalable, fault-tolerant and adaptive way.

The problem of designing distributed algorithms to evaluate RDF queries over structured overlay networks has been considered by many papers so far e.g., [2,3,4,5,6,7]. We can distinguish two scenarios for query processing in these papers. In the *one-time query* case [5,2,3], a user poses a query like "give me all songs by Leonard Cohen" and the system replies with a set of answers/pointers to nodes that hold related resources. In the *publish/subscribe scenario* [4,2,8], a user subscribes with a continuous query like "notify me when a new song of Leonard Cohen becomes available" and receives notifications when matching resources become available.

We consider RDF queries in the style of RDQL [9] using triples as the atomic construct. Our long term research goal is to create a set of algorithms for the publich/subscribe scenario that will support all useful query types in languages such

* This work was supported in part by the European Commission project Ontogrid (http://www.ontogrid.net/).

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 135–146, 2007.

as RDQL and RQL. In this paper we make a first step towards this direction by providing a set of algorithms that support the class of *conjunctive multi-predicate queries* and demonstrate that queries of this class are useful in applications. Conjunctive multi-predicate queries over a distributed DHT environment were first considered in [2] where algorithms for one-time query processing scenarios are proposed. Here we consider the publish/subscribe scenario for such queries over DHTs.

The contributions of this paper are the following. We propose two distributed algorithms for evaluating continuous conjunctive multi-predicate queries on top of DHTs. In our experiments we use Chord [10] as the underlying DHT due to its relative simplicity and widespread popularity. However, the implementation of our ideas which is underway, is *DHT-agnostic:* it will work with any DHT extended with the APIs we define. The case of conjunctive multi-predicate queries is an interesting one since more than one triples may be needed to answer a query. Since typically triples do not arrive in the network at the same time, the network should "remember" the queries that have been partially satisfied and create notifications only when all subqueries of a given query are satisfied. We introduce the notion of *query chains* to handle this problem. We argue that our algorithms are appropriate for *large networks* since their main emphasis is to *distribute the query processing load* to as many nodes as possible, while at the same time keeping the *network cost* in terms of overlay hops low. We experimentally evaluate and compare our algorithms in a simulated environment.

The organization of the paper is as follows. Section 2 describes Chord and our assumptions regarding the system and data model. Sections 3, 4 and 5 describe the two query processing algorithms. In Section 6 we experimentally evaluate the performance of our algorithms. Finally, Section 7 concludes the paper.

2 System Model and Data Model

We assume an overlay network where all nodes are *equal*, as they run the same software and have the same rights and responsibilities. Each node n has a unique key (e.g., its public key), denoted by $key(n)$. Nodes are organized according to the Chord protocol and are assumed to have synchronized clocks. This property is necessary for the time semantics we describe later on in this section. In practice, nodes will run a protocol such as NTP [11] and achieve accuracies within few milliseconds. Each data item i has a unique key, denoted by $key(i)$. Chord uses consistent hashing to map keys to identifiers. Each node and item is assigned an m-bit identifier, that should be large enough to avoid collisions. A cryptographic hash function, such as SHA-1 or MD5 is used: function $Hash(k)$ returns the m-bit identifier of key k. The identifier of a node n is denoted as $id(n)$ and is computed as follows: $id(n) = Hash(key(n))$. Similarly the identifier of an item i is denoted as $id(i)$ and is computed as follows: $id(i) = Hash(key(i))$. Identifiers are ordered in an *identifier circle (ring)* modulo 2^m i.e., from 0 to $2^m - 1$. Key k is assigned to the first node which is equal or follows $Hash(k)$ clockwise in the identifier space. This node is called the *successor* node of identifier $Hash(k)$ and

is denoted by $Successor(Hash(k))$. We will often say that this node is *responsible* for key k. A query for locating the node responsible for a key k can be done in $O(\log N)$ steps with high probability [10], where N is the number of nodes in the network. Chord is described in more detail in [10].

We use the API defined in [12,13] for implementing pub/sub functionality on top of Chord. [12] deals with languages from Information Retrieval while [13] with two-way equi-join queries and there is no overlap of [12,13] with the algorithms of this paper. Let us now shortly describe this API. Function *send* (msg, id), where msg is a message and id is an identifier, delivers msg from any node to node $Successor(id)$ in $O(logN)$ hops. Moreover, function *multiSend*(msg, I), where I is a set of $d > 1$ identifiers $I_1, ..., I_d$ delivers msg to nodes $n_1, n_2, ..., n_d$ such that $n_j = Successor(I_j)$, where $1 < j \leq d$. This happens in $d * O(logN)$ hops. Function *multiSend()* can also be used as, *multiSend*(M, I), where M is a set of d messages and I is a set of d identifiers. For each I_j, message M_j is delivered to $Successor(I_j)$ in $d*O(logN)$ hops. A detailed description and evaluation of this API can be found in [12].

In the application scenarios we target, each network node is able to describe in RDF the resources that it wants to make available to the rest of the network, by creating and inserting metadata in the form of *triples*. In addition, each node can *subscribe with a continuous query* that describes information that this node wants to receive *notifications* for. We use a very simple concept of schema equivalent to the notion of a namespace. Thus, we do not deal with RDFS and the associated simple reasoning about classes and instances. Different schemas can co-exist but we do not support schema mappings. Each node uses some of the available schemas for its descriptions and queries.

Each triple t has a time parameter called *published time*, denoted by $pubT(t)$, that represents the time that the triple is inserted into the network. Each query q has a unique key, denoted as $key(q)$, that is created by concatenating an increasing number to the key of the node that posed q. Each query q has a time parameter, called *subscription time*, denoted by $subscrT(q)$ that represents its creation time. Each subquery q_i of a query q is also assigned a subscription time $subscrT(q_i) = subscrT(q)$. A triple t can satisfy q_i iff $subscrT(q_i) \leq pubT(t)$, i.e., only triples that are inserted after a continuous query was subscribed can satisfy it. We will not have a complicated formal definition of notification as it might be appropriate for some applications.

We concentrate on the class of conjunctive multi-predicate queries. A conjunctive multi-predicate query q is a formula in the following form:

$$?x_1, \ldots, ?x_n : (?s, p_1, o_1) \wedge (?s, p_2, o_2) \wedge \cdots \wedge (?s, p_n, o_n)$$

where $?s$ is a variable, $p_1, \ldots, p_n$ are URIs and $o_1, \ldots, o_n$ are variables, URIs or literals. $?x_1, \ldots, ?x_n$ are variables and $\{x_1, \ldots, x_n\} \subseteq \{s, o_1, \ldots, o_m\}$. Variables will always start with the '?' character as in [2]. The formulas $(?s, p_1, o_1)$,...,$(?s, p_n, o_n)$ will be called *subqueries* of q. A query will be called *atomic* if it consists of a single conjunct.

A *substitution* θ is a finite set of the form $\{?v_1/c_1, \ldots, ?v_n/c_n\}$ where each $?v_i$ is a distinct variable and each c_i is a URI or literal. Each constant c_i is

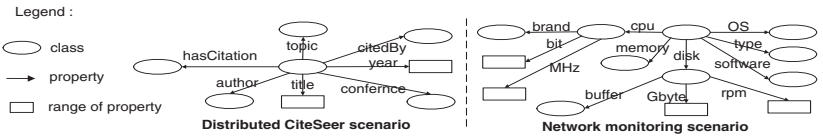


Fig. 1. Possible schemas for example applications

called a *binding* for $?v_i$. Note that we deal only with *ground* substitutions. Let q be a query and θ a variable substitution. Then $q\theta$ will denote the result of substituting each variable of q with its binding in θ.

A set of triples $T = \{t_1, \ldots, t_n\}$ *satisfies* a query $q = q_1 \wedge \ldots \wedge q_k$ with variable substitution θ, if for each $i = 1, \ldots, k$ there exists j, $1 \leq j \leq n$ such that triple t_j satisfies q_i with θ (i.e., $q_i\theta = t_j$) and $subscrT(q_i) \leq pubT(t_j)$. A triple t *satisfies* an atomic query q with variable substitution σ, if $q\sigma = t$ and $subscrT(q) \leq pubT(t)$.

Let T be an RDF database and q be a query in the above form. A substitution θ in variables $?x_1, ..., ?x_n$ is an *answer* to q if T satisfies q. A notification corresponding to a query q of the above form is just a substitution θ which is an answer to q.

Conjunctive multi-predicate queries over a distributed DHT environment were first considered in [2] for one-time query processing scenarios. Note that this class of queries allows join *only* on s (i.e., s is a subject *common to all* triples). Such queries can be used to express many interesting queries for P2P applications using RDF. For example, assume a distributed digital library that provides functionalities like those of CiteSeer. Library nodes could publish descriptions of academic literature in electronic format. The schema of the left graph of Figure 1 can be part of the schema used in such an application. Nodes can also subscribe with queries looking for publications with specific characteristics. A possible query could be: "Notify me when a paper by Smith is published that is related to P2P networks. List all citations in this paper". This is a conjunctive multi-predicate query that can possibly be expressed as follows:

$$?x, ?y : (?x, author, \text{“}Smith\text{”}) \wedge (?x, topic, \text{“}P2P\text{”}) \wedge (?x, citation, ?y)$$

It is well-known from systems such as EDUTELLA [5] that RDF is nicely suited for capturing digital library resource metadata. The fact that resource metadata may enter the network asychronously makes continuous query evaluation an incremental long-running activity (see Sections 3, 4 and 5). In reality, there will be applications where the metadata *about a specific resource* are all inserted in the network at *the same time* and applications where metadata are inserted in *steps*. For example, a digital library such as the ACM Digital Library might be expected to publish all metadata of a specific document (e.g., author, title, etc.) simultaneously. On the contrary, in the CiteSeer scenario, the system continuously crawls the web and collects information on Computer Science publications. In this case, as more details about a specific publication are created, previous CiteSeer entries will be updated.

Another example application where the class of queries studied is useful is Grid resource monitoring. In this application where computational resources (e.g., mainframes, personal computers, mobile devices, etc.) are connected in an overlay network. Users of this network would like to use cpu, memory, disk and other resources available in the overlay to carry out various computation- and data-intensive tasks. Part of the schema used in such a scenario could be the right graph of Figure 1. A continuous conjunctive multi-predicate query according to this schema might be "Notify me whenever a PC running Linux with the BLAST bioinformatics package installed, becomes available". This query can be expressed as follows:

$$?x : (?x, type, PC) \wedge (?x, OS, Linux) \wedge (?x, software, BLAST)$$

Similarly with CiteSeer, evaluating continuous queries for resource discovery in Grid environments needs data that might not be inserted in the system at the same time. Thus, algorithms have to "remember" previously inserted triples that partially satisfy a query. As new triples arrive, this memorized information is used to determine what queries have been fully satisfied. In general, applications where metadata is incrementally refined and updated seem to be prevalent in the Semantic Web and the Semantic Grid and can be nicely served by semi-structured data models like RDF and dynamic P2P networks.

3 A High-Level View of Our Algorithms

In our algorithms, when a continuous query is submitted, it is *indexed* somewhere in the network and waits for triples to satisfy it. Each time a new triple is inserted, the network nodes cooperate to determine what queries are satisfied and create notifications. The case of conjunctive multi-predicate queries is an interesting one, since a single triple may *satisfy* a query q only *partially* by satisfying a subquery of q. In other words, more than one triples may be needed to answer a query. Moreover, since the appropriate triples do not necessarily arrive in the network at the same time, the network should "remember" the queries that have been partially satisfied in the past (e.g., by keeping intermediate results) and create notifications only when all subqueries of a given query are satisfied.

We could index queries to a globally known node or set of nodes, but this would eventually overload these nodes. In a P2P environment we want as many nodes as possible to contribute some of their resources (storage, cpu, bandwidth, etc.) for achieving the overall network functionality. The resource contribution of each node will obviously depend on its capabilities, its gains from participating in the network, etc. In this paper we make the simplifying assumption that all nodes are altruistic, with equivalent capabilities, and, thus, can contribute to query evaluation in identical ways.

Let us first consider an atomic query $q = (?s_1, p_1, ?o_1)$. We can simply assign q to the successor node x of $Hash(p_1)$ by using the constant part p_1 of the query. Triples that have predicate value equal to p_1 will be indexed to x too, where they will meet q. Assume now the atomic query $q' = (?s_2, p_2, o_2)$. We can index q' either to node $x_1 = Successor(Hash(p_2))$ or to node $x_2 = Successor(Hash(o_2))$.

We prefer the second option since intuitively there will be more object values than predicate values in an instance of a given schema, which will allow us to distribute queries to a greater number of nodes. Another solution is to index q' to the node $x_3 = Successor(Hash(p_2 + o_2))$. We use the operator + to denote the *concatenation* of string values. This is the best option because the possible combinations of predicate and object values will be greater than the number of object values alone, so this will lead to an even better distribution of queries.

The difficulty with arbitrary conjunctive multi-predicate queries is that they demand more than one conditions to be satisfied before the whole query can be satisfied. As an example, consider the query $q = q_1 \wedge q_2 \wedge q_3$. Our approach is to *split* the query to the subqueries that it consists of, and to index each subquery *separately*. Then, three usually different nodes will be responsible for query processing regarding q. Each one will be responsible for a single subquery of q, e.g., nodes r_1, r_2 and r_3 will be responsible for q_1, q_2 and q_3 respectively. These nodes will form the *query chain* of q, denoted by $chain(q)$. Each one of these nodes will monitor the satisfaction of only the subquery that it is responsible for. To determine the satisfaction of q, we have to allow some kind of communication between these three nodes. In this way, as triples arrive and satisfy a subquery e.g., in node r_1, r_1 will forward *partial results* of q to r_2. Node r_2 will forward partial results that also satisfy the second subquery to r_3 and r_3 will realize that the whole query is satisfied and create a notification.

The first algorithm that we present creates a single query chain for each conjunctive multi-predicate query while the second one creates multiple query chains for a single query to achieve a better query processing load distribution. The presented algorithms are useful for the evaluation of conjunctive multi-predicate queries. However, the general idea of these algorithms is the base of our research towards the creation of distributed algorithms that will also support more general query types, e.g., arbitrary queries. In the following sections we describe our algorithms in detail.

4 The Single Query Chain Algorithm

In this section we introduce the *single query chain* algorithm (SQC). The main characteristic of this algorithm is that for *each* query, it creates a *single* query chain. Let us assume a node n that wants to subscribe with the query $q = q_1 \wedge q_2 \wedge ... \wedge q_k$ where each subquery q_j is of the form $(?x, p_j, ?o_j)$ or $(?x, p_j, o_j)$. We will use functions $subj(q_j)$, $pred(q_j)$ and $obj(q_j)$ to denote the string value of the subject, the predicate and the object of subquery q_j respectively.

Indexing a query. Node n will index q by creating a query chain and assigning responsibility for q to the nodes in the chain as follows. For each subquery q_j, n creates a message INDEX-QUERY$(q_j, key(q), key(n))$ and computes an identifier I_j using the elements of q_j that are constant. If q_j is of the form $(?x, p_j, ?o_j)$, then $I_j = Hash(pred(q_j))$, while if it is of the form $(?x, p_j, o_j)$, then $I_j = Hash(obj(q_j))$. The identifier I_j will lead to the node that will be responsible for subquery q_j. In this way, a set M of k messages is created and a set I of

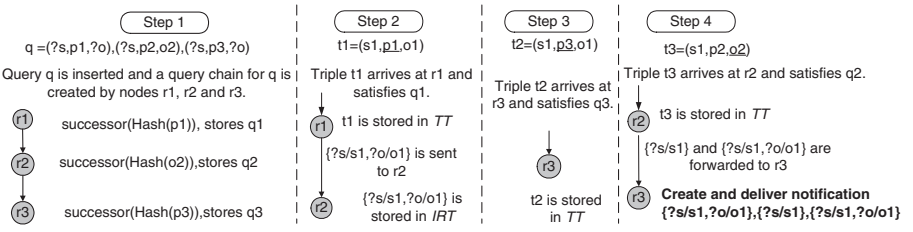


Fig. 2. The algorithm SQC in operation

k identifiers. The successors of those identifiers are called the *responsible* nodes for each subquery of q and form the query chain of q. Node n calls the function $multiSend(M, I)$ to index the query with complexity $k * O(logN)$ overlay hops. The *multiSend()* function sorts I in the clockwise direction starting from $id(n)$ so the query chain that will be created will require the minimum overlay hops when forwarding intermediate results [12].

Each node r that receives an INDEX-QUERY$(q_j, key(q), key(n))$ message stores q_j in its local *query table* (QT) along with $key(q)$, $key(n)$, and two other parameters: $next(q_j)$ and $position(q)$. Parameter $next(q_j)$ will be used by r to reach the next node in the chain when needed, i.e., $next(q_j)$ is the identifier of the next node. Parameter $position(q)$ is used to show the position of r in $chain(q)$. *position* takes the value $first$ or $last$ if r is first or last in $chain(q)$ respectively. Otherwise, *position* takes the value *middle*. The construction of query chains in SQC is shown graphically in Figure 2 through an example.

Indexing a new triple. A new triple has to meet all relevant queries. Since subqueries are indexed either according to their predicate or their object value, a new triple $t = (s, p, o)$ has to reach both $Successor(Hash(p))$ and $Successor(Hash(o))$ for SQC to be complete. Thus, a node that inserts a new triple t will use function $multiSend(msg, F)$, with msg =INDEX-TRIPLE(t, $key(n)$) and $F = \{Hash(p), Hash(o)\}$, to index t in $2 * O(logN)$ hops.

Forwarding intermediate results when new triples arrive. Let us now discuss how a node reacts upon receiving a new triple t. The node stores t in its local *triple table* (TT) and searches its QT for matching subqueries. We will first discuss what happens if this node is *first* in the query chain of a query q. For simplicity we assume that the nodes of the query chain are ordered as $r_1, \ldots, r_k$ and are responsible for subqueries $q_1, \ldots, q_k$ respectively. If t satisfies q_1 with substitution θ then a message msg=EXTEND-MATCHING $(\{q_1\}, \theta, key(q))$ is created and forwarded to the next node in $chain(q)$ with function $send(msg, next(q_j))$. Otherwise, triple t is ignored and there is nothing to be done.

If a message EXTEND-MATCHING $(\{q_1, \ldots, q_{j-1}\}, \theta, key(q))$ arrives at a node r_j in the middle of a query chain for some query q, then r_j tries to find out if the message can be forwarded further in the query chain. This can happen only if r_j is storing triples that satisfy the subquery q_j of q that r_j is responsible for. Thus, r_j searches its local TT for such triples. If there is a triple t' and variable substitution σ such that $q_j\theta\sigma = t'$, then the list of satisfied subqueries $\{q_1, \ldots, q_{j-1}\}$

can be extended with q_j and the next node in the chain should be notified with a message EXTEND-MATCHING $(\{q_1, \ldots, q_j\}, \theta\sigma, key(q))$. Furthermore, r_j stores $\{q_j\theta\sigma\}$ locally in its *intermediate results table* (IRT) which is necessary when triples arrive directly to r_j (see below).

Let us now discuss what happens when a node r_{j+1} in the middle of the query chain of a query q, receives a new triple t''. t'' will be stored in the local TT and r_{j+1} will search locally for satisfied subqueries. Assume that t'' satisfies a subquery q_{j+1} of query q with variable substitution λ, so that $q_{j+1}\lambda = t''$. The difference with the case of being first in $chain(q)$ is that r_{j+1} will not forward t'' to the next node unless the previous node in $chain(q)$ has already sent appropriate intermediate results. Thus, r_{j+1} will search its IRT for partially satisfied subqueries of q. If $\{q_1, \ldots, q_j\}$ such subqueries exist, a message EXTEND-MATCHING $(\{q_1, \ldots, q_j, q_{j+1}\}, \theta\sigma\lambda, key(q))$ will be created and forwarded to the next node in $chain(q)$ as in the previous paragraph.

When a node that is at the end of a query chain receives a message EXTEND-MATCHING $(\{q_1, \ldots, q_j, q_{j+1}\}, \theta\sigma\lambda, key(q))$, it will search its IRT for partially satisfied subqueries as in the previous paragraph, but then instead of forwarding intermediate results (there are no more nodes in the chain), it will use the key of the node that posed the query to deliver any notifications.

An example with SQC in operation is shown in Figure 2. Events take place from left to right, i.e., initially query q is indexed and then triples arrive. For readability reasons, only the steps that affect query q are shown.

Grouping queries. Since a large number of subqueries are expected to be similar, i.e., some of their components are identical, they are *grouped* together at each node. For example, all subqueries that have been indexed to a node r using predicate p will be satisfied when a triple with predicate p arrives (since the subject and object are variables), so r can locally store these subqueries as a group, and check their satisfaction in one step when such a triple arrives. In addition, when nodes send messages EXTEND-MATCHING (), subqueries with the same parameter *next* are grouped so that these results are delivered with a single message to reduce network traffic.

Links. Each node in a chain will contact more than once its next node, so nodes can maintain *pointers* (the IP addresses) to their next nodes for efficiency. This happens with a hash table based local data structure, called *query chain routing table* (QCRT). Thus, intermediate results are forwarded in a single hop.

5 The Multiple Query Chains Algorithm

In this section we present the *multiple query chains algorithm* (MQC). With this algorithm we extend the ideas of SQC to achieve a better distribution of the query processing load. MQC exploits the values of incoming triples to distribute the responsibility of evaluating a query to more nodes than SQC. More precisely, instead of creating a single query chain for a query q at the time that q is inserted, MQC indexes q to a single node r according to one of q's subqueries. Then, when a triple satisfying this subquery arrives at r, the value of its *subject* is used to

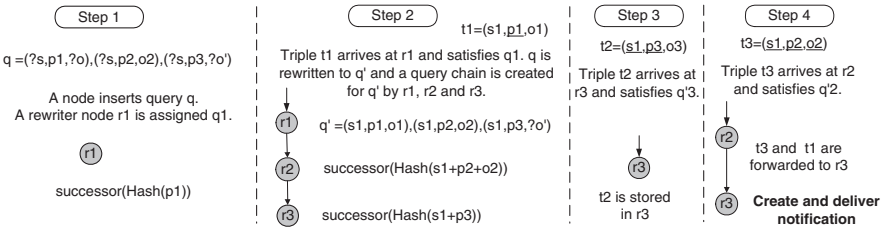


Fig. 3. The algorithm MQC in operation

rewrite q. For each *different* rewritten query derived from q, a *different* query chain is created. Thus, in SQC each subquery of a query q is assigned to a single node, while in MQC rewritten instances of each subquery are assigned to multiple nodes, namely to as many nodes as the distinct subject values in the arriving triples. In addition, MQC combines the known parts of a subquery to index it in order to achieve better distribution as discussed in Section 3.

Indexing a query. Assume a node n that wants to subscribe with the query $q = q_1 \wedge q_2 \wedge ... \wedge q_k$ that consists of k subqueries of the form $(?x, p_j, ?o_j)$ or $(?x, p_j, o_j)$. First, a subquery q_j of q is selected and q is indexed to a node corresponding to q_j. This node is $r = Successor(Hash(pred(q_j) + obj(q_j)))$ if both $pred(q_j)$ and $obj(q_j)$ are constant, or $r = Successor(Hash(pred(q_j)))$ if only $pred(q_j)$ is constant. We call node r the *rewriter* of q. This terminology comes from [13]. Later on, we will discuss good ways to choose a rewriter but at the moment we can assume that this is a random choice. The rewriter stores q in its local QT and waits for triples that satisfy q_j. Since the query is indexed to a single node, the cost is $O(logN)$ overlay hops. Each query has one rewriter, while all queries with the same indexed part have the same rewriter.

Indexing a triple. Since a query might be indexed using a combination of the constant parts of a subquery, we need a new tuple $t = (s, p, o)$ to reach the successor nodes of identifiers $I_1 = Hash(p)$ and $I_2 = Hash(p + o)$. In addition, since queries are rewritten according to their subject values (as we will see below) we also need new triples to reach the successor nodes of the identifiers $I_3 = Hash(s + p)$ and $I_4 = Hash(s + p + o)$. Thus, a node n_1 that inserts a new triple t will use function $multiSend(msg, I)$ to index t to these 4 nodes in $4{*}O(logN)$ hops, where $msg =$INDEX-TRIPLE$(t, key(n_1))$ and $I = \{I_1, I_2, I_3, I_4\}$.

Receiving a new triple. Let us now discuss what happens when a new triple t arrives at a rewriter node r. r checks if its QT contains any query q with a subquery q_j satisfied by t. For each such query q and subquery q_j, r does the following. It *rewrites* the formula $q_1 \wedge \ldots \wedge q_{j-1} \wedge q_{j+1} \wedge \ldots \wedge q_k$ by replacing the subject variable in each subquery with $subj(t)$ to arrive at a new query q'. Then, r uses $subj(t)$ to determine the nodes that participate in the query chain for q'. If this is the first time that q has been satisfied in r by a triple with $subj(t)$, then there is no chain yet for q and this subject value. *Multiple* chains are created for q and different subject values, as triples arrive. In order to create a query chain for a rewritten query, a rewriter node r performs a similar

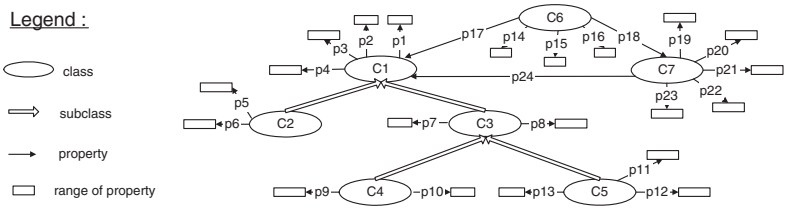


Fig. 4. The schema used in our experiments

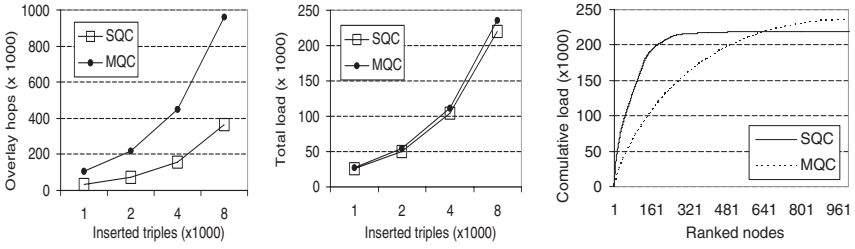
procedure with the one that a query node performs upon indexing a query in SQC only that this time the first node of the chain is already known, namely it is node r. In addition, instead of calculating the index identifier of each subquery according to predicate or object, the index identifier is calculated according to the concatenated string of $subj(t)$ with the predicate or the predicate/object combination of each subquery. As in SQC, the object option is preferred if the object is a constant. Also, index identifiers are sorted according to their distance from the identifier of r to minimize network traffic. From there on, query chains work exactly as in SQC. Each node is responsible for one of the subqueries in the rewritten query. Intermediate results flow in the chain as in SQC while notifications are created by the last node in the chain.

An example with MQC in operation is shown in Figure 3. Notice that a query chain is created in step 2 after the query is rewritten due to a new triple.

6 Experiments

In this section we experimentally evaluate our algorithms. We implemented a simulator of Chord in Java, on top of which we developed our algorithms. We synthetically create RDF triples and queries assuming the RDFS schema of Figure 4. Since our algorithms do not do RDFS reasoning, subclass links in Figure 4 are used to propagate instantiation links and make all class information explicit. We assume a set of 1000 subject values and randomly assign each subject to a class. We also assume a set of 1000 object values and randomly assign each one to the range of a property. To create an RDF triple t, we first randomly choose a class C. Then we randomly choose an instance of C to be $subj(t)$, a property p of C or of the superclasses of C to be $pred(t)$ and a value from the range of p to be $obj(t)$. We use conjunctive multi-predicate queries with three subqueries. To create a query of this type, we first randomly choose a class that the query will refer to. Then, we randomly choose three distinct properties of this class to be the predicates of the three subqueries. All subqueries have the same subject variable while the object parameter of each subquery can be a constant value or a variable. When an object is constant, we randomly choose a value that belongs to the specific property chosen for this subquery.

We present what happens while increasing the total number of triples in the network. Our metrics are (a) the number of overlay hops needed to insert a



(a) Overlay hops (b) Total load (c) Cumulative load (T=8K)

Fig. 5. Evaluating continuous conjunctive multi-predicate queries

number of triples, create and deliver notifications for all matching queries, (b) the total query processing load generated in the network and (c) the distribution of this load. The *query processing load* that a node incurs is defined as the sum of the number of triples that this node receives so as to check if locally stored queries are satisfied plus the number of subqueries that have to be compared against the triples locally stored in this node.

We design our experiment as follows. We create a network of 10^3 nodes and install 10^4 queries. Then, we insert $T = 1K$ triples and we evaluate the metrics described above. The last step is repeated three more times; each time we double T to reach 8K triples.

In Figure 5(a) we show the number of hops needed to insert a number of triples and evaluate all indexed queries. SQC outperforms MQC approximately by a factor of three. This is due to the fact that MQC creates more than one query chains for each query, which means that when nodes in SQC can use QCRTs, nodes in MQC have to create new chains and forward partial results using the Chord infrastructure or in other words in SQC nodes can train their QCRTs more quickly (there are less possible values). For both algorithms network traffic is linearly increased with the number of incoming triples. In addition, experiments where QCRTs are not used showed that MQC has similar performance with SQC (higher by a factor of 4 of the SQC performance with QCRT).

In Figure 5(b) we show the total load created by a number of incoming triples. We observe that the load increases linearly with the number of incoming triples. MQC creates a slightly higher load because more nodes have to be contacted and process messages. In Figure 5(c) we present the cumulative query processing load after 8K triples have been inserted. On the x-axis, nodes are ranked starting from the node with the highest query processing load. The y-axis represents the cumulative load, i.e, each point (a, b) in the graph represents the sum of load b for the a most loaded nodes. We observe that although MQC reaches a slightly higher total load, it achieves to distribute this load to a significantly higher portion of network nodes, i.e., in MQC there are 850 nodes (out of 1000) participating in query processing, while in SQC there are only 250 nodes.

MQC manages to fulfill our goals for a better load distribution which comes with a higher cost in total network traffic, as it is shown in Figure 5(a). However,

this extra network traffic is suffered by *more nodes* (that have to create and forward the extra messages) in MQC. In a longer version of this paper, various experiments are underway that explore how other parameters (i.e., larger network sizes, increasing numbers of indexed queries, skewed distributions etc.) affect the performance of the algorithms.

7 Conclusions

We deal with the problem of evaluating RDF queries over DHTs. We proposed novel algorithms for resolving continuous conjunctive multi-predicate queries with emphasis on distributing load and keeping network traffic low.

References

[1] Balakrishnan, H., Kaashoek, M.F., Karger, D.R., Morris, R., Stoica, I.: Looking up data in P2P systems. CACM **46** (2003) 43–48
[2] Cai, M., Frank, M., Pan, B., MacGregor, R.: A Subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management. J. Web Sem. **2** (2004)
[3] Aberer, K., Cudré-Mauroux, P., Hauswirth, M., Pelt, T.V.: GridVine: Building Internet-Scale Semantic Overlay Networks. (In: ISWC '04)
[4] Chirita, P.A., Idreos, S., Koubarakis, M., Nejdl, W.: Publish/Subscribe for RDF-based P2P Networks. (In: ESWC '04)
[5] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmer, M., Risch, T.: EDUTELLA: A P2P Networking Infrastructure Based on RDF. (In: WWW '02)
[6] Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M.T., Brunkhorst, I., Löser, A.: Super-peer-based routing strategies for RDF-based peer-to-peer networks. J. Web Sem. **1** (2004) 177–186
[7] Kokkinidis, G., Christophides, V.: Semantic query routing and processing in P2P database systems: The ICS-FORTH SQPeer middleware. (In: P2P&DB '04)
[8] Chirita, P.A., Idreos, S., Koubarakis, M., Nejdl, W.: Designing Semantic Publish/Subscribe Networks using Super-Peers. In: Semantic Web and Peer-to-Peer. Springer Verlag (Forthcoming)
[9] Miller, L., Seaborne, A., Reggiori, A.: Three implementations of SquishQL, a simple RDF query language. (In: ISWC '02)
[10] Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H.: Chord: A Scalable P2P Lookup Service for Internet Applications. (In: SIGCOMM '01)
[11] Bawa, M., Gionis, A., Garcia-Molina, H., Motwani, R.: The Price of Validity in Dynamic Networks. (In: SIGMOD '04)
[12] Tryfonopoulos, C., Idreos, S., Koubarakis, M.: LibraRing: An Architecture for Distributed Digital Libraries Based on DHTs. (In: ECDL '05)
[13] Idreos, S., Tryfonopoulos, C., Koubarakis, M.: Distributed Evaluation of Continuous Equi-join Queries over Large Structured Overlay Networks. (In: ICDE '06)

A Semantic Information Retrieval Advertisement and Policy Based System for a P2P Network

Giovanna Guerrini[1], Viviana Mascardi[2], and Marco Mesiti[3]

[1] DI, Università di Pisa, Italy
[2] DISI, Università di Genova, Italy
[3] DICO, Università di Milano, Italy
{guerrini,mascardi,mesiti}@disi.unige.it

Abstract. In this paper we propose a semantic based P2P system that incorporates peer *sharing policies*, which allow a peer to state, for each of the concepts it deals with, the conditions under which it is available to process requests related to that concept. The semantic routing approach, based on advertisements and peer behavior in answering previous requests, takes also into account sharing policies.

1 Introduction

In most P2P architectures, query answering is based on flooding algorithms, that propagate requests from one node to another till a given number of nodes has been reached. Typical routing protocols are based on distributed hash tables for improving routing efficiency. However, these indexes support a keyword based search rather than a *semantic* search. The advantages of a semantic routing, that keeps into account the semantics of data requests and shared resources, are well-accepted in terms of search effectiveness.

Whatever strategy is adopted for query routing, most existing systems are based on the assumption that, when connected to the network, peers are unconditionally available to share their resources with anyone interested in them. This assumption is, however, not reasonable in many contexts and for many reasons. Peers may wish to set some *sharing policies* depending on different factors such as temporal conditions (e.g., the time at which the request is received), internal state and connection conditions (e.g., the workload when the request is received), and conditions on the characteristics of the peer submitting the request (e.g., its membership to a group), that can typically be expressed through *credentials* [22]. A peer can thus customize its behavior by tailoring the general system behavior to its specific sharing needs and constraints.

In this paper, we propose a semantic routing approach in a P2P system that allows single peers to enforce their own sharing policies. The resources made available to the system may deal with many different subjects, or themes, and peers may register to one or more thematic groups. Relevant information retrieval is achieved through the use of a thematic global ontology (TGO) for each theme dealt with by the system; the TGO associates a semantics with the resources to be shared within the thematic group. All the peers that register to a thematic group share the TGO of the group. For the sake of clarity in the paper we will focus on a system with a single thematic group. Each peer associates instances of its local resource base with concepts of the TGO that better describe them. Peers actively push their expertises by sending *advertisements*, containing

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 147–154, 2007.

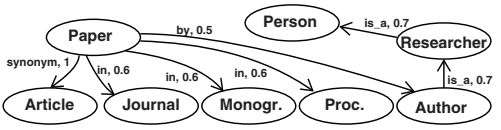


Fig. 1. A portion of a TGO for a computer science thematic group

the concepts of the TGO that better describe the resources they share. Semantic query routing is guided both by the advertised peer expertises and by the *relevance* of peer answers to previous requests. This relevance is quantified in a relevance degree associated with each concept of the TGO, which is updated each time a peer gets an answer to a request involving that concept.

This approach, as a novel feature, integrates in this context the sharing policy and credential notions, thus allowing a more flexible resource sharing mechanism. To allow a peer to enforce different policies for different resources, different sharing policies can be associated with different ontology concepts that the peer's user deal with. Policies associated with concepts of the ontology are actively pushed by the peer together with advertisements, so that other peers can avoid sending and forwarding requests that will not be processed. Thus, sharing policies also affect the routing algorithm.

In the remainder of the paper, Section 2 introduces basic concepts and Section 3 discusses the peer architecture and the system main functionalities. Section 4 compares our approach with related ones and concludes. For space constraints details of the developed approach can be found in [9].

2 Basic Concepts

In this section we introduce the basic notions our approach relies on. An XML format has been chosen for the representation and exchange of these components. The XML Schemas stating the exact format of each component can be found in [9].

Ontologies. In our system, all the peers that registered to a thematic group share the thematic global ontology of that group, TGO. TGO is a directed weighted graph, where nodes (V) represent concepts, arcs (E) represent relations between concepts (including the *is_a* relation), and weights, ranging in $[0, 1]$, represent *how similar* two related concepts are. Each peer P is characterized by a set of concepts of interest CoI such that $CoI \subseteq V$. For example, a portion of the TGO describing the computer science publication domain is shown in Fig. 1. The CoI of a peer $mike$ in this domain might be, for instance, $CoI_{mike} = \{Article, Proceedings\}$. A function Sim_C will be employed to measure the semantic distance between two sets of concepts. This function uses an auxiliary function sim_c for evaluating the similarity between a set of concepts and a single concept of the ontology. Both sim_c and Sim_C refer to the TGO for knowing the weights of the relations among concepts. Details of these functions are in [7].

Credentials and Policies. Credentials are a means to control resource access and to condition resource sharing to certain peer characteristics. A credential $c = (n, \{(p_1, v_1), \ldots, (p_k, v_k)\})$ is a named set properties, that is, name-value pairs. The XML document corresponding to a credential is shown in Fig. 2(a). The use of credentials asserting

```
<Credential name="DISI@UnigeAffiliation">
 <Property name="FirstName" value="Sonia"/><Property name="LastName" value="Pini"/>
 <Property name="Position" value="Researcher"/> <Property name="Office" value="5"/>
</Credential>                    (a)

 <Policy id="1">
  <TempConstDef name="TC1">
   <IntervalExpr  name="sinceJan1st"> <begin> 01/01/05:00 </begin> </IntervalExpr>
   <PeriodicTimeExpr name="9to13ofWorkingDays"> <StartTimeExpr>
     <Week> all </Week>
     <DaySet><Day>2</Day><Day>3</Day><Day>4</Day><Day>5</Day><Day>6</Day></DaySet>
     <Hour> 10 </Hour> <DurationExpr> <Hours> 4 </Hours> </DurationExpr>
    </StartTimeExpr></PeriodicTimeExpr>
  </TempConstDef>
  <InternalCondition type="state" prop="PendingRequests" op="LE" value="15"/>
  <InternalCondition type="state" prop="CPUIdleTime" op="L" value="50"/>
  <CertCondition prop="Position" op="EQ" value="Researcher"/>
 </Policy>                       (b)
```

Fig. 2. (a) An example of credential and (b) an example of policy

properties of individuals raises issues related to certification of properties, their authenticity and verification. These issues are beyond the scope of this paper, thus, in our system, we assume the presence of a peer that releases and certifies credentials of peers joining a thematic group.

Peers restrict their availability to share resources through *sharing policies*. Each policy is characterized by a temporal condition stating the time instants the policy is enabled. Temporal conditions are expressed, according to [3,16], as a $\langle [begin, end], P \rangle$ pair, where $begin$, end are time instants denoting the endpoints of a time interval and P is a periodic expression of the form $P = \sum_{i=1}^{n} O_i.G_i \rhd r.G_d$ where $G_d,\ G_1, \ldots, G_n$ are time granularities or calendars, such that G_d is finer than G_n, for $i = 2, \ldots, n$, G_i is finer than G_{i-1}, $O_1 = all$, $O_i \in 2^{\mathbb{N}} \cup \{all\}$ and $r \in \mathbb{N}$. Suppose for example we wish to represent the period between 9.00 and 13.00 of working days, starting from January 1, 2005 at 00. The corresponding temporal condition is: $[2005/01/01 : 00, \infty], all.Weeks + \{2, \ldots, 6\}.\ Days + 10.Hours \rhd 4.Hours$.

A policy is 4-tuple (id, tC, iC, cC), where tC is a temporal condition and iC, cC denote a conjunction of internal state/connection and credential conditions, respectively. Internal state/connection and credential conditions are of the form $(prop\ op\ value)$ where *op* is comparison operator in $\{\leq, \geq, <, >, =\}$. Suppose, for instance, that a peer wishes to share resources in the temporal period previously presented, but only when the pending requests are less than 15, the CPU idle time is below the 50% and the requester is a researcher. The XML representation of this policy is shown in Fig. 2(b).

A policy $p = (id, tC, iC, \{(n_1\ op_1\ u_1), \ldots, (n_m\ op_m\ u_m)\})$ is satisfied by a credential $c = (n, \{(p_1, v_1), \ldots, (p_k, v_k\})$ and a peer P if the current time instant belongs to set of time instants described by tC, $\forall\, i \in [1, m]\ \exists\, j \in [1, k]\ (n_i = p_j) \wedge (u_j\ op_i\ v_i)$, and P internal and network property values meet iC. For instance, consider a peer P_1 that receives on Monday, July 4, 2005 at 9:30 a data request with the credential of Fig. 2(a). If P_1 enforces the policy in Fig. 2(b), does not have pending requests, and is not performing any computation, then the policy is satisfied.

Advertisement and Data Request Messages. Messages exchanged among peers can be advertisements, data requests, and answer messages. Advertisement and data request

```
<Advs id="AdvChi2" TTL="5" BS="0.8" PeerId = "P457" TimeSent = "27/06/05:14:13">
 <Concept name="Paper"> </Concept>
 <Policies name="Chiara"> <Policy id="1"> see Fig. 2(b) </Policy> </Policies>
</Advs>                  (a)

<DataRequest id="QD2"  TTL="3" BS="1" PeerId = "P473" TimeSent = "27/06/05:14:13">
  <Query> <QueryPred op="EQ" value="Cardelli">
      <PathExpression> <Concept name="Paper">
            <Property name="by"/> <Property name="name"/>
        </Concept> </PathExpression> </QueryPred>
    <QueryPred op="EQ" value="1980">
      <PathExpression> <Concept name="Paper">
            <Property name="year"/>
        </Concept> </PathExpression> </QueryPred> </Query>
  <Credential> see Fig. 2(a) </Credential>
</DataRequest>           (b)
```

Fig. 3. (a) An example of advertisement and (b) an example of data request

messages that are forwarded to other peers are characterized by *Time To Live (TTL)* and *Broad Search (BS)* values, stating the maximal distance between the message sender and the last receiver, and the fraction of peers to forward the message to, respectively. Moreover, each message is characterized by an Id, by the Id of the sender peer, and by the time of the sending.

Advertisements are employed to disseminate information on expertises and sharing policies of the peer's user. An advertisement consists in concepts in the TGO that are related to resources the peer's user is willing to share, and a list of policies pL stating the sharing policies for resources related to these concepts. In checking satisfaction, policies in the list are considered in order, and the iC and cC conditions are checked for the first policy in the list for which now belongs to the set of instants described by tC. For example, Fig. 3(a) shows the XML document corresponding to the advertisement message sent by peer Chiara that shares papers under the previously presented policy.

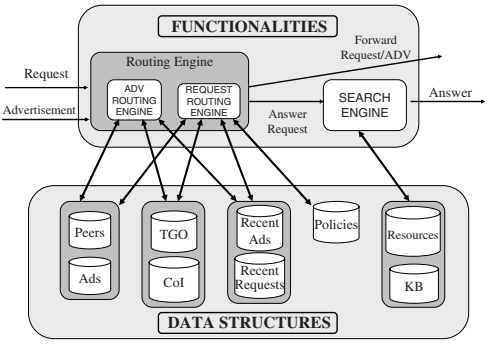
Data requests are characterized by a concept and a set of credentials. The concept belongs to the TGO, and may be optionally qualified with a number of predicates, interpreted as a conjunction, that allow to filter the resources of interest. Data request languages more sophisticated than ours can easily be accommodated in our framework. For example, Fig. 3(b) shows the XML document corresponding to the data request message sent by peer Sonia looking for papers published by Cardelli in 1980. The credential of Fig. 2(a) is attached to the request.

3 Architecture and Functionalities

In this section we describe the main functionalities of the system relying on the architecture graphically depicted in Fig. 4. More details can be found in [9].

Peer Registration

When a new peer wishes to register to a group of the P2P network, it connects to the "special peer". The peer Id is inserted in the list of peers known by the special peer, and the registering peer uses this list to initialize its local *Peers* structure (initially, with null global relevance and no concept-specific relevances associated with each peer). Then, a graphical interface showing the TGO is presented to the peer's user who can browse



Component	Description
TGO & CoI	the thematic global ontology and the concepts of interest. Some indexes are kept over the TGO allowing, given a concept, to directly retrieve its more specific/general concepts and the set of its properties.
Knowledge Base	the set of ontology instances together with their property values. It is handled and indexed through classical database technology.
Resources	the set of resources the peer is willing to share with other peers. Each resource is linked by an instance in KB.
Peers	information on the peers the peer is aware of: peer Id, global relevance degree and concept-specific relevance degrees. Some auxiliary structures allow a direct access to the relevance value of a concept-peer pair and to efficiently get the peers ordered by global relevance.
Ads	information on the advertisements received by other peers: sender peer Id, advertised concept set, sharing policies for those concepts, indexed to get a direct access to the sharing policies of a concept-peer pair.
Recent Requests/ Ads	information on the data requests (the advertisements, respectively) the peer recently received: Data request/Adv Id, sending time, sender peer Id, indexed on the message Id.
Policies	local peer sharing policies, indexed on the concepts they refer to.

Fig. 4. Peer architecture

the TGO, read the textual explanation associated with each concept, identify her concepts of interest (CoI, see Section 2), and realize the concepts that better describe the local resources she wishes to share. The TGO is then copied locally in the peer's data structures. Now the peer's user can, when she wishes, populate the peer's local knowledge and resource bases, as well as the *Policies* structure with the sharing policies to be enforced. The peer is now ready for sending advertisements and data requests, as discussed in what follows.

Advertisement Handling

– *Sending.* A peer wishing to advertise its expertises simply sends advertisement messages, as described in Section 2, to the peers it is aware of (stored in the *Peers* structure).

– *Receiving.* A peer receiving an advertisement message first of all checks whether it has already received it looking at the *RecentAds* structure. If so, it simply discards it. Otherwise, the message is inserted in the *RecentAds* and *Ads* structures. If the sender peer was not known, it is also inserted in the *Peers* structure (with null global relevance e no concept-specific relevance). Note that all the received advertisements are stored. A graphical interface, however, allows the peer's user to browse the *Ads* advertisement database, ordered either by sending time or similarity of the advertised topics with the peer concepts of interest in its CoI, computed through Sim_C, and delete some of them.

– *Forward.* A received advertisement is forwarded to a set of known peers according to the TTL and BS components of the message. Specifically, if TTL is greater than 0, the message is forwarded to BS peers with the TTL value decremented by 1. The peers to forward the message to are chosen among the known peers in the *Peers* structure. A

fraction is randomly chosen, whereas the others are the ones whose sets of advertised concepts (as stored in *Ads*) are most similar to the concepts in the advertisement to be forwarded, according to the similarity function Sim_C.

Data Request Handling

– *Peer Relevance.* When a peer gets an answer to one of its requests, it updates the information in the *Peers* structure related to the relevance of the sending peer to keep into account the new answer. The peer receiving some resources as answers to a data request evaluates them by stating which ones are relevant (and thus are accepted), and which others are not (and thus are discarded). A special *bonus* can be explicitly assigned for extremely relevant answers, through a parameter β whose default is 0. According to the evaluation of the peer P' getting a set of resources as answer to a request Q, the relevance degree got by a peer P sending the answer, related to a concept c belonging to the set of concepts appearing in Q, is a value in $[0, 1]$ computed as: $Relevance(P, c, Q) = \frac{accepted\_resources}{received\_resources} + \beta$. The $Relevance(P, c, Q)$ value contributes to the previous relevance of peer P and concept c, named $rel_{P,c}$, in the *Peers* structure of peer P', if such an entry was there. Otherwise a new entry for peer P, concept c and this value is inserted. The global relevance of a peer rel_P is the sum of the concept-related relevances $rel_{P,c}$ of the peer and is thus updated accordingly. The relevance of a peer P with respect to a set of concepts C is then obtained as $Rel(P, C) = \sum_{c \in C} rel_{P,c} + \sum_{c \in C,\ c \preceq c'} \alpha^d \cdot rel_{P,c'}$ where $\alpha \in [0, 1]$, $\preceq$ denotes the is_a relation in the ontology, and d is the distance between c and c' in the is_a hierarchy of the ontology. The basic principles in using relevance, inherited from [19], are indeed the following: *(i)* a data request is submitted to a peer that answered well to previous requests on the same concepts; *(ii)* a peer that answered well on a specific concept, is likely to be quite knowledgeable on more general concepts related to the same topic; *(iii)* a peer that answered well to previous requests on several different concepts, is likely to be well-informed in general (on any concept).

– *Sending.* When a peer wishes to submit a data request Q to the system, it may include any of its credentials in Q. Then, it selects the peers to send the request to, taking into account the advertisements it received and the peer relevance, for the concepts the data request involves. A list of peers is computed by ordering the set of peers in *Peers* according to their $Rel(P, C)$ value, being C the set of concepts involved in Q. This list is pruned by deleting the peers for which an advertisement has been stored for the involved concepts with associated policies whose credential conditions are not met by credentials in Q, obtaining a list L_r. A similar list L_g is obtained by taking into account the global relevance of the peer rel_P. A last list L_a is computed by ordering the peers in *Ads* according to the similarity of the advertised concepts and the concepts in data request Q, computed through function Sim_C, including only the peers for which the credential condition of an associated policy is met by a credential in Q. The request is sent firstly to the peers in L_r that also belong to L_a, then to other peers in L_r, then to other peers in L_a, then to peers in L_g not considered so far, till the desired number of peers is reached.

– *Receiving.* A peer receiving a data request Q first of all checks whether it has already received it looking at the *RecentRequests* structure. If so, it simply discards it.

Otherwise, Q is inserted in the *RecentRequests* structure and, if the sender peer was not known, it is also inserted in the *Peers* structure (with null global relevance and no concept-specific relevance). Then, the peer checks whether it can answer Q, checking the satisfaction of its own policies associated with the concepts in Q w.r.t. the current time, its current state, and the credentials in Q. If so, its own resources satisfying the data request conditions are sent to the requesting peer. In any case, the request is then forwarded to other peers, following the same behavior adopted for advertisement forwarding, for what concerns the TTL and BS values and the choice of forwarding to a fraction of randomly chosen peers. The other peers to forward the request to are selected with the same list-based approach discussed above for request sending.

4 Concluding Remarks

We have compared our system with FreeNet (`freenet.sourceforge.net`), Edutella [15,14], KEEx [4], Napster (`www.napster.com`), Piazza [12], the Trusted Computing P2P (TC-P2P) Architecture [18], and SWAPSTER [11,20], along the three features that characterize our proposal: *(i)* use of ontologies to answer data requests, and to better route them; *(ii)* use of advertisements to push information about a peer's expertise; *(iii)* use of sharing policies to allow a controlled flexible access to the peer's resources.

The choice of these systems has been driven by the will of considering a spectrum of heterogeneous proposals, where heterogeneity involves both the motivation and the nature of the proposal, and the intended application domain. Our comparison shows that very few systems address all the three aspects that characterize our proposal in a deep and exhaustive way, although most of them implement mechanisms to face at least two of them (see [9] for a full account of the results of our comparison). The originality of our proposal lies in addressing *all* of them into an integrated P2P system.

The system that is closer to ours is SWAPSTER, that has been used to implement two concrete applications: Bibster [11], and Xarop [20]; the developers of SWAPSTER also investigated several query routing strategies by simulation experiments.

Although it is not a P2P system, the framework developed inside the SEWASIE European project [2] shares some similarities with our proposal as far as the management of ontologies is concerned. In fact, in SEWASIE each SINode (a mediator-based system) provides a global virtual view (GVV) of the information sources managed within it, which may resemble the TGO of our proposal, and Brokering Agents integrate several GVVs from different SINodes into a Brokering Agent Ontology. In our proposal, TGO integration has not been investigated yet, but the adoption of a Brokering Agent Ontology suggested by SEWASIE could be a feasible direction to follow.

Most (although not all) of the systems that we have considered in our comparison have been tested on real applications. Although the implementation of our system is still to be completed, we have already implemented many crucial components such as those for evaluating the similarity between concepts, developed using Jena (`http://jena.sourceforge.net`). The main direction of our future work is thus completing the implementation, in order to release a first version, based on JXTA (`http://www.jxta.org`), in few months.

Acknowledgements. We acknowledge P. Bouquet, I. Clarke, E. Franconi, W. Siberski, and S. Staab for their precious advices in drawing the comparison with related work. We also thank C. Casanova for contributing with her Master's Thesis to the design of our system.

References

1. T. Andreasen, et al. On Ontology-based Querying. In *Proc. of the IJCAI Workshop on Ontologies and Distributed Systems*, 2003.
2. S. Bergamaschi, et al. The SEWASIE EU IST project. *SIG SEMIS Bulletin*, 2(1), 2005.
3. E. Bertino, et al. An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning. *ACM Transactions on Information and Systems*, 23(3):231–285, 1998.
4. M. Bonifacio, et al. KEEx: A Peer-to-Peer Solution for Distributed Knowledge Management. In *Proc. of Int'l Symposium on Knowledge Management*, 2004.
5. S. Castano, et al. Semantic Information Interoperability in Open Networked Systems. In *Proc. Conf. on Semantics of a Networked World*, 2004.
6. R. Chen and W. Yeager. Poblano, a Distributed Trust Model for Peer-to-Peer Networks. TR, Sun Microsystem, 2001.
7. V. Cordì, et al. Designing and Implementing an Ontology-Based Distance between Sets of Concepts. TR, DISI TR-05-11, Uni. di Genova, 2005.
8. E. Franconi et al. A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In *Proc. of the VLDB Workshop DBISP2P*, 2003.
9. G. Guerrini, V. Mascardi, and M. Mesiti. A Semantic Information Retrieval Advertisement and Policy Based System for a P2P Network. TR, DISI TR-05-10, Uni. di Genova, 2005.
10. P. Haase, et al. Peer Selection in Peer-to-Peer Networks with Semantic Topologies. In *Proc. of Conf. on Semantics of a Networked World: Semantics for Grid Databases*, 108–125, 2004.
11. P. Haase et al. Bibster – A Semantics-Based Bibliographic Peer-to-Peer System. In *Proc. of Int'l Semantic Web Conf.*, 122–136, 2004.
12. A. Halevy, et al. The Piazza Peer Data Management System. *IEEE Transactions on Knowledge and Data Engineering*, 16(7), 2004.
13. W. Nejdl, et al. Ontology-Based Policy Specification and Management. In *Proc. of European Conf. on Semantic Web*, 290–302, 2005.
14. W. Nejdl, et al. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In *Proc. of Int'l WWW Conf.*, 2003.
15. W. Nejdl et al. Edutella: A P2P networking infrastructure based on RDF. In *Proc. of Int'l WWW Conf.*, 2002.
16. M. Niezette and J. Stevenne. An Efficient Symbolic Representation of Periodic Time. In *Proc. of Int'l Conf. on Information and Knowledge Management*, 1992.
17. R. Rada, et al. Development and Application of a Metric on Semantic Nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30, 1989.
18. R. Sandhu, et al. Peer-to-Peer Access Control Architecture Using Trusted Computing Technology. In *Proc. of Symposium on Access Control Models and Technologies*, 2005.
19. C. Tempich, et al. Remindin': Semantic Query Routing in Peer-to-peer Networks Based on Social Metaphors. In *Proc. of Int'l Conf. on WWW*, 640–649, 2004.
20. C. Tempich et al. XAROP: A Midterm Report in Introducing a Decentralized Semantics-Based Knowledge Sharing Application. In *PAKM'04 Conf.*, 259–270, 2004.
21. G. Tonti et al. Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder. In *Proc. of the Int'l Semantic Web Conf.*, 419–437, 2005.
22. M. Winslett, et al. Using Digital Credentials on the World Wide Web. *Journal of Computer Security*, 5, 1997.

Cumulative Algebraic Signatures for Fast String Search, Protection Against Incidental Viewing and Corruption of Data in an SDDS

Witold Litwin[1], Riad Mokadem[1], and Thomas Schwarz[2]

[1] Université Paris Dauphine,
Centre d'Études et de Recherches en Informatique Appliquée,
Place du Maréchal de Lattre de Tassigny - 75775 Paris cedex 16, France
[2] Department of Computer Engineering,
Santa Clara University,
500 El Camino Real,
Santa Clara, CA 95053, USA

Abstract. Scalable Distributed Data Structures (SDDS) are a class of data structures for multicomputers (a distributed system of networked computers) that allow data access by key in constant time (independent of the number of nodes in the multicomputer) and parallel search of the data. In order to speed up the parallel search of the data fields of the records, we propose to encode the records of a Scalable Distributed Data Structure (SDDS) using pre-computed algebraic signatures. The encoding / decoding overhead is linear in the size of the records. It speeds up prefix searches, longest prefix matches, and string searches. In addition, the encoding protects the privacy of the SDDS data against the owners of the workstations that make up the multicomputer. Additional encoding protects the integrity of the data against malfunctions.

1 Introduction

High capacity networks allow the tight integration of many commodity workstations into a single system. The performance / price ratio of commodity PCs together with their speed allow us to use the combined RAM of workstations as a faster form of storage than traditional disk storage offers. Scalable Distributed Data Structures (SDDS) are designed for these systems; they store records identified by a key while offering key based access times that are independent of the number of computers used. In addition, they allow record searches in all nodes in parallel. The SDDS version of Linear Hashing [L80], LH*, implements a dictionary data structure, whereas RP* additionally allows for range queries [LNS94, LNS96]. Both store records in distributed RAM and achieve much better access than local disk can offer. Indeed, key search or insert speed can be three hundred times [DL00, LMS04] faster. This provides an experimental confirmation of Jim Grays old conjecture about the advantages of distributed RAM presented at UC Berkeley in 1992 based on the calculations in [G88].

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 155–162, 2007.

The workstation storing SDDS data might belong to a large organization. At present, the records are assumed to contain the original value of the user / application data. This exposes them to eyes of the administrator of the workstation on which they are stored, a situation with which neither workstation administrator nor data owner might be comfortable with. Furthermore, a malfunction at a workstation might corrupt the SDDS data. Most importantly, the most expensive operation is a search (in parallel over all workstations) of all local records for a substring. Our scheme greatly speeds up these searches, especially prefix searches in addition to protecting the data against corruption and accidental viewing. It does not protect the privacy of the data against a determined attacker with system access. However, the threat of social sanctions within an organization as well as the normal system defenses against outsiders should keep our data reasonably secure.

Our solution comes at minimal costs, namely the need to encode the record for insertion and to decode the record for accessing it at the SDDS client machine. Performance of key based access (search by key, insert, delete, update) to the record at the server is as fast as before. To protect against corruption, we insert additional data in the record. The storage overhead is small; a typical value would be 0.5% to pinpoint data corruption to blocks of 256B. Given a database workload (with a preponderance of string searches), our speed-up should prove very attractive.

2 Record Encoding

SDDS records consist of a key and a data field. Typically, keys are just Record Identifiers (RI). In this case, knowing a key does not divulge any information. The Data field (D-field) contains all the data in the record. We are not concerned about the internal format of the D-field.

In SDDS-2004, the SDDS prototype [C04], we use algebraic signatures for non-key data search. String searches are either partial or full. A partial search might be a prefix search where we match the first x symbols (bytes, Unicode characters, etc.) against the x-symbol search string.

We now recall briefly the algebraic signature calculus [LS04]. It is based on treating the characters that make up the record as elements of a Galois field. To recall, a Galois field is a finite algebraic structure $\mathcal{GF}(L)$ that allows addition, subtraction, multiplication, and division of its L elements, has a zero element and a one element, and exhibits the same rules for algebraic manipulations as the fields of rational, real, and complex numbers. In the case of ASCII characters, our field is $\mathcal{GF}(2^8)$, whereas in the case of Unicode characters, we use the field $\mathcal{GF}(2^{16})$. Our scheme works for other bit string lengths as well, but these two are the most common ones. In these Galois fields, sum and difference of two elements are both the bit-wise XOR of the operands, and the zero element 0 is the string zero. Multiplication and division are more involved, we refer to [LMS04] for a brief discussion on the way we implement them using logarithm and antilogarithm tables.

Let p_i be the i^{th} symbol in a data field P of l symbols, $i = 1, 2, \ldots, l$; $l < 2^f$. Let $\alpha \neq 0$ be an element of $\mathcal{GF}(2^f)$. Below, α is a *primitive element*, i.e. every non-zero element in $\mathcal{GF}(2^f)$ is a power of α. The *single symbol algebraic signature* is defined by the formula

$$\text{sig}_\alpha(P) = \sum_{i=1}^{l} p_i \alpha^i.$$

A *k-symbol signature* ($1 < k < 2^f$) is a vector

$$\mathbf{sig}_{(\alpha_1,\alpha_2,\ldots,\alpha_k)}(P) = \left(\text{sig}_{\alpha_1}(P), \ldots, \text{sig}_{\alpha_k}(P)\right).$$

A particular wise choice is $(\alpha_1, \alpha_2, \ldots \alpha_k) := (\alpha, \alpha^2, \ldots, \alpha^k)$. This k-symbol signature detects any difference of at most k symbols in a D-field of length up to 2^f *for sure.* For larger D-fields, the probability of an unnoticed error (collision probability, ...) is 2^{-kf}, if we can assume a uniform distribution of symbol values.

Unlike other signature schemes, the algebraic signature has algebraic properties including the certain detection of collisions we just mentioned. Other properties seem to be promising for other uses as well and we will encounter them in what follows. For instance, our signatures change whenever the symbols in the string are permuted, in contrast to calculating the bitwise XOR of all the symbols in the string. If a D-field is larger than $2^f - 1$, then we divide it into *pages* of at most that size. We calculate a k-symbol signature of the successive pages as the field signature.

We encode a given data field $p_1, \ldots p_l$ as $(s_1, \ldots s_l)$ with $s_i = \text{sig}_\alpha(p_1, \ldots p_i)$ Basically, the i^{th} symbol of the encoding is the signature of the prefix. We calculate our encoding incrementally & cumulatively by $s_{i+1} = s_i + p_i \cdot \alpha^{i+1}$ implemented as $s_{i+1} = \texttt{antilog}_\alpha(\texttt{log}_\alpha(p_i + i)) \oplus s_i$. Here, the i-th power is taken modulo $2^f - 1$. The addition in the implementation formula is integer addition and $\oplus$ is the normal bitwise XOR. The taking of the antilogarithm amounts to a table look-up.

Given an encoded data field $(s_1, \ldots s_L)$, we iteratively recoup the original string starting at the end by the formula $p_i = (s_{i+1} + s_i)/\alpha^i$. Thus, decoding is as fast as encoding.

In addition to encoding the D-field, we also embed a k symbol signature for each page at the end of the page. Assume a page size of N. The k symbol signature stored at the end of page i is then

$$\left(\text{sig}_\alpha(p_1, p_2, \ldots p_{iN}), \text{sig}_{\alpha^2}(p_1, p_2, \ldots p_{iN}), \ldots, \text{sig}_{\alpha^k}(p_1, p_2, \ldots p_{iN})\right),$$

that is, a latter page signature "includes" the earlier page signatures. However, the algebraic properties allow us to easily recover individual page signatures from the cumulative page signatures. A typical value would be $k = 4$ for $\mathcal{GF}(2^8)$ and $k = 2$ for $\mathcal{GF}(2^{16})$. In the first case, a page contains 256B and the overhead is 1/64, in the second case, a page contains $2^{17}B = 128KB$ and the overhead for large records is vanishingly small. In either case, the collision probability is 2^{-32}. If that probability is too high (or if the symbol distribution is too skewed), then we can simply increase k.

3 Searches

We describe now four types of searches in the data field, the full match, the prefix match, the string match, and the longest prefix search. A fifth search operation, looking for the longest common substring is also feasible, but we do not present it here for space reasons.

Full Match: A full (non-key) search is the search for all records with the entire, non-key field equal to the string provided by the application. In SDDS-2004, the client calculates the sequence of page signatures for the search string and sends these over to the servers. These compare the page signatures for all pages in the record and send the records for which they have a hit to the client. The client then verifies the accuracy of the match. Since in general a record will not contain many pages, the packets send by the client to the servers will be small.

Prefix Match: The application searches here for every record with the first l symbols of a D-field matching a search string of l symbols. The client sends the length of the search string and its one-symbol signature to all the servers. These compare the l^{th} symbol of each of their records with the signature sent by the client and send all matches to the client. In the case of 16b signatures, the number of false hits is small ($\approx 0.00015\%$ of all records), but in the case of 8b symbols, the number is higher ($\approx 0.39\%$ of all records), and it makes sense to weed out false positives at the server by sending the search string along or by adding symbols signatures to make up a k symbol signature. Even in this case, we still do not have to perform a standard string search for the majority of the records.

String Match: For a string match, the client sends the length l of the search string and the one symbol signature of the search string to all servers. Since $\text{sig}_\alpha(p_i, p_{i+1}, \ldots p_{i+l}) \cdot \alpha^i = \text{sig}_\alpha(p_1, \ldots p_{i+l}) - \text{sig}_\alpha(p_1, \ldots p_{i-1})$ the server multiplies the received symbol signature with α^i in order to see whether the substring starting at position i in the record's D-field matches. This takes one Galois field multiplication with α and one XOR. Thus, while the search takes proportionally to $n-l$), n being the record length, it is quite fast. As before for prefix matches, the size of the symbols used indicates the number of false positives to be expected. Based on this number, the servers might have to weed out false positives before sending the records found to the client.

Longest Prefix Match: For this search, the application provides a string $S = (s_1, s_2, \ldots s_l$ (in encoded form) and requests every record whose D-field shares with S the longest prefix with respect to all the records in the file. In a first stage, each server sends the length of the best local match to the client, who then selects the best match. We now discuss the first stage. Basically, we walk through the records matching against the longest prefix seen so far, and try to see whether any record provides a better match. To match at the first record with D-field $r_1, r_2, \ldots$ we compare s_1 with r_1, if successful, s_2 with r_2, then s_4 with r_4, s_8 with r_8 etc., i.e. increasing the index as the square. If we are unsuccessful, we use

binary search to determine the next index at which we compare. For example, if we fail matching s_{16} with r_{16}, then we would try matching s_{12} with r_{12}. The rationale behind this strategy is that initial matches should be a good predictor of future matching. When we match at subsequent matches, we of course start testing at the current best match, but then proceed in a similar way.

Our procedure runs the risks of collisions, i.e. of false positives. We resolve these collisions at each server, that is, each server checks whether the record with the longest prefix match indeed matches. It does so whenever it finds a record that seems to have a longer prefix match than seen before. Originally, we experimented with collision control at the client, which, depending on the probability of collision and the number of servers might be more effective.

4 Performance

We implemented our scheme on a small network comprising 2GHz Pentium machines using Windows 2000 and connected by a 1Gb/sec network. In our trials, we use records of with a 4B key and 100B D-fields. We found that both encoding and decoding are fast enough to constitute a small overhead ($\approx 15\%$) for SDDS inserts and record retrieval, taking around 0.045 ms/KB and consisting of a few table lookups and an XOR operation per symbol. That non-key string searches cost less for encoded data than for non-encoded data is especially attractive.

Table 1. Timing of String Match (left) and Prefix Match (right)

Record Position	Record Size	Prefix Size	Offset	Time (msec)
1	20	5	13	0.44
1	100	20	70	0.68
1	100	20	80	0.689
100	250	15	80	451
100	250	30	80	437
200	250	15	80	884

Record Position	Record Size	Prefix Size	Time (msec)
1	100	20	0.369
100	250	20	37.8
100	250	35	37.7
200	250	20	71.3
300	250	20	120.53
500	250	20	197.5

The string search performance depends on the location of the correct record within the bucket and to a small degree on the offset of the search string in the record we are looking for. It turns out that it takes about 400 msec to search through 100 records with D-fields of size 250B. (Table 1 left). In a comparison with the Karp-Rabin string search algorithms, [H71, KR87], our algorithm proved to be slightly faster. Searching records with 100 B D-fields, a search for a 15B string took on average 1.40 msec per record searched for our scheme compared to 1.45 msec for Karp-Rabin. Prefix searches are much faster, since we only need to look at a single position per record. It takes about 30 - 40 msec per 100 records (Table 1 right).

To test the longest prefix match, we made several experiments putting the longest prefix in a record at the beginning, in the middle, and at the end of a

Table 2. Longest Prefix Search

Record Position	Time (msec)
1/100	380
49/100	423
99/ 100	453

bucket with 100 records of size $250B$. The times reported in Table 2 are measured at the client talking to a single server. When we did the search on three servers simultaneously, the times slightly increased because the client needs to compare the results. For example, with three servers, the time in the last row of Table 2 increases to 470 msec. This small increase shows that our scheme is scalable.

5 Protection Against Incidental Corruption

The goal here is to detect silent record corruption and identify its location with a certain preselected location. We only need to worry about D-field corruption since the corruption of the key will be detected by normal SDDS operations. If the SDDS offers protection against bucket unavailability by adding parity records as in LH*RS, then silent corruption of one record can spread to all records recovered with the help of this record. A first level of protection is offered by the k-symbol signatures at the end of each page. However, pages can be large (2^{16} in case of Unicode characters, and therefore we might decide to insert k-symbol signatures for smaller fields. In each case, we will discover the silent corruption of up to k symbols in each field for sure and larger corruptions with very high probability.

6 Protection Against Accidental Viewing

Because record D-fields are encoded and because record identifiers have no intrinsic meaning (or are strongly encrypted otherwise), even a system administrator looking at a core dump file cannot see bits and pieces of records. However, a determined attacker with some idea about the structure of the D-field can break the encoding rather quickly. We developed a small test program found the α quickly by looking at a few hundred ASCII records just assuming that the beginning of the record was printable. The same program idea should work quickly for Unicode. Thus, our privacy protection is as strong as putting business letters in a sealed envelope. Almost anyone can break that level of security, but the gain stands no comparison with the social and legal consequences of being found out, and so very few coworkers steam open envelopes.

7 Related Work

At this stage we are not aware of any specific work sharing our assumption of incidental protection. Any trivial approach we are aware of does not match our

string search performance. For instance, this is the case of a table-based arbitrary substitution of characters at the client.

String matching has received attention for a long time [CL03]. The Karp-Rabin type schemes share with ours the principle of incremental calculation of the new value to match when the search moves to the next symbol [H71, KR87]. In addition, our component signature definition is quite similar to theirs. Like our scheme, several algorithms propose preprocessing the string to search in [C97]. Algorithms exist also that search compressed text, e.g., [NT04] and algorithms referenced by [NT04]. The performance of these algorithms varies strongly depending on the compression efficiency. A general comparison still seems to be lacking. However, our algorithms seem to be faster with their $O(1)$ prefix search complexity and the $O(n-l)$ string search complexity. All other major algorithms we know have higher complexity, of $O(l)$ and of $O(n)$ respectively, at best [CL03]. However, to be fair, we need to add the record decoding time of the matching records to our times.

For comparisons to be valid, we stress again the context of our non-key searches, namely that we are searching in a large file or database. In our context, searches are far more frequent than inserts and searches are highly selective. Thus, our encoding overhead is amortized over many searches and only a small fraction of records need to be decoded for a search.

Protection against corruption constitutes a form of data scrubbing [S+04]. Our record scrubbing occurs as part of a record access operation at the client and not preventively at the server. See [S+04] for related trade-offs.

8 Conclusions

We have presented a scheme that protects data stored in an SDDS against incidental viewing and against corruption at the server. The main idea is to encode the data field by storing as character i the single symbol algebraic signature of the prefix of characters 1 to i of the data field. This does not incur any overhead. Since the data field is no longer encoded in plain text, we incidentally protect the field against accidental viewing. In addition, we embed a k-signature for pages to protect against data corruption. This introduces only a slight storage overhead.

The speed of string matches opens interesting perspectives for the popular join, group-by, rollup, and cube database operations. We are in the process of investigating signature calculation speeds. Apparently, extensive, sophisticated use of tables and changing the definition of the single symbol signature without loosing the algebraic properties can speed up encoding and decoding considerably. Finally, one could investigate porting our encoding idea to the related, but slightly different original form of the Karp-Rabin hash function.

Acknowledgements

We thank Jim Gray for comments on early ideas and the support for this work through MS grant. We are also grateful to Lionel Delafosse for comments on the related work. We thank Peter Tsui for implementing the cracking algorithm.

References

[C04] SDDS-2004 Prototype. http://ceria.dauphine.fr/

[C97] M. Crochemore: Off-line serial exact string searching, in Pattern Matching Algorithms, A. Apostolico and Z. Galil, eds., Oxford University Press, New York, 1997) Chapter 1, pp 1–53.

[CL03] M. Crochemore and T. Lecroq: Pattern Matching and Text Compression Algorithms, in The Computer Science and Engineering Handbook, A.B. Tucker, Jr, ed., CRC Press, Boca Raton, 2003, Chapter 8.

[DL00] A. Diene and W. Litwin: Performance Measurements of RP* : A Scalable Distributed Data Structure For Range Partitioning. 2000 Intl. Conf. on Information Society in the 21st Century: Emerging Technologies and New Challenges. Aizu City, Japan, 2000

[G88] J. Gray: The Cost of Messages. 7th CAM Symposium on Principles of Distributed Computing, ACM Press. Aug. 1988, 1-7.

[H71] M. Harrison: Implementation of the Substring Test by Hashing. Communications of the ACM 14(12): 777-779(1971)

[KR87] R. Karp and M. Rabin: Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development, Vol. 31, No. 2, March 1987.

[L80] W. Litwin: Linear hashing : a new tool for file and table addressing. Proceedings of the sixth conference on very large databases (VLDB), 1980. Also in Readings in database systems, Morgan Kaufmann series in data management systems, 1994.

[L81] W. Litwin: Trie Hashing. ACM-SIGMOD 1981 Intl. Conference On Management of Data. Ann Arbor, USA (May, 1981), 19-29.

[LMS04] W. Litwin, R. Moussa, T. Schwarz: LH*RS: A Highly Available Distributed Data Storage System. Intl. Conf. On Management of Very Large Databases, VLDB-04, Toronto 2004.

[LNS94] W. Litwin, M-A. Neimat, and D. Schneider: RP*: A Family of Order-Preserving Scalable Distributed Data Structures. VLDB-94.

[LNS96] W. Litwin, M-A. Neimat, and D. Schneider: LH* - A Scalable Distributed Data Structure. ACM Transactions on Data Base Systems. December 1996.

[LS04] W. Litwin, T. Schwarz: Algebraic Signatures for Scalable Distributed Data Structures. IEEE Intl. Conf. On Data Eng., ICDE-04, 2004.

[NT04] G. Navarro and J. Tarhio: LZgrep: A Boyer-Moore String Matching Tool for Ziv-Lempel Compressed Text. Software Practice and Experience (SPE), to app.

[S+04] T. Schwarz, Q. Xin, E. Miller, D. Long, A. Hospodor, and S. Ng: Disk Scrubbing in Large Archival Storage Systems. 12th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS) 2004.

PARIS: A Peer-to-Peer Architecture for Large-Scale Semantic Data Integration

Carmela Comito[1], Simon Patarin[2], and Domenico Talia[1]

[1] University of Calabria, Rende, Italy
{ccomito,talia}@deis.unical.it
[2] University of Bologna, Bologna, Italy
patarin@cs.unibo.it

Abstract. We propose a novel peer-to-peer architecture, PARIS, aimed at exploiting the unprecedented amount of information available today on the Internet. In PARIS, the combination of decentralized semantic data integration with gossip-based (unstructured) overlay topology management and (structured) distributed hash tables provides the required level of flexibility, adaptability and scalability, and still allows to perform rich queries on a number of autonomous data sources. We describe the logical model that supports the architecture and show how its original topology is constructed. We also present the usage of the system in detail, in particular, the algorithms used to let new peers join the network and to execute queries on top of it.

1 Introduction

Our always more connected world makes available to everyone an unprecedented volume of information. The surge of the Semantic Web, online bibliographic databases, file sharing networks, etc. are only few among the many examples of today's data sources. These data sources are characterized by their heterogeneity, and their dynamic and autonomous nature. In this context, semantic interoperability and source organization are key challenges to be addressed to allow information search, access and retrieval. In other words, one needs to be able to write queries and to execute them efficiently, over all available data sources.

In the past years, solutions have been proposed that address each of these problems independently. In the domain of semantic integration, formal integration models have been defined, query languages have been designed, schema mediation techniques have been proposed [1,2,3,4,5,6]. All those systems focus on a decentralized integration approach: each peer represents an autonomous information system, and semantic data integration is achieved by establishing mappings directly among the various peers. Even if these systems achieve schema integration by adopting different formalisms, all of them abstract themselves from the underlying infrastructure and regard the system as a graph of interconnected data sources. Similarly, peer-to-peer topologies [7,8,9,10] have proven incredibly useful to manage and (self-)organize large networks of autonomous nodes. These systems only provide data sharing at file level, and a limited query

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 163–170, 2007.

language, usually based on file name search. So, little effort has been spent with respect to rich semantic representations of data and query functionalities beyond simple keyword searches.

Some projects exist that rely on peer-to-peer overlay networks and offer rich semantic for data sharing. However, we differentiate from them by the design of an original topology which makes use of both structured and unstructured peer-to-peer overlay techniques. PIER [11] proposes an architecture for relational query processing with an index based on CAN [12] but it does not offer any data integration functionalities. Edutella [13] is a schema-based peer-to-peer network that maintains a semantically enriched information system for the educational domain.

The PARIS (*Peer-to-peer ARchitecture for data Integration Systems*) architecture aims at filling this gap and proposes an integrated approach in which semantic integration, based on schema mapping, and peer-to-peer topology are tightly bound to each other. The combination of decentralized data integration techniques [14] with gossip-based (unstructured) overlay topology management [8] and (structured) distributed hash tables [9] enables rich queries to be performed on a number of autonomous data sources and makes their processing efficient. External efforts required to augment the system and to maintain it, are kept to a minimal level, while still providing the level of flexibility, adaptability and scalability required to cope with the targeted highly dynamic environment.

The rest of this paper is organized as follows. The system model is presented in Section 2. Then we present the PARIS topology design in Section 3. Section 4 describes the actual architecture of the system and Section 5 gives some concluding remarks, together with possible developments of this work.

2 System Model

The primary design goal of PARIS is to develop a decentralized network of semantically related schemas that enables the formulation of queries over autonomous, heterogeneous and distributed data sources. The environment is modeled as a system composed of a number of peers, each bound to a data source. Peer schemas are connected to each other through declarative mappings rules. PARIS builds on the data integration model introduced for the XMAP integration framework [14]. The underlying integration model of this framework is based on schema mappings to translate queries between different schemas. Query translation is performed through the XMAP query reformulation algorithm.

We model our system as a collection $\mathcal{P}$ of *peers* which are logically bound to data sources. Each data source D_p is represented by exactly one peer p and, conversely, each peer has access to a single data source, named its *local data source*. A *local schema* S_p is associated with the data source D_p. Each peer also holds a collection M_p of mappings from its local schema to other foreign schemas. Finally, a peer knows a list (*view*) of other peers (*neighbors*). Peers are able to execute the above mentioned reformulation algorithm (XMAP) and to answer locally the queries they receive.

From the point of view of a single peer, the other peers in the network can be classified into four "groups". (1) The peer **local** group is made of all peers p_i that share the same schema. We note L_p the local group of peer p and $L_p = \{ p_i \in \mathcal{P} \mid S_{p_i} = S_p \}$. Every peers in a local group L share the whole collection of mappings, denoted M_L, relative to the group schema S_L (the shared local group schema). (2) With respect to a peer local group L, the **semantic direct** group, D_L, is made of all local groups L_i with which a point-to-point mapping for the group local schema S_L is known. Formally: $D_L = \{ L_i \in \mathcal{L} \mid S_L \rightsquigarrow S_{L_i} \}$, where $\mathcal{L}$ is the set of all known local groups, and $S_A \rightsquigarrow S_B$ denotes the existence of a direct mapping from schema A to schema B. The local groups of a semantic direct group share all their mappings, so the mapping collection, M_{D_L}, of D_L is composed by the union of the mappings of each local groups L_i. (3) Again, with respect to a peer local group L, the **semantic transitive** group, T_L, is made of all local groups L_i whose schema is semantically related to the peer local schema S_L through a transitive mapping: $T_L = \{ L_i \in \mathcal{L} \mid (\exists\, L_0, \ldots, L_k \in \mathcal{L}^{k+1} \mid S_L \rightsquigarrow S_{L_0} \wedge S_{L_0} \rightsquigarrow S_{L_1} \wedge \ldots \wedge S_{L_k} \rightsquigarrow S_{L_i}) \}$. All members of a semantic transitive group share the same mapping collection M_{T_L} composed by the union of the mappings of each local group. (4) Finally, the peer **foreign** group is made of all remaining peers, i.e. all those that are not semantically related to the peer local schema. These notions are illustrated in the next section (Figure 1), when the topology is presented.

Schemas, queries, and peers are given unique identifiers. How schema identifiers are generated (central authority or decentralized hashing of the mapping contents) is not pertinent to our system, provided that we have some assurance that they are truly unique. Peer and query identifiers are to be considered relative to their schema, which means that these identifiers are in practice "prefixed" with their schema identifier. In the case of queries, the schema to be considered is the "original" schema, i.e. the schema over which the query was formulated when it was first submitted to the system. In both cases, we can assume that some local information (e.g. IP addresses, process identifiers, local time, etc.) combined with a random source should be enough to generate these identifiers safely.

3 Topology

We have made the design choice to keep topology management and actual data integration at two distinct levels. This separation of concerns allows us to exploit recent algorithmic advances in the later domain on top of a scalable and robust overlay. We have chosen an hybrid topology for PARIS that mixes both kinds of overlays: structured and unstructured ones. More precisely, local groups are organized in unstructured overlays, while peers (or a large subset of them) are also part of a DHT. This combination differentiates PARIS topology from previously proposed architectures that have not taken advantage of both kinds of overlay at the same time [15,16].

From the definition of local groups (see Section 2), it is clear that all peers in the same local group share the same schema. What we want is the reverse to

be also true: that all peers with the same schema are in the same group. It is therefore necessary to have strong guaranties that a group will remain connected even if a large number of peers fail. As outlined above, gossip-based membership protocols are particularly well-suited for this task. We have chosen Scamp [8] to implement this protocol. Besides its conceptual simplicity, it offers excellent robustness, load-balancing and error-recovery properties.

In parallel, peers also participate in a DHT. From the way we construct peer identifiers (i.e. prefixed by the schema identifier, see Section 2), it is clear that all peers sharing the same schema will be contiguous in the identifier space. Our usage of the DHT will be limited to the capacity of sending messages to a random peer with a specific schema. This means that we make use of the routing interface of the DHT, and ignore its storage capacities. We have chosen to use Chord [9] to implement this DHT. We have stated that, compared to unstructured overlays, DHTs do not support very well frequent peer connections and disconnections. To relax this constraint, we select only the peers that have the best performance within a local group to be part of the DHT. Naturally, a minimum number of peers within a group is required for this selection mechanism to take place. The nodes who do not take part in the DHT will maintain a set of peer addresses (within their own local group) that will act as "gateways" to the DHT. This list is maintained in an epidemic manner as it is the case for the standard peer view.

The resulting overall topology is depicted in Figure 1. In this figure, we assume that the following mappings exist: $S_A \rightsquigarrow S_C$, $S_C \rightsquigarrow S_B$, and $S_C \rightsquigarrow S_D$. From the point of view of the peer p we can classify the other peers as follows. The peer p belongs to the local group $L_p = A$ (i.e. whose shared local schema is S_A). Then, the local group A forms a semantic direct group with the local group C. The semantic transitive group of the local group A is composed of the local groups C, B and D. Finally, the local groups E and F are foreign groups for the peer p.

4 Functional Architecture

This section discusses the functional architecture of PARIS. We have discussed topology issues in the previous section and will not discuss here the exact mechanisms required to maintain it. Details may be found in the original descriptions of these algorithms [8,9].

4.1 Network Management

Letting nodes join (and leave) the network is obviously an important task to fulfill. We view this process as an iterative one, where the initial state is a network made of a single peer, to which nodes are added sequentially, one after the other.

For a node, joining the network means being inserted in its local group (the group corresponding to its local schema) and (possibly) in the DHT. In order to do so, it is sufficient to provide the node with the address of any single peer in the

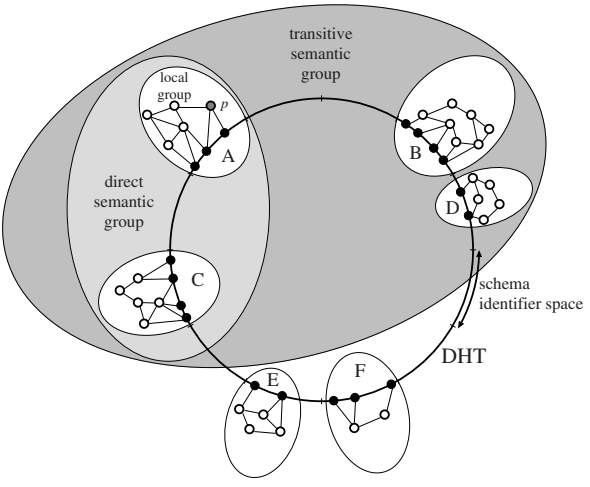


Fig. 1. PARIS hybrid topology. Black nodes are peers with the best connectivity properties that are selected to participate to the DHT. White nodes willing to send messages to another group must hop through a black node in their local group. Groups are named from the point of view of A, assuming the following mappings exist: $S_A \rightsquigarrow S_C$, $S_C \rightsquigarrow S_B$, $S_C \rightsquigarrow S_D$.

system. This peer will take the incoming node schema identifier to locate a peer belonging to the corresponding local group using the DHT (possibly through a gateway), according to the technique described in Section 3. Here, we must distinguish two cases, whether this localization phase succeeds or not.

In the case of a successful localization, the incoming node obtains the address of a peer that belongs to its local group. It will then use this "local contact" node to initiate the join protocol of the gossip-based overlay. The local contact (which is, by construction, part of the DHT) will decide next whether the incoming peer should be instructed to join the DHT or not. A failed lookup means that there exists no other peers that share the same schema as the incoming node. As a consequence, the incoming node will form a new group by itself and its first contact will make it join the DHT (helped by a gateway, if necessary).

4.2 Query Processing

Processing queries submitted by the clients of the system is the main task of PARIS. A query may be submitted to any peer able to understand it, which means that queries submitted to any given peer must be expressed over its local schema. This condition holds for queries internally forwarded by the system. Upon reception of a query, each peer executes the following algorithm.

The peer first looks up the query identifier in the processed query table and drops it if it has already been processed. Then it inserts the query identifier in the processed query table. If it detects that the request has been submitted directly

to it, it will apply the reformulation algorithm to recursively produce reformulated queries expressed over all schemas semantically connected to the schema over which the query is formulated.[1] After the reformulation it determines the associated local groups for each schema over which one or more reformulations of the original query have been produced. And, for each local group, it uses the DHT (possibly through a gateway) to send the reformulated queries to exactly one peer within the group according to the group local schema. Whether it has been reformulated or not, the query is broadcasted to neighbors in the local group before being executed locally and results returned to the originating client.

This algorithm is illustrated in Figure 2. We can see that the query Q_A is submitted to peer p (i). Q_A is reformulated by p into Q_C and, then, into Q_B and Q_D (ii). The reformulated queries are sent to G, P's gateway through the DHT (iii) and, from there, to peers in local groups B, C and D, respectively (iv). The original query is then broadcasted within the local group (v).

In this algorithm, contrary to the usual flood-based approach, the overlay is constructed in such a way that we know *a priori* that the neighbors of a given node are "interested" in the queries we forward to them only spurious messages exchanged are those sent to peers that have already seen the query. Even in this case, the processing overhead to reject such queries is minimal (a lookup in a table).

4.3 Mapping Management

In the steady state, we have said (Section 2) that all peers within a semantic transitive group share the same collection of mappings. We must maintain this invariant when the network evolves. As for request processing, we assume that mappings are submitted to a peer whose local schema is the source of the mapping.

Mappings are manually crafted by administrators or users of the system. When such a mapping is added to a running peer (or an existing one modified), the mapping is broadcasted to all the members of its local group. This mapping is also sent to one member of each local group in the semantic transitive group using the DHT. These members will then broadcast this information to the other members of their respective local groups. This protocol is called the "new mapping" protocol.

When a node joins the network, if its local group is not empty, its local contact will provide it with the current knowledge of the group. Respectively, the incoming peer will run the above new mapping protocol for each mapping it knows which is not yet known by the group.

Finally, when the first node of a local group is inserted in the system, it will use the mappings it knows to contact other peers in its semantic direct group.

[1] In detail, this means: first find all local groups in the semantic transitive group of the local group to which the peer that has received the query belongs to. Then, on the basis of the available mapping rules, determine for which schemas among them it is possible to reformulate the given query. Finally, for each of the latter ones, produce one or more reformulations of the original query.

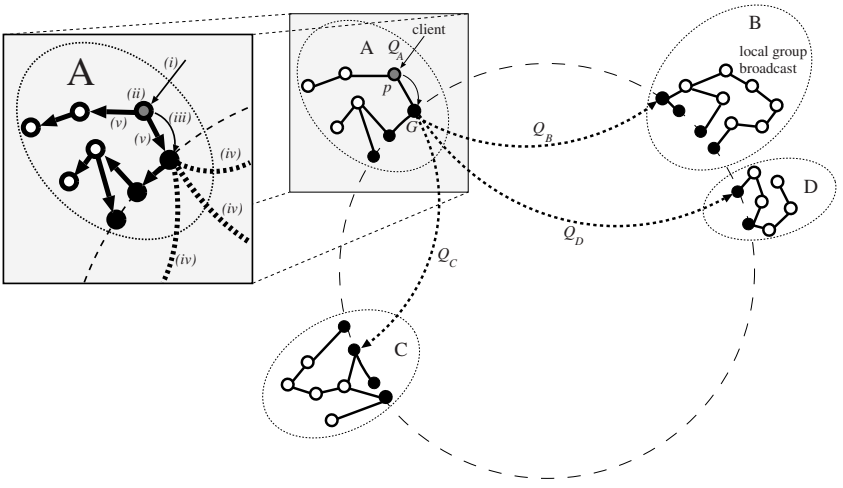


Fig. 2. Query processing in PARIS. In this example, we suppose that a query is submitted to a node of A and that we have the following mappings: $S_A \rightsquigarrow S_C$, $S_C \rightsquigarrow S_B$, $S_C \rightsquigarrow S_D$.

The first node it finds will be used to run the new mapping protocol. It might be the case that no neighbors are found, for example when there exists some group for which no peer has joined the network. Mappings for which no peer has been found yet are flagged as such and reformulated queries are also sent to those "still empty" groups. When a peer is eventually found in a previously empty group, the "new mapping" algorithm is run with this peer and the reformulation algorithm is re-run to take the newly acquired information into consideration.

5 Conclusion and Future Work

We have presented PARIS a peer-to-peer architecture that enables data integration in a large-scale network of data sources. Building on an original hybrid topology, PARIS proposes an efficient query processing framework over a set of semantically heterogeneous data sources. The management of peers itself is scalable as it requires only minimal information for a node to join the network. PARIS is on going work; in particular, some refinements could be brought to the limited broadcasting infrastructure: dynamically building a diffusion tree over an unstructured overlay is not very difficult, but making it robust enough to meet our requirements is. Load balancing issues should also be investigated: we are currently considering using different view sizes to reflect the different capacities of peers. Adaptive query processing could also help us balance the load over the data sources. We are in the process of evaluating the infrastructure through simulation and we plan to start the development of a software prototype in the short term.

References

1. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing : A vision. In: WebDB 2002. (2002) 89–94
2. Calvanese, D., Damaggio, E., Giacomo, G.D., Lenzerini, M., Rosati, R.: Semantic data integration in P2P systems. In: DBISP2P 2003. (2003) 77–90
3. Franconi, E., Kuper, G.M., Lopatenko, A., Serafini, L.: A robust logical and computational characterisation of peer-to-peer database systems. In: DBISP2P 2003. (2003) 64–76
4. Halevy, A.Y., Suciu, D., Tatarinov, I., Ives, Z.G.: Schema mediation in peer data management systems. In: ICDE 2003. (2003) 505–516
5. Kementsietsidis, A., Arenas, M., Miller, R.J.: Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In: SIGMOD 2003. (2003) 325–336
6. Melnik, S., Bernstein, P.A., Halevy, A.Y., Rahm, E.: Supporting executable mappings in model management. In: SIGMOD 2005. (2005) 167–178
7. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Middleware 2001. (2001) 329–350
8. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.: Peer-to-peer membership management for gossip-based protocols. IEEE Transactions on Computers **52** (2003) 139–149
9. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Transactions on Networking **11** (2003) 17–32
10. Jelasity, M., Guerraoui, R., Kermarrec, A.M., van Steen, M.: The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In: Middleware 2004. (2004) 79–98
11. Huebsch, R., Hellerstein, J.M., Lanham, N., Loo, B.T., Shenker, S., Stoica, I.: Querying the internet with PIER. In: VLDB 2003. (2003) 321–332
12. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Computer Communication Review. Volume 31., Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley (2001) 161–172
13. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: EDUTELLA: a P2P networking infrastructure based on RDF. In: WWW2002. (2002) 604–615
14. Comito, C., Talia, D.: XML data integration in OGSA grids. In: VLDB DMG'05. (2005) To appear.
15. Boncz, P.A., Treijtel, C.: Ambientdb: Relational query processing in a P2P network. In: DBISP2P 2003. (2003) 153–168
16. Löser, A., Naumann, F., Siberski, W., Nejdl, W., Thaden, U.: Semantic overlay clusters within super-peer networks. In: DBISP2P 2003. (2003) 33–47

Processing Rank-Aware Queries in P2P Systems

Katja Hose, Marcel Karnstedt, Anke Koch, Kai-Uwe Sattler, and Daniel Zinn

Department of Computer Science and Automation, TU Ilmenau
P.O. Box 100565, D-98684 Ilmenau, Germany

Abstract. Efficient query processing in P2P systems poses a variety of challenges. As a special problem in this context we consider the evaluation of rank-aware queries, namely top-N and skyline, on structured data. The optimization of query processing in a distributed manner at each peer requires locally available statistics. In this paper, we address this problem by presenting approaches relying on the R-tree and histogram-based index structures. We show how this allows for optimizing rank-aware queries even over multiple attributes and thus significantly enhances the efficiency of query processing.

1 Introduction

Schema-based Peer-to-Peer (P2P) systems, also called Peer Data Management Systems (PDMS), have recently attracted attention as a natural extension of federated database systems which are studied since the early eighties. PDMS add features of the P2P paradigm (namely autonomous peers with equal rights and opportunities, self-organization as well as avoiding global knowledge) to the virtual data integration approach resulting in the following characteristics: each peer can provide its own database with its own schema, can answer queries, and is linked to a small number of neighbors via mappings representing schema correspondences. However, the expected advantages of PDMS like robustness, scalability and self-organization do not come for free: In a large-scale, highly dynamic P2P system it is nearly impossible to guarantee a complete and exact query answer. The reasons for this are among others possibly incomplete or incorrect mappings, data heterogeneities, incomplete information about data placement and distribution, and the impracticality of an exhaustive flooding. Therefore, best effort query techniques seem to be more appropriate. By "best effort" we mean that we do not aim for exact results or guarantees but instead try to find the best possible solution w.r.t. the available local knowledge. Examples of such query operators are among others similarity operations, nearest neighbor search, top-N as well as skyline operators.

However, even if we relax exactness or completeness requirements we still need estimations about the error rate. In case of top-N queries this means that we give a probabilistic guarantee that x percent of the retrieved objects are among the top N objects that we would get if we asked all the peers in the system.

Assuming an astronomical application scenario with XML data from sky observations and XQuery as the common query language, a typical query would

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 171–178, 2007.

ask for astronomical objects that match a condition to a certain degree. For instance, a researcher could query the 10 stars closest to a given sky position as shown in the following top-N query:

```
for $s in fn:doc("sky.xml")//objects
order by distance($s/rascension, $s/declination, 160, 20)
limit 10 return ...
```

Here, `distance` is used as a ranking function and `limit` restricts the result set to the first N elements returned by `order by`.

Though one can easily combine multiple ranking functions it is often difficult to define the weights for the individual attribute rankings in order to determine the global rank. For this purpose, a more feasible operator is the skyline operator [1] returning the set of those points that are not dominated by any other point[1]. For example, a query asking for the brightest stars close to a given sky position could be formulated as follows:

```
for $s in fn:doc("sky.xml")//objects
skyline of distance($s/rascension,
                $s/declination, 160, 20), max($s/brightness)
return ...
```

Of course, the skyline operator can also be combined with the **limit** clause in order to restrict the size of the result set.

In order to process such queries in P2P systems in an efficient way, appropriate strategies are needed that reduce the number of queried peers as well as the size of the transferred (intermediate) result sets. The contribution of this paper is twofold: *(i)* we present a novel routing filter capturing multidimensional data summaries and *(ii)* we discuss strategies for processing top-N and skyline queries in P2P systems by exploiting these routing filters.

2 Multidimensional Routing Indexes Based on the QTree

Before presenting techniques for realizing the operators motivated in the previous section we will describe the principles of query processing in P2P systems. As sketched in the example in Section 1 we assume peers to export their data in XML and to be able to process queries based on XPath. We further assume the existence of correspondence links between pairs of peers representing schema mappings that we can use for query translation.

A first but naive strategy would be flooding the network, i.e., asking all the peers that are available in the P2P system. Of course, this works but results in high execution costs. These costs can be reduced by minimizing the number of asked peers. In P2P systems this is usually done by applying routing indexes [2,3] for routing the query to only those peers that are most likely to contribute to the final result. For this purpose, we use the concept of routing filters, as presented in [4]. These routing filters cover both schema and instance level and are based on one-dimensional histograms for numerical data.

[1] A point *dominates* another point if it is as good as or better in all dimensions and better in at least one dimension.

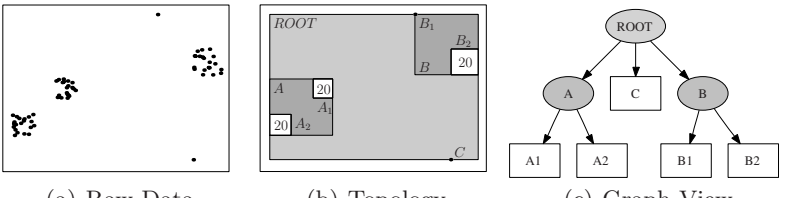
2.1 Routing Indexes

In general, routing indexes represent summarized information about the data a peer can provide. Thus, situations occur where we cannot exactly determine whether the indexed peer actually provides query relevant data or not. Consequently, in addition to reducing message volume and the number of round-trips (our algorithms are restricted to only one round-trip), further cost reduction can be achieved by not forwarding the query to such 'questionable' peers. This results in taking the risk of 'missing' some 'good' data items. The risk that the strategy takes can be quantified and output to the user as a guarantee for the result. In [5] we have presented a strategy that provides probabilistic guarantees for one-dimensional top-N queries. Due to limited space we will not discuss this approach more detailedly.

Having routing filters for one-dimensional queries based on one-dimensional histograms, using histograms for the multidimensional case seems to be the natural consequence. But as also discussed in [6] one-dimensional histograms are insufficient to adequately capture the necessary information about data distributions in the multidimensional case. However, as the main motivation for histograms is to anticipate the number of data items for selection and join queries, it is likely that one bucket covers a large area that contains only few data items and many buckets are used for approximating an area containing a large number of data items. Though this is a good approach for anticipating selection queries, it is not very clever for processing search queries like top-N and skyline. In that case it makes a big difference, if and especially where single data items are located in a bucket. Thus, we have developed a data structure (*QTree*) as a symbiosis of histograms and R-trees. It fulfills the following demands: *(i)* providing information about attribute correlation, *(ii)* being resource-adaptive and resource-efficient in terms of required disk space, *(iii)* being efficient in construction, maintenance and modification in terms of CPU cycles.

2.2 QTree-Based Routing Indexes

Each node in a QTree corresponds to a multidimensional rectangular bounding box. Just like in R-trees, a child's bounding box is completely enclosed in the box of its parent. Leaf nodes are represented by 'buckets' containing statistical



(a) Raw Data (b) Topology (c) Graph View

Fig. 1. QTree

information about those data points that are contained in the bucket's bounding box. The smallest buckets consist of only one point. In the following, we will consider buckets that only provide the number of data points as statistical information, though it is also possible to store mean value or other measures like standard deviation in a bucket. Each QTree has two parameters: (i) f_{max} maximum fanout, (ii) b_{max} maximum number of buckets. b_{max} limits the total number of a tree's buckets and thus its size and memory requirements. Fig. 1 illustrates a two-dimensional QTree with the following parameters: $f_{max} = 3$, $b_{max} = 5$. Fig. 1(a) shows the original data, Fig. 1(b) the QTree's bounding boxes, and Fig. 1(c) the QTree with its buckets and inner nodes.

The QTree represents the basis of the routing filters (*QRoutingFilters*) that we will use in the next section to process multidimensional top-N and skyline queries. A QRoutingFilter describes the data of all neighboring peers in just one index structure. The root node of a QRoutingFilter has one child for each neighbor of the filter owning peer. The data of each child is represented by a QTree and subsumes not only the data of the neighbor itself but also the data that is accessible via this neighbor within a specified hop count distance. The main benefit of maintaining the data altogether consists in the fact that the number of buckets for each neighbor can be chosen (even altered) dynamically.

3 Processing Multidimensional Top-N Queries

This section presents an algorithm based on QRoutingFilters that allows for efficiently processing top-N queries in P2P systems. Considering a peer's local data as well as the buckets of its QRoutingFilter we can efficiently determine the subset of neighbors that provide relevant data.

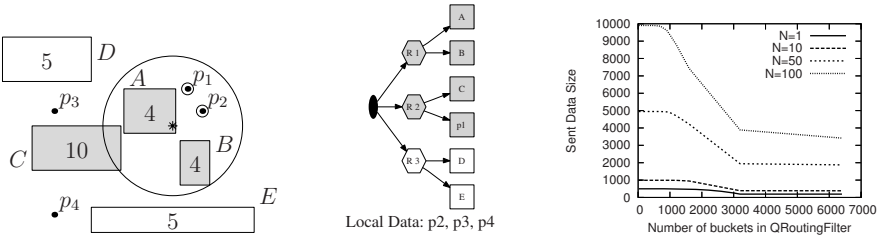
W.l.o.g. let us assume that the score value (that is assigned to a data item by the ranking function) has to be minimized. Let $s_{max}(B)$ and $s_{min}(B)$ denote the maximum and minimum scores that any point in bucket B might have. Furthermore, let $count(B)$ denote the number of data points in B and $\mathbf{B_{all}}$ the set of all buckets. The basic principles of the top-N algorithm are:

- determine a set $\mathbf{B_{suff}} \subseteq \mathbf{B_{all}}$ so that the worst score s is minimized and the following equation holds:
$$\sum_{B_i \in \mathbf{B_{suff}}} count(B_i) \geq N \, , \; s := \max_{B_i \in \mathbf{B_{suff}}} s_{max}(B_i)$$
- based on the worst score s determine all buckets $\mathbf{B_{add}} \subseteq \mathbf{B_{all}} \setminus \mathbf{B_{suff}}$ that might contain data items that have a better score than s:
$$\mathbf{B_{add}} := \{B_j \in \mathbf{B_{all}} \setminus \mathbf{B_{suff}} \,|\, s_{min}(B_j) < s\}$$
- $\mathbf{B_{top-N}} := \mathbf{B_{suff}} \cup \mathbf{B_{add}}$

Based on $\mathbf{B_{top-N}}$, the top-N algorithm is defined as follows:

1. Calculate the top-N result considering all local data and all the buckets of the routing filter

2. Forward a top-K query to all neighboring peers p owning buckets in $\mathbf{B_{top-N}}$, where $K := \min\left\{N, \sum_{p \text{ owns } B_i} count(B_i)\right\}$
3. Receive the answers of those neighbors and combine their results to a preliminary top-N result that is either sent to the query's sender or displayed to the user



(a) Algorithm (b) QRoutingFilter (c) Total Message Volume

Fig. 2. Top-N Algorithm using QRoutingFilters

Fig. 2(a) illustrates an example of our top-N algorithm: Assume we are looking for the top 10 elements near the asterisk. Fig. 2(b) shows the corresponding QRoutingFilter. p_2, p_3, and p_4 are local data items, hence they are not indexed by the filter. In the example, buckets A and B (each containing 4 data items), p_1, and p_2 would be sufficient to provide the top 10 elements: $\mathbf{B_{suff}} = \{A, p_1, p_2, B\}$. The worst-case score for any point in $\mathbf{B_{suff}}$ is visualized as a circle around the asterisk. $\mathbf{B_{add}}$ contains bucket C since this is the only bucket that might provide better data items than $\mathbf{B_{suff}}$. Considering $\mathbf{B_{top-N}} := \mathbf{B_{suff}} \cup \mathbf{B_{add}}$ only neighbors 1 and 2 have to be queried whereas no query has to be forwarded to peer 3.

Experimental Evaluation. For evaluation we analyzed the influence of the size that each routing filter is granted on the number of asked peers as well as on the total network traffic. Based on the attributes *rahour* (right ascension of the observation) and *dedeg* (declination of the observation) of our astronomical test data we calculated 3-dimensional x,y,z-coordinates using a fixed distance of 1000 as radius. Thus, all objects are located on the surface of a sphere. Together with the *vmag* (stellar magnitude in the V system) attribute we built 4-dimensional QRoutingFilters for each peer. The maximum number of buckets in our experiments varies whereas f_{max} is set to 10. All our top-N queries randomly choose a point on the sphere and minimize the sum of the Manhattan distances to the chosen point and to the maximum of *vmag*.

We also varied N in our tests. The results of our tests are shown in Fig. 2(c). The total number of data points sent through the network is shown as a function of b_{max}. The more elements are asked the more data has to be sent. Larger routing filters significantly reduce the network traffic. These results perfectly match our expectations. However, the total number of involved peers could only be reduced significantly when allowing a high number of buckets.

4 Skyline Queries

The main idea of our strategy for processing skylines based on QRoutingFilters is to generalize the "dominates" relation $\succ$ in a way that it can be applied not only on data items but also on arbitrary buckets A and B:

$$A \succ B :\Leftrightarrow a \succ b \ \forall a \in A, \forall b \in B$$

Single data items are interpreted as buckets with no extension. Based on this relation it is possible to calculate a skyline over the local data of a peer, enriched by all buckets of the QRoutingFilter. For this purpose $\succ$ over buckets can be determined using the idea of a worst and a best data item. If items $a_{worst} \in A$ and $b_{best} \in B$ can be constructed such that

$$\forall a' \in A : a' \succ a_{worst} \text{ and } \forall b' \in B : b_{best} \succ b'$$

then

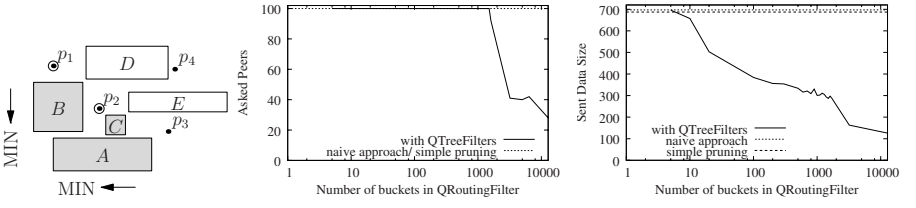
$$A \succ B \Leftrightarrow a_{worst} \succ b_{best}$$

The following fact directly leads to an algorithm for processing skyline queries in P2P systems: All buckets containing data items that are elements of the resulting skyline are elements of the skyline over buckets. The reason is that if an arbitrary bucket B contains a point b that would be part of the overall skyline, there cannot exist any other bucket A that dominates B. Assuming that such an A exists leads to a contradiction: All possible elements in A had to dominate all possible elements in B - thus, also b had to be dominated by all elements in A. Furthermore, as A is not empty, there exists at least one element $a \in A$. So we have found an element a for that $a \succ b$ holds. This is a contradiction to: b is part of the resulting skyline. The skyline algorithm for each peer can be defined accordingly:

1. Calculate the skyline over the local data and all buckets of the routing filter
2. Forward the query to those peers corresponding to the buckets of that skyline
3. Combine the peers' answers to a preliminary skyline that is sent back or displayed to the user

In order to further reduce the data volume that is shipped back during query execution, the data of some buckets is sent along with the query: those, that are likely to dominate a huge amount of the receiver's data. In order to choose these "most selective" buckets, one has to determine how many data items each bucket dominates. Calculating this exactly would be quite inefficient, so we only count for each bucket A how many elements are thrown out of the result skyline because of A. We use a nested loop algorithm for local skyline processing. Before each loop all buckets are sorted decreasingly according to the number of buckets they have already thrown out of the skyline. Thereby, those buckets that have already thrown out a huge amount of data are tested first. Consequently, "selective" buckets become even more "selective". The data of those buckets that superseded the most data items of the corresponding peer is forwarded to the selected neighbor peers. This data consists of a bucket's worst point since this is all information the receiver has to know about a bucket.

Fig. 3(a) shows an example of a skyline over buckets, the corresponding routing filter is shown in Fig. 2(b) with p_2, p_3, and p_4 being local data items. Assuming that both dimensions are to be minimized, buckets A, B, C as well as p_1 and p_2 are members of the resulting skyline. The reason is that only the following dominations occur: $A \succ p_3$, $C \succ E$, $C \succ p_4$, $p_2 \succ p_4$, $B \succ D$, and $B \succ p_4$. Notice that $A \not\succ p_2$ because a point in the bottom left corner of A would dominate p_2. Furthermore, $A \not\succ C$ since it might be possible that A only has data items that are located left of C. As a result the query has to be forwarded to only neighbors 1 and 2.



(a) Algorithm (b) Asked Peers (c) Total Message Volume

Fig. 3. Skyline Algorithm using QRoutingFilters

Experimental Evaluation. In our skyline tests we analyzed the total number of points that was sent through the network in order to answer a query. Like in the top-N experiments we varied the number of buckets for the QRoutingFilters. We queried the skyline over our astronomical test data using the Manhattan distance as ranking function. For each query we randomly chose a point P on the sphere and a *vmag* value. Whereas the distance was to be minimized the *vmag* value was to be maximized. The resulting skylines had an average of 10 data items. The approach like stated above uses QRoutingFilters for routing the query efficiently and sends the "most selective skyline points" along with the query. We used a threshold of 1 what means that all those skyline points that dominated at least one other data point are sent along with the query. In Fig. 3 this strategy is referred to as "*with QRoutingFilters*". It is compared to two other approaches: *naive* and *simple pruning*. *Naive* means that the network is flooded and each peer sends its local skyline to the initiating peer which then processes the final result. The *simple pruning* approach is based on the same principle but the skyline is already checked for dominance by those peers that forward the answer to the initiating peer. Thus, the answer is pruned by forwarding only not dominated skyline points.

Simulation results are shown in Fig. 3(b) and Fig. 3(c). The sum of all points that were sent through the network is displayed on the axis of ordinate whereas QRoutingFilter sizes are displayed in a logarithmic scale on the abscissa. The *naive* and *simple pruning* approaches do not differ significantly because pruning the answer results reduces costs only a little. Using routing indexes, especially QRoutingFilters, on the other hand can effectively reduce the network traffic: A filter that occupies not more than 100 buckets almost halves the amount of

sent data volume compared to a strategy without filter usage. Although these are good results w.r.t. the network traffic the number of asked peers starts decreasing only for large filter sizes (1600 and more buckets per peer), see Fig. 3(b). In order to further decrease the number of asked peers we are working on probabilistic algorithms that relax correctness and completeness requirements.

5 Conclusion

In this work we followed two main goals: processing rank-aware queries while reducing the number of involved peers as well as the amount of data sent through the network. We introduced strategies for achieving both of these goals and focused on the evaluation of a novel data structure called QTree. A QTree combines histograms with the advantages of R-trees. We have shown that the utilization of this structure allows for processing multidimensional top-N and skyline queries efficiently on numerical data. Main open issues and primary content of our current and future work are:

- Approximate query answering techniques, as they promise much more efficiency than always trying to provide exact answers.
- Details of the QTree which are in the first line the construction and maintenance of this innovative data structure.
- How to support rank-aware queries based on string data? This includes finding an index structure that efficiently represents strings, is easy to maintain, and supports lookups for arbitrary strings.
- How to support arbitrary combinations of attributes? This involves combinations of numerical data and string data.

Further aspects, but not in the primary focus of our current work, are the support of limited knowledge in P2P systems and a comparison of QTrees to multidimensional histograms.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline Operator. In: Proceedings of ICDE 2001. (2001) 421–430
2. Crespo, A., Garcia-Molina, H.: Routing Indices for Peer-to-Peer Systems. In: Proc. Int. Conf. on Distributed Computing (ICDCS 2002), Vienna, Austria. (2002) 23–34
3. Petrakis, Y., Koloniari, G., Pitoura, E.: On Using Histograms as Routing Indexes in Peer-to-Peer Systems. In: Proc. DBISP2P 2004. (2004) 16–30
4. Karnstedt, M., Hose, K., Stehr, E.A., Sattler, K.: Adaptive Routing Filters for Robust Query Processing in Schema-Based P2P Systems. In: IDEAS 2005, Montreal. (2005) 223–228.
5. Hose, K., Karnstedt, M., Sattler, K., Zinn, D.: Processing Top-N Queries in P2P-based Web Integration Systems with Probabilistic Guarantees. In: Proc. WebDB 2005. (2005) 109–114
6. Babcock, B., Chaudhuri, S.: Towards a robust query optimizer: A principled and practical approach. In: Proceedings of SIGMOD 2005. (2005) 119–130

Semantic Caching in Schema-Based P2P-Networks

Ingo Brunkhorst[1] and Hadhami Dhraief[2]

[1] L3S Research Center
Expo Plaza 1
D-30539 Hannover, Germany
brunkhorst@l3s.de
[2] Distributed Systems Institute, Knowledge Based Systems, University of Hannover
Appelstrasse 4
D- 30167 Hannover, Germany
dhraief@kbs.uni-hannover.de

Abstract. In this paper, we present the use of semantic caching in the environment of schema-based super-peer networks. Different from traditional caching, semantic caching allows the answering of queries that are not in the cache directly. The challenge of answering the queries using the cache is reduced to the problem of answering queries using materialized views. For this purpose, we implemented the MiniCon-algorithm, which delivers the maximally-contained-rewritings of a posed query based on the stored views. Using simulation and experimental results, we will show the benefit of semantic caching.

1 Introduction

P2P computing provides a very efficient way of storing and accessing distributed resources.

Our goal in the Edutella project [1] is designing and implementing a scalable schema-based P2P infrastructure for the Semantic Web. Edutella relies on the W3C metadata standards RDF[1] and RDF Schema (RDFS) and uses basic P2P primitives provided by Project JXTA[2].

Consider a group of researchers or students sharing metadata for papers they are currently discussing, or learning material they are using. Additionally, metadata repositories from universities and institutes are providing metadata for documents and courses. In this milieu where we deal with a significant number of participant peers and super-peers the network resources become more precious and the avoidance of superfluous queries and messages is crucial.

As a schema-based peer-to-peer network, Edutella constitutes an interesting application area for *semantic caching*. Applying caching techniques reduces the response time of the network and decreases the bandwidth and load for the end-user nodes. Different from traditional caching approaches, the semantic caches

[1] Resource Description Framework: http://www.w3.org/RDF/
[2] Project JXTA(TM) http://www.jxta.org/

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 179–186, 2007.

we propose have the benefit of being able to answer queries that are not in the cache directly. Our approach tries to rewrite the query so that available entries from the cache can be used to answer the request.

In this paper we will focus on the use of semantic caching in a schema-based system. Answering the queries using the cache is reduced to the problem of answering queries using materialized views. For this purpose, we implemented the MiniCon-algorithm, which delivers the *maximally-contained rewritings* of a given posed query based on the stored views. In the next section, we discuss semantic caching in general and how it compares to other caching strategies, followed by Section 3 in which we explain our approach to using semantic caching in the schema-based Edutella network. Section 4 gives a description of the simulation we used and presents the results. Section 5 concludes with a summary and directions for further work.

2 Related Work

Semantic caching in client-server or multi-database systems has received growing interest recently. Dar et al. proposes in [2] a model for client-side caching and replacement in a client-server database system and show the performance of the semantic caching approach compared to the traditional page caching and tuple caching. They mention the easy extension of their approaches to a multiple-server architecture or P2P network, but do not investigate the specific issues related to P2P networks, such as the placement of the cache data structures or the most appropriate cache replacement policy. Moreover, they only consider selection queries on single relations. Dealing with more complex queries is, however, an important issue in the context of semantic caching, and in our approach, we do consider them. More specifically, we consider scalable algorithms for answering conjunctive queries (even with arithmetic comparisons[3]) using views. Godfrey and Gryz present in [3] a general formal framework for semantic caching. The *key idea of semantic caching* is to remember the queries in addition to the query results. Semantic caching uses dynamically defined groups of tuples. We discern two types of semantic caching [4,2]: *Semantic Query Caching (SQC)* [3] and *Semantic Region Caching* [2]. Semantic caching can be considered as a passive caching strategy in contrast to active caching, a.k.a. *replication*, where the caches are filled using automatically generated queries during the periods of low network load. While replication aims at providing high data availability, semantic caching gives the priority to the optimization of response time.

In the context of P2P systems, semantic caching has so far not been investigated. There is some work on caching in P2P networks, mainly associated with replication and focusing on cache replacement strategies. Kangasharju and Ross present in [5] a set of distributed algorithms for replication and cache replacements policies (e.g. Top-K MFR and Top-K LRU) for P2P caches assuming an underlying DHT infrastructure. They show that Top-K MFR provides near optimal performance. Whether this cache replacement policy performs also in

[3] Not implemented in our current prototype.

schema-based super-peer networks, remains to be investigated. Wierzbicki et al. compare in [6] some conventional cache replacement policies in the case of P2P traffic and show that the LRU policy performs very well. An important additional challenge for P2P networks, and in particular super-peer networks is the placement of caches. While in client-server architectures the caches are placed at the clients, in P2P networks there is a need to define the most appropriate location. In addition, interesting for our implementation are semantic caching approaches investigated in the context of *web caches*. The traditional approach of web caches consists of using dedicated machines for centralized caching of web pages. A decentralized P2P-approach has been shown in SQUIRREL [7]. However, the course of action there does not use semantic caches or investigates the cache management at the peers. Lee and Chu present in [8] caching via query matching techniques for Web databases. They also only consider disjoint conjunctive predicates, the caches are located in the wrappers.

3 Semantic Caching in Super-Peer Networks

Super-peer based P2P infrastructures are usually based on a two-phase routing architecture, which first routes queries in the super-peer backbone, and then distributes them to the peers connected to the super-peers. Our routing mechanism is based on two distributed routing indices storing information to route within the super-peer backbone and between super-peers and their respective peers. Super-peers in Edutella [9] employ routing indices which explicitly acknowledge the semantic heterogeneity of schema-based P2P networks, and therefore include schema information as well as other possible index information. The HyperCuP [10] topology is used to organize the super-peers into a structure that enables efficient and non-redundant query broadcasts. Peers connect to the super-peers in a star-like fashion, providing content and content metadata. The characteristics of the super-peer backbone are exploited for maintaining the routing indices and executing distributed query plans.

3.1 Answering Queries Using Semantic Caches

Semantic caching is simply storing queries and their results. The stored queries and their respective results are called *views*. For our approach, we assume that "only" conjunctive queries are used, which are equivalent to *select-project-join* queries. Conjunctive queries are considered as the most used query class. In order to answer an incoming query from the cache, we have to rewrite it in terms of the stored views. The major obstacle for query rewriting is the *query containment problem*. The problem is known to be NP-complete [11], even for conjunctive queries without built-in atoms. The primary source of complexity is the fact that there are an exponential number of candidate rewritings that need to be considered. However, there are algorithms which efficiently generate maximally-contained rewritings (MCR) of a conjunctive query using a set of conjunctive

views. We decided to use the MiniCon algorithm to answer incoming queries using the cached views and content, since it is one of the established well-scalable algorithms for answering queries using views.

3.2 Cache Management Strategies in Edutella

Cache Locality: In order to place the caches and to implement semantic caching in Edutella, it is expedient to look at the different peer types and their characteristics. We distinguish two types of peers in Edutella: *Super-Peers*, which maintain the backbone structure of the network, they facilitate the best network connections and highest availability, as well as high capacity for processing messages. *Peers* are connected via slow connections and sometimes are only available for short durations only. These nodes are run on end-users' computers. The super-peers seem to be the ideal place for caching, in order to attend an optimal exploitation of caching. Due to their characteristics, the super-peers offer the best requirements for caching. Super-peers are responsible for query routing according to routing indices and query processing as well.

Cache Replacement Strategies: The cache replacement policy defines a strategy for replacement of items when additional space is needed in the cache. The caches that are placed on the super-peers must be maintained according to this strategy. The strategy is used to assign values to the items in the cache, the items with the lowest value are then replaced first. In the traditional systems we distinguish between *temporal locality* [2] and *spatial locality*. Temporal locality expresses the concept that items that have been referenced recently are likely to be referenced again in the near future. A "hard and fast" policy does not exist. Each of the replacement policies has both advantages and disadvantages. The choice of a certain replacement policy depends on the use case. The most known replacement strategies are FIFO - First In First Out, Random, *LRU - Least Recently Used* and LFU - Least Frequently Used. In our approach we only use the LRU-Policy, a strategy that is based on the temporal locality principle, i.e. the items which have not been used for the longest time are replaced.

Cache Misses Issues: If a query cannot be completely answered using the local cache, the query is forwarded to the neighboring super-peers according to the HyperCuP protocol. This technique of handling cache misses is referred to as *faulting*. The query and the obtained results are then cached at all super-peers on the path from the requester to the provider.

3.3 Answering Queries in Edutella Using MiniCon

Queries in Edutella are formulated in QEL[4], a query language based on datalog. The MiniCon algorithm [12] aims at finding the *maximally-contained rewriting* (MCR) of a conjunctive query using a set of conjunctive views. It starts similar to the bucket algorithm [13], i.e. in the first step it maps each query subgoal

[4] QEL Query Exchange Language: `http://edutella.jxta.org/spec/qel.html`

Table 1. Example MiniCon Descriptions (MCD)

V	h	φ	G
$V_1(A, B)$	$A \rightarrow A$, $B \rightarrow B$	$A \rightarrow X$, $B \rightarrow Y$	g_1, g_2, g_3
$V_2(C)$	$C \rightarrow C$	$X \rightarrow C$	g_1,g_2
$V_3(Z)$	$Z \rightarrow Z$	$Y \rightarrow Z$	g_3
$V_4(D, E, D)$	$D \rightarrow D$, $E \rightarrow E$, $F \rightarrow D$	$X \rightarrow D$, $Y \rightarrow E$, $X \rightarrow F$	g_1, g_2, g_3

to a view subgoal and determines if there is a partial mapping from the query subgoal to the view subgoal. Let's consider the following query:

$Q_3(X, Y) : -\text{hasSubject}(X, \text{“Edutella”}), \text{hasLanguage}(Y, \text{“de”}), \text{hasRating}(X, \text{“good”}),$

which can be separated into the subgoals $g_1(X) : -\text{hasSubject}(X, \text{“Edutella”})$, $g_2(Y) : -\text{hasLanguage}(Y, \text{“de”})$ and $g_3(X) : -\text{hasRating}(X, \text{“good”})$,

and the views:

$$V_1(A, B) : - \text{hasSubject}(A, \text{“Edutella”}), \text{hasLanguage}(B, \text{“de”}), \text{hasRating}(A, \text{“good”})$$
$$V_2(C) : - \text{hasSubject}(C, \text{“Edutella”}), \text{hasRating}(C, \text{“good”})$$
$$V_3(Z) : - \text{hasLanguage}(Z, \text{“de”})$$
$$V_4(D, E, F) : - \text{hasSubject}(D, \text{“Edutella”}), \text{hasLanguage}(E, \text{“de”}), \text{hasRating}(F, \text{“good”})$$

Once the partial mappings are found, the algorithm focuses on variables rather than on subgoals. The subgoal g of a query Q is mapped to a subgoal g_i of a view V according to specific rules (for details see [12]). The set of such query subgoals that have to be mapped to subgoals from one view (and the mapping information) is called a *MiniCon Description* (MCD).

The MCDs (generalized buckets) that only overlap on distinguished view variables are combined. Given a query Q, a set of views V and the set of MCDs C for Q over the views in V, the only combinations of MCDs that result in non-redundant rewritings of Q are of the form $C_1, C_2, ..., C_l$, where

$$\text{Subgoals}(Q) = \text{Goals}(C_1) \cup \text{Goals}(C_2) \cup \ldots$$
$$\forall i \neq j, \quad \text{Goals}(C_i) \cap \text{Goals}(C_j) = \emptyset.$$

Table 1 shows the corresponding MiniCon descriptions for the Query Q_3 and the Views V_1 to V_4 as presented above: The column h is representing a homomorphic mapping of variables from the head, and φ is showing the mapping of the variables from the query to the variables from the view. All combinations of the MCDs where the elements from G contain all the subgoals of the Query Q are valid rewritings of the query, in this example V_1, V_4, and the combination of V_2 and V_3.

4 Simulation

4.1 Experimental Setup

In addition to implementing and testing the Semantic Caching in the Edutella network, we simulated a larger network using a discrete simulation framework [14].

We extended the existing simulation framework [15] with an implementation for a semantic cache based on the MiniCon algorithm. Queries are routed using the HyperCuP protocols. For the simulation of caching strategies we only use a topology consisting of randomly distributed peers and super-peers. In the Edutella network we distinguish between *provider* and *consumer* peers. Provider peers are providing data to the other peers and can be queried for metadata, consumer peers are used to send queries to the network and retrieve the results.

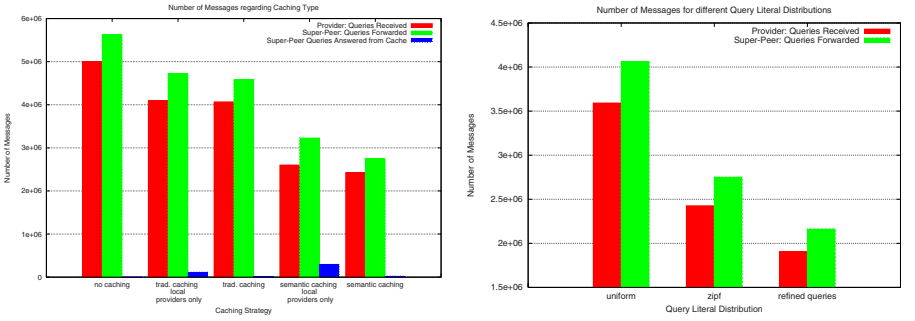
Assumptions: Provider and Consumer peers join and leave the network on a regular basis. Provider peers are considered more stable than consumer peers, and stay in the network for a longer duration. Super-Peers are considered well maintained and highly available, they remain integrated into the topology the whole time. Content is not simulated in the network, instead each provider peer has a fixed probability of 10 percent for generating a response for a received query. We assume that there won't be a fixed schema, only that the set of schema properties and values is identifiable and consists of arbitrary many named elements. Each of the query literals represents a combination of a schema property and a value, e.g. hasSubject(X, "Edutella"). We assume a Zipf-like distribution of schemas because it is a validated law on distributions [15] in many empirical studies and recent research has shown that it holds for the internet as well. Another assumption is that quite often new posed queries are refinements of previous queries, due to the fact that the initial query sometimes doesn't give the wanted results.

Hypothesis: Caching implemented in the super-peer backbone of the P2P network should reduce the amount of messages routed to the data providing peers. With semantic caching the benefit should be larger than with classic caching algorithms, since not only the exact matching cache entries are used to answer the request. The distribution of query literals should also have an effect on the hit/miss ratio of the cache, especially if new queries are refinements of already sent queries.

Experiments: For the experiments we simulated a network of 500 provider, 1000 consumer and 64 static super-peers. The Zipf-distribution used for this simulation consisted of 48 properties and 16 values for constructing query literals. Each query consisted of up to three different literals.

Different caching strategies: The first simulation compares five different caching strategies, including a network without caching. For classic and semantic caching, two different configurations were used. In the first setup only data from providers connected to the corresponding super-peer is cached, messages in the super-peer backbone are transmitted without caching. In the second approach all messages on a super-peer are cached.

Influence of query property/value distribution: To show the influence of the type of distribution of query literals, a simulation was performed with three different configurations. For the first run the properties and values of query literals were uniformly distributed. The second and third simulation uses the same



(a) Caching strategies (b) Query distributions

Fig. 1. Simulation Results

Zipf distribution as in the first experiment. Additionally, in the third simulation a new query is always generated by extending a previously sent query.

4.2 Results

Figure 1a shows the total (accumulated) number of messages in the network after the consumers sent 10000 queries. Every caching strategy reduces the amount of messages transmitted in the system. Without caching the providers alone had to process approx. 5.5×10^6 messages to answer the posed queries. While traditional caching in the super-peers reduces the load for the provider to approx. 4.5×10^6 messages, the use of semantic caching further reduces the amount of messages to 2.7×10^6 (for more details see [16]). Figure 1b shows the results for three different query literal distributions. Semantic caching works best for Zipf distributions, and where new queries are refinements of previously posed queries.

5 Summary and Future Work

Our simulations and experiments have shown, that semantic caching can be used to optimize routing in our schema-based P2P infrastructure. Especially for networks, where a large number of queries is similar, i.e. distributed according to Zipf's law, the semantic caching approach improves the efficiency more then classic caching does. By implementing the MiniCon algorithm we have an efficient way to find the maximally-contained rewriting of a given query using the views contained in the caches. An interesting extension would be a more efficient handling of cache misses using *remainder queries*. Instead of forwarding the query in case answering from the cache is not possible, the query is checked if it can be split (rewritten) in a "cache answerable" and a "cache non-answerable" part. The "cache non-answerable" part, called remainder query is forwarded to the other super-peers in the backbone. If the query can completely be answered from the cache, then we obtain a null remainder query, i.e., no communication with other

super-peers is needed. The advantage of this approach is a lower network load. However, the query rewriting implies an increased complexity.

References

1. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmr, M., Risch, T.: EDUTELLA: A P2P Networking Infrastructure based on RDF. In: Proc. 11th WWW Conference, Hawaii, USA (2002)
2. Dar, S., Franklin, M.J., Jónsson, B., Srivastava, D., Tan, M.: Semantic data caching and replacement. In: Proc. 22th VLDB, Morgan Kaufmann Publishers Inc. (1996) 330–341
3. Godfrey, P., Gryz, J.: Answering queries by semantic caches. In: Proc. 10th DEXA, Florence, Italy (1999)
4. Luo Li, Birgitta König-Ries, N.P., Makki, K.: Strategies for semantic caching. In: Proc. 12th DEXA. Volume 2113 of Lecture Notes in Computer Science., Springer (2001) 99–106
5. Kangasharju, J., Ross, K.W., Turner, D.A.: Adaptive replication and replacement in P2P caches. Technical Report EURECOM+1102, Institut Eurecom, France (2002)
6. Wierzbicki, A., Leibowitz, N., Ripeanu, M., Wozniak, R.: Cache replacement policies revisited: The case of p2p traffic. In: 4th. GP2P Workshop, Chicago, Il. (2004)
7. Iyer, S., Rowstron, A., Druschel, P.: Squirrel: A decentralized peer-to-peer web cache (2002)
8. Lee, D., Chu, W.W.: Towards intelligent semantic caching for web sources. J. Intell. Inf. Syst. **17** (2001) 23–45
9. Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Loser, A.: Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In: Proc. 12th WWW Conference, Budapest, Hungary (2003)
10. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: HyperCuP—Hypercubes, Ontologies and Efficient Search on P2P Networks. In: International Workshop on Agents and Peer-to-Peer Computing, Bologna, Italy (2002)
11. Halevy, A.Y.: Answering queries using views: A survey. The VLDB Journal **10** (2001) 270–294
12. Pottinger, R., Levy, A.Y.: A scalable algorithm for answering queries using views. In: VLDB '00: Proc. 26th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc. (2000) 484–495
13. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogeneous information sources using source descriptions. In: VLDB '96: Proc. 22th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc. (1996) 251–262
14. Cowie, J., Liu, H., Liu, J., Nicol, D.M., Ogielski, A.T.: Towards realistic million-node internet simulation. In: PDPTA. (1999) 2129–2135
15. Siberski, W., Thaden, U.: A simulation framework for schema-based query routing in p2p-networks. In: Proc. Workshop on P2P Computing and Databases, EDBT, Heraklion, Greece (2004)
16. Brunkhorst, I., Dhraief, H.: Semantic caching in schema-based p2p-networks. Technical report, L3S Research Center (2005) `http://www.l3s.de/php/publikation.php`.

Aggregation of a Term Vocabulary for P2P-IR: A DHT Stress Test*

Fabius Klemm and Karl Aberer

School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
{Fabius.Klemm,Karl.Aberer}@epfl.ch

Abstract. There has been an increasing research interest in developing full-text retrieval based on peer-to-peer (P2P) technology. So far, these research efforts have largely concentrated on efficiently distributing an index. However, ranking of the results retrieved from the index is a crucial part in information retrieval. To determine the relevance of a document to a query, ranking algorithms use collection-wide statistics. Term frequency - inverse document frequency (TF-IDF), for example, is based on frequencies of documents containing a given term in the whole collection. Such global frequencies are not readily available in a distributed system. In this paper, we study the feasibility of aggregating global frequencies for a large term vocabulary in a P2P setting. We use a distributed hash table (DHT) for our analysis. Traditional applications of DHTs, such as file sharing, index keys in the order of tens of thousands. Aggregation of a vocabulary consisting of millions of terms poses extreme requirements to a DHT implementation. We study different aggregation strategies and propose optimizations to DHTs to efficiently process large numbers of keys.

1 Introduction

Performing Information Retrieval (IR) on top of Peer-to-Peer (P2P) systems has become an active research field in recent years. In such systems, the peers organize to jointly build a distributed index. Most of the work in P2P-IR has concentrated on efficiently distributing the index. In [12], for example, the authors use a distributed hash table (DHT) to map keywords to responsible peers for indexing. However, this and similar approaches assume that global statistics of the term vocabulary are available and ready to use for, e.g., calculating top-k results.

Another indexing technique is presented in [15] and is based on CAN [11]. Documents and queries are represented as latent semantic indexing (LSI) vectors in a Cartesian space. This space is mapped into a structured P2P network keeping semantically related indexes co-located. Once again, global statistics of

* The work presented in this paper was carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European FP 6 STREP project ALVIS (**002068**).

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 187–194, 2007.

the term vocabulary, which are necessary to compute the weights for the vectors in the Cartesian space, are assumed to be available.

In this paper we study the aggregation of global statistic of a large term vocabulary using DHTs. Our main contributions are: a) We introduce optimization strategies that improve the performance of a DHT to efficiently handle concurrent insertions of very large numbers of keys. b) We present an analysis of term vocabulary aggregation for Internet-scale full-text retrieval.

Our paper is structured as follows: Section 2 gives a brief overview of structured P2P systems. In section 3 we introduce our optimization strategies for DHTs. Results of some practical experiments are presented in section 4. Sections 5 and 6 finish with discussion and conclusions.

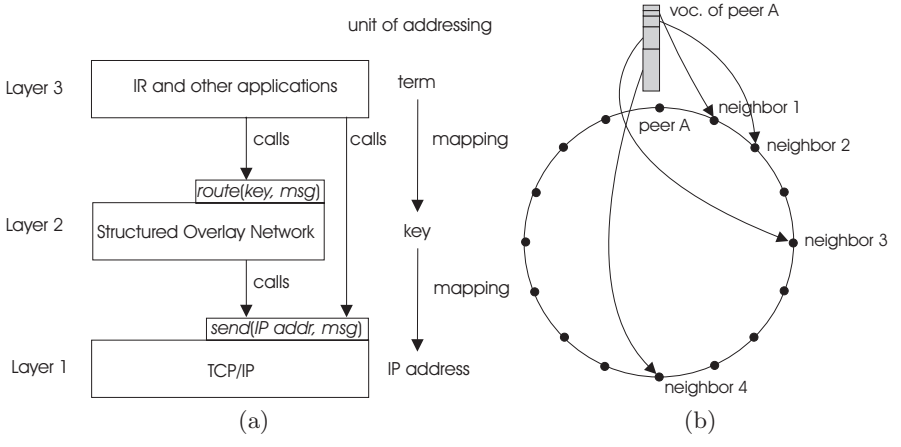


Fig. 1. a) 3-layered architecture b) Insertion of a term vocabulary

2 Overview of Structured P2P Systems

We first provide a short introduction to distributed hash tables (DHTs), also called structured overlay networks. For a clearer presentation, we structure a peer into three layers (figure 1(a)).

The lowest layer provides communication between two peers using TCP/IP. It provides the service *send(IP address, message)*, which sends a message to the peer listening at the given IP address. This service is used by the structured overlay network on layer 2 as well as applications on layer 3.

Layer 2 is the routing layer. It provides the service $route(key, message)$, which routes a message to the peer responsible for the key. It creates and maintains a routing table, which, given a key, determines the IP of the next hop peer for forwarding the message. Therefore, layer 2 provides a key → IP mapping. Most DHTs, such as CHORD, Pastry, or P-Grid [14,13,1] create routing tables of size

$O(log(n))$, where n is the number of peers in the system. The routing entries are chosen in such a way that the resulting graph has small world properties [10,8]. Routing a message between any two peers is then guaranteed to take $O(log(n))$ overlay hops on average.

On layer 3 we have the application that is using the DHT. In our case, it is an IR application, which inserts a local term vocabulary into the DHT using a $route(key, message)$ function provided by layer 2. To perform the mapping of a term to a key layer 3 uses a hash function, which is usually provided by layer 2.

3 Aggregation of Term Vocabulary

This section describes aggregation local document frequencies frequencies to global frequencies. We will first describe the usage scenario and then discuss insertion strategies.

3.1 Usage Scenario

Each peer stores a local document collection. From its local document collection, each peer creates a local term vocabulary. For each term in the local vocabulary, a peer determines the local document frequencies, i.e. the number of documents the term appears in. Each peer inserts its complete vocabulary together with the local frequencies into the DHT. Each term and its local frequency are packed into a message and routed using a key that has been created by hashing the term. The peer responsible for the key receives the message and stores the containing (term, frequency) pair in its local database. For each term, there is exactly one responsible peer. Therefore, for a given term there is one peer that will receive all local frequencies of this term to calculate its global frequency.

Assume a small document collection of 200,000 documents per peer, which has a term vocabulary of about 190,000 terms [1]. All peers concurrently insert their vocabulary into the DHT. We will now present several strategies to handle such a flood of messages and discuss their advantages and disadvantages.

3.2 Blunt Message Handling

When an application calls $route(key, message)$, the straight-forward procedure is to use the routing table to map the key onto the next-hop IP and pass the message to layer 1 to be sent to the next hop. This strategy works fine when only a couple of hundreds to a few thousands of messages have to be inserted and when those messages are reasonably large. However, when inserting a term vocabulary, sending millions of small messages containing only a (term, frequency) pair individually is extremely inefficient. The overhead of message headers is high and message compression is ineffective for small messages.

[1] We computed this value from a sample collection of Reuters news articles available at http://about.reuters.com/researchandstandards/corpus/. The growth of the vocabulary follows Heap's law [9].

3.3 Splitting the Vocabulary into Blocks

Our second strategy optimizes the insertion process by processing (term, frequency) pairs in blocks. Figure 1(b) shows for peer A how the vocabulary is divided into blocks. Most DHTs use randomized hashing to achieve a uniform distribution of keys. In this case, about 1/2 of the terms in the vocabulary maps to peers on the left side of the circle and are therefore sent to neighbor 4 as next hop. 1/4 will be sent to neighbor 3, 1/8 to neighbor 2, 1/16 to neighbor 1, and 1/16 maps to peer A itself. In general, $O(Log(n))$ blocks have to be sent.

This scheme has the following advantages: a) some of the blocks are large enough to be efficiently compressed. b) Shipping few large packets over TCP/IP is faster than shipping many small packets of only a few bytes.

However, in which layer of the architecture should we split the vocabulary into blocks? If we do it in layer 3, it has to know about the key $\rightarrow$ IP mapping of layer 2, which should be hidden to upper layers. Handing the whole vocabulary down to layer 2 would require making the interface of layer 2 application-specific, which is also not a desirable solution. Therefore, we propose a third strategy, message queueing at layer 1.

3.4 Message Queuing

In this strategy layer 3 and 2 do not have to deal with message packing at all. Messages with single (term, frequency) pairs are handed over to layer 1 with the *send(IP address, message)* function. Layer 1 takes care of efficiently shipping messages to their next-hop IP. To build blocks of messages layer 1 maintains a queue for each outgoing IP address. As the size of the routing table at layer 2 is $O(log(n))$, we also need $O(log(n))$ outgoing IP queues at layer 1. Each queue stores messages according to the following scheme: Each queue has a timer and a threshold. Messages are delayed in the queue until either the threshold is reached or a timeout occurs. A timer is started when a message is inserted into an empty queue. When the message threshold is reached or a timeout occurs all messages in the queue are packed together. This pack of messages is then compressed and sent to the next-hop IP as one large packet.

This approach has the following advantages: a) it is completely hidden to upper layers: many small messages can be inserted into the DHT in bursts and efficiently processed at layer 1. b) It is more flexible compared to approach 2: messages from other peers that have to be forwarded can be packed together with messages originating from the same peer.

The threshold should be set high enough for compression to be effective, but not too high to avoid unnecessary delays. The queue threshold can be defined in either number of message or number of bytes. To avoid that time critical messages, such as queries, are delayed, we added a message flag that specifies whether a message can be delayed. Time-critical messages are instantly sent, irrespective of timeout and threshold.

3.5 Avoiding Flooding

When all peers in the network concurrently start to insert their vocabularies, the network can be flooded and break down if no additional measures are taken. In this subsection, we will present mechanism to avoid such an overload of the system.

Priority queues. As first mechanism to avoid overload we propose using priority queues: We have two types of messages: a) the messages that are already in the system and are travelling (over several overlay hops) to their final destination. b) The messages that are about to be inserted into the system (by an application on layer 3). The first type of messages should have priority over the newly inserted messages. We give higher priorities to messages that have been in the system for a long time. Such messages are close to their final destinations and therefore will soon get out of the system to make space for new messages.

Receiver feedback. The second mechanism is receiver feedback. We are in an environment of heterogeneous peers, i.e. some peers have more processing power than others. It is therefore important to avoid that slower peers are flooded with messages. TCP flow control already does some work, however, cannot fully prevent slower peers from being overloaded. Therefore, we introduced a feedback mechanism on top of TCP. A peer (at layer 1) can forward the next message to the same IP address only after having received an acknowledgement, which the receiver returns after the message has been processed. The delay of this acknowledgement thus depends on the current load of the receiving peer.

4 Experimental Results

We performed experiments in the local EPFL gigabit LAN on 64 SUN Ultra 10 with 360 MHz CPU and 256 MB RAM. The results are for 4, 8, 16, 32, and 64 peers, one peer running per machine. We implemented our DHT in JAVA. Messages are objects, which are serialized and compressed at layer 1. For compression we use the java GZIP classes. The queue threshold is 100 messages and the timeout 1s. Figure 2(a) shows the results of our runs for vocabularies of 96,000 and 192,000 terms per peer.

The first observation we can make is that the execution time to insert all vocabularies is about twice as long for 192K terms as for 96K terms. We therefore conclude that our implementation is stable.

Second, we observe that the insertion time grows considerably slower than the total number of terms: with 192K terms per peer, increasing the total number of terms from $4 * 192K \approx 770$K to 64 * 192K $\approx$ 12M, a factor of 16, increases the total insertion time from 460s to 1600s, which is a factor of 3.5.

Figure 2(b) shows experiments with varying queue thresholds of 1, 10, and 100 messages. A threshold of 1 means that messages are not queued, i.e. each messages is sent individually. The number of keys inserted by each peer is 19,2K, i.e. 10% of the amount in figure 2(a). With queue threshold t set to 10, the

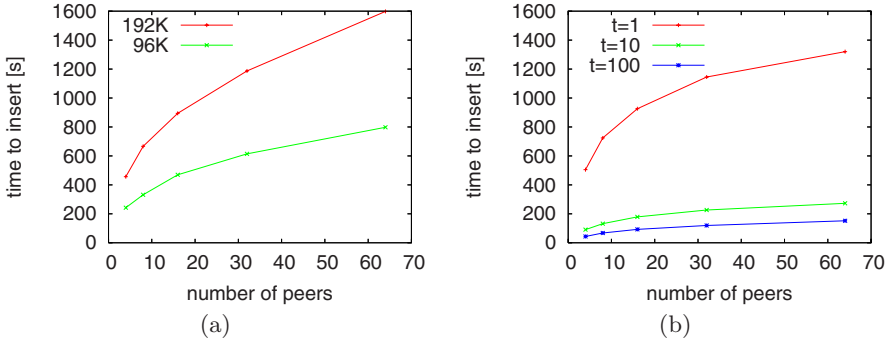


Fig. 2. Insertion times for a) varying voc. sizes and b) varying queue thresholds

insertion time decreases already significantly by 80%. A queue threshold of 100 messages leads to a decrease of 89%. The decrease in bandwidth consumption is 87% for $t = 10$ and 97% for $t = 100$. The reason for this dramatic decrease is that Java produces very large object serializations, which can be very well compressed. Compression of multiple small messages can therefore increase the performance significantly.

5 Discussion

5.1 Redistribution of Aggregates

Once the local term frequencies are aggregated, the global frequencies have to be distributed to interested peers. If all peers are interested in the same vocabulary, we could simply broadcast the global frequencies. Efficient broadcast strategies in DHTs have been presented in recent research papers, such as in [6,7]. However, such an assumption is not realistic in large networks. Another possibility is that aggregates are streamed to interested peers. This could be done using a multicast protocol, such as presented in [4,5]. The integration of such a protocol is part of future work.

5.2 Fighting Malicious Peers

As all peers are allowed to insert term-frequency pairs, some peers could try to insert false values to change ranking to their advantages. Trust in P2P is not the focus of this paper. Nevertheless, we sketch possible solutions: a) There exist environments where all peers in the network are trusted, e.g. when they belong to a company network or a closed P2P group of cooperating universities. Malicious behavior would lead to the exclusion from the group. b) False frequencies that are excessively high could be detected by comparing to former values of the global frequencies of this term. However, a peer could still repeatedly insert

small values for the same term to increase its frequency. A solution could be to monitor, which peer is inserting which frequencies to detect malicious pees.

Trust in P2P is a large research area on its own. We believe that our application of DHTs is not fundamentally different from other applications and that solutions in trust management would therefore be applicable.

5.3 Updating Term Frequencies

Local document collections and therefore term vocabularies keep changing over time. It is necessary to update term frequencies. One possible approach might be to re-run the complete aggregation process, e.g. every couple of days or weeks, depending on how fast the local document collections change. Another option might be to instantly insert updates. The responsible peer that aggregates frequencies for a certain term would then have to estimate update rates to calculate approximate global term frequency. Further problems during the aggregation process can arise when peers fail. Handing peer failures and replication of data, however, is orthogonal to this work. We leave improvements in these areas to future work.

5.4 Scaling It Up

In our experiments the size of the local vocabularies was about 200K terms, which corresponds to a collection of roughly 200K documents. The insertion time with 64 peers was approx. 30 min. Let's assume a peer stores 5 million documents. The corresponding term vocabulary would contain about 1 million terms (according to Heap's law [9]) and could therefore be inserted in less than 3 hours with 64 peers (five times the time necessary than for the 200K vocabulary). In a network of 2000 peers the insertion time would be approx. 5 hours. Such a network could thus maintain the global term vocabulary for a collection of 10 billion documents, about the size of the Google index a the time of writing.

6 Conclusions

In this workshop paper we showed that it is possible to aggregate an Internet-scale term vocabulary with P2P technology. This result is important as many P2P-IR systems require global document frequencies of terms for efficient indexing and ranking. We proposed mechanisms to improve standard DHTs to handle very large numbers of messages. These strategies can serve as suggestions for improving existing DHT implementations. Aggregation of a term vocabulary is only one (though very important) possible application that can benefit from our improvements. In principle our techniques using message packing, priority queues, and receiver feedback are necessary for efficiently implementing any distributed application that sends very large numbers of small messages.

As future work, we are planning experiments in a more "hostile" environment than the university Intranet, e.g. in PlanetLab [2]. In such an environment we

[2] www.planet-lab.org

expect more unevenly loaded peers and large variations in network delays as well as peer failures, which will require refinements of our queueing strategies.

References

1. K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. *Sixth International Conference on Cooperative Information Systems*, 2001.
2. K. Aberer, F. Klemm, M. Rajman, and J. Wu. An Architecture for Peer-to-Peer Information Retrieval. *27th Annual International ACM SIGIR Conference (SIGIR 2004), Workshop on Peer-to-Peer Information Retrieval*, 2004.
3. A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 353–366, New York, NY, USA, 2004. ACM Press.
4. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03)*, Berkeley, CA, 2003.
5. M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. In *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, October 2002.
6. S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks. In *IPTPS*, pages 304–314, 2003.
7. A. Ghodsi, L. O. Alima, S. el Ansary, P. Brand, and S. Haridi. Self-correcting broadcast in distributed hash tables. In *Series on Parallel and Distributed Computing and Systems (PDCS'2003), ACTA Press, Calgary*, 2003.
8. S. Girdzijauskas, A. Datta, and K. Aberer. On small world graphs in non-uniformly distributed key spaces. 2005.
9. H. S. Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., Orlando, FL, USA, 1978.
10. J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.
11. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. 2001.
12. P. Reynolds and A. Vahdat. Efficient Peer-to-Peer Keyword Searching. *Middleware03*, 2003.
13. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, 2001.
14. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. *In Proceedings of ACM SIGCOMM*, 2001.
15. C. Tang, C. Xu, and S. Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *SIGCOMM*, 2003.

Peer Group-Based Dependency Management in Service-Oriented Peer-to-Peer Architectures

Sascha Alda

Department for Applied Computer Science, University of Bonn
Roemerstrae 164, 53117 Bonn, Germany
alda@cs.uni-bonn.de

Abstract. Dependency management in service-Oriented peer-to-Peer architectures aims at handling functional dependencies between a public service hosted by a service providing peer and all consumer peers that rely on that service. The analysis of dependencies on consumer peers is important for supporting the adaptation of a public service. The uncoordinated adaptation of public services potentially leads to malfunctions in the environment of depending peers. In this paper, a novel way for handling service dependencies in peer-to-peer architectures is proposed. This approach suggests that peers of a peer group agree on a common adaptation policy that prescribes how service providing peers have to deal with potential dependencies before an adaptation can be pursued.

1 Introduction

Service-oriented peer-to-peer architectures leverage the original notion of centralized service-oriented architectures (SOA) by enabling hosts (*peers*) to function as provider, consumer, and registrar of services at the same time. This architectural style facilitates new ways for the composition of services towards applications. A peer may compose services from service providing peers with local components, which in turn may serve as the implementation of a new service provided by the same peer. Other third party peers locating and using that service thus hold a direct and a transitive dependency on both provider peers given in this scenario. In peer-to-peer architectures, dependencies cannot be assumed as static but can be violated due to two reasons:

- owing to the fluctuating topology of peer-to-peer architectures (peers tend to fail or are unavailable), services can become unavailable. Affected peers that deploy applications relying on a failed service are no longer able to offer a correct run of these applications and malfunctions may occur.
- the operator of a provider peer is always capable of adapting the interface or the implementation of a peer service without any prior notice to depending consumers. Malfunctions can also occur within the consumer's environment.

An efficient dependency management for service-oriented peer-to-peer architectures turns out to be indispensable for guaranteeing a stable environment in

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 195–202, 2007.

which applications are accomplished to operate in a reliable manner. However, existing implementations of peer-to-peer architectures cover this management issue insufficiently. Existing general-purpose notions for describing and identifying dependencies among services (or software entities in general) prove to be inflexible for peer-to-peer architectures. The intention of this paper is to discuss the requirements of dependency management accurately and, moreover, to explain why existing approaches fail to serve as a solid solution for this management activity. This present paper also outlines main aspects of the DeEvolve peer-to-peer platform [1] [2] that incorporates novel approaches for managing dependencies within a peer-to-peer architecture. Integral part of DEVOLVE is a runtime environment for the deployment of component-based peer services. DeEvolve features an adaptation environment enabling component assemblers to flexibly define and to adapt compositions of peer services during runtime. Any attempt to adapt a composition has to precede an analysis of consumer dependencies, to reveal dependencies to current and potential consumers. Dependencies impose an adaptation policy that prescribes how provider peers within a distinct group have to account for dependencies, and how these are to be resolved.

The goal of this paper is to present our recently unpublished material for analyzing consumer dependencies in service-oriented peer-to-peer architecture. In the next section, we first summarize some related work. In section 3 the DeEvolve platform is introduced for a better understanding of the rest of the paper. The fourth section features a description of the analysis of consumer dependencies. Section 5 concludes the paper by summarizing the main aspects as well as future directions.

2 Related Work

Recent conventional implementations of service-oriented architectures have been realized in terms of Web Services (architectures) [3]. In a typical Web Services scenario, providers of services are not aware of clients that have located and are making use of their services. Hence, intended adaptations to public Web Services deployed on a provider host (e.g. changing the WSDL description) cannot be communicated with any depending client. The DEVOLVE platform enables consumer peers to subscribe into a list of depending peers that is maintained by each service providing peer. The service providing peer is then capable of instantly analyzing all registered dependencies and of contacting these dependent peers in case of any conflicting changes. The ability of peers to organize themselves into self-governed peer groups constitutes one of the distinguishing characteristics of peer-to-peer architectures. The concept of self-organization has mainly been implemented by the JXTA framework [4] making it possible for peers to detect existing peer groups, to apply for a group membership and finally to join a group. Recent projects have utilized this framework for instance for building student forum systems [5]. The JXTA framework typically supports peer group

organization with respect to functional properties (e.g. retrieving and sharing documents). In our approach, we adopt the peer group idea for enabling peers to agree upon a common adaptation strategy and, thus, a non-functional property. The management of dependencies between distributed services has been studied by different authors [6] [7] [8]. In accordance to our work, all approaches represent dependencies in a graph-like structure, thus allowing the analysis of transitive dependencies. Ensel et al. [6] provide an extensive catalogue of possible parameters and metrics for rating dependencies. Our approach adopts the idea of rating dependencies to allow a fine-grained analysis of dependencies. In contrast to Ensel's work, we provide more accurate meanings (semantics) of the applied parameters. All mentioned approaches assume global and fixed strategies and conditions for reacting on dependencies. In our approach, however, these strategies and conditions can be formulated dynamically in terms of adaptation policies by peer groups. Thus, peer groups are able to react to new or updated conditions in their self-governed application environment.

3 The DeEvolve Platform

The DeEvolve peer-to-peer architecture is a runtime environment for deploying component-based services. These services consist of local and remote components. Remote components serve as an interface for external peer services that are provided and deployed by other third-party remote peers. Any composition can be published as a peer service that can in turn be discovered and integrated by other consumer peers. A peer adopts two different roles within a peer architecture: the role of a provider and that of consumer of peer services.

DeEvolve supports two different types of compositions, orchestration and choreography composition (figure 1). Orchestration corresponds to the conventional client-server architectural style, but with more than one server (i.e. service providing peer) involved. Choreography composition allows for workflow-like structures with several peers transitively connected and without a single

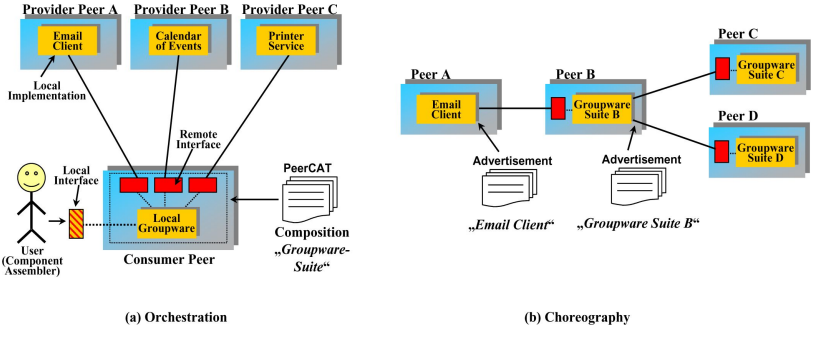


Fig. 1. Composition Types supported by DeEvolve

established control peer. We have developed a new composition language called PeerCAT [1] for describing the composition of services to new applications. A description of a composition includes meta-information about the structure of a composition, about the involved components and the dependencies among them. We build tools like the DeEvolve Console that allows to define compositions in both textual and graphical ways.

DeEvolve is built on top of the JXTA framework [4], in particular to realize fundamental mechanisms for publishing and discovering peer services within a peer-to-peer network. For the actual description of peer services, so-called advertisements are used, which contain any necessary information to bind the peer service (e.g. network address, port number) as well as an unstructured text used to envision the semantics of a distinct peer service. These advertisements are published to a number of well-known rendezvous peers which are in the scope of the providing peer (see [4] for more information of these discovery mechanisms). DeEvolve also adopts JXTA's peer group concept enabling peers to organize into self-governed peer groups despite any network boundaries. Like peer services, peer groups are described by advertisements which are used to announce the existence of peer groups. Having located these advertisements, peers are capable of applying and of joining to that group.

DeEvolve realizes a API to provide operations for adapting single components as well as compositions of local and remote components (peer services) during runtime. The idea is to adopt operations for the creation of component-based applications (e.g., setting parameters of components, adding components, connecting components) also for the adaptation of these (e.g. changing the connection of two components, removing components). The incentive for adapting compositions is provoked either by user-triggered demands or by exceptions that have occurred in the environment (e.g. unavailability of provider peers). DeEvolve is responsible to detect exceptions and to determine the consequences to affected local components. Handler routines can be defined for establishing reactions to handle occurred exceptions. These routines can either be executed autonomously by DeEvolve or in interaction with an operator.

4 Dependency Analysis in DeEvolve

This section elucidates the approach on analyzing consumer dependencies in a service-oriented peer-to-peer architecture. The model that our approach is based on presumes two assumptions. The first assumption implies that any consumer peer can subscribe to a list maintained by a provider peer if the peer relies on a public service offered by that provider. If an adaptation is planned, the operator of a provider peer is able to consult all subscribed peers before the adaptation can be carried out. The second assumption states that peers of a peer group have agreed to a common adaptation policy in the run-up to the adaptation of a service. The purpose of a policy is to clarify how a provider peer should handle existing dependencies if an adaptation to a service is planed.

4.1 Registration of Consumer Dependencies

The consumer of a peer service is able to determine the dependency value of each used peer service and to register this value to the provider of the consumed peer service. The attribute can hold one of the following five values:

No_Dependency. The peer service has a dependency on a consumed service, but yet no concrete importance value has been determined.

Interest_Dependency. The peer operator has indicated a consumable peer service as an interest service. An interest service is not directly used by local components, but is designated for a later usage or for an upcoming composition with local components towards a new peer service.

Low_Functional_Dependency. The peer operator has composed a consumable peer service with other local components towards a new peer service that is used directly by the peer operator. The importance of this functional dependency as low.

Strong_Functional_Dependency. The operator has composed a consumable peer service with local components towards a new peer service that is used directly by the operator. The importance of this dependency is strong.

Transitive_Functional_Dependency. The operator has composed a peer service with other local components towards a new peer service that has also been located and used by other third-party peers. So, transitive dependencies exist within the topology of the peer-to-peer architecture.

These dependency values are sorted according to the impact the dependent peer service has on the local peer service: while we assume that the impact of the first dependency value (No_Dependency) is low, the impact of the last value (Transitive_Functional_Dependency) is quite high. Consequently, an unheralded adaptation of a peer service with one transitive functional dependency would cause the most critical violations of behaviour in a peer-to-peer network.

Dependency values are determined and transferred automatically to the respective provider peer of a service. For instance, the indication of a located peer service to become an interest service within the peer operator's environment causes the triggering of a notification message to the provider peer. During the composition of an interest peer with local components to a new service, the value is updated. The operator can choose between a strong and a weak dependency value. If a third party peer has located and used this new service in its local environment, the value is updated to a transitive functional dependency.

4.2 Adaptation Policy

An adaptation policy is represented as an aggregation of two concepts, that is, an adaptation condition and an adaptation strategy. An adaptation strategy denotes an explicit procedure describing how a service providing peer has to proceed in case of dependent consumer peers. An adaptation condition dictates

when a selected strategy can be executed. We distinguish between four different types of adaptation strategies:

Conservative Adaptation an adaptation can only be executed if no dependencies are available. This strategy presumes no consultation with dependent peers. It can be applied to application with high demands on availability (24x7) and reliability or with only little maintenance support.

Negotiation an adaptation can only be carried out if all dependent consumer peers have been consulted. Moreover, all consulted peers have to acknowledge the adaptation request. This strategy can be utilized in application scenarios in which many high-value dependencies on consumer peers can be expected.

Notification an adaptation can only be carried out if all dependent consumer peers have been notified in advance. Note that there is no approval necessary. This strategy can be applied in structures with high maintenance support where a circumventive reaction to adaptation requests can be expected.

Liberal Adaptation an adaptation can be carried out directly without any notification of or consultation with dependent peers. This strategy can be applied to scenarios with only little or even no degree on decentralization.

Adaptation strategies preceding negotiation or notification assume communication between the service provider and all pertaining consumers. Negotiation requires the establishment of a bi-directional channel (like a synchronous chat) between provider and consumers. Such channel can be used many times in either direction to debate on details of the planned adaptation.

The effectuation of an adaptation strategy may be varied through the insertion of an adaptation condition. An adaptation condition can be formulated in terms of both the number of current dependencies on the consumable service and the potential dependency values available for these dependencies. Also, a target variable tv is to be declared in order to state when a condition is fulfilled or not. An adaptation condition can also be expressed as a function that takes all registered dependent remote peer services rs that are dependent on a consumable service ls provided by a peer i, whereas all dependent services and the local service belong to peer group m (PS denotes the set of all peer services):

$$cond_m \ : \ PS^n \times \mathbf{R} \rightarrow (true, false),$$
$$cond_m(ls^m_{i,j}, rs^m_{i,j,1}, ..., rs^m_{i,j,n}, tv_m) = \begin{cases} true & \text{if condition is fulfilled} \\ false & \text{else} \end{cases} \quad (1)$$

Function $cond_m$ denotes the adaptation condition that was previously defined and prescribed by a peer group with index m. The term $rs^m_{i,j,k}$ denotes the $k-th$ of n remote services that are dependent on the $j-th$ consumable peer service of peer with index i. The function itself is able to internally apply various auxiliary functions to see if the condition is fulfilled or not. For instance, the function $value(rs^m_{i,j,k})$ returns the corresponding (numerical) dependency value of the depending remote service $rs^m_{i,j,k}$. Function $count(ls^m_{i,j})$ returns the total number of dependent remote services for a local service $ls^m_{i,j}$. The following example of

an adaptation condition computes a weighted average of dependency values. If the result is greater than tv, the condition is true:

$$cond_m(ls^m_{i,j}, ..., tv_m) = \begin{cases} true & \text{if} \frac{1}{count(ls^m_{i,j})} \sum_{k=1}^{n} value(rs^m_{i,j,k}) \geq tv_m \\ false & \text{else} \end{cases} \quad (2)$$

Note again that the adaptation condition states whether or not the defined strategy is valid. For instance, if *cond* function in (2) delivers true as a result, the strategy selected by a peer group can be put into effect. For strategies Notification, Negotiation, and Conservative Adaptation we assume the rule that if the condition is false, the next lower strategy can be executed. The function in (3) models the entire adaptation policy that takes the result of the condition (1) and the chosen strategy to compute which strategy is valid for a given application scenario. We assume that each strategy has an index indicating its priority (conservative (3) to liberal adaptation (0)):

$$policy_m : (true, false) \times \mathbf{N} \rightarrow \mathbf{N},$$
$$policy_m(cond_m, strat_m(i))) = \begin{cases} strat_m(i) & \text{if } cond_m(..) = true \\ strat_m(i-1) & \text{if } cond_m(..) = false \wedge i \geq 1 \end{cases} \quad (3)$$

For a consumer peer service that associates many peer groups with adaptation policies and, thus, strategies of different order, the strategy with the highest order has preference during analysis.

4.3 Prototypical Implementation

We implemented a first prototype of our analysis model for the DeEvolve platform. Once a composed application is published as a peer service, consumer peers can subscribe to the pertaining provider peer. Before an adaptation of a

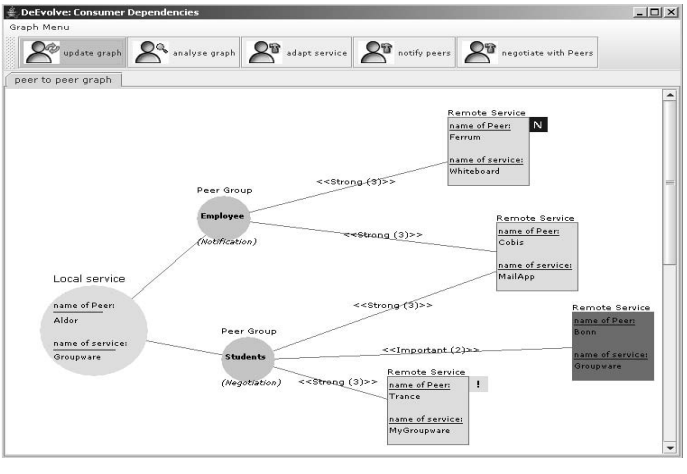


Fig. 2. Snapshot of the Visualization Tool for Dependency Analysis

peer service is executed, the peer provider has to generate a graphical visualization of existing dependencies to consumer peers with the DeEvolve Analysis tool (figure 2). Starting from the left node representing the local peer service, all dependent remote consumer peer services are visualized on the right side. Consumer peers are assigned to the respective peer groups they belong to. Attached to each peer group node is the adaptation strategy imposed by the group. The annotations on the edges denote dependency values that declare the importance of a dependency between provider and consumer. Based on the current visualization of the dependency graph, the peer operator can run an analysis of the dependencies to determine if an adaptation can be executed. In the example of figure 2, the policy of peer group Employee allows for adaptation after notification, while the policy of group Students requires negotiation with all dependent peers. The overall result points out that the adaptation cannot be carried out, as there is at least one policy that does not allow an immediate adaptation.

5 Conclusion

In this paper a new approach for handling consumer dependencies in service-oriented peer-to-peer architectures has been proposed. Apparently, this approach can be enhanced by existing reputation models. A reputation model could thereby be utilized, to make assumptions about the continuity a peer is providing its services. Any intended violation of consumer dependencies could limit the reputation of a provider in a given community.

References

1. Alda, S., Cremers, A.B.: Strategies for component-based self-adaptability model in peer-to-peer architectures. In: Proc. of 4th International Symposium on Component-based Software Engineering (CBSE7), Springer (LNCS 3054) (2004) 59–67
2. Alda, S., Cremers, A.B.: Towards composition management for peer-to-peer architectures. In: Proc. of Workshop Software Composition (SC 2004), affiliated to the 7th European Joint Conference on Theory and Practice of Software (ETAPS 2004). (2004) 59–67
3. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., , Weerawarana, S.: Unraveling the web services web: An introduction to soap, wsdl, and uddi. IEEE Internet Computing **6** (2002) 86–93
4. Sun: Jxta v2.0 protocols specification (2003) URL: http://spec.jxta.org/v2.0/.
5. Halepovic, E., Deters, R.: Building a p2p forum system with jxta. In: Proc. of 2nd IEEE International Conference on Peer-to-Peer Computing (P2P2002). (2002)
6. Ensel, C., Keller, A.: Managing application service dependencies with xml and the resource description framework. In: Proc. of 7th International IFIP/IEEE Symposium on Integrated Management (IM 2001). (2001)
7. Hasselmeyer, P.: Managing dynamic service dependencies. In: Proc. of 12th International Workshop on Distributed Systems: Operations / Management (DSOM' 2001). (2001)
8. Kon, F., Campbell, R.H.: Dependence management in component-based distributed system. IEEE Concurrency **8** (2000)

LEAP-DB: A Mobile-Agent-Based Distributed DBMS Not Only for PDAs

Peter Ahlbrecht and Andreas Bothe

Technical University of Braunschweig, Institute of Information Systems,
P.O.box 3329, D-38023 Braunschweig, Germany
{p.ahlbrecht|andreas.bothe}@tu-bs.de

Abstract. Mobile devices are subject to severe constraints like restricted IO, processing and storage capabilities or intermittent connectivity. These issues of course also pertain to applications running on the mobile hardware, and for instance with regard to processing and displaying vast amounts of data or the special requirements imposed by supporting transactions, database and information systems are particularly affected. Mobile agents have frequently been suggested as a remedy for at least some of these problems, but only few implementations actually utilising them are known. In this paper, we present the design and implementation of a distributed database system applying mobile agent technology to support mobile devices. In particular, being based on the agent platform JADE-LEAP, it allows a P2P usage of PDAs, which is so far not supported by major DBS vendors.

1 Introduction

For the portable types of mobile hardware, from notebook computers and Personal Digital Assistants (PDAs) to smart and mobile phones, functionality related to information systems—like accessing a database from or running one on them—is becoming more and more popular. However, though these devices continuously gain more power, they still show severe constraints when compared to stationary computers, and IS-related software is particularly vulnerable to these constraints [1], [2]. The mismatch between restricted resources and the need to obtain or process a large set of data is obvious. Furthermore, data-intensive applications and distributed systems relying on an unstable communication network with low bandwidth are said to have a special need for replication techniques [3], and in order to overcome unreachableness of mobile devices or to allow for better performance when being faced with long-lived transactions, several transaction models have been suggested which relax some of the ACID properties commonly associated with database transactions; cf. [4] for an overview on these models.

Mobile agents—software objects at user level which may migrate from one computer in a network to another, provided a suitable runtime environment is available on those nodes—have frequently been suggested as a means to overcome at least some of these obstacles [5], [6]. In moving to and computing the result at the place where the data is stored, they reduce processing and storage

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 203–210, 2007.

requirements at the mobile device, and by doing this in an asynchronous way they relax the issue of intermittent connectivity: once dispatched to the target, they work autonomously and wait to return until a connection to their source becomes available again.

However, despite these convincing theoretical advantages, only very limited practical implementations utilising mobile agents for information/database systems with mobile devices seem to be available. To the best of our knowledge, so far only the prototype described in [7] realises this combination, based on the Aglets mobile agent platform. Our approach also employs mobile agents, but is based upon JADE-LEAP [8], [9], which compared to Aglets comes with a built-in support for mobile hardware like cellular phones and PDAs. It therefore also allows a peer-to-peer (P2P) usage on PDAs, while major vendors of database systems like IBM, Oracle or Microsoft by now only support a mobile client and a stationary backend-server, or a stand-alone database application on the mobile device [10], [11].

This paper is organised as follows: Section 2 presents some background material on issues pertaining to distributed database systems and the agent platform underlying our implementation. Section 3 details the design and section 4 covers aspects of the implementation of the system. Finally, section 5 concludes this survey and points out future work.

2 Databases on Mobile Devices and JADE-LEAP

From the large variety of different types of mobile devices available today, our focus is on those which can be carried around by a user and support access to or running a database system on it. We are particularly interested in PDAs because of their increasing popularity and the upcoming database-system-related functionality provided on them: As already mentioned in the introduction, major vendors have begun to develop "light" versions (e.g. DB2 Everyplace by IBM, Oracle Database 10g Lite by Oracle, SQL Server CE by Microsoft) of their database system software, which can be used on mobile devices as clients to access backends on stationary servers, and open source projects (eXtremeDB, hSQLDB, etc.) started working on database systems usable as stand alone applications on PDAs [11]. Our aim, however, is to realise a distributed database system which allows PDAs to be used as peers, either on their own or in conjunction with notebook and stationary computers, where we pick up the definitions from [1] of a *distributed database* (DDBS) as "a collection of multiple, logically interrelated databases distributed over a computer network", and a *distributed database management system* (DDBMS) as "the software system that permits the management of the DDBS".

As already mentioned, mobile agents ultimately need a runtime environment supplied by *hosts,* which conform to some agent API or agent system. They enable creation, execution and migration of as well as communication between agents and other hosts. Standardisation efforts by the Object Management Group (OMG) and the Foundation for Intelligent Physical Agents (FIPA) regarding the

infrastructure and provided actions have led to the *Mobile Agent System Interoperability Facility (MASIF)* specification [12] and the *FIPA* set of standards [13], respectively, where the latter is concerned with agent-related topics (including migration) from the area of Artificial Intelligence, while the former focuses on mobile agents in particular. In order to exploit mobility, usually a number of hosts will be used to provide a distributed agent environment.

The *Java Agent DEvelopment Framework (JADE)* [8] is on the one hand a FIPA-compliant software framework for developing agent systems, on the other hand it provides a runtime environment for such agent applications [14]. With regard to the latter, an instance of the runtime environment is referred to as a "container", and several containers from one or more computers can be used to create a distributed agent platform. The container instantiated as the first one of a platform, the so-called "main container", has to be active and available throughout the existence of the platform, and all other containers joining the platform have to register with it. Avoiding to do so will make a newly instantiated container become another main container, and thereby also instantiate another platform. Communication between agents on different platforms is possible, but migration of agents is restricted to hosts of the same platform. JADE is entirely implemented in Java, with the minimal system requirements being version 1.4 of its runtime environment or the JDK.

As several agent development environments were available, but none was capable of running on mobile devices with restricted resources, the objective of the *Lightweight and Extensible Agent Platform (LEAP)* project was to bridge this gap [9]. This objective was realised by providing a set of libraries that can be used to modify parts of the JADE kernel—resulting in three different profiles, which can then be used to develop applications for certain classes of devices: 1. The *midp profile* is targeted towards devices offering the Mobile Information Device Profile of the Java 2 Micro Edition, frequently found for instance on Java enabled mobile phones. 2. The *pjava profile* is to be used with hardware supporting PersonalJava, which includes many types of PDAs. 3. The *j2se profile* plays the LEAP counterpart to plain JADE. It is geared towards stationary PCs and servers, and can be used with the other profiles within the same platform.

3 LEAP-DB Design

We now describe the design of our DDBMS based on the JADE-LEAP agent platform. Due to space restrictions, our focus will be on schema integration for heterogeneous databases.

The system should be usable from mobile as well as stationary devices linked via wireless or wireline connections. Due to the selection of the agent platform JADE-LEAP as the basis for the implementation, exactly one of the devices has to run the platform's main container, while all others will participate by joining with a simple container. An autonomous database plus the corresponding DBMS may exist on any device, but it is also permissible to use a client solely for querying—without itself providing a data source.

The system should allow formulating global queries and running global transactions, but prior to be able to do so first a global database schema has to be determined. All the tasks are carried out by various stationary and mobile agents. Re-using the concepts developed in [7], the stationary types of agents are the *managerAgent,* the *wrapperAgent* and the *transactionManager,* while the *brokerAgent* and the *runnerAgent* are of the mobile kind.

The *managerAgent* acts as the main coordinator on a device, governing interactions with the user, creation and termination of mobile agents, and communication between brokerAgent or runnerAgent on the one hand and wrapperAgent on the other. In addition to this, the managerAgent running within the main container is also responsible of integrating the individual local schemata into the global schema and distributing the latter to the various participants. The *wrapperAgent* provides access to the local data. For reasons of simplicity, we assume a relational database system to be available on every device—noting, however, that it is also possible to store data in other formats like files with comma separated values or XML-structures [15]. Accordingly, we strive to implement a *jdbcWrapperAgent,* which will provide access to the local data using the JDBC interface. The *transactionManager* is created by the managerAgent and will be in charge of splitting a global transaction into subtransactions, which will then be carried out by the mobile *runnerAgents* governed by the former. Finally, the *brokerAgents* are mainly needed in the process of forming the global database schema. Upon joining the system, the managerAgent on a device will create a brokerAgent and provide it with the information concerning the local database schema. The brokerAgent then migrates to the main container, forwarding its schema information to the global managerAgent, who then integrates this schema into the global schema and returns the new global schema to the brokerAgent. The latter then returns to the source device, while the global managerAgent creates brokerAgents which will update the schema information at the other participants. Fig. 1 illustrates this design with a possible scenario including two PDAs and a stationary computer by a deployment diagram in the Agent UML [16], an extension of the Unified Modeling Language (UML) intended to support the development of agent systems.

As our system was to be based on agents, using an ontology for exchanging the schema information seemed natural. We designed a DatabaseOntology, whose

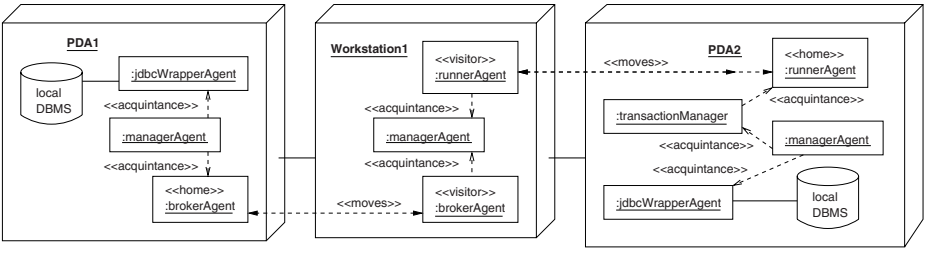


Fig. 1. Scenario in Agent UML illustrating the agents making up the system

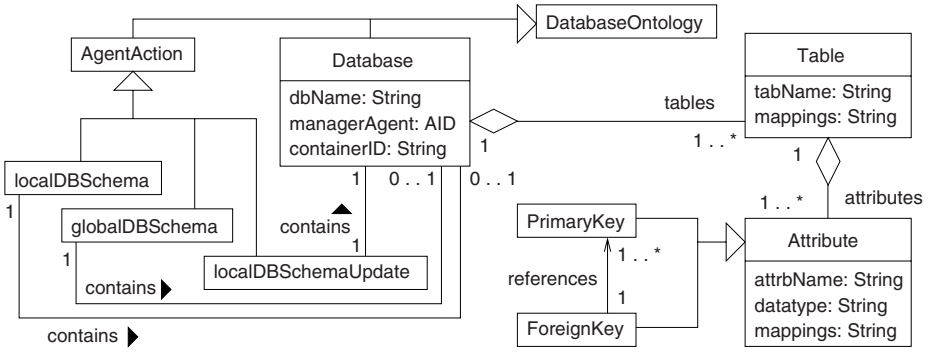


Fig. 2. DatabaseOntology for exchange of database schema information

structure is shown in fig. 2. Unsurprisingly, a database is identified uniquely by its name, the managerAgent in charge of it and the container it lives on. A database may comprise one or more tables, which in turn may consist of one or more attributes. The latter may possibly function as primary or foreign keys. In addition, tables and attributes are also characterised by their respective names, and attributes furthermore by their datatypes. The attribute *mappings* is only used in descriptions of the global database schema, providing the origin of the object it qualifies.

The ontology also contains agent actions, which tell an agent how to handle a message it just received. To be precise, an agent action forms one part of a message exchanged between agents. The format of the messages in the FIPA compliant *Agent Communication Language (ACL)* is standardised to contain the following fields:

- the *sender* and *receiver* of the message
- the so-called *performative,* indicating the sender's intention—examples being: a REQUEST, denoting that the sender would like the receiver to execute some action; an INFORM, signalling that the receiver is to be informed about something; etc.
- the *contents* as the actual information of the message—depending on the performative this could be, e.g., facts in conjunction with an INFORM, the action to be executed for a REQUEST, etc.
- the *language,* determining the syntax for the message contents
- the *ontology,* providing the vocabulary and semantics for the symbols
- additional fields for determining the flow of control

Owing to the agent-based-approach, we devised several interaction protocols in which the messages and agent actions are used; fig. 3 illustrates the protocol for the construction of the global database schema, which will be explained thoroughly in section 4. The diagram is again given in the Agent UML, this time by a protocol diagram [17], where the important points to note with regard to the conventional UML are that these diagrams combine sequence and state diagrams, that a life line or message can be split up in order to allow the description of

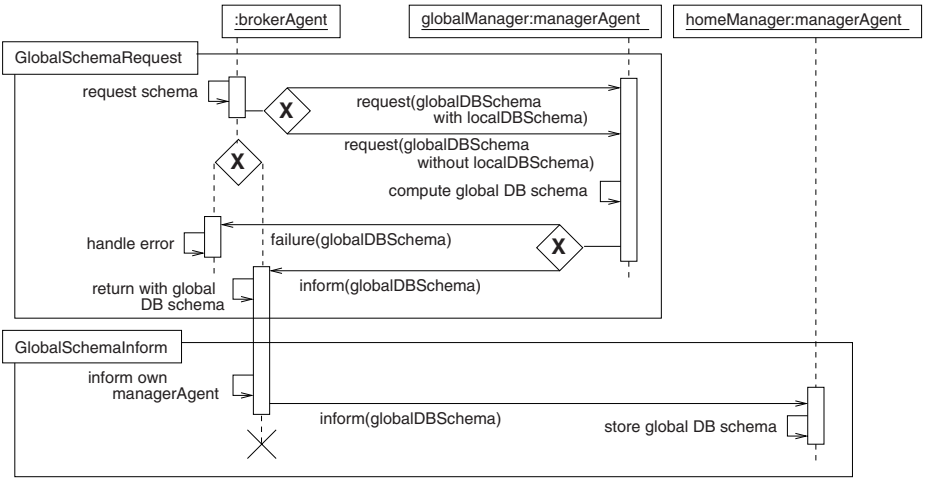


Fig. 3. Interaction protocol in Agent UML

parallel or alternative behaviours—indicated by a fork bar or a diamond which may show an "X" to describe AND, OR or eXclusive-or branching, respectively—and that they can be packaged to facilitate re-use.

4 Aspects of the Implementation

In this section, we describe the implementation of the schema integration, as it is the first item to be considered in a DDBMS unless the entire system including the databases can be designed from the very beginning, which will rarely be the case for mobile systems.

Our algorithm for the schema integration is build upon the work by Lawrence and Barker [18]. However, in order to ease a prototypical implementation, we so far avoided realising the dictionary suggested by them, and made the following simplifying assumptions with regard to the different types of conflicts which may occur during integration [2]: tables and attributes from different databases representing the same aspect of the universe of discourse are identified by the same name ("no descriptive conflicts"); attributes representing the same aspect of the universe of discourse use the same data type and range of value ("neither type, nor value conflicts"); and integrity constraints and user rights are so far not considered.

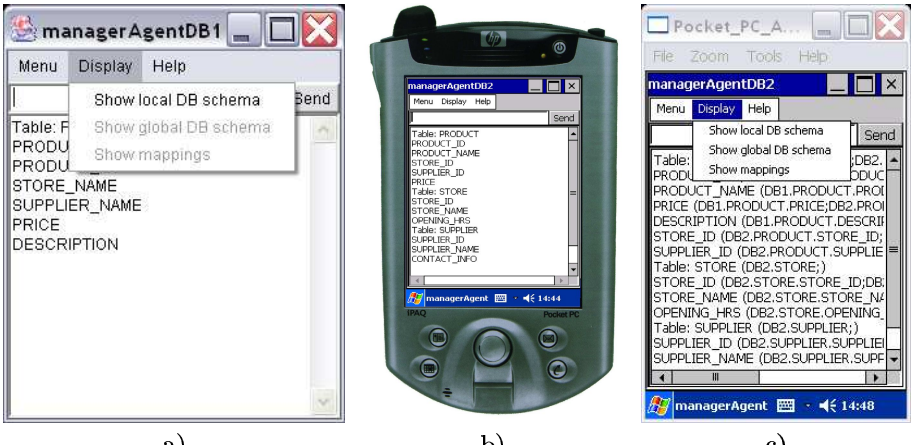
As described in section 3 and illustrated in fig. 3, the algorithm is carried out by the global managerAgent when receiving the request about the database schema from a brokerAgent, which itself may or may not provide a local schema. In case of the latter, the algorithm works on an empty DatabaseOntology and the global one as the input parameters.

For every table of the schema to be integrated, it first has to be checked whether a table with the same name already exists in the global schema. If this is not the case, the table is included into the global schema and a corresponding

mapping is added to the latter. The mapping, hidden from the user, stores the origin of the table, allowing to determine the required sources for answering queries later on. The attributes of the table are processed in the same manner. They can be added straightforwardly to the newly created table of the global schema, but attention has to be paid when creating the corresponding mappings: for attributes bearing a foreign key, on the one hand also a link to the referred attribute has to be added, on the other hand the mapping of the referred attribute has to be adjusted. The adjustment, however, is postponed until the end of the algorithm, as the table containing the referenced attribute may not have been added to the global schema so far.

If a table with the same name already exists, the mapping for the table is added in the same way. Attributes, however, are treated differently. For an attribute without a foreign key but with an existing attribute of the same name, only the mapping is added. If an attribute is not contained within the table under consideration, but in a table referenced by an already included attribute, then the mapping of the new attribute is only added to the table holding the included one. If the attribute so far does not occur in the global schema, it is added to the table together with the corresponding mapping.

Fig. 4 illustrates this process for a scenario corresponding to the one modelled in fig. 1 with PDA1 replaced by a notebook computer. The global managerAgent runs on a stationary PC. A database "DB1" stores the data on the notebook, and the one on the PDA (= PDA2 in fig. 1) is named "DB2", both being administered by hSQLDB. With the managerAgents realising the user interface, fig. 4 a) and b) show the schemas of the local databases used on the notebook and PDA, respectively, prior to integration, while fig. 4 c) depicts the global schema at the PDA after integrating both mobile devices into the system. The mappings added to the tables and attributes indicate the successful construction of the global schema.



a) b) c)

Fig. 4. Screenshots taken during schema integration; c) taken from a redirected display of the PDA in order to increase readability

5 Conclusion

With this paper, we showed that is possible to realise a distributed database management system for mobile devices like notebook computers and PDAs on top of mobile agents. The emphasis was on presenting the overall design of the system, but aspects of the implementation based on the JADE-LEAP agent platform were also discussed, namely the schema integration utilising various agents and an algorithm for the metadata management. Of course, realising large parts of the implementation, e.g. the transaction management, remains as future work. Furthermore, as security is a major issue particularly in the area of mobile technology, investigating how to include the security add-on JADE-S into the system would be highly interesting.

References

1. Öszu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Prentice Hall (1999).
2. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. Addison-Wesley (2000).
3. Huang, Y., Sistla, P., Wolfson, O.: Data Replication for Mobile Computers. In: Proc. 1994 ACM SIGMOD Int. Conf. on Management of Data. (1994) 13–24.
4. Seydim, A. Y.: An Overview of Transaction Models in Mobile Environments. Internet (2002). `http://engr.smu.edu/~yasemin/mobile_trans.pdf`
5. Chess, D. et al.: Itinerant Agents for Mobile Computing. In Huhns, M.N., Singh, M.P., eds.: Readings in Agents. Morgan Kaufmann (1997) 267–282.
6. Brewington, B. et al.: Mobile agents in distributed information retrieval. In Klusch, M., ed.: Intelligent Information Agents. Springer (1999) 355–395.
7. Brayner, A., Filho, J.: Sharing Mobile Databases in Dynamically Configurable Environments. In: CAiSE 2003. LNCS 2681 (2003) 724–737.
8. JADE. Java Agent DEvelopment Framework. `http://jade.tilab.com/`.
9. Bergenti, F., Poggi, A.: LEAP: A FIPA Platform for Handheld and Mobile Devices. In: ATAL 2001. Proc. 8th Int. Workshop. LNCS 2333 (2002) 436–446.
10. Laberge, R., Vujosevic, S.: Building PDA Databases for Wireless and Mobile Development. Wiley (2003).
11. Mutschler, B., Specht, G.: Mobile Datenbanksysteme. Springer (2004). In German.
12. MASIF. OMG Doc. orbos/98-03-09: `ftp.omg.org/pub/docs/orbos/98-03-09.pdf`.
13. FIPA. The Foundation for Intelligent Physical Agents. `http://www.fipa.org/`.
14. Bellifemine, F., Poggi, A., Rimassa, G.: JADE: A FIPA2000 Compliant Agent Development Environment. In: Agents'01. Proc. 5th Int. Conf. on Autonomous Agents. LNCS 1222 (2001) 216–217.
15. Ahlbrecht, P., Röver, J.: Specification and Implementation of Mobile-Agent-Based Data Integration. In: Databases and Information Systems II, Kluwer Academic Publisher (2002) 269–283.
16. Manson, G. et al: FIPA Modeling Area: Deployment and Mobility. Internet (2003). `http://www.auml.org/auml/documents/DeploymentMobility.zip`.
17. Odell, J., Parunak, H.v.D., Bauer, B.: Representing Agent Interaction Protocolls in UML. In Wagner, G., ed.: Proc. of AOIS-2000 at CAiSE*00. (2000) 29–40.
18. Lawrence, R., Barker, K.: Automatic Integration of Relational Database Schemas. Technical Report 2000-662-14, University of Calgary (2000)

Models and Languages for Overlay Networks

Stefan Behnel and Alejandro Buchmann

Databases and Distributed Systems Group,
Darmstadt University of Technology (TUD), Germany
{behnel,buchmann}@dvs1.informatik.tu-darmstadt.de

Abstract. Implementing overlay software is non-trivial. In current projects, overlays or frameworks are built on top of low-level networking abstractions. This leaves the implementation of topologies, their maintenance and optimisation strategies, and the routing entirely to the developer. Consequently, topology characteristics are woven deaply into the source code and the tight coupling with low-level frameworks prevents code reuse when other frameworks prove a better match for the evolving requirements.

This paper presents *OverML*, a high-level overlay specification language that is independent of specific frameworks. The underlying system model, named "Node Views", abstracts from low-level issues such as I/O and message handling and instead moves *ranking nodes* and *selecting neighbours* into the heart of the overlay software development process. The abstraction decouples maintenance components in overlay software, considerably reduces their need for framework dependent source code and enables their generic, configurable implementation in pluggable EDSM frameworks.

1 Introduction

Recent years have seen a large body of research in decentralised, self-maintaining overlay networks like P-Grid [1], ODRI [2], Chord [3] or Gia [4]. They are commonly regarded as building blocks for Internet-scale distributed applications.

Contrary to this expectation, current overlay implementations are built with incompatible, language specific frameworks on top of low-level networking abstractions. This complicates their design and prohibits code-reuse in different frameworks. It also hinders the comparison and exchangeability of different topologies within an application.

Our recent work [5] promotes a data abstraction as a much cleaner foundation for the implementation of overlay software. It decouples components for routing and maintenance and enables abstract, framework independent specifications.

This paper presents OverML, a new set of languages that were specifically designed for the framework independent specification and implementation of adaptable overlay networks. Section 2 briefly overviews the underlying data driven system model that is more thoroughly explained and motivated in [5]. The remaining sections then describe OverML, a set of abstract XML specification languages for the major parts of overlay implementations: topology specifications, event flows, messages and node attributes.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 211–218, 2007.

2 Node Views, the System Model

We model overlay software as data management systems by applying the well-known Model-View-Controller pattern [6]. It aims to decouple software components by separating the roles for data storage (model), data presentation (views) and data manipulation (controllers).

In our case, the *model* is an active local database on each node, a central storage place for all data that a node knows about remote nodes. The major characteristics of the overlay topology are then defined in *node views* of the database. They represent sets of nodes that are of interest to the local node (such as its neighbours). Different views provide different ways of selecting and categorising nodes, and therefore different ways of adapting topology characteristics to application requirements.

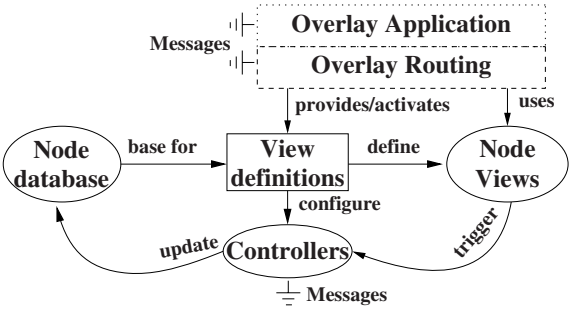


Fig. 1. The system model

The *controllers* are tiny EDSM states that are triggered by events like timeouts, incoming or leaving messages or changes in the views. They perform simple maintenance tasks like updating node attributes when new data becomes available or sending out messages to search new nodes that match the view definitions.

Controllers and other overlay components like message handlers or routers use node views for their decisions. Database and views decouple them from each other and simplify their design considerably. Even more so, as this architecture can provide powerful operations like selecting and adapting topologies with a single view selection command. The abstract view definition becomes the central point of control for the characteristics of the overlay.

3 OverML, the XML Overlay Modelling Language

We propose the XML Overlay Modelling Language *OverML* for specifying the four portable parts of overlay software: node attributes, messages, view definitions and EDSM graphs. Because of space limitations, only the first three are presented here. Schema definitions are provided at `http://www.dvs1.informatik.tu-darmstadt.de/research/OverML/`, which is also the XML namespace that we abbreviated in the examples.

3.1 Slosl, the View Specification Language

The view definitions implement adaptable topologies which makes them the key components in overlay software. They are expressed in Slosl, the SQL-Like Overlay Specification Language. As the XML representation of Slosl is relatively straight forward (using Content MathML [7]), we will stick to the more

readable representation. We start with a simple example, an implementation of an extended Chord graph [3].

```
1 CREATE VIEW chord_fingertable
2 AS SELECT node.id, node.ring_dist, bucket_dist = node.ring_dist−2^i
3 RANKED lowest(nodes+i, node.msec_latency / node.ring_dist)
4 FROM node_db
5 WITH log_k = log(𝒦), nodes = 1
6 WHERE node.supports_chord = true AND node.alive = true
7 HAVING node.ring_dist in (2^i : 2^(i+1))
8 FOREACH i IN (0:log_k)
```

While most clauses behave as in SQL, the new clauses RANKED and HAVING–FOREACH were added to provide simple statements for highly expressive overlay specifications. A more detailed description of the example follows, leaving out the obvious clauses CREATE VIEW and FROM.

SELECT. The interface of this view contains the attributes *id* and *ring_dist* of its nodes (ID and distance along the ring), as well as a newly calculated attribute *bucket_dist*.

RANKED. To support **topology adaptation**, the nodes in the created view are chosen by the ranking function *lowest* as the $nodes + i$ top node(s) that provide the lowest value for the given expression. Rankings are often based on the network latency, but any arithmetic expression based on node attributes or even user defined functions can be used.

WITH. This clause defines variables or options of this view that can be set at instantiation time and changed at run-time. Here, *log_k* will likely keep its default value, while *nodes* allows adding redundancy at runtime.

WHERE. Any SQL boolean expression based on node attributes can be used in this clause. It constrains nodes that are valid candidates for this view (**node selection**). Here we use an attribute *supports_chord* that is true for all nodes that know the Chord protocol. The second attribute, *alive*, is true for nodes that the local node considers alive.

HAVING–FOREACH. This pair of clauses aggregates valid candidates into buckets for **node categorisation**. In the example, the HAVING part states that the ID distance must lie within the given half-open interval (excluding the highest value) that depends on the bucket variable i. The FOREACH part defines the available node buckets by declaring this bucket variable over a range (or a list, database table, ...) of values. It can define a single bucket of nodes, but also a list, matrix, cube, etc. of buckets. The structure is imposed by the occurrence of zero or more FOREACH clauses, where each clause adds a dimension. Nodes are selected into these buckets by the HAVING expression (which is optional and defaults to true). A node can naturally appear in multiple buckets if the HAVING expression allows it.

The bucket abstraction is enough to implement graphs like Chord, Pastry, Kademlia or de-Bruijn in less than a dozen lines and should be just as useful for a large number of other cases. It is not limited to numbers and ranges, buckets can be

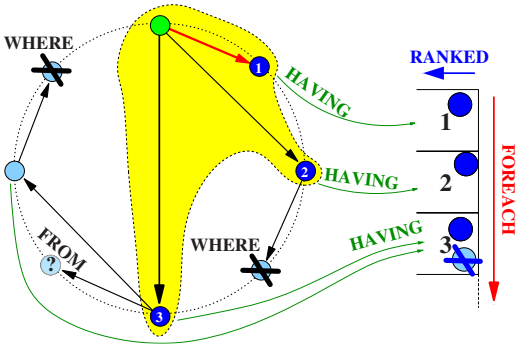


Fig. 2. Implementing the chord topology in SLOSL

defined on any list or even on a database table. Numbers are commonly used in structured overlays (which are based on numeric identifiers), while strings could be used for topic-clustering in unstructured networks.

The clauses FROM, WHERE, HAVING–FOREACH, RANKED and SELECT directly impact the nodes and attributes in the view. We will therefore explain their interaction semantics in terms of an execution order as illustrated in figure 2. Note, however, that SLOSL is a declarative language and that query optimisers and source code generators may handle specific SLOSL statements quite differently.

1. FROM selects all nodes from the parent view. Note that the database may be globally incomplete or outdated (as the missing node in figure 2 suggests).
2. WHERE restricts this set to nodes matching a boolean expression.
3. FOREACH sorts the selected nodes into buckets by evaluating the HAVING expression for each node and each value of the bucket variable(s).
4. RANKED restricts the maximum number of nodes in each bucket and selects only those nodes for which the ranking expression yields the best results.
5. SELECT finally selects the attributes of each node that are visible in this view. Note that SLOSL inherits SQL's powerful capability of calculating attribute values based on arbitrary SQL expressions. If bucket variables are used in these expressions, the same node can carry different attribute values in different buckets of the created view.

3.2 NALA, the Node Attribute Language

Attribute definitions can currently utilise the data types of XML Schema [9] or SQL [10].[1] For SQL types, OverML allows the definition of custom data types based on the predefined types as follows.

[1] The 2003 SQL/XML standard [11] defines a mapping between the two.

```
<types xmlns:sql="OverML/sql">
  <sql:composite   type_name="tcpaddress" /> <!--composite data type-->
    <sql:inet      name="address" />
    <sql:shortint  name="port" />
  </sql:composite>
  <sql:decimal type_name="id128" bits="128"/> <!--restricted decimals-->
  <sql:decimal type_name="id256" bits="256"/>
</types>
```

Line 6 and 7 define customised descendents of the normal **sql:decimal** data type that are restricted to a fixed size to represent node IDs. Note that for specific restrictions, some implementations may not support an exact mapping to storage types and may need to do range checking. The **sql:composite** meta type allows composing multiple simple types into one new structured type.

Any base type or custom type can be used for node attributes. Attributes have a name and a number of flags as shown in the following example.

```
<nala:attributes xmlns:nala="OverML/nala" xmlns:sql="OverML/sql">
  <nala:attribute name="id" type_name="id256" selected="true">
    <nala:static/> <nala:transferable/> <nala:identifier/>
  </nala:attribute>
  <nala:attribute name="knows_chord" type_name="sql:boolean"
                  selected="true">
    <nala:static/> <nala:transferable/>
  </nala:attribute>
  <nala:attribute name="latency" type_name="sql:interval"
                  selected="true" />
</nala:attributes>
```

For easier extensibility, all flags except '*selected*' are represented as XML elements. Their meaning is as follows:

identifier. The attribute uniquely identifies a node. If a node carries multiple identifiers, each one is treated independently as a unique identifier. This allows different levels of identification, most notably physical and logical addresses. Note that multiple types (like IP address and port) can be combined into a single new type, which can then be used as identifier.

static. The attribute is static and does not change once it is known about a node. All identifiers are implicitly static, but not all static attributes fulfil the uniqueness requirement of an identifier.

transferable. The attribute can be sent in messages. Some attributes (like network latency) only make sense locally and should be marked non-transferable.

selected. Selects the attribute for use in the database. Unselected attributes can reside in the specification without actually being used during execution. They can be dynamically activated at need, just like SLOSL statements.

3.3 HIMDEL, the Hierarchical Message Description Language

Messages combine attributes and other content into well defined data units for transmission. Their definition follows a hierarchy rooted in the top-level header, followed by a sequence of other headers and finally a sequence of content fields. Being an XML language, OverML presents this hierarchy in a natural way.

```
<msg:message_hierarchy xmlns:msg="OverML/msg" xmlns:sql="OverML/sql">
  <msg:container type_name="ids">
    <msg:attribute access_name="source" type_name="id" />
    <msg:attribute access_name="dest"   type_name="id" />
  </msg:container>
  <msg:header access_name="main_header">
    <msg:container-ref access_name="addresses" type_name="ids" />
    <msg:message type_name="join_request" />          <!--1st message-->
    <msg:message type_name="view_message">            <!--2nd message-->
      <msg:viewdata structured="true" access_name="fingertable"
                    type_name="chord_fingertable" />
    </msg:message>
    <msg:header>
      <msg:content access_name="type" type_name="sql:smallint" />
      <msg:message type_name="typed_message">         <!--3rd message-->
        <msg:content access_name="data" type_name="sql:text" />
      </msg:message>
    </msg:header>
  </msg:header>
  <msg:protocol access_name="tcp" type_name="tcp">
    <msg:message-ref type_name="view_message" />
    <msg:message-ref type_name="typed_message" />
  </msg:protocol>
  <msg:protocol access_name="udp" type_name="udp">
    <msg:message-ref type_name="join_request" />
  </msg:protocol>
</msg:message_hierarchy>
```

In this representation, a message becomes a path through the hierarchy that describes the ordered data fields (i.e. **content** and **attribute** elements) that are ultimately sent through the wire. Message data is encapsulated in the hierarchy of headers that preceed it along the path. Headers and their messages are finally encapsulated in a network protocol, apart from their specification. This makes it possible to send the same message through multiple channels and to decide the best protocol at runtime.

Multiple independent messages can be defined within the same header. Messages and headers branching away from a message path are completely ignored, i.e. the 'joined_message' in the example is not part of the 'view_message' and the second header is not part of any of them.

This means that the tag order on the message path is important. It describes the field order when serialising data, but it also defines the data fields that are actually contained in a message. If a header is extended by content or **container** elements after the definition of a message, the preceeding messages will *not* contain the successor fields, as they are not on its path. In the example, the 'view_message' will not contain the content field named 'type'. This field is, however, available in the 'typed_message' and all messages that are defined later under the same **header** tag.

As shown in the example, **container** elements can also be used as children of the **message_hierarchy** tag. Here, their definition is not part of the message hierarchy itself. They only predefine container modules for replicated use in headers and messages where they are referenced by their **type_name** attribute.

The Source code interface to messages and their fields. is defined using the **access_name** attribute. Accessing the fields of a message from an object oriented language should look like the following Python snippet.

```
def receive_view_message(view_message):
  net_address  = (view_message.tcp.ip, view_message.tcp.port)
  main_header  = view_message.main_header
  source_id    = main_header.addresses.source
  finger_nodes = view_message.fingertable
```

The following rules define the access paths. They allow for a concise, but nevertheless structured and well defined path to each element.

1. As the basic unit of network traffic, a message is always the top-level element.
2. Everything defined within the message becomes a second level element, referenced by its access name.
3. Entries within containers are referenced recursively, namespaced by the access name of their parent.
4. Following the path from the message back to the root header, all headers and the protocol become second-level elements, referenced by access name. Their child fields and container elements are referenced recursively as before. Children of nameless headers become elements of the message itself.

The message specification allows EDSM states to be triggered by framework-independent subscriptions to message names or header hierarchies. A simple subset of the XPath language [8] lends itself for defining these subscriptions. Note that even expensive abbreviations like '//' can be resolved at compile time or deployment time using the message specifications.

The network serialisation of messages depends on framework and language, whereas the specification above does not. There is a huge number of network representations for messages that are in more or less wide-spread use. In any case, the specified message hierarchy can directly be mapped to an XML serialisation format. But also in flat serialisations like XDR [12], mapping the message specification is straight forward when laying out the data fields depth-first along the message path. Depending on the attribute '*structured*', Views are serialised either as bucket structure (XML subtrees or XDR arrays) or as plain node data. The latter can avoid duplicate data if nodes appear unchanged in multiple buckets.

4 Conclusion, Current and Future Work

This paper presented OverML, a language for abstract overlay specification. Based on a Model-View-Controller architecture, it provides portable, framework-independent abstractions for the major components in overlay software. The achieved modularisation facilitates the development of generic components which enables pluggable development and integration of overlay systems.

The SLOSL language lifts the abstraction level for overlay design from messaging and routing protocols to the topology level. Its short, SQL-like statements meet the requirements for design-time specification, topology implementation and run-time adaptation of highly configurable overlay systems.

OverML and SLOSL make the design of the main characteristics of overlay software simple, fast, and independent of languages and frameworks. Our prototype implementation comprises a graphical editor for OverML specifications as well as a proof-of-concept runtime environment. Their combination moves the development of the remaining framework specific software components to the very end of the design process and supports it with a powerful and generic high-level API.

Future work will include better mechanisms for view and query optimisation. Our current PostgreSQL implementation maps SLOSL statements to rather complex, generic SQL queries. Building on the large body of literature on query modification and optimisation, we can imagine a number of ways to investigate for pre-optimising these statements. This is most interesting for recursive views and for merging view definitions when sending them over the wire.

We believe that high-level, integrative overlay design is an interesting new field that builds upon major achievements in the areas of databases, networking and software engineering. We would be glad to seed interest in new implementations of OverML compatible frameworks, SLOSL optimisers, source code generators, as well as possible mappings to existing frameworks.

References

1. Aberer, K.: P-Grid: A Self-Organizing access structure for P2P information systems. In: Proc. of the Sixth Int. Conference on Cooperative Information Systems (CoopIS 2001), Trento, Italy. (2001)
2. Loguinov, D., Kumar, A., Rai, V., Ganesh, S.: Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. [13]
3. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. of the 2001 ACM SIGCOMM Conference, San Diego, California, USA (2001)
4. Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S.: Making gnutella-like p2p systems scalable. [13]
5. Behnel, S., Buchmann, A.: Overlay networks - implementation by specification. In: Proc. of the Int. Middleware Conference (Middleware2005), Grenoble, France (2005)
6. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons (1996)
7. The World Wide Web Consortium: Mathematical Markup Language (MathML) Version 2.0 (Second Edition). (2003)
8. The World Wide Web Consortium: XML Path Language (XPath) Version 1.0. (1999)
9. The World Wide Web Consortium: XML Schema Part 2: Datatypes Second Edition. (2004)
10. ISO Document ISO/IEC 9075:2003: Database Language SQL. (2003)
11. ISO Document ISO/IEC 9075:14-2003: Database Language SQL - Part 14: XML-Related Specifications (SQL/XML). (2003)
12. Srinivasan, R.: XDR: External Data Representation Standard. RFC 1832 (Draft Standard) (1995)
13. The 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM). (2003)

A Peer-to-Peer Membership Notification Service*

Roberto Baldoni and Sara Tucci Piergiovanni

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Rome, Italy
{baldoni,tucci}@dis.uniroma1.it

Abstract. The problem of providing a peer with a good approximation of the current group membership in a peer-to-peer (p2p) setting is a key factor to the successful usage of any application-level multicast protocol (e.g. gossip based protocols). A p2p setting makes this problem hard to be solved due to the its inherent dynamic and asynchronous nature. This paper studies the problem of implementing a fully distributed, also called p2p, Membership Notification Service (MNS) which is able to handle any number of simultaneous join and leave while allowing reliable delivering of messages among peers which remain permanently alive inside the group.

1 Introduction

We are interested in studying group membership in very large scale peer-to-peer environments formed by processes sharing a common interest. In this setting, hundreds of thousands of processes communicate through application-level multicast protocols over an overlay network formed by the peers themselves [10,4]. This environment is inherently asynchronous and dynamic because peers continuously join and leave the system. This implies that the multicast protocol, to be effective, has to rely on a group membership service [1] to individuate at each point in time the set of intended peer receivers for each multicast message.

Group membership has been extensively studied in the literature in the context of group communications. Traditionally, group membership [5] supports process group communication [2] with the following two objectives [11]: (i) determining the set of processes that are currently up and (ii) ensuring that processes agree on the successive values of this set. These group membership approaches require "long enough" periods of time in which (i) no membership changes occur and (ii) the underlying system model shows a synchronous behavior [6]. The scale and

* The work described in this paper was partially supported by the Italian Ministry of Education, University, and Research (MIUR) under the IS-MANET project.

[1] The multicast can also directly embed the membership management, but in the following we maintain separated these two concerns: multicast communication of application information and group membership management.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 219–226, 2007.

dynamic nature of a p2p environment make the requirement of a "long enough" period of stability and synchrony problematic to discharge in practice.

Recently, in the context of WAN, Anker et al. [1] proposed the notion of *Membership Notification Service* (MNS) which provides each process with an approximation of the current group membership, without being synchronized with the message stream. This approach allows handling any number of simultaneous join/leave events concurrently and allows message reliability among those members that remain permanently alive against on-going membership changes. The authors proposed a client/server implementation of a MNS. More specifically, there is a set of servers and each server is in charge of (i) being the access points for joining nodes, (ii) tracking the departures of processes (both failures and voluntarily leaves) and (iii) providing views of the membership to whom requested them.

The aim of this paper is to study what are problems which arise in implementing a fully distributed (or p2p) MNS in a p2p setting and, then, to propose a solution.

The contribution of the paper is twofold, the paper firstly presents two impossibilities results that delimitate under which assumptions a p2p MSN implementation can be realized. Secondly, it introduces a p2p MNS solution that manages concurrent leaves. The presented solution dynamically builds and maintains an overlay topology in which processes are partially ordered by a *rank* which is assigned to a process at join time. This weak order is taken into account at leave time. The algorithm serializes any two concurrent leave operations executed by neighbor processes in the overlay topology that could lead to a partition of the topology itself. These departures are ordered by process rank. All the other concurrent leave operations are not serialized.

The paper is organized as follows. Section 2 presents the system model including the group membership management. Section 3 formally introduces the MNS specification, the impossibility results and the circularity problem. Section 4 shows a p2p MNS implementation.

2 System Model

The system consists of an unbounded but finite set of processes Π[2]. Any process may fail by crashing. A process that never fails is correct. The system is asynchronous: there is no global clock and there is no timing assumption on process scheduling and message transfer delays. Each pair of processes p_i, p_j may communicate along point-to-point reliable links. To simplify the description without losing generality, we assume the existence of a fictional global clock, whose output is the set of positive integers denoted by $\mathcal{T}$.

Group Membership. Each process $p_i \in \Pi$ may become member of a group G. Once member, it may decide to leave the group. To this aim p_i may invoke the following operations: `join(G)` to enter G and `leave(G)` to exit the group. The

[2] Note that in the informal parts of the paper we use the term "peer" as a synonym of "process".

set of processes constituting the group G at time $t \in \mathcal{T}$ is denoted as $G(t)$. At any time t, $G(t)$ is a subset of Π with size unbounded but finite. The rules defining the membership of G are the following:

1. a process $p \in \Pi$ becomes a member of G immediately after the completion of `join(G)`.
2. a process p ceases to be member of G immediately after the completion of `leave(G)`.
3. a process p may become member of G at most once [3].

A group member $p \in G$ is *stationary* if it is correct and never invokes `leave(G)`. A group member $p \in G$ is *transient* if it is correct and eventually executes `leave(G)`.

Note that, since $G(t)$ is unbounded and finite at any point of time, the number of stationary processes is unbounded and finite, as well.

Membership Management. To abstract in a general manner the membership management we consider that each process can locally access a distributed oracle. Each process p_i invokes the `join(G)` and `leave(G)` operations through the *MNS local module* MNS_i that is in charge of the actual execution of these operations. Each MNS_i module provides p_i with a *local view* of the group, i.e. a *list of processes* representing the current membership of the group as perceived by MNS_i. We assume that MNS_i crashes only when p_i crashes.

Upon the invocation of `join(G)` by p_i, MNS_i generates an event denoted as inv_i(`join(G)`), then MNS_i is granted permission to access the group on behalf of p_i. After that, MNS_i returns to p_i with an upcall by generating the event res_i(`join(G)`), thus at this point $p_i \in G$.

Upon the invocation of `leave(G)` by p_i, denoted as inv_i(`leave(G)`), MNS_i obtains permission for p_i to leave the group. After that, MNS_i returns to p_i with an upcall res_i(`leave(G)`), thus from this time on p_i is no longer a member of G.

Note that MNS_i may provide p_i with a local view even when p_i is not a group member. In this case we call the view $access\_view_i$. As long as p_i belongs to the group, the local view is called $group\_view_i$. Only between events res_i(`join(G)`) and res_i(`leave(G)`) we say that $p_i \in group\_view_i$.

3 MNS Specification

The view information for one group can be represented as a knows-about directed graph $K = (\Pi, E)$ [9]. For each pair of processes p_i, p_j, there will be an edge (p_i, p_j) in the graph if $p_j \in group\_view_i$, and an edge (p_j, p_i) if $p_i \in group\_view_j$. There exists an edge (p_i, p_i) for every process p_i such that $p_i \in group\_view_i$ [4]. This graph actually represents the overlay network to be used as underlying communication network by an application-level multicast protocol in a peer-to-peer environment.

[3] This is not a restriction because the process may join with another identifier.
[4] Note that there exists an edge (p_i, p_i) even for each faulty member p_i that crashes before generating res_i(`leave(G)`).

3.1 Specification

Safety. Since view information is propagated along edges of the knows-about graph, once joins and leaves cease, every stationary member p_i belonging to the graph should have for each stationary member at least one path formed by stationary members [5]. This is necessary because even though leaves and joins no longer occur, crashes are still possible. Such crashes could partition the set of stationary members. Therefore, if this condition is satisfied, the view of each stationary member eventually includes all stationary members. Formally,

Property 1 (Safety). Let $K = (\Pi, E)$ denote the knows-about graph at time t s.t. no edge (p_i, p_i) will be added or removed for each $t' > t$ (i.e., joins and leave cease at time t). Let us consider the subgraph $K_s = (S, E_s)$ such that

(i) $p_i \in \Pi$ and p_i is stationary $\Leftrightarrow p_i \in S$
(ii) $\forall\ p_i, p_j \in S,\ (p_i, p_j) \in E \Leftrightarrow (p_i, p_j) \in E_s$.

Then, $\forall p_i, p_j \in S$ there exists an edge (p_i, p_j) in the transitive closure of E_s for each $t' > t$.

Liveness. A trivial group membership implementation may maintain *safety* by blocking the completion of each `join(G)/leave(G)`.

Then, to avoid *static* implementations the following property holds:

Property 2 (Liveness). The execution of the *`join(G)` and `leave(G)`* operations requires *finite* time.

3.2 Impossibility Results

The following impossibility results stem from the general assumption that $access\_view_i$ is a random set of processes belonging to Π, without any relation with the current membership. Unfortunately, as we see later, to guarantee the MNS specification even $access\_view_i$ has to satisfy some property (stated in Corollary 1). While this property is very lightweight, it nonetheless necessarily introduces a circularity problem.

Impossibility Result 1. *If there exists a time $t \in \mathcal{T}$ s.t. $G(t) \equiv \emptyset$, the MNS specification cannot be guaranteed.*

Proof. (sketch) Let us suppose by contradiction that at some point of time t, $|G(t)| \equiv \emptyset$.

Assume a process p_i executing `join(G)` produces the inv_i(`join(G)`) event while $|G(t)| \equiv \emptyset$. p_i does not know whether the group is empty or not as $access\_view$ is neither complete nor accurate. p_i can send a JOIN message to its $access\_view$ but cannot get any acknowledgement (like any concurrent joining process) since $G(t)$ is empty. To respect Liveness, p_i has become a member after a finite amount of time exploiting a time-out strategy [6]. Then, at time $t + T$ p_i

[5] Our Safety specification is partially inspired by the group membership specification in [9].

[6] The strategy can encompass mechanisms such as setting a timeout T or retransmitting the JOIN message k times.

concludes to be alone in G and includes in $group\_view_i$ only itself. Because of the asynchrony of the underlying system, another process p_j with $p_j \notin access\_view_i$ and $p_i \notin access\_view_j$ can decide to join. As p_j does not "see" p_i, it uses the same strategy and generates $res_i(\texttt{join(G)})$ including in $group\_view_j$ only itself at time $t + T$. If both p_i and p_j are stationary no edge connects them at time $t' \geq t + T$. If no other join and leave occur there is no way to add that edge at a later moment. Hence, no edge will connect them for each $t' \geq t + T$ violating Safety.

Lemma 1. *Let us suppose that $|G|$ is never empty. Then, any process p_i cannot generate $res_i(\texttt{join(G)})$ until there exist at least one edge $(p_i, p_j) \in E$ and one edge $(p_j, p_i) \in E$.*

Proof. (sketch) Let us suppose that $res_i(\texttt{join(G)})$ is generated at time t and that $G(t)$ contains a stationary member p. By the way of contradiction, let us suppose that does not exist any edge (p_i, p_j) in E at time t. However, after $res_i(\texttt{join(G)})$ p_i has an edge $(p_i, p_i) \in E$ at time t. If p_i is also stationary then $G(t)$ contains two stationary processes and no edge in the transitive closure of E. If no other join and leave occur there is no way to add that edge in a successive moment. No edge will connect them for each $t' \geq t + T$, violating Safety.

Impossibility Result 2. *If there exists a time $t \in \mathcal{T}$ s.t. $G(t)$ contains no stationary member, the MNS specification cannot be guaranteed.*

Proof. (sketch) Let us suppose by contradiction that there exists a point of time $t \in \mathcal{T}$ s.t. $G(t)$ does not contain stationary members. From Lemma 1 every joining process has to establish two edges with a process p_j, before generating $res_i(\texttt{join(G)})$. From Liveness it has to establish those edges in a finite time. Without loss of generality suppose that at time t, $G(t)$ comprises k faulty members and c transient processes. Taken any subset $S(t) \subseteq G(t)$, of one process p_j, that process is either faulty or transient. Let us assume that:

1. p_j is transient and belongs to G between times t_j^J and t_j^L.
2. p_i generates $inv_i(\texttt{join(G)})$ and sends at time t a $JOIN$ message to each member of $G(t)$.

As the system is asynchronous, the delay experienced by $JOIN$ on the fair lossy link connecting p_i to p_j, could be greater than $t_j^L - t_j^J$ and then the message of p_i would not reach p_j. Moreover, p_j does not yet know p_i, so p_j cannot communicate with p_i before the $JOIN$ arrives to p_j. Then, p_i cannot establish any edge with p_j. p_i cannot establish in a finite time any edge unless some other stationary member will join the group. However, no stationary member can surely join in a finite time for the same reason that blocks p_i. Thus, p_i waits for an infinite time violating Liveness.

From Impossibility Result 2, the following Corollary holds:

Corollary 1. *If $access\_view_i$ does not eventually contain at least one stationary member, the MNS specification cannot be guaranteed.*

This constraint on *access_view* poses a *circularity problem* when the MNS is implemented in a pure p2p fashion, i.e. it is implemented in a fully decentralized manner by members themselves and no process plays a special role from the beginning. In this case, to fill the *access_view* in order to be compliant with Corollary 1, a run time discovery has to be performed. This discovery cannot be push-based (from the current members of the group to the newcomer): none can indeed provide the newcomer with a view as no member of the group knows the newcomer [7]. Thus, to discover a current peer, the newcomer has to contact someone (e.g. a special process) that knows some peer. Following a pure peer-to-peer approach (where there is no special process), only a peer may have this knowledge. Then, to know a peer, the newcomer must already know a peer: a classic instance of the hen-and-egg problem.

Circularity, in these systems, may be avoided by assuming either that eventually the newcomer will somehow know someone inside the group or the existence of special processes constantly known by all other processes from the beginning—at the cost, however, of losing a pure peer-to-peer approach.

4 A p2p MNS Implementation

In this section we provide a p2p MNS implementation. In particular, the MNS is implemented by the peers themselves where each peer only has only a partial view of the group[8,7]. The interested readers are referred to [3] for a performance analysis of the algorithm and its comparison with [8].

The proposed algorithm may concurrently handle join/leave operations generating, in a decentralized manner, knows-about graphs respecting Safety. The resulting graphs show a particular structure in which each member has around itself a clique of at least $f + 1$ members, where f is the number of tolerated failures. The other important feature of the algorithm consists in imposing a partial order on processes to manage concurrent leaves that may partition the graph. The algorithm also exploits heartbeat messages to monitor node failures.

Data Structures. The variable $group\_view_i$ is the union of two different variables: $sponsors_i$ and $sponsored_i$. $sponsors_i$ is a list of processes (identifiers) which guarantee to p_i the connection[8] to the group, i.e. upon the join operation the list contains all processes the grant p_i the permission to enter the group, then if some of these sponsors leaves the list will contain some other process that replaces the left one. $sponsored_i$ is a list of processes (identifiers) which p_i is responsible for in terms of connection. A variable $rank_i$ gives an indication of the position of p_i in the graph, inducing a partial order on nodes. A boolean variable *leaving* is initialized to $\perp$.

[7] The number of potential newcomers is unbounded. As a consequence the identifiers of potential newcomers cannot be available at design time.

[8] The connection is intended here as the connection to the overlay in terms of knows-about relation.

Initialization of the group. A set of processes $\{p_1, ...p_{f+1}\} \subseteq \Pi$ totally interconnected and defined in the initialization phase instantiates the group[9]. All these processes have rank $rank_i = 0$. They are special processes, they never leave the group.

Join Management. Rules of the algorithm:

- MNS_i sends a JOIN message to $access\_view_i$ [10]
- When MNS_i receives a JOIN message from MNS_j and $p_i \in group\_view_i$: (1) MNS_i inserts p_j in $sponsored_i$; (2) it sends an acknowledgement to p_j along with its own rank $rank_i$.
- When MNS_i receives $f+1$ acknowledgments: (1) MNS_i includes in $sponsors_i$ all the senders and p_i; (2) it sets $rank_i = max(rank_k, \forall senderp_k)+1$ and (3) returns to p_i generating $res_i(\texttt{join(G)})$. From this time on with an heartbeat mechanism all $sponsors_i$ are monitored. Each time a sponsor is suspected to be faulty, MNS_i tries to re-establish the missed connection searching another sponsor p_j with $rank_j < rank_i$.

Leave Management. Rules of the algorithm:

- MNS_i (i) sets $leaving_i = \top$ and (ii) sends a LEAVE message to $sponsors_i$, so composed $\langle LEAVE, sponsored_i, rank_i \rangle$;
- When MNS_i receives a LEAVE message $\langle LEAVE, sponsored_r, rank_r \rangle$ from MNS_j and $rank_j > rank_i$ and $leaving_i = \bot$: (1) MNS_i inserts $sponsored_r$ in $sponsored_i$; (2) it sends an acknowledgment to p_j and (3) sends a message $\langle NEWSPONSOR, oldsponsor = p_j \rangle$ to $sponsored_r$.
- When MNS_i receives an acknowledgment from its sponsors: (1) discards p_i from $sponsors_i$ and (2) returns to p_i generating $res_i(LEAVE(G))$.
- When MNS_i receives $\langle NEWSPONSOR, oldsponsor_r \rangle$ from MNS_j and $oldsponsor_r \in sponsors_i$: MNS_i includes p_j in $sponsors_i$ and discards $oldsponsor_r$ from $sponsors_i$.

Thanks to ranks, it is possible to induce a partial order on the nodes. In practice, when two nodes p_i, p_j with rank $rank_i < rank_j$ want to concurrently leave, a partition may occur if they actually leave at the same time. The algorithm sequences the leaves, by allowing a leave of a process of rank $rank_j$ only if none of its sponsor p_i with rank $rank_i < rank_j$ is concurrently leaving. Note that p_j remains blocked as long as new sponsors of p_j are concurrently leaving. Eventually, if all processes with rank lower than $rank_j$ leave, then p_j will have as sponsors processes with rank 0. Since by construction these processes never leave (they are stationary), then also p_j eventually will leave (Liveness).

Each process, if no failures occur, maintains at any time a knows-about graph with connectivity at least equal to $f + 1$. If some failure occurs during overlay changes, a recovery mechanism restore the connectivity of graph. If overlay

[9] Impossibility results are circumvented because of the presence of these processes.

[10] The mechanism to fulfill *access_view*, addressing Corollary 1, will be discussed in the reminder of this Section.

changes (joins/leaves) subside, the resulting knows-about graph has connectivity $f + 1$ and remains always connected until f failures occur. Safety is maintained until these f failures occur. Anyway, a restoring mechanism will restore connectivity until only stationary processes are in the overlay. From this time on connectivity among stationary members is always guaranteed.

References

1. Tal Anker, Danny Dolev and Ilya Shnayderman: Ad Hoc Membership for Scalable Applications. Proceedings of 16th International Symposium on DIStributed Computing, (2002)
2. Kenneth Birman and Robert van Renesse: Reliable Distributed Computing with the Isis Toolkit. IEEE Computer Society Press (1994)
3. Roberto Baldoni, Adnan Noor Mian, Sirio Scipioni and Sara Tucci Piergiovanni: Churn Resilience of Peer-to-Peer Group Membership: a Performance Analysis. International Workshop on Ditributed Computing (2005), to appear.
4. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Antony Rowstron: Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure. IEEE Journal on Selected Areas in communications (2002)
5. Gregory Chockler, Idit Keidar, Roman Vitenberg: Group Communication Specifications: a Comprehensive Study. ACM Computing Surveys 33(4): 427-469 (2001)
6. Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernardette Charron-Bost. On the Impossibility of Group Membership. In 15th Annual ACM Symposium on Principles of Distributed Computing, (1996)
7. Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Petr Kouznetsov, Anne-Marie Kermarrec: Lightweight Probabilistic Broadcast. ACM Transactions on Computuer Systems 21(4): 341-374 (2003)
8. Ayalvadi J. Ganesh, Anne-Marie Kermarrec, Laurent Massoulié: Peer-to-Peer Membership Management for Gossip-Based Protocols. IEEE Transactions on Computers 52(2): 139-149 (2003)
9. Richard A. Golding and Kim Taylor: Group Membership in the Epidemic Style. Technical Report UCSC-CRL-92-13, University of California, Santa Cruz (1992).
10. John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, James W. O'Toole: Overcast: Reliable Multicasting with an Overlay Network. Proceedings of the 4th Symposium on Operating System Design and Implementation, San Diego (2000)
11. André Schiper and Sam Toueg: From Set Membership to Group Membership: A Separation of Concerns, Technical Report, EPFL, Lausanne, (2003)

Querying Communities of Interest in Peer Database Networks

Md. Delwar Hossain and Iluju Kiringa

School of Information Technology and Engineering, University of Ottawa
800 King Edward St.
Ottawa, Ontario, Canada, K1N 6N5
{dhossain,kiringa}@site.uttawa.ca

Abstract. Peer databases are individual, independently developed databases that contain local data, but are also linked to each other through acquaintances for the purpose of sharing data. Many researchers have tackled the problem of query processing in P2P networks. Only recently have researchers started to exploit the idea of community-based querying in peer database networks. P2P communities are formed using common claimed attributes. We develop a community-based search algorithm for peer databases. The algorithm combines an existing community formation and discovery algorithm with an existing query translation mechanism for peer databases. We have implemented the algorithm and report on preliminary experiments.

1 Introduction

The P2P paradigm of computing is defined as the distribution of computer resources and information through direct, point-to-point exchange. A P2P network is composed of nodes (peers), each of which contains both strictly local resources as well as resources to be shared. Those shared resources are also said to be exported by the peer. A node can work as a client as well as a server. Efficient P2P search techniques for P2P networks are proposed in [17]. In addition to efficient search techniques, it has been noticed that peers usually tend to form P2P communities [3,7,10,16]. The term "P2P community" comes from common interests. Here, *common interests* are the same resources or attribute that are exported by a set of peers. To follow [10], "a non-empty set N of peers forms a peer-to-peer community if all peers in N claims the same attributes". Moreover, an elaborated search mechanism based on the idea of communities of interest is proposed in [10]. In the sequel, the term community will refer to the set of attributes that characterizes the common interests. Some others are outlined in [3,7].

Meanwhile, database management systems (DBMSs) are being built around the P2P computing paradigm [1,6,13]. Such DBMSs are called peer DBMSs [1]. Among a few others (e.g., [6,13]), the Hyperion project [1] develops algorithms for peer database management. Peer DBMSs form a network of nodes (peers) which coordinate querying, updating, and sharing of data at run-time. One of the most important features of a network of peer DBMSs is that peers establish and abolish

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 227–234, 2007.

links at run-time. In this context, the set of all peer DBMSs breaks down into multiple communities of interest formed around common, shared attributes. However, existing P2P query mechanisms do not specifically exploit the idea of communities of interest in networks of peer DBMSs. Community-based querying assumes guidelines on how users will search in an interest-based locality. The issue is two fold: (1) to discover and form communities; (2) to apply community-based search algorithms. We provide an algorithm for community-based query processing in peer DBMSs. The idea is to exploit the concept of community of interest to enhance query processing in a network of peer DBMSs. The main contributions are: combining the community formation and discovery algorithms described in [10] with a query processing algorithm developed in [9] by considering explicit communities; and implementing the algorithm in a peer database network. Our algorithm forms peer communities by using an escalation method [10] which consists in having a peer claim as many relational attributes as possible. After discovering the formed communities, our algorithm uses a query translation technique inherited from [9] to get results from other peers in targeted communities. The query processing algorithm translates a query posed against a peer to the vocabulary of all the peers that belong to the same communities which have been formed and discovered using an extension of the method of [10]. Since these communities are formed and discovered in advance, they are explicit communities.

2 Querying Communities of Interest

2.1 Motivating Example and Preliminary Concepts

We consider a movie domain with ten peers P1, ..., P10 (Figure 1). We assume the relational data model. Each peer has strictly local (or personal) attributes and shared (or claimed) attributes.

The movie domain contains the following attributes, which are denoted as letters: Movie name = N, Film Director = D, Movie type = T, Country of movie = C, Movie award = A, Actor/Actress = H, Year of movie = Y. Each peer claims a subset of its personal attributes to be shared with other peers. For example, peer P1 claims the set [Y,C,T,N]. We now show portions of the schema of the aforementioned movie domain. To be brief, we only give schemas of peers P1 and P7.

Schema of peer P1:
Movie (MovieID, DirName, TypeID, CountryCode, AwardType, ActName, Year)
MovieName (MovieID, MovieTitle)
MovieType (TypeID, Type)
Director (DirName, CountryCode, MovieID)
Country (CountryCode, CountryName, MovieID)
Award (AwardType, AwardName, MovieID)
Schema of peer P7:
MovieDesc (MovID, Country, MovCategory, AwardCategory,
ActorName,ActressName, MovieYear)

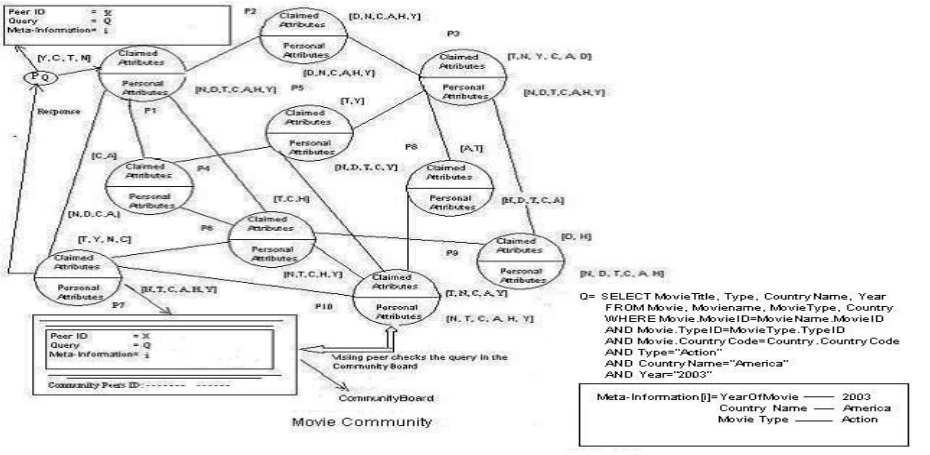


Fig. 1. A Movie Domain

Movie (MovID, MovName)
Category (MovCategory, CategoryName)
Award (AwardCategory, AwardTitle)

Attribute mappings: Different peers may contain the same attributes under different names. Through attribute mappings, attributes of two different peers are matched.

Mapping tables: Mapping tables provide a mechanism for sharing data and for associating values residing in different peers [11]. We use them for discovering communities in some cases, but they are mainly used for query translation of different peers. A mapping table may say, e.g., that the value "Action" of the attribute "Type" is mapped to both "War" and "Fighting".

Item file: The item file is used to store the description of attributes of the database. More specifically, it provides the full name of each attribute of the database. Each peer database has an item file. A main table is a master file where all necessary information is kept. It contains data of some or all of the attributes as a code for security purposes. It is a main table of the database. The item file is needed to recognize the claimed attributes in every peer. Peers declare claimed attributes through the item file, but the processing related to communities (e.g. calculation of link weight, discover communities, etc.) is done by the related database.

Community board: A community board is a data structure that stores community peers identity (e.g., peer name, signature of claimed attributes set). Each peer has a community board, but, as an example, Figure 1 shows only the community board of peer P7. In Figure 1, a SQL query Q, and the related meta-information are also shown. We assume that communities are formed using an attribute escalation method where attributes are claimed through the item file

of peer databases. Attribute escalation is a process in which attributes are promoted from a private attribute of a peer to a claimed attribute set to form communities. Communities are discovered by computing the link weight and comparing the latter with predetermined threshold values. The link weight is the weight calculated for each claimed attribute of a peer V based on the number of links from V that can reach, after at most one indirection, other peers that claim the same attribute [10]. Threshold value remains a "magic number" that depends on a number of observations. In our implementation, we set the threshold value at 60%.

2.2 Community-Based Querying Algorithm

In the context of our running movie community example, we now informally explain the community-based query processing algorithm. We assume that attribute mapping and mapping tables are all in place. The main ingredients of our community-based query processing approach are queries, communities, and peer databases. Our solution has two steps: (1) Forming and discovering communities; and (2) translating a given query to be sent to the desired communities.

Step 1: Communities are discovered by examining database attributes. The attributes of peer databases are described in the item file of the peer. Each peer claims some attributes from the item file; then it calculates the link weight of these claimed attributes. After that, it compares this link weight with a globally predetermined threshold value. If the link weight is greater than the threshold value then the peer belongs to a community with those attributes as signature. Each peer may belong to one or more communities. Details of this algorithm are given in [10]. At the end of Step 1, every peer knows about communities to which it belongs and which attributes it claims. For example, peer P7 will know that it belongs to the community [P1,P7,P10] formed by peers P1, P7, and P10.

Step 2: Suppose that the SQL query Q is posed at peer P1. From the query Q, peer P1 will get the following meta-information: year (2003), movie type (Action), and country (America). If peer P1 cannot execute Q, it just discards it; else peer P1 executes Q and sends the results back to the user and copy the query in its community board. Now, peer P7, which knows that it belongs to the same community as P1, will download the query Q from peer P1. Peer P7 will then compare its claimed attributes with the meta-information of query Q. Since the meta-information of Q matches the attributes claimed in peer P7, the latter will first translate Q into its schema, using the mapping tables. Peer P7 stores the translated query Q' in its community board, execute Q' and send the results back to peer P1. Each peer in the network proceeds in a similar manner.

Periodically and asynchronously each neighboring peer of P7 which has a common community with P7 will visit P7's community board. If a neighbor can answer any query that is stored in that community board, it copies the given query into its own community board. This process repeats until all peers of the community have been checked. Algorithm 1 summarizes the community-based query processing mechanism described above.

Algorithm 1. Community-Based Query Execution and Propagation

Community-based search using Query Translation
/* This algorithm is run at each peer Pi */
Begin
Assume that the list of communities has already been compiled and let this list be COM; each one of the members of COM is a set of attributes;
  Periodically do the following:
    For each acquaintance Pj of Pi do
      Check any query Q stored in Pj's community board CBj
      Set identifier PQ of the peer that initiated Q
      Extract the meta-information MI(Q) of Q
      If MI(Q) matches COM[k] for some k then
        Translate Q using the algorithm of [11];
        Let Q' be the translation of Q;
        Copy Q' the community board of Pi;
        Execute Q' and send results back to PQ
      End-If
    End-For
End

3 Implementation

3.1 System Architecture

Figure 2 extends the Hyperion DBMS architecture [1] with a community-based search component. In the Hyperion architecture, a peer consists of three main components: an interface (GUI) for posing queries to the system and performing administrative tasks, a P2P layer that encompasses the peer-oriented data management functionalities, and an underlying DBMS. The P2P layer includes a Query Manager (QM) used to process and propagate queries, and an Acquaintance Manager (AM) for establishing acquaintances between peers. Through the AM, the P2P layer allows a peer database to establish or abolish acquaintances at run time, thereby forming a logical peer-to-peer network. The underlying DBMS manages a database containing local databases, as well as mapping tables and attribute mappings which enable data exchange with other peers.

The P2P layer of the Hyperion architecture is augmented a Community Discovery layer (CD). The CD layer has four sub-components: an Attribute Escalator/Item Matcher (AE/IM), a Community Formator (CF), a Link weight Calculator (LC), and a Community Discovery Rule Manager (CDRM). The CD layer is connected with the QM and AM for propagating the given query into a desired community. The AE/IM escalates attributes by intersecting personal items with claimed items. The CF is responsible for forming communities. The LC calculates link weights of each of the claimed attributes. Finally, the CDRM discovers communities by comparing link weights with given threshold values.

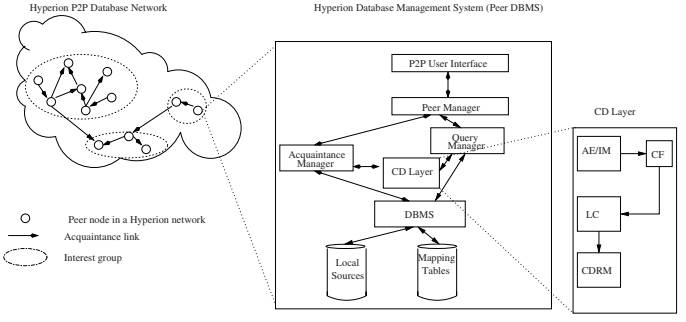


Fig. 2. Architecture for a Community-Based Hyperion Peer

3.2 Implementation and Experiments

We have implemented our community-based query algorithm which is mentioned in section 2.2. We used MySQL 4.1.9 as our underlying DBMS for the local peer databases. Following the existing Hyperion infrastructure [18], our prototype is built on top of JXTA [8] using Java SDK 1.4.2. We provide a user interface for declaring claimed attributes, discovering communities, and posing queries to the peer database network. We are mainly concerned with measuring the correctness and efficiency of discovering communities in a peer database network. We created 10 peers on 4 different machines using the JXTA package. To measure the efficiency and effectiveness of our algorithm, we did some experiments. Because of the unreliability of JXTA pipes at the time of the experiments, and motivated by the desire to create a signification amount of peers, we decided to create two networks of 100 and 300 peers, respectively, all on one single machine. We collected data for evaluating the following: the community refreshment within different time intervals; the effect of community changes on query propagation; and the time complexity of our algorithm. Due to space limitations, we only report on the findings with respect to community refreshment.

For plotting percentage of community changes, we gathered two sets of data: one is the change induced by peers that join or leave the networks; the other is the variation of peer database attributes within different time intervals. In our networks, each peer contains a maximum of 12 personal attributes, and

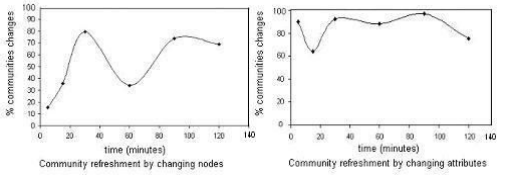


Fig. 3. Community Refreshment by Changing Peers and Attributes

a peer was explicitly placed into only one group. Our algorithm sets the link threshold at 60%. Figure 3 shows community refreshment within different time intervals. The curves of Figure 3 are in different shapes. The percentage of communities change depends on how many peers change their status (i.e. joining or leaving the network, and changing claimed attributes) after community refreshment. A large time interval (typically 30 minutes) for community refreshment shows more community changes than shorter intervals. Our observation determines that peers change their attributes more frequently than they join or leave networks.

4 Related Work

Peer database networks are a serious option for the next generation of multidatabase systems [11]. The work reported in the present paper contributes to the implementation of the idea of run-time (as opposed to design time) multidatabase systems [1]. The vision reported in [6] presents the first complete articulation of the idea of data coordination based on the concept of community of interest. In [4,5], web communities are presented along with their own concepts of community formation. The community formation and discovery techniques that we used are inherited from the work reported in [10,16]: we adapt the algorithms described there to our context of relational peer databases. In our context, communities are formed using common claimed relational attributes. In addition, we refined the attribute escalation and community discovery algorithms: in [10], all the subsets of a peer's claimed attributes are checked in order to determine community membership; on the contrary, we do introduce an optimization that allows to discard a membership in, e.g., [A,B,C] whenever a peer does not belong to, e.g., [A]. Doing so, our algorithm reduces the number of iterations during community discovery.

5 Conclusion and Future Work

We have addressed the problem of using the concept of peer communities to enhance query propagation in a network of peer databases. We discover communities by taking into account relational attributes claimed by peers on the network. Such communities are explicit. A query translation scheme uses mapping tables to translate a given query to the vocabulary of every peer community. Attributes mentioned in the given query are used in the choice of the targeted communities. We implemented the algorithms for explicit communities on top of JXTA and did some experimentations. As future work, we plan to implement the algorithm for discovering communities on the fly. This amounts to discovering communities in a distributed way as queries are being propagated. Currently, the system accepts only queries with simple condition parts, but queries with more elaborated conditions could be incorporated in our algorithm as well.

Acknowledgment

We would like to thank Mujtaba Kambatti for making his community simulator available to us and for other useful feedbacks.

References

1. Arenas, M., Kantere, V., Kementsietsidis, A., Kiringa, I., Miller, R.J., and Mylopoulos, J.: The Hyperion Project: From Data Integration to Data Coordination. In SIGMOD Record (2003)32(3): 53-58
2. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., and Zaihrayeu, I.: Data Management for Peer-to-Peer Computing: A Vision. Proc. of the 5th International Workshop on the Web and Databases (WebDB) 2002
3. Doucet, A. and Lumineau, N.: A collaborative approach for query propagation in peer-to-peer systems. Proc. of SWDB 2003
4. Flake, G.W., Lawrence, S., Giles, C.L., and Coetzee, F.M.: Self-organization and identification of web communities. IEEE computer (2002)35(3):66-71
5. Gibson, D., Kleinberg, J., and Raghavan, P.: Inferring web communities from Link topology. Dept. of Computer Science. UC Berkeley Berkeley USA, 1999
6. Giunchiglia, F. and Zaihrayeu, I.: Making peer databases interact - A vision for an architecture supporting data coordination. Proc. of CIA 2002
7. Gnasa, M., Alda, S., Grigull, J.,and Cremers, A.B.: Towards Virtual Knowledge Communities in Peer-to-Peer Networks. Proc. of the SIGIR Workshop on Distributed Information Retrieval, 2003
8. JXTA Project: http://www.jxta.org
9. Kementsietsidis, A. and Arenas, M.: Data sharing through query translation in autonomous sources. Proc. VLDB 2004
10. Khambatti, M.: Peer-to-Peer communities: Architecture, Information and Trust Management. Ph.D. Thesis, Arizona State University, December 2003
11. Litwin, W., Mark, L., and Roussopoulos, N.: Interoperability of multiple autonomous databases. ACM Computing Surveys (1990)22(3):267-293
12. Lumineau, N. and Doucet, A.: Sharing Communities Experiences for Query Propagation in Peer-to-Peer Systems. Available at http://www-poleia.lip6.fr/padoue
13. Piazza PDMS: http://data.cs.washington.edu/p2p/piazza
14. Serafini, L., Giunchiglia, F., Mylopoulos, J., and Bernstein, P.A.: Local relational model: A logical formalization of database coordination. Technical Report DIT-03-002, Informatica e Telecomunicazioni, University of Trento, June 2003
15. Siong Ng, W., Ooi, B.C.,Tan, K.L., and Zhou, A.: PeerDB: A P2P-based system for distributed data sharing. Proc. of ICDE 2003
16. Vassileva, J.: Supporting peer-to-peer user communities. Proc. of CoopIS 2002
17. Yang, B. and Garcia-Molina, H.: Efficient Search in Peer-to-peer Networks. International Conference On Distributed Computing System, 2002
18. Rodriguez-Gianolli P., Garzetti M., Jiang L., Kementsietsidis A., Kiringa I., Masud M., Miller R., and Mylopoulos J.: Data Sharing in the Hyperion Peer Database System. Proc.VLDB 2005

Middleware for Reliable Real-Time Sensor Data Management

Vana Kalogeraki

Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521
vana@cs.ucr.edu

Abstract. The integration of sensors in networks of embedded systems revolutionize distributed networked applications in a variety of disciplines. The development of the appropriate middleware components and tools that simplify the programming of the applications and enable reliable and timely communication is a key challenge. This paper describes challenges and basic building blocks in the design of such middleware. We first examine in-network data management computations that can be performed in a sensor network. Then we discuss the various challenges that must be addressed in providing a convenient and powerful middleware environment that simplifies the development of distributed applications on sensor networks, concentrating on in-network storage and real-time data dissemination. Such a sensor network middleware is expected to significantly promote the wide adoption of distributed sensor network applications.

1 Introduction

Advances in wireless technology and embedded sensor devices are revolutionizing the way that sensor data is collected and analyzed. Large-scale deployments of sensor network devices have already emerged in environmental monitoring (such as atmosphere and habitat monitoring [1,2], seismic and structural monitoring [3]), industry manufacturing [4,5], healthcare and disaster recovery missions.

Consider for example a sensor network deployment for environmental monitoring. Imaging and non-imaging sensors can be used to capture continuous streams of data such as temperature, position, movement, as well as images or other conditions related to the observed phenomenon. In many of these deployments, it is often more energy efficient to store locally a large window of measurements (at the generating site) and transmit specific readings to a centralized server (*i.e.,* the sink) only when requested. For instance, biologists analyzing a forest are usually interested in the long-term behavior of the environment (*e.g.,* forests, plant growth). Therefore the sensors are not required to transmit their readings to the sink at all times. Instead, the sensors can work unattended and store their reading locally until certain preconditions are met, or when the sensors receive a query over the radio that requests specific data.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 235–246, 2007.

Another example is a disaster recovery scenario. In military battlefields, there is need for reliable and high quality communication to ensure that high priority events are delivered to the operations center in a timely manner. The sensor devices are either placed manually at predetermined locations, or (in some environments where human intervention is not possible) the sensors can be thrown from an airplane to self-organize into a network. Sensor nodes in military deployments can be useful to sense and monitor data streams of interest or to detect and track objects as they move through the battlefield. Such deployments require reliable and real-time communication. Time requirements are usually expressed in the form of deadlines, where the typical objective is to ensure that packets are delivered end-to-end, from the sources to the operations station, within their deadline. Once deployed, sensors are prone to failures due to manufacturing defects, environmental conditions (such as fires) or battery depletion. In such circumstances, the data (e.g., sensors' reports) may become stale or get lost. Thus, reliable functioning of wireless sensor devices is of paramount importance to guarantee correct operation and ensure that critical events are accurately reported.

Compared to peer-to-peer networks, sensor networks have the following key characteristics:

- *Ad hoc sensor network topology.* The network is peer-to-peer, composed of several wireless sensor devices deployed in a particular region. A static sensor network environment is typically assumed, although a small number of autonomous mobile devices could also be part of the network. Sensor networks are based on wireless technology. Sensors are often deployed in large numbers (to increase redundancy). The density of the nodes depends on the type of application. Nodes can directly communicate with other devices within their transmission radius, but no direct communication with remote nodes is possible. New sensor nodes may be added or removed during the network lifetime. Furthermore, nodes may turn on and off based on some sleep pattern or because of failures. As opposed to peer-to-peer systems, failed or damaged devices can affect the correct operation of the application.
- *Limited resource capabilities.* Sensors are expected to be battery equipped. Each device has limited processing, storage and communication capabilities and thus sophisticated management policies are not appropriate. Resource constraints are much more stringent in sensor networks, compared to peer-to-peer networks. This is due to the fact, that, sensor networks are often expected to be deployed in environments where human intervention or network maintenance may not be possible. The resource constraints of the nodes (such as bandwidth and memory constraints) also limit the degree to which data can be collected and communicated to other nodes in the network. For example, since multiple nodes share the same communication channel, the bandwidth available per node depends on the area density as well as the traffic in the network. Furthermore, the local RAM memory of the sensor devices is limited. Also, the non-volatile on-chip flash memory featured by most sensors today is also very limited. However, several sensor devices, such

as the RISE hardware platform [6] and the TSAR storage architecture [7,8], propose to equip sensor devices with off-chip flash memory which supplements each sensor with several Megabytes of storage. We believe that future sensor nodes will feature more SRAM and flash storage, which will allow larger volumes of data to be stored and analyzed in-situ.

- *Unreliable environment.* Failures are inevitable in sensor networks. Failures can be caused due to hardware or software faults (such as manufacturing defects, battery depletion, program errors) or due to communication errors. Failures can be addressed by including a significant amount of redundancy in anticipation of possible failures during the application operation. The sensor network must be able to detect and recognize failures and be able to dynamically adapt to changing conditions and maintain correct operation.
- *Aperiodic initiation of data streams.* Sensor network applications can vary from periodic applications, such as environmental monitoring, to event-based applications, including target tracking, emergency response and military applications. In event-based sensor networks, events are aperiodic and are likely to be generated at random times and at random places in the network (*e.g.,* sensors tracking a moving object). Different types of applications have different requirements in terms of resource and rate demands. For example, in a disaster recovery situation, there might be need for high quality video surveillance and prioritized flow delivery between event sources and control locations. In such scenarios, a key question is how to handle bursty data and data rates far in excess of that currently available for communication.

  Thus, the limited resource capabilities of the sensor devices, the lack of centralized control, the unreliable communication environment and the variable resource requirements and rate demands of the applications, generate significant challenges to sensor data management.

The remaining of this paper is structured as follows: We first provide a brief description of data management computations in sensor networks (Section 2). In Section 3 we discuss the middleware components required to support distributed data management applications. Related work is discussed in Section 4 and finally Section 5 concludes the paper.

2 Data Management on Sensor Networks

In this section we give an overview of in-network data management computations in sensor networks.

2.1 Attribute-Based Queries

The ability to query data streams produced by a sensor network has been a topic of much research in recent years. Previous approaches require data streams to be transmitted to a centralized server for further processing, storage and analysis. This approach has been widely utilized in peer-to-peer data management systems. However, in a sensor network setting, these approaches suffer from high

communication cost and latency since they require that all the data must be transmitted to a central location. To overcome these limitations, systems like Cougar [9] and TinyDB [10] process attribute queries using a declarative SQL language. Their goal is to monitor continuous streams of data produced at geographically distributed sources and use sensor resources in an efficient way when collecting the query results. In-network aggregation techniques [11,12] improve the centralized approaches, by pushing partial computation to the internal nodes along the path from the data sources to the centralized server. These techniques can be efficiently used when raw sensor data is required, or when one wants some aggregate data from multiple nodes (such as AVG, MAX, MIN and SUM).

Although collecting attribute query results is an important problem, these techniques still require some amount of computation at the server. Furthermore, to perform more intelligent computations such as top-k queries and spatial queries, more efficient techniques are required: (1) First, although attribute queries can be used for collecting spatial query results, we can significantly reduce the energy consumption in processing spatial queries by exploiting the fact that the sensors with the query result are usually located in the same geographical area. (2) Second, some applications may not require the collection and transmission of the entire volume of sensor data, rather they are interested in identifying and monitoring a relatively small subset of data of interest (*e.g.,* top-k results). However, to compute top-k queries in a distributed environment, it may be necessary to retrieve data from multiple sensors and compare their values to generate the results of interest.

2.2 Spatial Queries

Spatial queries are a subset of queries in which the database or the sensor network is queried by location rather than an attribute. Spatial queries are used to answer questions, such as *"find the average temperature in a geographical area or count the number of sensors within one mile of a point of interest"*. In a sensor network, however, spatial queries have to be processed in a distributed manner. Because of the resource limitation of sensor nodes, it is desirable to process the query only on those sensors that have relevant data or are used to route the data to the centralized server (*i.e.,* sink). The idea behind these techniques is to create a *spatial index* on groups of objects which are geographically related, and use this index to process spatial queries. The unique characteristics of the sensor networks generate new challenges for processing spatial queries in sensor network settings:

- Distributed query execution. Queries must be computed in a distributed manner, because sensor data are distributed in the network, and there is no global collection of the data.
- Distributed and dynamic index maintenance. The high energy cost of communication requires to decide where to compute the queries to optimize the resource usage. In addition, the spatial index must reorganize itself to respond to dynamic changes in the environment or sensor failures.

To be able to process spatial queries efficiently, we have developed a distributed SPatial IndeX (SPIX) over the sensor network[13]. SPIX imposes a hierarchical structure in the network and is maintained by the sensors to efficiently evaluate spatial queries. This distributed spatial index is built once by flooding a message into the network and maintained dynamically when sensors fail or new sensors join. Spatial queries are disseminated from a centralized server. The server is responsible for preparing the query, submitting it into the sensor network and getting the result back. The spatial query will be disseminated into the routing tree and the result will be sent back to the server. When a sensor receives the query, it decides if the query applies locally and/or needs to be submitted to one of its children in the routing tree. A query applies locally if there is a non-zero probability that the sensor produces a result for the query. To determine this, each sensor node in SPIX maintains a minimum bounded area (MBA), that represents the geographical area covered by the sensor. The sensor's MBA covers itself and the nodes below it. When a node receives a spatial query, it intersects the query area to its MBA. If the intersection is not empty, it applies the query and forwards the request to its children. SPIX has a number of advantages. It allows us to: (1) Bound the branches that do not lead to any result, (2) Find a path to the sensors that might have a result, and (3) Aggregate the data in the sensor network to reduce the number of packets transferred and save energy. We can also appropriately select the MBA sizes of the sensors and intelligently choose the parents in the spatial index, which allows us to further optimize the construction of the tree and save more energy in processing spatial queries.

2.3 Top-K Queries

Top-k queries are useful in sensor networks when we are interested in evaluating queries over multiple sources and only retrieving those results that match well the given query. An example of a top-k query might be *"Find the three moments on which we had the highest average temperature in the last month?"*. The objective of a top-k query in a sensor network is to find the k highest ranked answers to a user defined similarity function. Since the execution of a query is typically associated with the transfer of data over the wireless network, the objective is to accomplish this in an energy efficient manner, that is, by transferring as fewer readings to the sink as possible and using a fixed number of rounds.

Computing distributed top-k queries is significantly more complicated than aggregating the local readings of individual sensors. The problem is that the top k data values of a single sensor are not necessary the top k values of the entire sensor network. Thus, multiple rounds of data exchanges may be required to answer such queries. We have studied the problem of distributed TOP-k query processing in sensor network environments. We have developed the Threshold Join Algorithm (TJA), which is an efficient solution to this problem[14]. *TJA* resolves top k queries in the network rather than in a centralized fashion, minimizing the communication between nodes but also exploiting well the inherent parallelism of the network itself. We have shown that our approach is highly efficient and accurate in computing distributed top-k queries.

3 Middleware Components

In this section we present our middleware components for supporting distributed data management applications. Our goal is to provide a general purpose middleware that can be used to support a variety of applications. In this section we focus on middleware components for in-network data storage and real-time data dissemination.

3.1 In-Network Data Storage

The high energy cost of transmitting continuous data streams in sensor networks has motivated the development of in-network data storage schemes that facilitate both energy savings and long-term data storage. The challenge in current sensor network architectures is that the local RAM memory of the sensors is both volatile and very limited. Furthermore, the non-volatile on-chip flash memory featured by some sensors today is also very limited. To overcome these limitations, recently, sensor network architectures have been proposed that equip sensor devices with off-chip flash memory. Examples of such architectures are the RISE hardware platform [6] and the TSAR storage architecture [7,8]. These aim to: (1) supplement each sensor with several Megabytes of storage, and (2) assist in creating an energy-efficient distributed storage platform suitable for resource-constrained sensor nodes. The use of flash memory in the sensor nodes offers new opportunities for pervasive monitoring of the environment. Local storage on the flash memories enable the devices to store and manage data, as generated at the sensors, and do that in an energy efficient manner.

Data storage has been a basic research topic of systems research. A number of large-scale, peer-to-peer data storage systems have been proposed in the literature. Data-centric sensor networks differ from traditional peer-to-peer data storage systems in two important ways: (1) in peer-to-peer systems data objects can be available at multiple nodes in the network and each node is responsible to store a number of objects, regardless of where the data was generated in the network, and (2) the resource limitations of the sensor devices require economical usage of resources and the design of efficient mechanisms to retrieve the data. When developing in-network storage support on flash memories, the following main questions arise: (i) what file structures and index structures are required to allow us to store and efficiently retrieve data from the flash, (ii) how to address the unique characteristics and constraints of flash memories. We have proposed the MicroHash index[15], an efficient external memory structure for wireless sensor devices. MicroHash exploits the asymmetric read/write and wear characteristics of flash memory and offers high performance indexing and searching capabilities in the presence of a low energy budget which is typical for the devices under discussion and at a small cost of constructing and maintaining the index.

3.2 Real-Time Data Dissemination

Support for timely packet dissemination is a fundamental requirement for a number of sensor network applications. Typically, a sensor network integrates, within the same network, a rich collection of sensing devices with different requirements in terms of transmission rates, bandwidth and jitter demands. In these settings, data from a multiplicity of sensor-detected events, will typically have to flow via data paths that are largely interference-coupled. Collisions of packets from simultaneous flows may create congested hotspots, which in turn, cause flows to experience arbitrary delays and packet drops. The problem becomes more critical in situations where we have multiple types of traffic and some flows are more important than others and thus need to be delivered with bounded delay[16]. The characteristics of the sensor networks such as limited availability of resources, interference-coupled paths as well as the aperiodic initiation of event-based flows, make the problem more challenging. Current congestion control or admission control approaches are not adequate to address these problems. The majority of these techniques are reactive in nature and are implemented only after congestion has already occurred and, thus, may not be able to avoid excessive packet losses. Furthermore, these approaches fail to take into consideration the different requirements of the individual classes of flows.

In our work we have addressed the problem of congestion and rate control in the presence of multiple classes of interfering flows[17,16]. In particular, we have proposed: (i) distributed mechanisms for identifying congestion control and, (ii) rate allocation mechanisms to effectively and fairly manage flows from multiple classes of traffic. Our goal is to dynamically regulate sensor traffic to control congestion and schedule the flows end-to-end based on the sensor traffic characteristics and the real-time requirements. Our congestion control mechanisms are based on a collective and distributed estimation of the current *traffic intensity* in the network to identify the onset of congestion. Our estimation technique is simple, decentralized, lightweight and should be implemented efficiently within the sensors' localized scope. To support multiple classes of sensor traffic, in the case that where many flows are present in the network, our rate allocation mechanism assigns lower rates to low priority flows. Using such an approach, our techniques allow us to effectively allocate and adjust the rates of the flows based on the sensor traffic requirements and current network conditions.

4 Related Work

In this section we discuss related work in the areas of data management, in-network storage and real-time data dissemination.

Data Management. Several attribute-based query processors have been developed for sensor networks. Madden et al [10,12] have proposed an Acquisitional Query Processor (ACQP) that executes attribute queries over a sensor network. ACQP builds a semantic routing tree (SRT) that is conceptually an attribute index on the network. It stores a single one-dimensional interval representing the

range values beneath each of its children in each node. Every time a query arrives at a node s, s checks to see if any child's value overlaps the query range. If so, it prepares and forwards the query. SRT provides an efficient way for disseminating queries and collecting query results over constant attributes. Systems such as TinyDB [18] and Cougar [9] achieve energy reduction by pushing aggregation and selections in the network rather than processing everything at the sink. Both approaches propose a declarative approach for querying sensor networks. These systems are optimized for sensor nodes with limited storage and relatively short-epochs, while our techniques are designated for sensors with larger external flash memories and longer epochs.

Range queries have also been studied in dynamic [19] and large-scale environments [20,21,22]. However, because of the resource limitation of the sensor devices, building an efficient centralized index or distributed index to execute queries presents a number of challenges. Ferhatosmanoglu et al [23] have proposed peer-tree, a distributed R-Tree method using peer-to-peer techniques. The basic idea is that the sensor network is partitioned into hierarchical rectangle shaped clusters, and then they implement joins/splits of clusters when the number of items (sensor nodes) in the cluster satisfies certain criteria. The authors have shown how to use the peer-tree structure to answer Nearest Neighbor queries. In [23], the peer-tree is created bottom up by grouping together nodes that are close to each other. Each group of nodes selects a representative, which acts as the parent of this group of nodes, and these representatives are in turn grouped together at the next level. As a result, the connections between parents and children become progressively longer, and there is no way to guarantee that they can be implemented as single hops in the sensor network unless we make the assumption that the nodes' transition range is in the order of the dimensions of the sensor field. We note however that such long transmission ranges would have large energy costs. Our technique, on the other hand, operates in a top down fashion when constructing the hierarchy, and guarantees that each parent to child connection is only one hop away.

In-network Storage. A few techniques have been proposed for data storage and indexing in sensor networks. Matchbox, is designed to provide a simple file system for mote-based applications. The advantage is that it hides the lower details of wear-leveling and provides a pointer to each file (page in our context) on the flash memory. Matchbox works with the packaged with the TinyOS distribution [24]. This approach has the disadvantage that it requires a large footprint to keep track of these pointers. An efficient and reliable file storage system, called ELF (Efficient Log Structured Flash File System) [25] has been proposed for sensor networks. ELF is a log-like file structure designed for wear-leveling. It works by updating desired file page and writing it into a new flash memory space. The $TSAR$ indexing scheme [7] stores data locally at sensor nodes and indexes them by higher tier nodes called proxies. Distributed Index of Features in Sensor networks (DIFS [26]) and Multi-dimensional Range Queries in Sensor Networks (DIM [27]) extend the data-centric storage approach to provide spatially distributed hierarchies of indexes to data. All these techniques provide

index topologies at the network level, but do not provide details on how to efficiently write the index information into the sensor flash memories as we do in our approach. Furthermore, flash-based file systems have been proposed in the last few years, such as the Linux compatible Journaling Flash File System (JFFS and JFFS2) [28], the Yet Another Flash File System (YAFFS) [29] specifically designed for NAND flash with it being portable under Linux, uClinux, and Windows CE.

Real-time Data Dissemination. A few projects have investigated techniques for real-time data communication in sensor networks. Most of these techniques have focused on providing Medium Access Layer (MAC) or routing solutions. MAC–based scheduling solutions, however, may fail to consider two important pieces of information: (a) the urgency of a packet to be transmitted and (b) the queuing delay. Furthermore, localized MAC protocols are limited as they cannot consider the congestion state of the flows and delays along their complete routing path. Localized time-division-based solutions (TDMA) can be employed to guarantee deterministic access times but are quite complex and require coordination among the sensors which is expensive in terms of message exchanges. An alternative approach would be to provide the higher priority packets with lower back-off times that would allow them to access the channel with higher probabilities [30][31]. But in spite of this, due to interference range effects,low priority packets contend with those of high priority. Furthermore, loss of control packets still could occur, causing route failures and other associated effects.

Routing protocols for real-time delivery have also been proposed [32,33]. These techniques attempt to schedule the packets either at the MAC or routing layer to guarantee timely delivery. The SPEED protocol [32] utilizes greedy Geographical Forwarding and attempts to maintain a required speed between hops as packets are propagated in the network. SPEED uses rerouting when the delay becomes excessive. One disadvantage of SPEED is that it can suffer from instability caused by sudden congestion, which can cause highly variant delay measurements. The RAP protocol [33] relies on MAC prioritization to provide soft guarantees by locally considering the velocity at a node which can be dynamically readjusted. An extension to SPEED has been recently proposed in [34] where reliability requirements are considered. In [35] the authors proposed DEED, a dissemination tree which is constructed while considering end-to-end delay constraints.

In order to support high delivery ratios, congestion control [36][17] in combination with rate control [37] techniques have been proposed for sensor networks. In these works, the goal is to adjust the rate using Additive-Increase-Multiplicative-Decrease (AIMD) or similar schemes to react in the case of network overload. In [38], fairness is also achieved by regulating and balancing the incoming load introduced from each sensor. This technique can be extended to consider different priority classes, however a tree–based routing protocol is required to achieve this functionality. In [16], the joint problem of prioritization and scheduling has been studied for real-time traffic management in sensor networks. The goal is

to dynamically regulate real-time traffic to control congestion and schedule the flows end-to-end to meet application real time requirements.

5 Conclusions

In this paper we have studied middleware components for supporting distributed data management computations in sensor networks. Our middleware includes in-network data storage and real-time data dissemination mechanisms that simplify the development of distributed applications and allow us to achieve reliable, real-time sensor data management. Middleware for sensor networks is an emerging and very promising research area. We believe that as future sensor devices will feature more resources, more complex in-network processing applications will be developed which will further increase the resource demand and wide adoption of sensor network applications.

Acknowledgments

This work has been supported by NSF Awards 0330481 and 0627191.

References

1. Szewczyk, R., Mainwaring, A., Polastre, J., Anderson, J., Culler, D.: An analysis of a large scale habitat monitoring application. In: SenSys'04. (2004)
2. Sadler, C., Zhang, P., Martonosi, M., Lyon, S.: Hardware design experiences in zebranet. In: ACM SenSys'04. (2004)
3. Xu, N., Rangwala, S., Chintalapudi, K., Ganesan, D., Broad, A., Govind an, R., Estrin, D.: A wireless sensor network for structural monitoring. In: ACM SenSys'04. (2004)
4. Crossbow'05: Crossbow Technology Inc., (http://www.xbow.com/)
5. Madden, S., Franklin, M., Hellerstein, J., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. In: OSDI 2002, Boston, MA (2002)
6. Banerjee, A., Mitra, A., Najjar, W., Zeinalipour-Yazti, D., Kalogeraki, V. an d Gunopulos, D.: Rise co-s : High performance sensor storage and co-processing architecture. In: IEEE SECON'05. (2005)
7. Desnoyers, P., Ganesan, D., Shenoy, P.: Tsar: A two tier sensor storage architecture using interval skip graphs. In: SenSys'05. (2005)
8. Mathur, G., Desnoyers, P., Ganesan, D., Shenoy, P.: Ultra-low power data storage for sensor networks. In: IEEE SPOTS'06. (2006)
9. Bonnet, P., Gehrke, J., Seshardi, P.: Toward sensor database systems. In: MDM 2001, Hong Kong (2001)
10. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: The design of an acquisitional query processor for sensor networks. In: SIGMOD 2003, San Diego, CA (2003)
11. Considine, J., Li, F., Kollios, G., Byers, J.: Approximate aggregation techniques for sensor databases. In: ICDE 2004, Boston, MA (2004)
12. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. In: OSDI 2002, Boston, MA (2002)

13. Soheili, A., Kalogeraki, V., Gunopulos, D.: Spatial queries in sensor networks. In: International Symposium on Advances in Geographic Information Systems (GIS 2005) , Bremen, Germany, November. (2005)
14. Zeinalipour-Yazti, D., Vagena, Z., Gunopulos, D., Kalogeraki, V., Tsotras, V., Vlachos, M., Koudas, N., Srivastava, D.: The threshold join algorithm for top-k queries in distributed sensor networks. In: Intl. Workshop on Data Management for Sensor Networks DMSN (VLDB'2005), Trondheim, Norway (2005)
15. Zeinalipour-Yazti, D., Lin, S., Kalogeraki, V., Gunopulos, D., Najjar, W.: Microhash: An efficient index structure for flash-based sensor devices. In: 4th USENIX Conference on File and Storage Technologies (FAST 2005), San Fransisco, CA (2005)
16. Karenos, K., Kalogeraki, V.: Real-time traffic management in sensor networks. In: In Proc. of RTSS, Rio de Janeiro, Brazil (2006)
17. Karenos, K., Kalogeraki, V., Krishnamurthy, S.V.: A rate allocation framework for supporting multiple classes of traffic in sensor networks. In: In Proc. of RTSS. (2005)
18. Madden, S., Franklin, M., Hellerstein, J., Hong, W.: The design of an acquisitional query processor for sensor networks. In: SIGMOD'03. (2003)
19. Tao, Y., Papadias, D.: Spatial queries in dynamic environments. ACM Trans. Database Systems **28** (2003) 101–139
20. Sahin, O.D., Gupta, A., Agrawal, D., Abbadi, A.E.: A peer-to-peer framework for caching range queries. In: Proc. of ICDE 2004, Boston, MA (2004)
21. Zheng, B., Xu, J., Lee, W.C., Lee, D.L.: Grid-partition index: A hybrid ap-proach to nearest-neighbor queries in wireless location-based services. VLDB Journal (2004)
22. Gupta, A., Agrawal, D., Abbadi, A.E.: Approximate range selection queries in peer-to-peer systems. In: CIDR 2003, Asilomar, CA (2003)
23. Ferhatosmanoglu, H., Demirbas, M.: Peer-to-peer spatial queries in sensor networks. In: 3rd IEEE International Conference on Peer-to-Peer Computing (P2P 2003), Linkoping, Sweden (2003)
24. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for network sensors. In: ASPLOS'00. (2000)
25. Dai, H., Neufeld, M., Han, R.: Elf: an efficient log-structured flash file system for micro sensor nodes. In: ACM SenSys'04. (2004)
26. Greenstein, B., Estrin, D., Govindan, R., Ratnasamy, S., Shenker, S.: Difs: A distributed index for features in sensor networks. In: Elsevier Journal of Ad Hoc Networks 2003. (2003)
27. Li, X., Kim, Y., Govindan, R., Hong, W.: Multi-dimensional range queries in sensor networks. In: In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems. (2003)
28. Woodhouse, D.: Jffs : The journalling flash file system. (In: Red Hat Inc. http://sources.redhat.com/jffs2/jffs2.pdf)
29. Wookey: Yaffs - a filesystem designed for nand flash. (In: Linux 2004 Leeds, U.K)
30. Yang, X., Vaidya, N.H.: Priority scheduling in wireless ad hoc networks. In: MobiHoc '02: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing. (2002) 71–79
31. Woo, A., Culler, D.E.: A transmission control scheme for media access in sensor networks. In: Proc. MobiCom. (2001) 221–235
32. He, T., Stankovic, J.A., Lu, C., Abdelzaher, T.F.: SPEED: A stateless protocol for real-time communication in sensor networks. In: Proc. of ICDCS. (2003)

33. Lu, C., Blum, B.M., Abdelzaher, T.F., Stankovic, J.A., He, T.: RAP: A real-time communication architecture for large-scale wireless sensor networks. In: Proc. IEEE RTAS. (2002)
34. Felemban, E., Lee, C.G., Ekici, E., Boder, R., Vural, S.: Probabilistic QoS guarantee in reliability and timeliness domains in wireless sensor networks. In: INFOCOM. (2005)
35. Kim, H.S., Abdelzaher, T.F., Kwon, W.H.: Dynamic delay-constrained minimum-energy dissemination in wireless sensor networks. ACM Transactions on Embedded Computing Systems **4** (2005) 679–706
36. Wan, C.Y., Eisenman, S.B., Campbell, A.T.: CODA: Congestion detection and avoidance in sensor networks. In: Proc. ACM SenSys. (2003)
37. Sankarasubramaniam, Y., Akan, O., Akyildiz, I.: ESRT: Event-to-sink reliable transport in wireless sensor networks. In: Proc. of ACM MOBIHOC. (2003)
38. Ee, C.T., Bajcsy, R.: Congestion control and fairness for many-to-one routing in sensor networks. In: Proc. ACM SenSys. (2004)

Oscar: Small-World Overlay for Realistic Key Distributions*

Sarunas Girdzijauskas[1], Anwitaman Datta[2], and Karl Aberer[1]

[1] Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland
{sarunas.girdzijauskas,karl.aberer}@epfl.ch
[2] Nanyang Technological University (NTU), Singapore
anwitaman@ntu.edu.sg

Abstract. The research on P2P systems which support skewed key distributions has rapidly advanced in the recent years. Yet, the assumptions on the skews we are dealing with remained pretty simple: most of the existing literature assumes simple monotonous key distribution skews. However, this is not always the case. For example, Gnutella filename traces show that complex key-distributions rather than monotonous skews occur in practice. We show that one of the seminal P2P systems which support skewed keys - Mercury [7], performs poorly given such complex distributions generated from the trace of Gnutella filenames. We discuss the shortcomings of such state-of-the-art techniques. We present an overlay network *Oscar*, based on a novel overlay construction mechanism, which does not depend on the key-distribution complexity. We demonstrate through simulations that our technique performs well and significantly surpasses Mercury for such realistic workloads.

1 Introduction

The rapid spread of broadband Internet allowing numerous users to equally participate in the communication process uncovered the broad potential of P2P networks. Free of the drawbacks of centralized systems and with a potential of providing scalable, fault tolerant services, P2P systems flourished in the recent years. A wide range of P2P systems were proposed (e.g. [1,15,17]) and P2P became widely recognized as a fundamental computing and networking paradigm. Structured P2P systems started being considered as the next generation application backbone on the Internet. Although a vast majority of structured P2P systems are conceptually similar, they differ in the rules which describe how to choose the neighboring links at each peer (to form routing tables). These rules

* The work presented in this paper was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935. The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 247–258, 2007.

heavily depend on the nature of the peer identifiers (keys). To ease the network construction task most of the structured P2P systems use uniform hash functions for assigning identifiers to the peers. This, however, limits the use of data-oriented P2P systems, e.g. only simple keyword search capabilities are supported. One of the biggest challenges in such data-oriented systems is to preserve semantic relationships among resource keys (such as ordering or proximity), or to allow semantic data processing (such as complex queries or information retrieval). This implies that the construction of (efficient) overlay networks has to support the use of non-uniform hashing functions.

However this is not a trivial task and up to now there exist only very few overlay networks which support non-uniform key distributions. Some examples are CAN [8], Mercury [7], P-Grid [1], skip graphs [5,11] and their derivatives [4,9]. These however suffer from other problems, e.g. the search efficiency in terms of the number of overlay hops cannot be guaranteed in CAN for an arbitrary partitioning of the key-space (zones). Because of its storage load-balancing, P-Grid has highly imbalanced peer in/out-degree. Skip graphs need to have $O(\log N)$ level rings at each peer and do not provide a possibility to choose routing table entries in a randomized manner (level ring neighbors are determined by a peer's membership vector and the existing skew in the system). In such a way skip graphs are lacking the flexibility which is provided by the truly randomized approaches (e.g. based on Small-World construction principles like Mercury) and cannot address some of the heterogeneity issues, e.g. different constraints of bandwidth at each peer. Meanwhile Small-World construction principles do not restrain a peer on having fixed amount of links: the out-degree of a peer can vary, still providing guarantee of efficient search. Because the links are chosen randomly, the in-degree of a peer can also be easily adjusted. Such features of Small-World approaches enables exploiting the heterogeneity of the peers not only storage-wise but also bandwidth-wise. In such a way peers are free to choose the amount of outgoing and incoming links locally, based on their bandwidth or other constraints.

Although Mercury can address most of the aforementioned issues and have many good properties, the sampling technique that Mercury uses does not scale given complex distributions, which however often occur in practice. Mercury can easily deal with "simple" skewed distributions but does not address the fact that "real-world" distributions are highly complex. Later in the paper we will show that in certain cases Mercury nodes will suffer of in-degree imbalance which results in decrease of search performance and increase of the hot-spots in the network. In general, the problem of most of the current state-of-the-art approaches dealing with non-uniform key distributions is the resulting node degree imbalance.

The aforementioned problem with Mercury is that for the construction of the P2P network it uses an *approximation* of the global key distribution from a limited set of uniform samples. Since it is not possible to correctly approximate the distribution if it is highly complex, the resulting P2P networks will have poor performance. Therefore we suggest a novel overlay network *Oscar* (*O*verlays using *SCA*lable sampling of *R*ealistic distributions) based on overlay construction

algorithms which use a *scalable* sampling technique and are capable of efficiently constructing and maintaining "routing efficient" networks given any complex key distribution function. Such "routing efficient" networks will enable to enjoy all the benefits of the system which supports non-uniform key distribution (e.g. range queries) and will not suffer of node in-degree imbalance hence exhibiting nice lookup performance. We will show in an evaluation that for a network of 12000 peers we outperform Mercury by having 20 times lower routing latency in the network, yet using sample sets of comparable sizes and similar workload characteristics.

The paper is organized as follows. In Section 2 we give the background on structured P2P concepts and observe the occurrence of complex distributions. In Section 3 we analyze the drawbacks of existing approaches and illustrate some of their shortcomings with examples. Then in Section 4 we present our proposed solution: a scalable sampling for construction of routing-efficient networks and the corresponding algorithms for building *Oscar* overlay. In Section 5 we show the performance of our proposed approach in the simulation environment and we conclude our findings in Section 6.

2 Background

Before going into details let us introduce basic concepts which are widely used in structured P2P systems and will be used later on in the paper. To facilitate the understanding in the following we will use the notations of concepts which were generalized and classified in [2].

Basic concepts for structured P2P. A structured overlay network consists of set of peers $\mathcal{P}$ ($N = |\mathcal{P}|$) and set of resources $\mathcal{R}$. There exists an identifier space $\mathcal{I}$ (usually on the unit interval $\mathcal{I} \in [0..1)$, e.g. Chord [17], Symphony [15]) and two mapping functions $F_{\mathcal{P}} : \mathcal{P} \rightarrow \mathcal{I}$ and $F_{\mathcal{R}} : \mathcal{R} \rightarrow \mathcal{I}$ (e.g. SHA-1). Thus each peer $p \in \mathcal{P}$ and each resource $r \in \mathcal{R}$ are associated with some identifiers $F_{\mathcal{P}}(p) \in \mathcal{I}$ and $F_{\mathcal{R}}(r) \in \mathcal{I}$, respectively. There exists a distance function $d_{\mathcal{I}}(u, v)$ which indicates the distance between a peer u and a peer v in $\mathcal{I}$. Each peer p has some short-range links $\rho_s(p) \subset \mathcal{P}$ and long-range links $\rho_l(p) \subset \mathcal{P}$ which form a peer's routing table $\rho(p) = \rho_s(p) \cup \rho_l(p)$. There exists a global probability density function f setting the manner of how peer identifiers are distributed in $\mathcal{I}$. Any resource $r \in \mathcal{R}$ in the P2P system can be located by issuing a query for $F_{\mathcal{R}}(r)$. In structured P2P systems queries are usually routed in a greedy fashion, i.e. always choosing the link $\rho \in \rho(p)$ which minimizes the distance to the target's identifier.

Complex Distributions. As discussed earlier in data-oriented P2P applications using a uniform hash function $F_{\mathcal{R}}$ (e.g. SHA-1) is not adequate and we will have to deal with hash functions, which produce highly skewed key distributions. In the following we will show an example by which we illustrate the necessity of having key spaces with complex skews.

Let us assume a data-oriented P2P system where the resources are identified and looked-up by filenames. A widely used technique in P2P systems is the

following: each peer p and each resource r have identifiers $F_{\mathcal{P}}(p)$ and $F_{\mathcal{R}}(r)$ on a 1-dimensional ring $\mathcal{I} \in [0..1)$. Each peer is responsible for all the resources which map to the identifier range $D(p) \in \Big[F_{\mathcal{P}}(p), F_{\mathcal{P}}(p_{succ})\Big)$, where the peer p_{succ} is the successor of the peer p on the identifier ring $\mathcal{I}$. We cannot use any uniform hash function such as SHA-1, as used in DHTs like Chord and Pastry, since we want the function $F_{\mathcal{R}}$ to preserve ordering relationships between the resource keys (e.g. enabling the straightforward use of range queries), i.e. $F_{\mathcal{R}}(r_i) > F_{\mathcal{R}}(r_j)$ iff $r_i > r_j$. Such a order preserving function will lead to very skewed distribution of resource identifiers over the identifier ring $\mathcal{I}$. For example in Figure 1(a) (dotted line) we can see a distribution function of filename identifiers in $\mathcal{I}$ extracted from Gnutella trace (20'000 filenames crawled in 2002). Despite the complex skew, we would like that each peer p would be responsible for a "fair" (or equal) amount of resources and be storage-load balanced, i.e. $|\mathcal{R}_{p_i}| \approx |\mathcal{R}_{p_j}|$ for any i and j, where $\mathcal{R}_p \subset \mathcal{R}$ and $\forall r \in \mathcal{R}_p | F_{\mathcal{R}}(r) \in D(p)$. In such a way the peer identifiers will have to reflect the distribution of resource identifiers. Hence the peer identifier distribution will have a similar shape as the resource identifier distribution. Since in general the resource identifier distributions are usually non-uniform and exhibit complex skews, the resulting peer identifier distribution will have to have a complex skew as well.

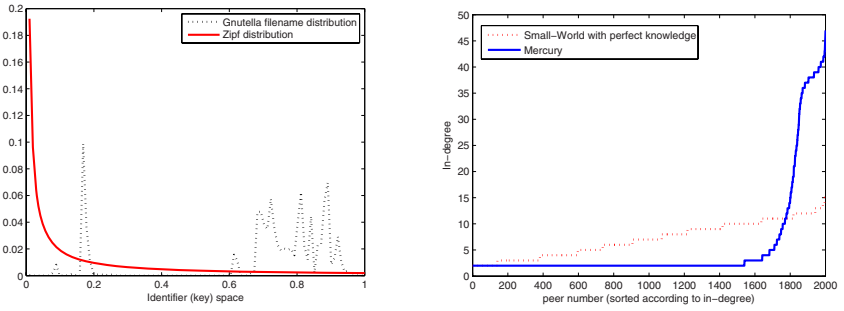
3 Problems with Existing Solutions

Dealing with skewed spaces. In the seminal work of Kleinberg [12] it has been shown how to construct "routing efficient" network on d-dimensional mesh. The follow up works [6,15] showed how to adopt the "kleinbergian" network construction principles for P2P systems with uniform key distribution. In [10] it has been shown that it is indeed possible to construct "routing efficient" small-world network in the 1-dimensional space even if the peers are non-uniformly distributed on the unit interval. For doing so it was shown that a peer u has to choose a peer v as long-range neighbor with a probability that is inversely proportional to the integral of the probability density function f between these two nodes, i.e.

$$P[v \in \rho_l(u)] \propto \frac{1}{|\int_{F_{\mathcal{P}}(u)}^{F_{\mathcal{P}}(v)} f(x)dx|}. \tag{1}$$

However, it is non-trivial to apply this technique in practice because it requires at each peer the global knowledge about the data load in the system, hence the key distribution f. Therefore, an efficient algorithm to gather the global distribution locally is needed. One possible technique was proposed in Mercury [7]. Since Mercury has the most advanced features comparing to other similar approaches (as discussed in the Introduction) in the following we will compare our work only to Mercury.

Problems with Mercury. The idea in Mercury was to uniformly sample the network and approximate the distribution function f. However, this approach assumes very "simple" distribution functions where the peer load changes



(a) Probability density functions: *Zipf (solid) vs Gnutella filename(dotted)*

(b) In-degree of the resulting network of 2000 peers (sorted according to in-degree)

Fig. 1.

monotonously over the key space, e.g. Zipf (Figure 1(a), solid line). In fact, as previously discussed, the load distributions are much more complex in the "real-life". It is clear that to correctly approximate a "simple" monotonous Zipf distribution we need only a few samples k. But when the distribution becomes more complex, a larger number of samples becomes necessary. This hinders the scalability of the system. It is easy to notice that the more resources are in the system, the more complex the distribution function f becomes. Such distributions might be totally arbitrary and the only sufficient "approximation" of the distribution would be gathering in a sample set the complete set of values. That of course would not scale. Thus we will be forced always to use "inaccurate approximations" which in turn would lead to poor performance of the resulting system. The network constructed based on a inaccurate approximation of the distribution function will suffer heavy in-degree imbalance, resulting in a high increase of the messages in the system.

Evaluation of Mercury. Our intuition is validated by simulations using real traces of the aforementioned Gnutella file name distribution. We started the network from the scratch and using Mercury's technique we repeatedly added peers to the network until it reached a network size of 2000 peers. In one case we assumed "perfect" knowledge of the distribution and in the second we relied on the random sample set gathered at each peer, similarly to Mercury's sampling technique. In Figure 1(b) we can see the distribution of the node in-degrees in both cases. It can be clearly seen that using only a partial view of the distribution some peers were heavily loaded while others almost did not have any incoming links. The search cost of the network constructed using a partial view of the distribution had also increased. In this network on average 2.9 times more messages have been generated during search than in the network with perfect knowledge.

The problem with this example is that Mercury seeks to approximate the whole distribution. But as we show, such an attempt is not feasible - distributions are too complex and Mercury's sampling technique will never scale. In

the following we will present a scalable distribution sampling technique together with the modified overlay construction algorithm which scales given a probability density function of any complexity.

4 Network Construction Using Scalable Sampling

Since it is obvious that "approximating" the whole distribution function is not scalable, one could ask the question: "Do we really need to know the distribution function for the whole identifier space with the same granularity?". Maybe it is enough to "learn" well only some regions of the identifier space while leaving some others vaguely explored. Indeed, it is. We will present here a novel P2P construction technique which will require the information only regarding a very small part of the global distribution.

4.1 The Insight and the Proposed Method

According to the continuous Kleinberg's approach [6,15,10] for construction of "routing efficient" network in 1-dimensional space, each node u has to choose two short-range neighbors and one or more long-range neighbors. Short-range neighbors of u are immediate successor and predecessor on the unit ring. A peer u chooses it's long-range neighbor v in the unit interval with the following pdf $g(x)$, where $x = d_{\mathcal{I}}(u, v)$:

$$g(x) = \int_{\frac{1}{N}}^{1} \frac{1}{x \ln N} dx \tag{2}$$

It has been proven that a network constructed in such a way is a "routing-efficient" network, where a greedy routing algorithm on expectation requires $O(\frac{\log_2^2 N}{l})$ hops, where l is the number of long range links at every peer. This means that a node u will tend to choose a long-range neighbor v rather from its close neighborhood than from the further regions. The pdf $g(x)$ according to which the neighbors are chosen also has one nice property when we partition it into logarithmic partitions. If we partition the identifier space into $\log_2 N$ partitions $A_1, A_2, ..A_{\log N}$, such that the distance between the peer u and any other peer v in $A_i$is bounded by $2^{-i} \leq d_{\mathcal{I}}(u, v) < 2^{-i+1}$ the peer v will have equal probability to be chosen from each of the resulting partitions (Figure 2). Indeed, the probability that v will be chosen by u in some interval A_i is exactly $\frac{1}{\log N}$ and does not depend on i:

$$P(F_{\mathcal{P}}(v) \in A_i) = \int_{2^{i-\log N-1}}^{2^{i-\log N}} \frac{1}{x \ln N} dx = \frac{1}{\log N} \tag{3}$$

In practice choosing non-uniformly at random but according to some continuous pdf is complicated. Thus equation 3 gives us an insight of how to modify a network construction algorithm, in which the neighbors will be chosen not by some continuous pdf $g(x)$, but uniformly at random in certain regions. If we do

not violate the pdf characteristics all the desirable properties of the "routing efficient" network will be preserved. We propose the long-range link acquiring procedure in the following way: *each peer u first chooses uniformly at random one logarithmic partition and then within that partition uniformly at random one peer v. This peer v will become a long-range neighbor of u.* Of course, this approach perfectly fits the case with uniform key-distributions. In such cases the partitions can be recalculated in advance at each peer. However, when assuming skewed key-spaces, it is not straightforward how to define logarithmic partitions, hence how to choose the long-range link.

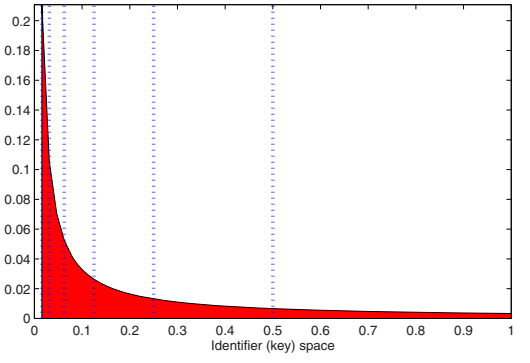


Fig. 2. pdf $g(x)$ (solid bars) and the $A_1, A_2, .., A_{\log_2 N}$ partitions separated by the dotted lines

In the case of uniform peer identifiers the expected number of peers within some range of length d is actually equal to $d \cdot N$ assuming unit length identifier space. Thus, the division of such an identifier space into logarithmic (base-2) partitions is nothing else but recursively halving the peer population. That means a peer u with identifier 0 ($F_{\mathcal{P}}(u) = 0$) will define the partition A_1 which will contain half of the peer population, i.e. all the peers which identifiers are bigger than $\frac{1}{2}$, A_2 – all the peers which identifiers are bigger than $\frac{1}{4}$ and smaller than $\frac{1}{2}$ and so on. This technique can be easily adopted to the case with any identifier skew, just instead of using the predefined borderlines between the logarithmic partitions we will use a median value of the logarithmically decreasing peer populations, i.e. the border between A_2 and A_1 will be the median m_1 of the whole peer population $\mathcal{P}$, the border between A_3 and A_2 will be the median m_2 of the subpopulation $\mathcal{P} \setminus A_1$ etc. In general the border value between A_{i+1} and A_i will be the median m_i of the subpopulation $\mathcal{P} \setminus B_i$, where $B_i = \cup_{j=1}^{i-1} A_j$. For finding the median values we can employ the method proposed by Mercury for collecting the set random peer identifiers in the network, i.e. by issuing random walkers with $TTL = \log_2 N$. To sample the subsets of the population B_i the random walkers will have to be slightly modified. The random walkers will have to choose randomly not any link from $\rho(p)$ of some current peer p, but a random

peer from $\rho'(p)$, where $\rho'(p) = \{u \in \rho(p) | F_{\mathcal{P}}(u) \in \left[F_{\mathcal{P}}(p), m_i\right)\}$. Simulation experiments show that such a technique yields very good results in practice even with very low sample sizes (~ 5 samples per subset as we show in Section 5).

4.2 The Algorithms for Long-Range Link Construction

Here we will formally describe *Oscar* network construction model and the algorithms. Each peer has to perform three operations: *join*, *rewireLongLinks* and *leave*. For the successful construction of a P2P network each peer has to establish short-range links and long-range links. Since we use the same techniques for short-range link establishment and maintenance (e.g. self-stabilizing, ring maintenance algorithms [3,13,14,16,17]) as similar approaches such as Chord, Symphony or Mercury we will not discuss it in detail. The establishment of short-range links ensures correctness of the greedy routing algorithm[1], while long-range links are peculiar to different approaches and serve as "routing optimization" links. Thus we will focus mainly on the main method of our proposed approach, the *rewireLongLinks* algorithm.

In *Oscar*, as in many other P2P approaches for joining the network a peer u has to know at least one peer already present in the system and to contact it. The joining peer is assigned with some identifier $F_{\mathcal{P}}(u)$. For joining the network u issues a query with it's identifer $F_{\mathcal{P}}(u)$ and it inserts itself into the unit ring between the responding peer and its successor (establishment of the short-range links). Afterwards the peer u establishes l long-range links using the *longRangeLink* algorithm.

The *randomBoundedWalk* algorithm. For successful usage of the *longRangeLink* algorithm it is necessary to be able to sample not only the whole population of peers $\mathcal{P}$ but also some subpopulation of peers B. Therefor a specific random walk algorithm is needed. The algorithm will produce random walkers which would be able to "walk" only within a subpopulation of peers $B \subset \mathcal{P}$ restricted by a predefined scope variable *range*, such that $p_B \in B$ iff $F_{\mathcal{P}}(p_B) \in range$. Such an algorithm *randomBoundedWalk* (Algorithm 1) is a modified random walker, where the message is forwarded not to any random link of the current message holder u, but to a randomly selected u's link p, which satisfies the condition $F_{\mathcal{P}}(p) \in range$(Algorithm 1, line 3).

The *longRangeLink* algorithm. To choose a neighbor according to the proposed technique the *longRangeLink* algorithm (Algorithm 2) has to define $O(\log_2 N)$ logarithmic partitions (ranges) in the identifier space. To find the fist partition the *longRangeLink* algorithm starts with issuing k random walkers within the defined *range* of the identifer space using the *randomBoundedWalk* algorithm. Initially the defined *range* spans the whole identifier space starting from the identifier of peer's u successor on the identifier ring up to the peer's u

[1] Establishment of short-range links results in a virtual ring topology in such a way ensuring the correctness of the greedy routing algorithm (a message *always* can be forwarded closer to a target).

Algorithm 1. Bounded Random Walk Algorithm $[r] = randomBoundedWalk(u, range, TTL)$

1: **if** $TTL > 0$ and $R \neq \oslash$ **then**
2: $TTL = TTL - 1$
3: $R = \{p \in \rho(u) | F_{\mathcal{P}}(p) \in range\}$;
4: $next = chooseRandomly(R)$
5: $[r] = randomBoundedWalk(next, range, TTL)$
6: **else**
7: $r = u$
8: **end if**

identifier itself (Algorithm 2, line 2). After the collection of the random sample set P_{sample} the peer u finds the median value of all the peer identifiers of the set P_{sample} (line 9). Having the median value the peer u can define the first, furthest, partition A_1 which will span the identifier space from the found median value up to the peer's u identifer value (line 14). The *range* value for performing the next random walk within the subgraph $\mathcal{P} \setminus A_1$ is reduced (line 18) and the algorithm continues by repeating the same steps (lines 4- 19) to find the successive partitions $A_2, A_3, ..$ etc. The algorithm stops finding the partitions when the median value is equal to the identifier of the u's successor $F_{\mathcal{P}}(u_{successor})$ (line 10). In such a way the algorithm acquires on expectation $\log_2 N$ partitions. To assign the long-range link, the *randomBoundedWalk* algorithm chooses uniformly at random one of the partitions A_i(line 20) and then assigns the random peer v from that partition using *queryToRange* algorithm (line 21). The *queryToRange* algorithm is a greedy routing algorithm, which minimizes distance to the given range A_i and terminates whenever the first peer v in that range is reached. Since the algorithm requires k samples per each logarithmic partition, the expected number of needed samples per peer in total is $O(k \log N)$.

Note that the algorithm does not require knowledge or estimation of the total number of nodes in the network. The only place where theoretically the estimation of N is needed is the TTL value of a random walk. As explained in [7] the TTL should be set to the value of $\log_2 N$. However, the simulations show that it is sufficient to set the TTL value equal to the number of previously calculated partitions. In such a way *Oscar* algorithms are tuned to be independent of the estimation of the network size N.

Since churn exists in P2P networks and the peers join and leave the system dynamically each peer has to rewire it's long range links from time to time. This can be done either periodically or adaptively. One of the dynamic techniques could be monitoring the average amount of hops in the routing path. If the number of routing hops increases above some threshold the long-range link rewiring can be triggered locally and autonomously by individual peers. In such a way the *Oscar* system can self-optimize its performance under dynamically changing network conditions. However for the correctness of the system, as discussed previously, we rely on the already devised self-stabilizing algorithms (e.g. [3,13,14,16,17]) which maintain the virtual ring (short-range links) under churn.

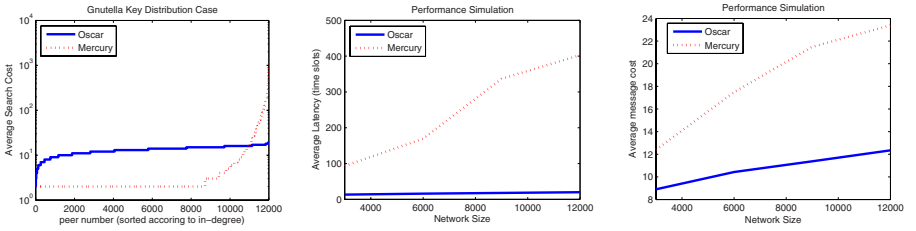
Algorithm 2. Long range link construction algorithm $[v] = longRangeLink(u)$

$\rho(u) = \oslash; i = 0;$
$range = \Big[F_{\mathcal{P}}(u_{successor}); (F_{\mathcal{P}}(u) + 1) \bmod 1\Big);$
$notEnoughPartitions = true;$
while $notEnoughPartitions$ **do**
  $i = i + 1; P_{sample} = \oslash;$
  for j=1 to k **do**
    $P_{sample} = P_{sample} \cup randomBoundedWalk(u, range, TTL)$
  end for
  $m(i) = median(P_{sample})$
  if $m(i) = F_{\mathcal{P}}(u_{successor})$ **then**
    $notEnoughPartitions = false;$
  end if
  if i=1 **then**
    $partitions(i) = \Big[m(i); (F_{\mathcal{P}}(u) + 1) \bmod 1\Big);$
  else
    $partitions(i) = \Big[m(i); m(i-1)\Big);$
  end if
  $range = \Big[F_{\mathcal{P}}(u_{successor}); m(i)\Big);$
end while
choose random partition $randPart$ uniformly from $partitions$;
$[v] = queryToRange(u, randPart)$

5 Simulations

Here we show that the network built according to our proposed technique performs well and does not suffer of the drawbacks that state-of-the-art systems like Mercury have. We base our experiments on a simulation of the bootstrap of the *Oscar* network starting from the scratch and simulating the network growth until it reaches 12000. Since our goal is to show the ability of our proposed technique to construct "routing efficient" networks and dealing with churn is an orthogonal issue, we have simulated a fault-free environment, i.e. system without crashes. Each peer was assigned with an identifier randomly drawn from the Gnutella filename pdf (as in Figure 1(a)).

During the growth of the networks we were periodically rewiring long-range links of all the peers. When the networks have reached the size of 12000 nodes, we have performed three random rewiring on each node and then measured the performance of the networks. For the construction of *Oscar* network we have set the number of samples for each logarithmic partition (parameter k) equal to 5. As suggested in Mercury [7] we have set the parameters k_1 and k_2 values to $\log_2 N$ for constructing the Mercury network. Each peer in our Mercury simulation has constructed a distribution approximation from the sample set of $k_1 \cdot k_2$ random walks. In such a way we have approximated the exchange of Mercury's estimates in an epidemic manner where each peer issues k_1 random walks and each of the selected nodes reports back the k_2 most recent estimates. With such settings the



(a) Distribution of in-degree in the networks (b) Average latency given different network sizes (c) Average search cost given different network sizes

Fig. 3. Simulation Results

average amount of sampling messages are similar in both networks ($5 \log N$ in *Oscar* and $\log^2 N$ in Mercury) what provides comparable simulation conditions.

The results have shown that the *Oscar* network had a much better distribution of in-degree and the resulting lower message cost and routing latency in the network. In Figure 3(a) it can be seen that the original approach used in Mercury has a significantly higher in-degree imbalance. Since in reality each peer can process only limited amount of requests per one time slot, having such high imbalance causes significant routing latency in the network. We have simulated the delay in the networks where each peer could process at most $2 \log N$ requests per one time slot and $\log N$ random queries were issued at each peer per one time slot. After performing the simulation the Mercury network consisting of 12000 peers had a 20 times higher routing latency, and twice higher message cost cost for routing the queries. As shown in Figures 3(b,c) such trends also can be observed given different network sizes (from 3000 to 12000 peers).

6 Conclusions

In this paper, we have addressed the problem of dealing with skewed key distributions in data-oriented P2P environments. We argue that in such environments it is much more likely to encounter highly complex key-distributions, rather than simple monotonic skews. We have shown that with such complex distributions nowadays approaches (e.g. Mercury) cannot cope successfully. In this paper we have presented a novel overly network *Oscar* based on construction algorithms which are capable of constructing "routing efficient" networks given any complex key distribution using a novel technique of scalable sampling. We have shown in our experiments that the proposed approach performs well and outperforms Mercury. As for now, we are working on the analytical part which will theoretically support our proposed technique. Also we are pursuing several possibilities to further improve network construction algorithms. One of them is to reduce the network sampling even more by aggregating already existing sample sets in the network (e.g. using sample sets of a peer's immediate neighbors) instead of sampling the network from the scratch.

References

1. K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *CoopIS*, 2001.
2. K. Aberer, L. Onana Alima, A. Ghodsi, S. Girdzijauskas, M. Hauswirth, and S. Haridi. The essence of p2p: A reference architecture for overlay networks. In *P2P2005, August 31-September 2 2005, Konstanz, Germany.*
3. D. Angluin, J. Aspnes, J. Chen, Y. Wu, and Y. Yin. Fast construction of overlay networks. In *In 17th ACM Symposium on Parallelism in Algorithms and Architectures(SPAA 2005),Las Vegas, NV, USA*, 2005.
4. J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load balancing and locality in range-queriable data structures. In *Twenty-Third Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, 2004.
5. J. Aspnes and G. Shah. Skip graphs. In *SODA*, 2003.
6. L. Barriere, P. Fraigniaud, E. Kranakis, and D. Krizanc. Efficient routing in networks with long range contacts. In *DISC 01, pp 270-284*, 2001.
7. A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *ACM SIGCOMM, Portland, USA*, 2004.
8. P. Fraigniaud and P. Gauron. The content-addressable network d2b. Technical Report Technical Report LRI 1349, Univ. Paris-Sud, 2003.
9. Prasanna Ganesan, Mayank Bawa, and Hector Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *VLDB*, 2004.
10. S. Girdzijauskas, A. Datta, and K. Aberer. On small world graphs in non-uniformly distributed key spaces. In *NetDB 2005, April 8-9 2005 Tokyo, Japan*, 2005.
11. N. J. A. Harvey, Jones, M. B., S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, Seattle, WA*, March 2003.
12. J. Kleinberg. The Small-World Phenomenon: An Algorithmic Perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, 2000.
13. X. Li, J.Misra, and G. Plaxton. Active and concurrent topology maintenance. In *In the 18th Annual Conference on Distributed Computing (DISC)*, 2004.
14. D. Liben-Nowell, H. Balakrishnan, and D. R. Karger. Analysis of the evolution of peer-to-peer systems. In *In Proceedings of the twenty-first Annual Symposium on Principles of Distributed Computing (PODC-02), pages 233242, New York, ACM Press*, 2002.
15. G. S. Manku, M. Bawa, and P. Raghavan. Symphony: Distributed hashing in a small world. In *4th USENIX Symposium on Internet Technologies and Systems, USITS*, 2003.
16. Ayman Shaker and Douglas S. Reeves. Self-stabilizing structured ring topology p2p systems. In *Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, 2005.
17. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM*, 2001.

Keyword Searching in Structured Overlays Via Content Distance Addressing

Yu-En Lu, Steven Hand, and Pietro Lió

University of Cambridge Computer Laboratory
{firstname.lastname}@cl.cam.ac.uk

Abstract. We present a novel keyword search scheme for file sharing applications based on *content distance addressing* (CDA). Through CDA, we are able to associate node IDs with content distances and thus reduce several complex keyword queries to routing problems on structured overlays. Unlike traditional approaches, we require neither set intersection process nor replication for query processing as a result of content distance addressing. In this paper, we present the design and theoretical analysis of CDA for keyword search as well as simulation results using real world parameters.

1 Introduction

There have been extensive studies for keyword query processing in both structured and unstructured overlay design, primarily motivated by the potential to build a P2P search engine. The fundamental query primitive in such system is:

> Given a set of keywords, return *all* objects with description containing these keywords.

Unstructured overlays, such as Gnutella, build local indexes and use flooding or random walks over the overlay to give a complete result. This approach is simple and is effective for objects with lots of identical copies in the network. However, it still requires a broadcast to ensure the completeness of query results. Also, it is not clear as to how exhaustive this approach is for objects of only moderate occurrances.

Structured overlays, in contrast, adopt the distributed data structure paradigm, one of the most recognised approach being distributed hash tables (DHT). DHTs give unique identifiers to each object stored and place them at a unique address in the overlay. This approach gives good performance for exact keyword queries where the *exact* description is given. In fact, Castro et al. [2] showed that structured overlays can give better performance even for flooding-based search schemes.

With an eye on this fundamental difference, several systems have been proposed to combine the best of both worlds. Hybrid search systems [12] proposed to distiguish the use of structure and unstructured overlays by the popularity of data. eSearch [19] publishes each document by each of its keywords and expand search range on DHT by associated metadata (keywords as well).

We argue that the inherent difficulty here is that identifiers in DHT do not pose for any query semantics but uniqueness. Thus, objects containing the same keywords is indifferent from everything else. We argue that whilst identifiers should be made random to give good load-balancing properties, they should not be made pair-wise independent. Instead, identifiers should be able to reflect the notion of distance between objects.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 259–272, 2007.

In this paper, we propose a simple scheme *Content Distance Addressing*, a bundle of a new hash function and additional overlay links, that gives identifiers the notion of distance in terms of Hamming distance. Like DHTs, objects are addressed in the overlay by their identifiers. Unlike DHTs, the object address now reflects its keyword composition using CDA. Like unstructured overlays, we use a kind of multicast to find relevant objects to the keyword query. Unlike unstructured approaches, each hop in the multicast now yields precise query semantics.

The contributions of this paper are three-fold. Firstly, we propose to associate object IDs with distance semantics. A new vaiant of locality preserving hash function, Summary hash, is presented for identifier generation. Modelled as a set of keywords, object descriptions with overlaps in keywords would have close hash values in terms of Hamming distance. We show that Summary hash not only preserves distance semantics, but also provides low collision probability for dissimiliar objects in Section 3.2.

We then formulate the keyword search problem in to Hamming proximity routing problems in a hypercube graph in Section 3.1. We show that the query space defined by these problems corresponds to topologically close regions in the overlay in Section 3.3. Therefore, the keyword query processing problems are reduced to a overlay routing problem. Interestingly, we show that the required primitive is inherent in standard hypercube routing primitives and how to make this scheme applicable to most DHTs today (section 3.5).

Finally, we give a reference implementation of such structured overlay and study its performance under real world dataset via simulation in Section 4.2 and 4.3.

2 A High Level Presentation

2.1 Hashing

Consider each object as b bit string where the value of each bit is initially '0'. Summary hash produce the hash value of an object by flipping a certain bit if the corresponding keyword is present. For example, suppose the corresponding bit for "Harry" and "Potter" is 2 and 3, then the object "Harry Potter" can be encoded as 0110. In Figure 1(a), we can see what hash values the "Harry Potter" related objects may have.

To evaluate the query "Harry Potter" is therefore to first land at the node 0110. Then we may find similar objects related to it by further querying nodes 0111 and 1110 and so on. This parameter b, as we shall see later, turns out to be the parameter for the degree of clustering of the resulted address space. We give the analyses of Summary hash in section 3.2.

To insert an object, we simply computes the Summary hash value and place the object pointer at the corresponding node. Thus, removing an object from the system would require only the deletion of the pointer. Here, we assume that the routing scheme is built into the overlay, we present the details of the overlay in section 3.4.

2.2 Routing in Hypercube Graphs

A b-dimension hypercube C_b is a graph (V, E) where V is the set of nodes labelled by $\{0, 1, 2, \ldots, 2^b - 1\}$ and E is a pair of nodes in V denoting the set of links. Given any

two nodes u, v, there exists an edge $(u, v) \in E$ if and only if the labels of u and v differs at exactly one bit. To route a message from node 0000 to 1111, 0000 can first try to "fix" the first bit by sending the message to 0001, where it is forwarded to 0011 and so on.

Therefore, to probe all neighbours of address within Hamming distance α, it suffices to first send the query message to the neighbours of the seed. These neighbours, each of Hamming distance 1, further propagate the message to their neighbours. Thus, in α hops, all nodes with label within Hamming distance α will be reached.

In practice, however, the number of machines network nodes is unlikely exactly 2^b. A mechanism to emulate a complete hypercube graph is therefore needed. We propose a variant based on a special case of [14]. A *surrogate* is uniquely picked for every hypercube labels by matching their IDs. A node x is said to be the surrogate of y if x's ID has the longest matching prefix to y and x is the smallest in lexicographic order among all nodes with the same matching prefix. Details for how various types of queries perform on the hypercube are presented later in section 3.5. Due to space constraints, we give only the lemmas in the following texts and leave their proofs in the appendix. For a detailed description of the surrogate mechanism, we give an implementation in a declarative language: P2 [13] in appendix.

2.3 Query Processing

We can now introduce the query processing algorithm. The query processing consists of two phases– the landing phase and the probe phase. In the landing phase, one hashes the query to find the node responsible , which we shall refer to as the *seed node*. One can arrive at the seed node by using standard DHT routing schemes. Then one may proceed into the probe phase in which nearby nodes are probed to find objects similar to the query.

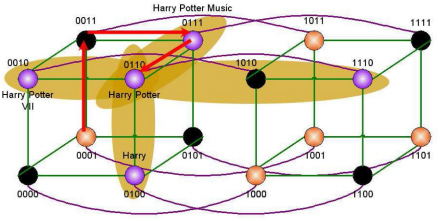
The probe phase is subject to a user specified argument α, which specifies the probing radius in the probe phase.[1] The probe phase would start from the seed node and send query messages to all nodes with overlay address within Hamming distance α. To see how this may be made efficient, we need to introduce the notion of hypercube graphs first.

2.4 Dataset Characterisation

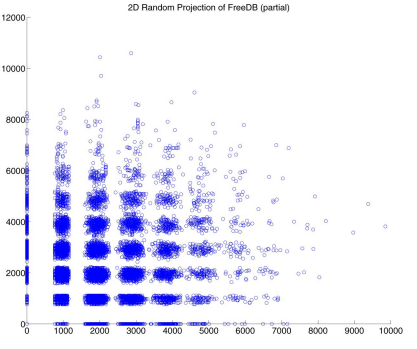
We evaluate our results for the FreeDB [4] database, a free online CD album database containing 21 million songs which shall be representative for current P2P information retrieval applications for its size and diversity.

Analysing the dataset, we have observed that there is a significant skew in the number of keywords for most songs and keyword frequency. It is clear that most songs can be described by only two thousand keywords , and contain no more than thirty keywords. In fact, eSearch [19], a web IR system, evaluated system performance with number of keywords no more than 20 and [6] reported roughly 6 keywords per document in average in Chinese web corpus.

[1] This parameter clearly depends on the characteristics of the dataset and user requirements. For text applications such as information retrieval and online file sharing, having $\alpha = 30$ would suffice, as reported in [19] and confirmed by our statistic of FreeDB.



(a) Object placements as a result of Summary hash over a four dimension hypercube. The edges represent links between a pair of logical nodes. The red path denotes the path from query to seed node on DHT (DHT phase) and shaded area covers nodes to be probed (Probe phase).



(b) A 2D random projection of FreeDB (partial) songs from $\{0,1\}^{60}$ by Summary hash. We project each collection of 30 dimensions to X and Y axis. Notice that the value of X and Y axis means little but the result of random projection. Here, we mean to visualise the clustering property as a result by Summary hash in the high dimension space.

Fig. 1. An example song placement in a hypercube and characterisation of the data space

We demonstrate the main data characteristics in Figure 1(b). Observe that since most songs would cluster around each other in the hash space, our land and probe search mechanism effectively exploits the clustering property as a result of Summary hash.

3 Our Approach

3.1 Query Model

In this subsection, we give a more formal description of our query model. Let $K = \{k_1, k_2, \cdots, k_m\}$ denote the set of all possible keywords and $\mathcal{K}$ be the power set of K, containing all possible sets of keywords. Any keyword query Q must therefore be in $\mathcal{K}$. We denote the set of all objects as O in which every element o has an associated keyword set $K(o)$.

We define several types of query as following:

k-Related objects query: given a query string Q, return the set: $\{o \in O | Q \subseteq K(o), |K(o) \backslash Q| \leq k\}$.
k-Partial match query: given a query string Q, return the set $\{o \in O | K(o) \subseteq Q, |Q \backslash K(o)| \leq k\}$.
(f,k)-Flawed query: given a query string Q and let F be the set of flawed keywords, return the set $\{o \in O | Q \backslash F \subseteq K(o), d(K(o), Q \backslash F) \leq k, |F| \leq f\}$

Notice that the definition of flawed query above assumes the existence of a number of correct keywords in the query. Under this definition, flawed query may be seen as a larger case of k-partial match query.

3.2 Summary Hash

We propose a simple hash method — the *Summary Hash* — that preserves keyword information in the hashed name of an object. The basic idea is to consider b bit string initially set to 0, each keyword is assigned a bit position. Given a keyword description P, we iterate each element p_j and flip the corresponding bit.

More formally, let $\pi : K \rightarrow \{1, 2, \ldots, b\}$ be uniformly random and $P \in \mathcal{K}$. Let a_i denote the number of keywords in P such that $\pi(p_j) = i$. We specify Summary hash $H : \mathcal{K} \rightarrow \{0,1\}^b$ as:

$$h(p) = \sum_{i=1}^{b} 1_{\{a_i \bmod 2=0\}} \cdot 2^{b-i} \tag{1}$$

where a_i is the number of keywords in p such that $\pi(p_j) = i, \forall p_j \in p$ and $1_{\{.\}}$ is the characteristic function. Below, we show that Summary hash (1) is *locality preserving* under Hamming distance metric.

Lemma 1. $\forall p_1, p_2 \in \mathcal{K}$, $d_H(p_1, p_2) \leq d_H(h(p_1), h(p_2))$.

While Summary hash does give similar hash values for similar descriptions, we shall also consider the collision probability, i.e., the chances for dissimilar descriptions to have similar hash values. Let m be the number of keywords. Here, we consider each object as a m-dimensional $\{0,1\}$ vector and Summary hash serves as the projection function to $\{0,1\}^b$. We give the collision probability below, notice that we do not need m to be known. Real world dataset results are shown later in Section 4.3.

Lemma 2 (Discrimination). *Let $B \subset \{0,1\}^m$, $|B| = c2^{b/2}$ where $0 \leq c \leq 1$. Pr[$h_{|B}$ is one to one] $\geq 1 - c^2$ when $m >> b$ and m is the number of keywords.*

In addition to collision probability, it is also important to consider the load-balancing property. We show that, at least in theory, Summary hash partitions the keyword space to hash space fairly below.

Lemma 3 (Equal Division of Space). *Let $h : \{0,1\}^m \rightarrow \{0,1\}^b$ and $m \geq b > 0$.* $\forall v \in \{0,1\}^b, |h^{-1}(v)| = 2^{m-b}$.

3.3 Keyword Query Reduction

Given the basic properties of the hash function, in this section, we deduce the set of points covered by the query to the nodes in $C_b = (V, E)$ where C_b is a b-bit hypercube. Let $B_v(k) = \{u | d_H(u, v) \leq k, u, v \in V\}$ (the ball of radius k around node v). We bound the range of these nodes in C_b in Theorem 1.

Theorem 1. *Both k-related objects and k-partial match query Q can be answered by probing $B_{h(Q)}(k)$. (k, f)-flawed query can be answered by probing $B_{h(Q)}(k + f)$.*

One may wonder if Summary hash end up packing most of the objects into the same subcubes in C_b. The following lemma bounds the overlapping region for two arbitrary Hamming balls of radius α. This result shows us that clusters of related objects are safely

stored in Hypercubic manifolds of their own [2]. Lemma 4 implies that given two clusters, say documents related to "Harry Potter" and "Prince of Persia", would have very little overlapping and thus each cluster tends to be assigned to different groups of nodes.

Lemma 4. *Let $u, w \in V$ and $d(u,w) = \Delta$, $0 \leq \Delta \leq 2\alpha$ where α is a constant. Then we have*

$$|B_u(\alpha) \cap B_w(\alpha)| = \sum_{a=0}^{\alpha} \sum_{t} \binom{\Delta}{t} \binom{m-\Delta}{a-t}$$

where
$t = [\max\{0, \lceil \frac{a+\Delta-\alpha}{2} \rceil\}, \min\{a, \Delta\}]$ *if* $\lceil \frac{a+\Delta-\alpha}{2} \rceil \leq \min\{a, \Delta\}$. *Otherwise, t = nil.*

Now we can turn our attention to the flawed query. Whilst most typos in keywords may be corrected by using a dictionary, this technique is useful for recovering other potentially important keywords in the system as well. In fact, in traditional IR systems relevance feedback– the process to produce more accurate query string by adding more keywords – is done by picking terms from the retrieved documents. Here, we show how this may cost in Summary hash based systems.

Recall that f is the number of flawed keywords or keywords to expand and Q is the set of query keywords. Intuitively, suppose $f = |F|$ is not large, the seed $h(Q)$ will not miss the correct seed too much and therefore shall still cover a fraction of the correct set of nodes. Therefore, one shall be able to collect objects that is in $B_{h(Q)}(\alpha) \cap B_s(\alpha)$ and therefore recover the f flawed keywords. The following lemma comes directly by the application of lemma 4 and standard Chernoff bound. It gives the precise probability we can recalibrate to the correct seed node s from its neighbours according to their distance.

Lemma 5. *Let D_i be the number of objects of distance i to s, i.e., $|\{v|d(s,v) = i\}|$ and f be the number of flawed keywords in the query Q. Let $\mathcal{E}_i$ denote the event that query Q cannot find 1 object containing s in $\{v|d(s,v) = i\}$.*
$Pr[\bigcap_{i=0}^{\alpha} \mathcal{E}_i] \leq \prod_{i=0}^{\alpha} (D_i p)^{1/D_i p}$
where $p(f,b,\alpha) = \frac{\sum_t \binom{2f}{t}\binom{b-2f}{\alpha-t}}{\binom{b}{\alpha}}$ *and* $t = [\max\{0, \lceil \frac{i+2f-\alpha}{2} \rceil\}, \min\{i, \Delta\}]$ *if* $\lceil \frac{i+\Delta-\alpha}{2} \rceil \leq \min\{i, \Delta\}$. *Otherwise, t = nil.*

3.4 Keyword Edges and Emulation

So far, we have discussed how keyword queries can be processed on the hypercube graph C_b. Here, we would like to discuss possible technical realisations of C_b in current overlays. The easiest way is to embed C_b into any of the current DHT overlays. Such an embedding may be realised by using the surrogate mechanism as described in Section 2.2. Similar to [14], such design would require at most b edges for each node and therefore incurrs rather small maintainence overhead.

Another approach is to emulate a hypercube from current DHTs. With little modification, Chord [18] can simulate hypercube as well. Observe that in contrast to Hypercube,

[2] Numerical results suggests that with suitable b and α, for example $b = 100, \alpha = 30$ the overlapping region is roughly 10^{-5} of the volume of the ball.

each i-th finger of node x in Chord links nodes of ID $x + 2^i \mod 2^b$ on a ring. The neighbours of a node in the hypercube is either the direct i-th successor of a node or its i-th predecessor in Chord. Also, it is also possible for other existing DHTs to emulate hypercube. For example Hypercubic graphs such as de Bruijn and Butterfly graphs can simulate hypercube with only constant slowdown [9]. Therefore the same algorithms can be used for Koorde [7] to simulate the hypercube.

Our implementation choice here is to directly simulate the DHT with the hypercube using the surrogate definition as in [14]. Due to space limitations, we provide the detailed implementation using P2 [13] in appendix.

3.5 Probing the Ball

In section 3.3, we have shown that processing these types of query are equivalent to performing the query on nodes of a bounded Hamming distance to the seed. Therefore, the fundamental primitive supporting these queries is to "multi-cast" the query to every node within that distance from the seed.

More specifically, one first choose the seed node using $h(Q)$ where Q is a query (which may be expanded by the search application) and multi-cast from seed the query to every node within distance α. We can see that the search is essentially first reaching the seed via DHT routing and then route to all neighbours within Hamming distance α from seed.

Routing toward $h(Q)$ can be done similarly to the standard hypercube routing discussed in Section 2.2. At each hop, each node picks either one of its neighbors or itself by the length of matching prefix. If the length is the same, it forwards the message towards the node with smaller ID. Routing terminates when a node finds no other but itself to forward to.

The probing procedure for complete hypercube graphs can be revised to fit our surrogate routing as in Algorithm 1.

The correctness of Algorithm 1 is as following:

Lemma 6. *Let $Probe(u, \alpha)$ denote the set of nodes reached by Algorithm 1 where $u \in V$ and α is constant. $Probe(u, alpha) \equiv B_u(\alpha)$*

Algorithm 1. Probing $B_u(\alpha)$

```
Procedure Probe(Message msg, String hv, Integer pos, Integer step)
  processMessage(msg);
  if step==0 then
    return
  end if
  for each i-th neighbor y of x where i ≥ pos do
    if y equals x or d_H(y, hv) ≤ α then
      return
    end if
    y.Probe(msg, hv, i+1, step −1)
  end for
```

Now, we analyse the message and time complexity of our search algorithm under the synchronous network model. Notice that we give analyses under synchronous network model so that we may compare with and characterise different systems. The algorithm can clearly operate asynchronously. The following bound for query performance can be obtained by standard Chernoff bound and lemma 6. Observe that $p(b, \alpha)$ controls the fraction of the nodes to be probed. Therefore, even with α can be as high as 30 and above, the actual number of nodes required to probe is only proportion to the ratio of the probing ball and the size of the entire hypercube.

Theorem 2. *Let n be the total number of physical nodes in the Peer-to-Peer name space, b be the dimension of hypercube graph, and $p(b, \alpha) = \sum_{t=1}^{\alpha} \binom{b}{t}/2^b$. The message complexity of the keyword search algorithm with radius α is $\Omega\left(\log n + np(b, \alpha)\frac{\log n}{\log \log n}\right)$ with probability at least $1 - n^{-1}$ on a hypercube overlay and the time complexity is $O(\log n + \alpha)$.*

4 Evaluation

So far, we have described the concept of content distance addressing (CDA). Firstly, Summary hash produces hash values for object descriptions in such a way that similar descriptions would have similar hash values in terms of Hamming distance. Eyeing on the similarity between the keyword query semantics and hypercube routing, we provide reductions from the keyword query semantics to hypercube routing. We then demonstrated how we can built a hypercube routing graph in the face of insufficient nodes and present the routing mechanisms necessary for query processing.

In this section, we evaluate CDA in terms of its retrieval capability measured by *recall rate* [3] determined by α against network cost. Since the radius α would determine the range to be probed in the hypercube, we show how α is related to the recall rate below in section 4.2. Notice that since most songs in FreeDB contains less than 30 keywords, having $\alpha = 30$ would, in most of the cases, give recall rate 1.0. We then evaluate network costs for any given recall rate in section 4.3. Recall that the dimensionality b of the hypercube and the number of nodes populating the address space would collectively decide the number of nodes involved in query processing as shown in Theorem 2. Therefore, we conduct our experiments in both evaluations with different network configurations characterised by $\ll b, n \gg$.

In the figures presented, we show results under various configurations $\ll b, n \gg$ with parameters $\ll 12, 4096 \gg, \ll 20, 4096 \gg, \ll 20, 16384 \gg$, and $\ll 60, 16384 \gg$. We provide these configurations so that one may observe the individual and combined effect of b, n.

4.1 Experimental Setup

In this section, we present simulation results of the proposed scheme. We instantiate a network with n nodes to simulate a b-dimensional hypercube. End to end round trip

[3] The ratio of the number of relevant objects retrieved and number of all relevant objects in the system.

time are derived from the King dataset [5] , measurements to 1,724 DNS servers, to reflect the topologically dispersed nature in P2P systems. We use these entries in King dataset to model inter-domain latency. Each node is assigned to a random domain, and then an intra-domain latency (1-100 ms) is given in order to model the latency occurred inside a domain. That is, the end to end latency from A to B is the sum of A and B's intra-domain latency and the inter-domain latency between them. Although intra-domain latency is not always a uniform distribution, we would like to point out that this range is larger than most LAN based connections and thus shall suffice representing the network in most cases.

Below, we define the metrics for evaluation:

- Recall: Percentage of relevant objects retrieved
- Percentage of Nodes Probed: Percentage of nodes that received message during both DHT and Probe phase.
- Number of Messages
- Elapsed Query Processing Time: The duration of the whole query process: from the time a random node issues a query to the time the last message is processed.
- Bandwidth/Data ratio: Ratio of bandwidth used and actual data size

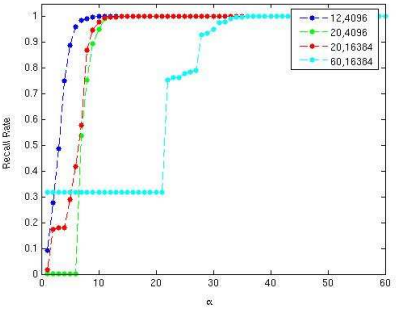
Notice that the precision metric, the ratio of relevant documents retrieved and total number of relevant documents, is always 1.0 under the definition in Section 3.1. This is due to that the query is propagated through the nodes so that the nodes can simply return only the documents with matching keywords.

4.2 Retrieval Performance and Data Placement

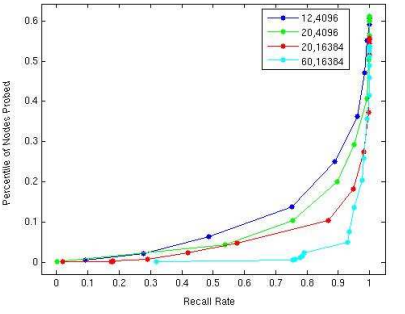
Firstly, we present in Figure 2(a) the relationship between probing radius α and the average recall rate. Notice that whilst this alpha may represent a very large hypercube address space, the number of nodes to probe is in fact dictated by the ratio between the volume of the probing ball and the size of the whole hypercube as suggested by Theorem 2.

As shown in Figure 2(a), the recall rate reaches 1.0 in our sample song set when α reaches around 30 with $\ll 60, 16384 \gg$, confirming our previous observation. Also, most songs may be retrieved with alpha around 20 as shown by $\ll 60, 16384 \gg$. This reflects the underneath data characteristics that most songs contains less than 20 keywords. The curve of $\ll 60, 16384 \gg$ as well as $\ll 20, 4096 \gg$ exhibit stepping behaviour compared to their lower dimension counter-parts. This is due to the fact that surrogate nodes are assigned a larger region in the hypercube and thus one can retrieve more (distant) songs by just visiting one node.

We present the percentage of nodes probed in Figure 2(b). As predicted in Theorem 2, the number of nodes to probe follows the accumulated binomial distribution. As the dimensionality increases, each node is responsible for increasing size of keyword space and thus give dramatic jumps in recall rate whilst using much less nodes. For example, the recall jumps from 0.3 to 0.72 in $\ll 60, 16384 \gg$ when $\alpha = 22$. However, its merely a few more node to probe as shown in Figure 2(b). Moreover, we can observe that CDA scales well as it is insensitive to the size of network , as the cases $\ll 20, 4096 \gg, \ll 20, 16384 \gg$ show. Compare to random walk approaches in unstructured network, we



(a) Recall against α (X-axis) with various configurations $\ll b, n \gg$. Notice increasing the size of the network does not affect recall curve much while change in dimensionality in herently changes the alpha required. The stepping behavior shown by $\ll 60, 16384 \gg$ is due to the effect of surrogate routing.



(b) Percentage of nodes to probe against recall rate (X-axis) required under configuration $\ll b, n \gg$. We may see that CDA is insensitive to the size of network by comparing configurations under same dimensionality with different n.

Fig. 2. Recall rate and its associated costs

can see that CDA gives a precise semantics to the query and the network costs and scales smoothly without using super nodes that stores most of the information in the network.

One may wonder if the scalability demonstrated above is achieved by trading load-balanceness. In Figure 3, we show that this is the case. Indeed, reducing the number of nodes to query implies concentrating data objects onto a smaller region in the overlay and hence the imbalance. However, from the figure, we can observe that this imbalance in fact reflects the data characteristics. Nevertheless, the imbalance is still smooth when we change the dimensionality b or n.

We may see that the load distribution is not entirely uniform in Figure 3. This is due to that although Summary hash does partition the keyword space equally, the load distribution reflects the fact that most songs are clustered around certain topics such as "Mozart" and "Harry Potter". That is, although different clusters are assigned onto different regions in the hypercube by Summary hash, there exists great difference in density among clusters.

In comparison, consider using distributed inverted index where each object is published on per keyword basis. One would find that most documents are described by the most common keywords which could be a few thousand in total. In such cases, there would exist only a few thousand keys in the DHT and therefore adding nodes beyond that scale does not help sharing the load and the load imbalance could thus be even worse. Also, the cost for using bloom filters for large collections has been proven unfruitful as in [21].

This can be observed in Figure 3 where we also simulated an inverted index publishing each song using 6 keywords [4]. We can see that with 6 times more storage, its load distribution, though more even, corresponds to the top 5% loaded nodes in some cases and that can be as low as 2% when we use $\ll b, n \gg = \ll 20, 16384 \gg$.

[4] This number can go up to 20 in [19] and [6] reported 6 in Chinese corpus.

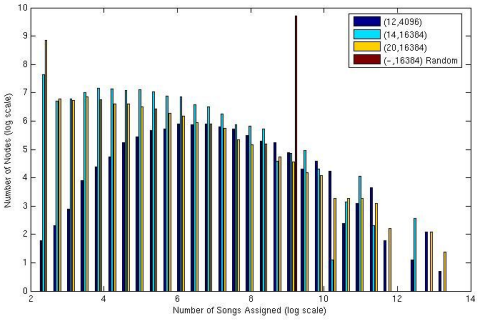


Fig. 3. This is a histogram of distribution of songs in FreeDB where X-axis is the number of songs in log scale and Y-axis is the number of nodes. Notice that the curve of distribution reflects the clustering property of the dataset. Also, the increased imbalance in higher dimension hypercubes reflects the fact that most songs are clustered in low dimension manifolds. We can see that an inverted index using standard DHTs, though spread loads fairer, uses significantly more storage that every node would have the load correspond to the top 5% nodes in $\ll 20, 16384 \gg$.

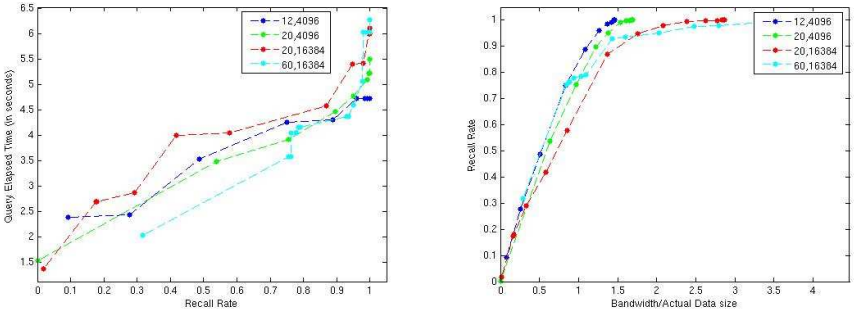
Therefore, we believe that our scheme is better for large scale systems since we do not require expensive intersections as in inverted index and the induced load-imbalance is acceptable even for large datasets such as FreeDB.

4.3 Performance

In this subsection, we look at the recall attained as a function of network cost in terms of number of messages, query elapse time, and overhead ratio. Notice that b is actually the parameter to control the degree of clustering. Therefore, the number of messages required along with the elapsed time improves as b grows larger as suggested by Theorem 2. This is due to the fact that the number of nodes required to attain a given recall rate decreases, as the load distribution become more and more imbalanced as in Figure 3.

Following Theorem 2, the elapse time grows linearly as recall rate increases (see Figure 2(a)). Observe that the seemly dramatic increase in elapse time after recall is greater than 0.7, this is due to that there more far more nodes probed as well as in Figure 2(b). We may see that b value again controls the elapse time by reducing the number of messages required to attain the given recall rate. Notice that when the size of the network increases, the latency does not increase much under the same dimensionality. This is due to the increase of n is comparatively ineffective as the size of the hypercube and probing phase is parallell. We may compare the significant reduction in elapsed time in $\ll 20, 16384 \gg$ and $\ll 60, 16384 \gg$.

Last, but not least, we look at the overhead of our protocol. We can see that the actual data size is roughly equal with processing overheads in large scale network as in Figure 4.3. However, when recall reaches 1.0, larger networks incur larger overhead as eventually the size of the search messages out grow the actual data to be transmitted. In our sampled song clusters such as "Mozart" and "Dvorak" have merely $26,038$ and $5,222$ songs each of which contains less than 70 bytes. Consider the IP packet header



(a) Simulated query elapse time (in seconds) with recall rate (X-axis) using α from 1 to 30 in settings of different $\ll b, n \gg$.

(b) Recall rate (Y-axis) against total bytes spent in settings of different $\ll b, n \gg$. The maximum absolute bandwidth consumption is 70 Kbytes in our simulation. Notice that this is totally acceptable for network of such size in simulation.

Fig. 4. Query Performance Simulations

and control variables to be carried in each query and answer message in a large network, this is not surprising. On the other hand, we would would like to point out that the absolute performance is entirely feasible.

5 Related Work

The most commonly adopted approach to support keyword search on DHTs follows the distributed inverted index paradigm [10,16,19,17,21]. The document space is partitioned by keywords, and each node is responsible for a set of keywords and holds the list of documents containing the keyword. Retrieval is therefore done by a set intersection among the lists of keywords. Tackling the overheads during set intersection, several techniques such as bloom filters [10,16] and top-k pruning [21] are proposed to reduce network bandwidth required for the set-intersection stage and optimise network latency. Some common problems are: inverted indices require storage a constant factor times the size of data and thus the problem of consistency maintenance. [19] adopts a hybrid index scheme to reduce the necessity for excessive set-interaction and storage.

An important approach to tackle the object location problem is to associate network with contents which is sometimes referred to as *semantic routing* [8,3]. Attenuated bloom filter is used in [8] to summarise contents, and [3] establishes the relevance of nodes via statistical estimators.

Recently, various new proposals such as [6,1] take it further, giving query semantics in the network structure. Joung et al. [6] consider keyword queries as binomial tree routing on hypercube graphs. Instead of using Bloom filters for generating identifiers as in [6], we proposed Summary hash which yields provably good collision and load-balancing properties (as shown in Section 3.2) for content distance addressing. Our query semantics are different from the one proposed in [6] and gives a precise control over cost spent. We propose the multicast scheme for *incomplete* hypercube graph instead of assuming DHTs underneath.

Other search scheme exploiting data locality are [20,11,1]. Tang et al. [20] uses very high-dimensional CAN [15] to process queries in the object space. Based on Locality Sensitive Hashing (LSH), [1] constructs the search tree based on hash prefix and gives fast retrieval for similarity searches.

6 Conclusion and Future Works

In this paper, we present a new Peer-to-Peer keyword search scheme based on content distance addressing. CDA associates object identifiers in structured overlays with distance semantics. Therefore, several classes of keyword search problems can be reduced to routing problems on hypercube graphs. Our new hash function significantly improves prior search scheme and we also give methods for constructing hypercubic overlays for efficient multicasting without resorting to an additional layer of DHT. Together, our approach provides precise keyword query semantics with efficiency and effectiveness of structured overlays.

We aim to study in detail on the performance and storage aspects of our scheme as well as comparison with major structured and unstructured keyword search systems in the future. Also, an important problem is with the dimensionality b. As shown in previous analyses, b controls the degree of clustering in the node space and thus suitable α would depend b. The choice of b depends on the scaling property required by the system developer and is preferrable since it enables customisation to suit for various applications. However, $alpha$ would depend on the underneath dataset characteristics. We have shown that our current construct is feasible for online file sharing and IR applications in this paper by evidences in both prior work and statistic in FreeDB. In the future, we plan to investigate how we may extend this scheme further for multimedia databases.

References

1. Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 651–660, New York, NY, USA, 2005. ACM Press.
2. M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proceedings of the Second Symposium on Networked Systems Design and Implementation*, Boston, USA, May 2005.
3. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proceedings of the Twenty Eighth Conference on Distributed Computing Systems*, July 2002.
4. FreeDB.org. FreeDB database . http://www.freedb.org.
5. Thomer M. Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. King dataset. 2004. http://pdos.csail.mit.edu/p2psim/.
6. Yuh-Jzer Joung, Chien-Tse Fang, and Li-Wei Yang. Keyword search in DHT-based peer-to-peer networks. In *Proceedings of the Twenty-fifth International Conference on Distributed Computing Systems*, 2005.
7. M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Second International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Feburary 2003.

8. John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*. ACM, November 2000.
9. Frank Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann, San Mateo, CA 94403, 1992.
10. J. Li, B. Loo, J. Hellerstein, F. Kaashoek, D. Karger, and R. Morris. The feasibility of peer-to-peer web indexing and search, 2003.
11. Xin Li, Young Jin Kim, Ramesh Govindan, and Wei Hong. Multi-dimensional range queries in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 63–75, New York, NY, USA, 2003. ACM Press.
12. Loo, Huebsch, Stoica, and Hellerstein. The case for a hybrid P2P search infrastructure. In *International Workshop on Peer-to-Peer Systems (IPTPS), LNCS*, volume 3, 2004.
13. Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Implementing declarative overlays. *SIGOPS Oper. Syst. Rev.*, 39(5):75–90, 2005.
14. C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the ninth annual ACM Symposium on Parallel Algorithms and Architectures*, pages 311–320. ACM Press, 1997.
15. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172. ACM Press, 2001.
16. Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.
17. S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, and M. Chen. Making peer-to-peer keyword searching feasible using multi-level partitioning. In *Proceedings of International Workshop on Peer-to-Peer Systems*, San Diego, CA, USA, February 2004.
18. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160. ACM Press, 2001.
19. Chunqiang Tang and Sandhya Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Proceedings of the First Symposium on Networked Systems Design and Implementation (NSDI '04)*, pages 211–224, 2004.
20. Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186. ACM Press, 2003.
21. Jiangong Zhang and Torsten Suel. Efficient query evaluation on large textual collections in a peer-to-peer environment. In *Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, Konstanz, Germany, August 2005. IEEE Computer Society Press.

XML Query Routing in Structured P2P Systems

Leonidas Fegaras, Weimin He, Gautam Das, and David Levine

University of Texas at Arlington, CSE
416 Yates Street, P.O. Box 19015, Arlington, TX 76019
{fegaras,weiminhe,gdas,levine}@cse.uta.edu

Abstract. This paper addresses the problem of data placement, indexing, and querying large XML data repositories distributed over an existing P2P service infrastructure. Our architecture scales gracefully to the network and data sizes, is fully distributed, fault tolerant and self-organizing, and handles complex queries efficiently, even those queries that use full-text search. Our framework for indexing distributed XML data is based on both meta-data information and textual content. We introduce a novel data synopsis structure to summarize text that correlates textual with positional information and increases query routing precision. Our processing framework maps an XML query with full-text search into a distributed program that migrates from peer to peer, collecting relevant document locations along the way. In addition, we introduce methods to handle network updates, such as node arrivals, departures, and failures. Finally, we report on a prototype implementation, which is used to validate the accuracy of our data synopses and to analyze the various costs involved in indexing XML data and answering queries.

1 Introduction

In the past few years, the P2P model has emerged as a new and popular computing model for many Internet applications, such as file sharing, collaborative computing, and instant messaging. A P2P network consists of a large number of nodes, called peers, that share data and resources with other peers on an equal basis. Peers are connected through a logical network topology implemented on top of an existing physical network, which may dynamically adapt to cope with peers joining and departing, as well as with network and peer failures. In contrast to traditional client-server architectures, a node in a P2P network can act as both a service provider and a client. Compared to traditional client-server systems, P2P systems are more scalable, flexible, fault-tolerant, and easier to deploy. Since no central coordination exists in a P2P system, there is no central point of failure. Additionally, network resources can be fully utilized and shared, and the server workload can be distributed among all the peers in the system.

Despite their benefits when compared to centralized systems, the most important challenge in designing a P2P system is search efficiency, especially in the presence of complex data and sophisticated search queries. Based on their search techniques, P2P systems can be classified into two main categories: unstructured and structured P2P systems. In unstructured P2P systems, which include file

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 273–284, 2007.

sharing systems, such as Napster, Gnutella, KaZaA, and Freenet, there is no global protocol for data placement and the network topology is not tightly controlled. Their basic search strategy is to partially flood the P2P network with search messages or to randomly traverse the network until the desired data are found. Unstructured P2P systems offer easier deployment, more flexibility, and lower cost of maintenance than structured systems. However, they usually have poor search performance and do not scale very well, because the query load of each node grows linearly with the total number of queries, which in turn grows with the number of nodes in the system.

In a structured P2P system, the location of data is determined by some global scheme, such as a global hash function that hashes a search key to a node. Thus, keys are distributed to peers, forming a virtual Distributed Hash Table (DHT). By limiting the routing state of each peer to a small number of logical neighbors (typically logarithmic to the network size), structured P2P systems form overlay networks in which both the lookup time for a key and DHT maintenance take a logarithmic number of routing hops between peers. Thus, by adding an acceptable amount of lookup overhead, structured P2P systems offer higher availability and better scalability than traditional client-server architectures and unstructured P2P systems. Moreover, the well-distributed DHT-based data placement strategy naturally leads to load balancing in the system. Examples of DHT-based P2P systems are Pastry [9], Chord, and CAN.

While the benefits of structured P2P systems are significant, regardless of the type of data, there has been recent interest in indexing and querying data that are far more complex than the simple keys supported by DHT-based P2P networks, such as relational data [5,4] and XML [3,1]. The greatest challenges faced by these systems are data placement and query processing, because queries over these data are typically complex and, if not carefully optimized, may involve routing and processing massive data.

In this paper, we consider the problem of data placement, indexing, and querying large schema-less XML data repositories distributed over an existing P2P service infrastructure. Instead of developing a new special-purpose P2P architecture from the ground up, we are leveraging existing P2P technology, namely DHT-based P2P systems. We have chosen to work on XML because XML is now the language of choice for communication among cooperating systems. In our framework, any peer may publish XML documents, by making them available to all participating peers, and may submit queries against the published data. Although the published XML documents remain at the publication site, the P2P infrastructure serves as a distributed index for routing queries originated by any peer to the document sources that contain the query answers, rather than for retrieving the actual XML fragments of the answer. The query language considered by our framework is XPath, extended with simple syntactic constructs for full-text search. An XML document in our framework is indexed on both its textual content and its structural makeup, called the structural summary, which is a concise summary of all valid paths to data in the document. Even though a formal schema, such as a DTD,

would have been useful information for indexing and accessing data, our framework does not require it. The textual content of a document is summarized into data synopses, which capture both content and positional information in the form of bit matrices across the dimensions of indexed terms and their positions in the document. These matrices are small enough to accommodate frequent document publishing but precise enough to reduce the number of false positives in locating documents that satisfy the structural and content constraints of a query.

Although there are a number of earlier proposals on indexing and querying XML data distributed over a P2P network [3,1], there is no work reported on complex XML query processing with full-text search on P2P networks that uses data synopses to selectively route queries to peers. The key contributions of our work can be summarized as follows:

1. We develop a framework for indexing XML documents based on summarized data (structural summaries and data synopses) and for mapping XML queries with full-text search into distributed programs that migrate from peer to peer, collecting relevant document references along the way.
2. We introduce novel data synopsis structures that have low data placement and maintenance overheads, and a high query routing precision. Our synopses correlate content with positional information, which results to a more accurate evaluation of textual and containment constraints in a query, when compared to regular Bloom filters. To the best of our knowledge, they are the first synopses to effectively address containment relationships between predicates in a query.
3. We introduce novel methods to handle network updates, such as node arrivals, departures, and failures.
4. Finally, we report on a prototype implementation to analyze the various costs involved in indexing XML documents and answering queries, and to validate our accuracy and scalability claims.

2 System Functionality

Although our framework is independent of the underlying DHT-based P2P architecture, our system is implemented on top of Pastry [9,7]. A peer in our framework can make any number of its local XML documents public to the other peers through the process of *publishing*. Once published, an XML document is available to any peer for querying, until is removed by the document owner through the process of *unpublishing*. Our P2P network serves as a distributed index for routing queries to the peers who own the documents that can answer the queries. Document updates are done by unpublishing the entire document and publishing it again (as is done in most IR systems), rather than updating the distributed index to reflect the document updates.

Our main addition to the XPath syntax is the full-text search predicate $e \sim S$, where e is an arbitrary XPath expression, that returns true if at least one element

from the sequence returned by e matches the *search specification*, S. A search specification is an IR-style boolean keyword query that takes the form

$$\text{“term”} \mid S_1 \text{ and } S_2 \mid S_1 \text{ or } S_2 \mid (\ S\ )$$

where S, S_1, and S_2 are search specifications. A term is a keyword that must be present in the text of an element returned by the expression e. For example, the XPath query, QUERY:

```
//biblio//book[author/lastname ~ "Smith"][title ~ "XML" and "SAX"]/price
```

searches for books in biblio documents authored by Smith that contain the words “XML” and “SAX” in their titles and returns their prices. The result of the query is the set of addresses of all peers who own documents that satisfy the query.

3 Document Indexing

When published by a peer, the data of an XML document are indexed over the DHT of the P2P network on both the structural summary and the data synopses of the textual content of the document. A *label path* of an XML document is a simple path expression that contains child/attribute steps only and can distinguish a non-empty set of data nodes in the document. Each label path of a document is associated with a single node in the document's structural summary and with a single data synopsis. For example, using XML representation for convenience, one possible structural summary related to the bibliography documents searched by QUERY is shown at the left of Figure 1. In this figure, node 8 is associated with the label path /biblio/book/title. We use two types of data synopses: The structural summary nodes that contain text are associated with bit matrices, called *content synopses*, across the domains of text terms

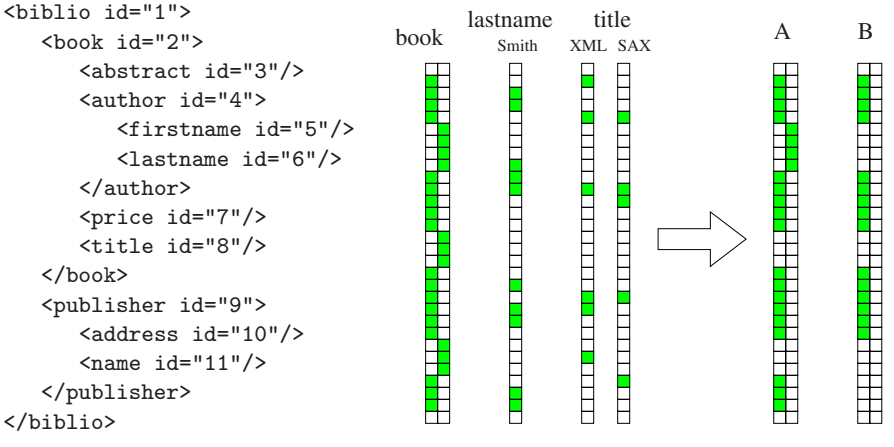


Fig. 1. A Structural Summary, and Testing QUERY Using Data Synopses

(keywords) and their positions in the document, while the internal structural summary nodes are associated with bit vectors, called *positional filters*, across document positions. Both structural summary information and data synopses are distributed over the P2P network in a way that facilitates query evaluation, involving a small number of peers while achieving a good load balance. The DHT keys used for indexing a structural summary in a P2P network are the different tagnames in the summary, while the DHT key of a data synopsis is its label path. Given an XPath query, our framework is able to find all the plausible structural summaries applicable to the query using only one DHT lookup. In a typical DHT-based P2P system, this DHT lookup takes a number of routing hops logarithmic to the network size. A plausible structural summary is one that matches the *structural footprint* of the query, which is a path expression that captures all structural restrictions in the query.

The problem that we examine first is, given an XML document and an XPath query that contains search specifications, is the document likely to match the query? Since we are using synopses, our goal is to find an approximation method that reduces the likelihood of false positives, does not miss a solution, and does not require high placement and maintenance overheads. Since we are interested in evaluating IR-style search specifications, the document content is broken into simple terms (keywords), followed by stemming and stop word elimination. The resulting terms, called the indexed terms, are summarized in content synopses. More specifically, for each unique label path in a document that directly reaches text, we create one content synopsis that summarizes the indexed terms in this text along with their positions. The positional information is derived from the document order of the begin/end tags of the XML elements in the document. That is, the position of an element tag is the number of the begin and end tags that precede this tag in the document. The positional range of an XML element, on the other hand, consists of the positions of the begin/end tags of the element, while the positional range of an indexed term is the positional range of the enclosing element. That is, terms that belong to the same XML element have the same positional range. All positions in a document are scaled and mapped into a bit vector of size L, called a *positional bit vector*, so that the last position in the document is mapped to that last bit in the vector.

The positional dimension of the synopses is necessary due to the containment restrictions inherent in the search specifications of a query. For example, the search specification $e \sim$ "t_1" and "t_2" for two terms t_1 and t_2 becomes true if and only if there is at least one document node returned by e that contains both terms. Using one-dimensional term bitmaps alone, such as Bloom filters, and checking whether both the t_1 and t_2 bits are on, will give us a prohibitive number of false positives (as is shown in our experimental evaluation). For instance, using Bloom filters, our QUERY might have returned all documents that have one book whose author last name is Smith, a second book whose title contains XML, and a third book whose title contains SAX. Therefore, term position is crucial in increasing search precision and reducing false positives.

Given a document, the content synopsis H_p associated with a label path p is a mapping from an indexed term t to a positional bit vector. In our implementation, a content synopsis is summarized into a bit matrix of fixed size $W \cdot L$, where W is the number of term buckets and L is the size of bit vectors. Then, $H_p[t]$ is the matrix column associated with the hash code of term t. If there are instances of two terms t_1 and t_2 in the document that have the same positional ranges (ie, they are contained in the same element), then the $H_p[t_1]$ and $H_p[t_2]$ bit vectors should intersect (ie, their bitwise anding should not be all-zeros). For example, we can test if a document matches the search specification title $\sim$ "XML" and "SAX" by bitwise anding the vectors H_8["XML"] and H_8["SAX"], which correspond to the node 8 (the book title) in Figure 1. If the result of the bitwise anding is all zeros, then the document does not satisfy the query (the opposite of course is not always true).

But given the bit vectors H_8["XML"], H_8["SAX"], and H_6["Smith"], how can we enforce the containment constraint in the QUERY that the book whose author last name is Smith must be the *same* book whose title contains XML and SAX? We cannot just use bitwise anding or oring. A first attempt is to use a positional filter F_p of size L associated with a label path p, so that for each XML element in the document reachable by the label path p, the bits in the bit vector that correspond to the element's positional range are all on. Unfortunately, this may result into bit overlaps of consecutive elements in the document, when their mapped positional ranges intersect. We address this problem by using M bit vectors F_p so that the positional range of the ith book goes to the i mod M bit vector. That way, two consecutive books in the document, the i and $i+1$ books, are placed to different bit vectors, thus reducing the likelihood of overlapping, which may in turn result to false positives. M should be at least 2, since it is very common to have a situation similar to that of consecutive books.

The basic operation in our framework to test element containment is called *Containment Filtering*, $CF(F, V)$, that takes a positional filter F and a bit vector V as input and returns a new positional filter F'. Function CF copies a continuous range of one-bits from F to F' if there is at least one position within this range in which the corresponding bit in V is one. For example, the right of Figure 1 shows how the data synopses of a document are used to determine whether this document is likely to satisfy QUERY (assuming $M = 2$). This is the case when at least one bit vector of B is not all zeros, where A is $CF(F_2, H_6[\text{"Smith"}])$, B is $CF(A, and(H_8[\text{"XML"}], H_8[\text{"SAX"}]))$, F_2 is the positional filter for node 2 (of books), and '*and*' is bitwise anding.

4 Data Placement and Query Processing

Our data synopses are used to route queries to peers who are likely to own documents that satisfy a query. To accomplish this, we introduce methods for placing and indexing data synopses over a P2P network and for locating documents relevant to a query based on these indexes. Our placement strategy for structural summaries is very simple: they are routed to peers using every distinct tagname

from the structural summary as a routing key. Thus, locating all structural summaries that match the structural footprint of a query is accomplished by a single DHT lookup by choosing a tagname uniformly at random from the query footprint as the routing key. A data synopsis is placed on the P2P network using its label path as the routing key. Since a label path may belong to multiple documents, all the relevant synopses from all these documents are placed at a single peer, the one whose Node Id is numerically closest to the Id of the label path. Thus, locating all document locations that satisfy the simple query $p \sim$ "term", for a label path p, is accomplished by a single DHT lookup by using the label path p as the routing key. Then, the document locations that satisfy this simple query are those whose content synopses, projected over "term", give a non-zero positional filter. To handle more complicated XPath queries, all the structural summaries that match the query footprint are extracted and all the distinct label paths that participate in the query's search specifications are collected and used as routing keys. Thus, the query is routed to the peers that contain the relevant data synopses, collecting and filtering document locations along the way. This is done over multiple documents at once. That is, the unit of communication between peers is a list of triples (peer,document,positional-filter), containing the owner and the id of a matching document along with the document positions that satisfy the query at the current point of query evaluation. This list is shorten at each visited peer by removing documents whose positional filters are all zeros. At the end, the query client collects all document locations that match the query and routes the query to the document owners for evaluation.

5 Handling Network Updates

There are three types of network updates that need to be handled by any P2P system: arrival, departure, and failure of nodes. While it is very important to maintain the integrity of the routing indexes, the integrity of data, which in our case are document references, is of secondary importance, as is apparent in web search engines that may return outdated links to documents. Both arrivals and departures can be handled without disrupting the query routing process. When a new node joins the overlay network and is ready to send and receive messages, it invokes the Pastry method notifyReady(). Our implementation of this method includes code that sends a message to the new node's successor to transport parts of its database to the new node. The node successor is derived from the node's Leaf set and is basically the immediate neighbor in the Id space with a larger Id. When the successor receives the message, it moves all structural summaries and data synopses whose routing Ids are less than or equal to the new node's Id to the new node. Therefore, the arrival of a new node requires two additional messages to transport data to the new node. When a node willingly leaves the overlay network, it routes all structural summaries and data synopses to its successor using one message only. The departing peer leaves the references to the local documents dangling and let the system remove them lazily.

A node failure is the most difficult network update to handle. When a peer P_1 receives a search request based on a tagname tag_1 to find all structural summaries that match a query footprint and does not find one, there are two possibilities: either there was really no matching structural summary indexed in the DHT, or the predecessor node, who was closest to tag_1, had failed. Given that P_1 knows when its predecessor in the Id space fails (since, when this happens, it will receive a special message from Pastry), it can always distinguish the two cases: if the tagname Id is smaller than the failed predecessor Id, then it is the latter case. In that case, P_1 will choose another tagname tag_2 uniformly at random from the query footprint and relay the search request to another peer P_2 under the new key tag_2. In addition to the message relay, P_1 sends another message to P_2 asking to return all structural summaries associated with tag_1 to be published in P_1 (since P_1 now gets the requests for tag_1 keys). That way, the structural summaries of the failed peer are republished one-by-one lazily and on demand. Similarly, when a peer gets a request for a data synopsis based on the label path p and does not find one, and its predecessor had failed, it will check whether the Id from p is less than the Id of the failed predecessor. If so, it will abort the query and will scan the list of document publishers from the hit list routed along with query and will send a message to each publisher to publish the data synopses for path p again. Therefore, the restoring of data synopses associated with a failed peer in the P2P network is done lazily and on demand, and each time only one query has to be aborted.

6 Related Work

The closest work to ours is by Galanis *et al* [3] on XPath query routing in large P2P systems. Like our framework, the target of their distributed indexing is the location of data sources that contain the answer, rather than the actual XML fragments of the answer. Their structural summaries are similar to ours, since, for each tagname in an indexed document, they index all possible distinct paths that lead to this tagname in the document. This mapping is distributed in a DHT-based system using the tagname as the search key. Similarly, for each XML element that contains text, they store the text in the DHT index using the tagname of the XML element as the search key. This is contrary to our approach in which text is broken into terms before is indexed and label paths are used as keys. Unfortunately, the authors do not address the indexing cost, since their design is based on the assumption that querying is far more frequent than data placement. We believe that their framework is more suitable for data-centric XML data rather than to document-centric ones, since the latter may include large text portions inside specific tagnames, which results to the routing of large parts of a document to the same nodes. In their system, XPath queries are routed to peers based on the last tagname in the query, which serves as the DHT lookup key. Contrary to our approach, their evaluation of our example query will return even those nodes who own documents that have one book written by Smith

and another with XML and SAX in their titles. That is, their method does not address containment relationships between predicates.

Another related framework is XP2P [1], which indexes XML data fragments in a P2P system based on their concrete paths that unambiguously identify the fragments in the document (by using positional filters in the paths). The search key used for fragment indexing is the hash value of its path. Thus, XP2P can answer simple, but complete XPath queries (without predicates or descendant-of steps) very quickly, in one peer hop, using the actual query as the search key. The main drawback of this method is that retrieving a complete data fragment with all its descendants would require additional hops among peers by extending the query with the child tagnames of each retrieved fragment recursively, until all descendants are fetched. The descendant-of step requires even more searching by considering the parts of the query that do not have descendant-of steps and appending to them the child tagnames of the retrieved fragments (which makes it impossible to answer queries that start with a descendant-of step). More importantly, this technique has not been extended to include XPath predicates.

Although there are other proposed synopses and value distribution summaries for XML data, such as XSketch [8], their main use is in selectivity estimation, rather than in query routing in a P2P network. In [2], the structural summary (called Repository Guide) is served as a global schema that indicates how XML data are fragmented and distributed over the network. DBGlobe [6] exploits multi-level Bloom filters based on the structural summary of XML documents to route path queries to peers but they are based on structure information only. To the best of our knowledge, none of the synopses proposed by others summarizes both content with positional information in the same structure.

7 Performance Evaluation

We have built a prototype system to test our framework. It is built on top of Pastry [7] and uses Berkeley DB Java Edition as a lightweight storage manager. It is available at http://lambda.uta.edu/xqp/. The platform used for our experiments was a 3GHz Pentium 4 processor with 1GB memory on a PC running Linux. The simulation was done using Java (J2RE 1.5.0) with a 768MBs maximum memory allocation pool, from which 60% was allocated to the Berkeley DB cache. The experiments were performed over a cluster of 100, 1000, and 2000 peers in a simulated network on a single CPU. We used four datasets for our experiments, which were synthetically generated by the XMark and XMach benchmarks (see Figure 2).

Our query workload consisted of 1000 random XPath queries generated from 50 files selected uniformly at random from each dataset. More specifically, each selected file was read using a DOM parser and was traversed using a random walk, generating an XPath step at each node. When a text node was reached and a search specification was selected, search terms were picked from the text uniformly at random. That is, each generated query was constructed in such a way that it satisfied at least one document (the parsed document). Based on these four datasets and query workload, we derived the measurements in

	files	size	file size	tags	paths	msg/file	syn/file	syn size	msg/qry
XMark1	230	1.7 MBs	7.6 KBs	83	341	41.7	17.9	161 bytes	3.93
XMark2	2300	17 MBs	7.6 KBs	83	424	41.3	17.6	166 bytes	3.63
XMark3	11500	82 MBs	7.3 KBs	83	436	41.4	17.6	165 bytes	3.87
XMach	5000	87 MBs	17.8 KBs	1616	9206	28.9	17.7	547 bytes	3.52

Fig. 2. Benchmark Measurements

Figure 2, where msg/file is the average number of messages needed to publish one file from the dataset, syn/file is the average number of data synopses produced by each file in the dataset, syn size is the average size of an uncompressed synopsis, and msg/qry is the average number of messages needed to evaluate one query from the query workload. Note that these measurements are independent of the number of peers; they simply indicate the number of messages/synopses generated, rather than the number of distinct peers that receive these messages.

Note that the work by Galanis *et al* [3] does not capture positional information and does not address containment relationships between predicates, which, as we will see in our measurements, can considerably reduce the number of false positives. Furthermore, our indexing scheme is based on label paths (tagname sequences), which results to better load balancing than their scheme, which is based on single tagnames (the endpoints of a query). Nevertheless, we evaluated their system based on our datasets and query workload in terms of numbers of messages. Since their system was not available at the time of writing, we calculated the number of messages based on the analysis given in their paper. For XMark, they needed 17.6 msg/file, were each message had size 442 bytes, and 4.58 msg/qry. For XMach, these numbers were 17.7 msg/file, 1030 bytes, and 3.39 msg/qry. We can see that their system requires fewer messages for publishing than ours, although each publishing message is a little bit larger than ours on the average (since they have to publish the entire text). With our approach, in return, we gain better accuracy (as shown below) and better load balancing.

Based on the XMark datasets and query workload (1000 random queries), we measured the load distribution in our system for a network of 100, 1000, and 2000 peers. The results are shown in Figure 3. More specifically, for each one of the 3 XMark datasets, we grouped and counted the peers based on 1) the distribution of the number of messages needed to publish all the documents in a dataset; 2) the distribution of the number of content synopses after all the documents in a dataset have been published; 3) the distribution of the number of messages needed to evaluate 1000 randomly generated queries. These results can be easily explained given that the documents generated by XMark match a single DTD, resulting to a very small number of distinct tagnames and text label paths. For instance, from a network of 2000 peers, at most 436 peers are expected to receive all data synopsis placement/search requests, while the rest get none. For a network of 100 peers, though, the load is more evenly distributed. This load balancing skew, however, is not likely to happen with a heterogeneous dataset, when the number of distinct label paths is comparable to the network

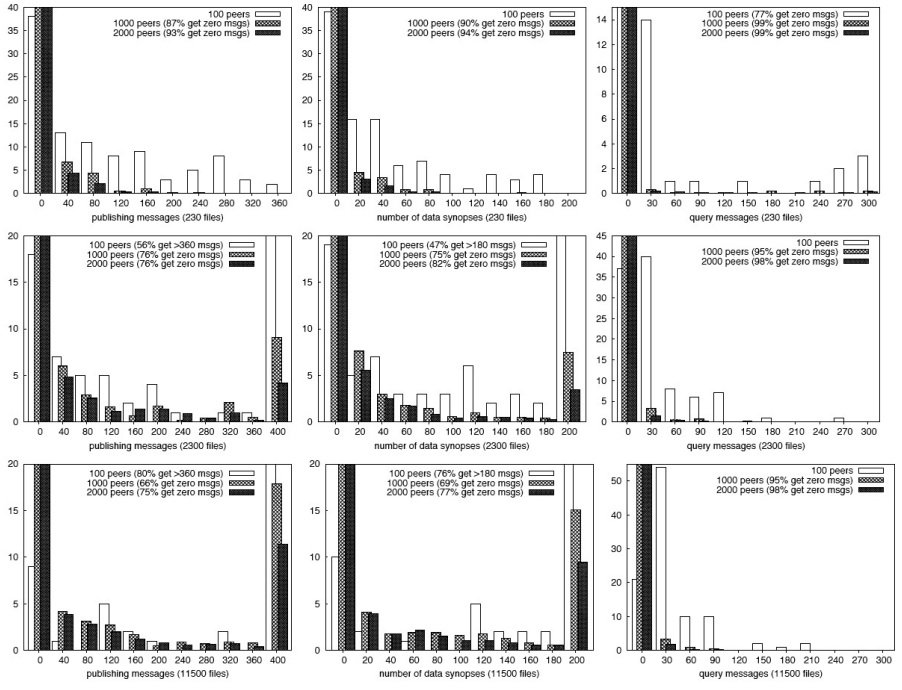


Fig. 3. Load Distribution based on the XMark Datasets (y Axis is % of Involved Peers)

size. For example, the XMach dataset, which uses multiple DTDs, gives a better load distribution in processing 1000 randomly generated queries, as shown in Figure 4. More specifically, out of 2000 peers, 52.2% receive between 1 and 15 messages and 47.1% receive no messages (while for XMark3, 98% receive no messages, leaving the burden of query processing to 2%).

The second set of experiments was designed to measure the accuracy of data synopses. It was based on the XMark1 dataset on a single peer (since precision is not affected by the network size). The results are shown in Figure 4. For the first precision experiments, we used queries that match only one document from the dataset. The plot at the top right of Figure 4 shows the average number of false positives for various sizes of data synopses. Given a label path and a document, the width of its content synopsis is the number of document elements reachable by this path multiplied by the width factor. The height factor, when multiplied by the document size, gives the content synopsis heights. When the height factor was set to zero, then Bloom filters were used instead of content synopses (zero height). The size M is the height of positional filters. When $M = 0$, then no positional filters were used. We can see that, for random queries, the width factor affects precision more than the height factor (ideally, the number of false positives should be zero.) The plot at the bottom right of Figure 4 indicates that using Bloom filters (height factor = 0) yields twice as many false positives as when the height factor is ≥ 0.1.

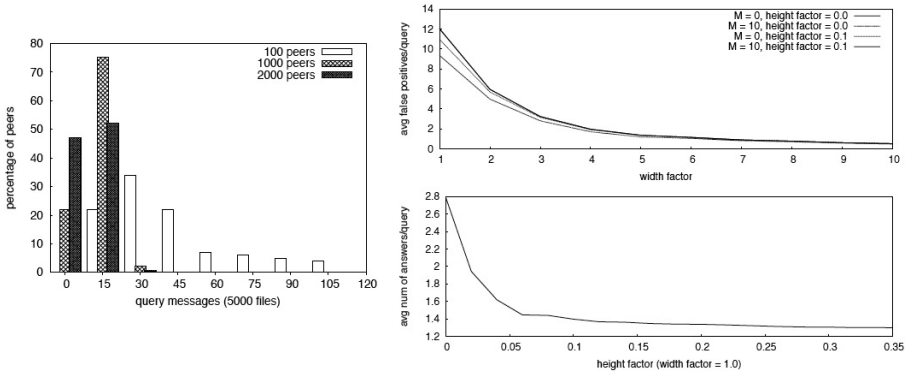


Fig. 4. Query Load Distribution for XMach and Data Synopsis Accuracy

8 Conclusion

We have presented a scalable architecture for indexing and querying XML data distributed over a DHT-based P2P system. The main contribution of our work is the development of a framework for indexing XML data based on the structural summary and data synopses of data, and for mapping an XML query with full-text search into a distributed program that migrates from peer to peer, collecting relevant data and evaluating parts of the query along the way. As a future work, we are planning to develop relevance ranking functions based on content synopses and use one of the known top-k threshold algorithms to reduce network traffic during querying.

References

1. A. Bonifati, *et al.* XPath Lookup Queries in P2P Networks. WIDM 2004.
2. J.-M. Bremer and M. Gertz. On Distributing XML Repositories. WebDB 2003.
3. L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating Data Sources in Large Distributed Systems. VLDB 2003.
4. A.Y. Halevy, Z.G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. IEEE Trans. Knowl. Data Eng. 16(7): 787-798 (2004).
5. R. Huebsch, *et al.* The Architecture of PIER: an Internet-Scale Query Processor. CIDR 2005.
6. G. Koloniari and E. Pitoura. Content-Based Routing of Path Queries in Peer-to-Peer Systems. EDBT 2004.
7. Pastry. http://freepastry.rice.edu/.
8. N. Polyzotis and M. Garofalakis. Structure and Value Synopses for XML Data Graphs. VLDB 2002.
9. A. Rowstron and P. Druschel. Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. International Conference on Distributed Systems Platforms 2001.

Reusing Classical Query Rewriting in P2P Databases*

Verena Kantere and Timos Sellis

School of Electr. and Comp. Engineering, National Technical University of Athens
{vkante,timos}@dbnet.ece.ntua.gr

Abstract. Sharing of structured data in P2P overlays is a challenging problem, especially in the absence of a global schema. The nature of structured data stored in the overlay enforces strict methods of querying. However, the intention of the user is usually to obtain information that is semantically relevant to the posed query and not information that strictly complies to structural constraints. The rewriting mechanisms for structured data were initially developed for tasks such as data integration and mediation, which by nature dictate strict consistency. In this work we propose ways of preprocessing a query in order to produce a version that can be rewritten in the classical way. We propose preprocessing guidelines that produce a new query that is most similar to the initially posed query. Accordingly, we discuss thoroughly query similarity aspects from a structural point of view. Finally, we present an algorithm that selects the most appropriate mappings in order to perform query rewriting.

1 Introduction

In contrast to data integration architectures, P2P data sharing systems do not assume a mediated schema to which all sources of the system should conform in order to share data. In such a system, where peers share (semi)-structured data, each is an autonomous source that has a local schema. Sources store and manage their data locally, revealing only part of their schemas to the rest of the peers. Due to the lack of global schema, they express and answer queries based on their local schema. In order to be able to exchange queries, during the acquaintance procedure the two peers exchange information about their local schema and create a mediating mapping semi-automatically [9].

We are interested in data exchange issues in pure (i.e. without super-nodes) P2P database systems. We assume that each peer owns a relational peer schema (i.e., the only internal mappings are foreign key constraints) that it thoroughly exports to its immediate neighbors, hereafter *acquaintees*. Each pair of acquaintees holds peer mappings between their schemas. Peer mappings are considered to be of the well-known GAV-LAV-GLAV (i.e. Global, Local, Global and Local As View) form (we limit our study to mappings that can be expressed as SPJ queries). Hence, a peer mapping is a view with the head of it belonging to the global schema and the body to the local one (GAV) or the opposite (LAV). For clarity reasons, we remind that in GAV/LAV definitions for the P2P setting, the global schema is the schema of the peer on which the query is initially

* This research has been partly supported by the European Union - European Social Fund and National Resources under the PENED/EPAn program (Greek Secretariat of Research and Technology).

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 285–297, 2007.

posed and the local schema is the schema of the peer on which the query is rewritten. In order to provide answers to a query, peers express queries on their local schema, answer the query and at the same time propagate the query to its acquaintees in the P2P system. At each step, a query is successively reformulated through mappings.

For LAV mappings we assume that the query reformulation is performed by a query rewriting mechanism based on the well known algorithms for answering queries using views [12], [13] and [1]. For GAV mappings, we assume the straightforward substitution of the view head with the body as it is done in Piazza [8]. GLAV mappings use a combination of LAV and GAV query rewriting techniques [8].

The goal of the reformulation mechanism is to transform a query so that it can be answered partly or completely by an acquaintee, i.e. an one-hop neighbor in the overlay. The available query rewriting algorithms restrict their usage to queries that can be completely rewritten under a set of mappings, meaning that these algorithms can rewrite queries only if all 'select' attributes and 'where' conditions of the original query can be rewritten through the available mappings. Yet, this is not suitable for a P2P environment. In many common P2P applications, peers are satisfied with retrieved information with characteristics similar to those of their query and not necessarily with exactly the same characteristics (as is the case with search engines, popular P2P file-sharing applications, etc.) Therefore, it is reasonable to assume that it is preferable for our P2P database system to operate in the same manner as [15] does, and as a result we would like peer queries to be reformulated and propagated even if they can be only partly satisfied.

Moreover, existing rewriting algorithms have been created to serve the problem of data and schema integration and thus do not allow partial query rewriting. These algorithms are driven by the assumption that the correct rewriting of a query is the maximally-contained version of it [11]. In P2P database environments, peers are not interested in answers from other peers that are contained in the answers they can retrieve locally; peers are interested in answers that would be semantically relevant to the originally posed query.

Motivating Example

Envision a P2P system where the participating peers are databases of private doctors of various specialties, diagnostic laboratories and databases of hospitals. Figure 1 depicts a small part of this system, where nodes are: DavisDB - the database of the private doctor Dr. Davis, LuDB - the database of pediatrician Dr Lu and StuartDB - the database of the pharmacist, Mr Stuart. On top of each database sits a P2P layer, which is responsible for all data exchange of this peer with its acquaintees. Among others, the P2P layer is responsible for the creation and maintenance of mappings of local schemas during the establishment of acquaintances towards the line of [9]. Moreover, each peer owns a query rewriting mechanism. The schemas of the databases are shown on Figure 1.

Suppose that Dr Davis would like to collect from the system *general* information about patients that have had diseases. He expresses the following query on his database: Q_{orig}:

```
SELECT  V.Pid, D.DisDescr, D.Ache, T.Drug, T.Dosology
FROM    Disease D, Treatment T, Visits V
WHERE   V.Did = D.Did AND D.Did = T.Did
```

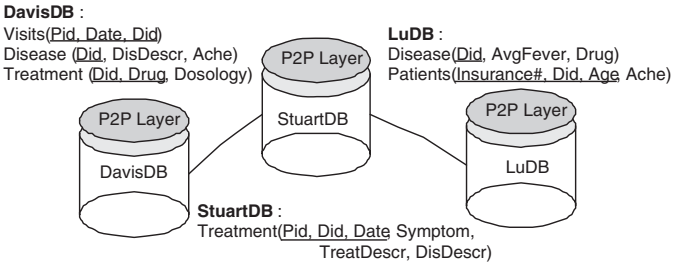


Fig. 1. Part of a P2P system with peer-databases from the health environment

Having only one acquaintance, the pharmacist's database, Dr. Davis's database propagates Q_{orig} to it. We assume the following LAV mapping between DavisDB and StuartDB databases:

$M_{StuartDB\_DavisDB}$:

Treatment(Pid, _, _, Symptom, TreatDescr, DisDescr):-Visits(Pid, _, Did),Disease(Did, DisDescr, Ache), Treatment(Did, Drug, _),

where correspondences Symptom = Ache, TreatDescr = Drug are implied. [1] Thus, the rewritten query on StuartDB is the following:

$Q_{StuartDB\_sr}$:

```
SELECT  T.Pid, T.DisDescr, T.Symptom, T.TreatDescr
FROM    Treatment T
```

Obviously the new query has lost the attribute referring to information about drug dosology, since it cannot be mapped in StuartDB. However, if classical query rewriting were used, the query could not be rewritten at all. Thus, Dr Davis would not be able to get any information at all from Mr Stuart.

The node of Mr Stuart passes the rewritten version $Q_{StuartDB\_sr}$ to Dr Lu with whom he has the following GAV mapping:

$M_{StuartDB\_LuDB}$:

Treatment(Pid, _, _, Symptom, _, _):- Disease(Did, AvgFever, _), Patients(Insurance♯, Did, _, _), Age $<$ 13

where correspondences Pid = Insurance♯, Symptom = AvgFever are implied. Thus, the rewritten query on LuDB is the following:

Q_{LuDB_sr}:

```
SELECT  P.Insurance#, D.AvgFever
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

[1] The mapping is actually a view defined on StuartDB.Treatment, which is matched with a join on DavisDB relations such as:
View1(Pid, Symptom, TreatDescr, DisDescr):-Treatment(Pid,Did, Date, Symptom, TreatDescr, DisDescr)
View1(Pid, Ache, Drug, DisDescr):- Visits(Pid, Date, Did),Disease(Did, DisDescr, Ache), Treatment(Did, Drug, Dosology)
Due to lack of space we summarize mappings by omitting view definitions and introducing '_' for attributes that are not needed.

Obviously the new query has lost more attributes, which refer to the description of the disease and the respective treatment. Moreover, the new query is more restrictive than the original, since it has an additional condition on 'Age'. Yet, Dr Davis has the chance to retrieve some information relevant to his initial query. In case of classical query rewriting, Dr Davis would not get any information at all from the overlay, even though his intention was to learn as much as he can about patients and their diseases.

Contributions

The presented situation points out that a major problem in unstructured P2P DBMSs is that, even though peers may need general information (i.e. it is loosely coupled with some constraints), the nature of structured data stored in the overlay enforces strict methods of querying. Moreover, the rewriting mechanisms for structured data were initially developed for tasks such as data integration and mediation, which by nature dictate strict consistency. Thus, the use of the classical query rewriting algorithms in P2P database settings deviates from the initial purpose of the users; the result is that queries cannot be propagated in the overlay if there is not a chain of mappings that matches them thoroughly to other peer schemas. Since such a requirement is too strict for a P2P environment, classical rewriting algorithms cannot be used. Given that these algorithms encapsulate a huge experience on reformulation of queries on structured data, it would be naive to ignore all this work and 'reinvent the wheel'. Thus, we have to adapt the experience on classical rewriting in the context of P2P databases.

This paper makes a contribution towards this step. Specifically, we propose the preprocessing of the queries posed on P2P databases in order to produce versions that can be classically rewritten to other peer schemas. In this context, we investigate the notion of query similarity based more on structural features rather than semantics itself. We propose criteria that can be used to form a query similarity function, i.e. a metric to judge how much a query is (semantically) close to the original query . In the same spirit we discuss guidelines for the preprocessing of queries and we present an algorithm that considers GLAV, GAV and LAV mappings in order to select those mappings that can rewrite the query in the best way.

2 Query Similarity

In order for a peer to answer an incoming query, it has to translate it with respect to its local schema. Usually, the resulting query is not a complete translation of the original one, but is somehow 'similar' to it. In the literature [11], the similarity between rewritten queries is measured according to the containment of the results of the rewritten query to the results of the original one. However, this kind of query similarity cannot be effective in the context of this work. The reason is that the queries we want to compare are written on different schemas: Q_{orig} (the original query) is written on the schema of the initiator and the rewritten version Q_{rewr} on the schema of a remote peer. Moreover, our goal is not to answer the queries in order to measure their similarity, but to measure their similarity in order to decide which one to answer. Thus, we have to rely on query characteristics rather than the answer to the query in order to determine similarity.

Query similarity has been explored in several works in the recent past. Due to lack of space we cannot make a thorough discussion. Some of these works deal with

keyword matching in the database environment [2, 5] or with the processing of imprecise queries [14, 6, 10]. The work in [3] talks about attribute similarity but focuses on numeric data and on conclusions about similarity that can be deduced from the workload. Furthermore, in [7] queries are classified according to their structural similarity; yet, the authors focus on features that differentiate queries with respect to optimization plans. The only work relevant to ours is that of [4], where overall semantic similarity of queries is explored. Yet, our focus is on query versions that are produced through the use of mappings, and we are interested on the effect of the mappings in query similarity.

In order to measure the similarity of two queries, Q_{orig} and Q_{rewr} there is a need for a function that quantifies their semantic relativeness, $M_{sim}(Q_{orig}, Q_{rewr})$. Next, we discuss the guidelines along which such similarity functions should be constructed and the factors that affect it.

2.1 Aspects of Query Similarity

The similarity of two queries each on a distinct schema is not only a matter of different query characteristics, but also of why these different characteristics exist. Specifically, since we are interested in incomplete rewritings of queries, query similarity has to take into consideration under which data exchange conditions new elements in the rewritten version are inserted or old elements are missing.

(1) 'select' attributes
For example, remember the query on StuartDB, $Q_{StuartDB\_sr}$, and the successively rewritten version on LuDB, Q_{LuDB_sr}; First, two of the 'select' attributes of $Q_{StuartDB\_sr}$, T.DisDescr and T.TreatDescr are missing in the 'select' clause of Q_{LuDB_sr}. It is obvious that the lack of rewriting of these two attributes is due to either the lack of corresponding attributes in the schema of LuDB, or the lack of mappings between StuartDB and LuDB that encapsulate the correspondence of these attributes. Nevertheless, the rewritten version has never additional 'select' attributes, compared with the original one.

Observation: *In any case, it is clear that 'select' attributes missing in the rewritten query version influence negatively the overall similarity of the queries.*

(2) 'where' conditions
Yet, things are more vague with the query conditions. There are several situations and we consider each separetely.

(2a) additional value constraints: In our example Q_{LuDB_sr} has the additional condition $P.Age < 13$. Is this condition an additional constraint to the query compared with the non-conditional $Q_{StuartDB\_sr}$? In order to find out, we have to consider the circumstances under which the mapping that contributed this condition, $M_{StuartDB\_LuDB}$, was created. In our case, Dr Lu is a pediatrician, and, thus, he stores in his database information about kids, i.e. P.Age < 13 for all data in LuDB. Therefore, the corresponding additional condition in Q_{LuDB_sr} is not actually an additional constraint, since the set of returned tuples is the same if the posed query includes or not this condition. However, if Dr Lu is a family doctor, but for some reason his database maintais the mapping $M_{StuartDB\_LuDB}$ with StuartDB, the condition on $P.Age$ is an actual additional constraint, since Q_{LuDB_sr} returns less tuples than if the condition is eliminated. In general, additional value conditions in rewritten queries either restrict or do not influence the result

of the query. This depends on the reasoning that created the mappings used for the query rewriting.

Observation*Since we are not able to know the logic beneath mapping creation, we consider additional value conditions as restrictive, and, therefore, that they decrease the similarity of the queries.*

(2b) additional joins on non-key attributes: Beyond value conditions, SPJ queries have joins either on relation keys or just plain attributes. As far as additional joins on plain attributes are concerned, we can follow the same rationale as in the previous paragraph. We can conclude that, in the same way as with additional value conditions, additional joins on plain attributes can be considered as more restrictive, in the general case.

Observation:*Hence, additional joins on non-key attributes decrease query similarity.*

(2c) additional joins on key attributes: However, is this the case for joins on relation keys? Consider again the successive rewritings $Q_{StuartDB\_sr}$ and Q_{LuDB_sr} of the motivating example. The second query has a join on relation keys, $D.Did = P.Did$, whereas the first does not have any. Easily, we can see that this join is necessary in order to 'select' both attributes $P.Insurance\sharp$ and $D.AvgFever$; thus, it does not restrict the query answer.

Yet, a minor objection to this reasoning is that two relations joined by their keys do not coincide with one relation that contains all their attributes: the first contains only the tuples of the two joined relations that have common key values, whereas the second can contain even the tuples of the two relations that have non-matching key values [2].

Furthrmore, a join on keys restricts the query answer, if the rewritten version does not contain 'select' attributes from both parts of the join. For example, suppose that $Q_{StuartDB\_sr}$ does not contain in the 'select' clause the attribute $P.Insurance\sharp$. However, the only way to rewrite it is through the available mapping $M_{StuartDB\_LuDB}$. Hence, the rewritten version on LuDB will be:
Q'_{LuDB_sr}:

```
SELECT  D.AvgFever
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

In this case, even though the additional join is on relation keys, it does not serve as an associative action and it does restrict the query answer: in absence of it the query would return additionaly the tuples of relation *Disease* that do not have a matching *Did* with a the *Pid* of a tuple in relation *Patients*.

Observation: *Based on the above, we consider additional joins on relation keys as neutral to query similarity, if the existense of 'select' attributes is based on the existense of the join; otherwise, we consider additional joins on relation keys as a negative influence to query similarity.*

(2d) absent joins on key attributes: Finally, let us consider joins on relation keys that exist in the source query, but are absent in the rewritten version. For example, this is

[2] Assume that there are two relations $R1(x,y)$ and $R2(x,z)$ with the same key, x, and a relation that contains all attributes of $R1$, $R2$: $R(x,y,z)$ with key x. The tuples of each one of $R1/R2$ that cannot be joined with a tuple of $R2/R1$, would have a corresponding tuple in R, for which the attributes corresponding to $R2/R1$ would be null (outer join).

the case of Q_{orig} and $Q_{StuartDB\_sr}$. The two joins on relation keys in the former are not present in the latter. Should this element absence in the rewritten version be considered as a reformulation failure? The answer is no, obviously: since the query rewriting is performed using the mapping $M_{StuartDB\_DavisDB}$ that combines the relations *Visits*, *Disease* and *Treatment* from DavisDB in a correspondence to the *Treatments* relation in StuartDB, the two joins of Q_{orig} are "*consumed*", in a way, during the complete rewriting through the discussed mapping. Thus, the absence of the two joins in the rewritten version does not mean that the mechanism failed to rewrite them, but that they are encapsulated in the mappings used for the rewriting, and they are not needed in the new query.

But, what if the the joins of Q_{orig} were not present in the mapping $M_{StuartDB\_DavisDB}$? For example suppose that StuartDB uses for the rewriting Q_{orig} the mappings:
$M1'_{StuartDB\_DavisDB}$:

Treatment(Pid, _, _, Symptom, _, DisDescr):- Visits(Pid, _, Did),Disease(Did, DisDescr, Ache)
and $M2'_{StuartDB\_DavisDB}$:

Treatment(_, _, _, _, TreatDescr, _):- Treatment(_, Drug, _)[3],
The rewritten query on StuartDB would be:
$Q'_{StuartDB\_sr}$:

```
SELECT  T1.Pid, T1.DisDescr, T1.Symptom, T2.TreatDescr
FROM    Treatment T1, Treatment T2
```

The second join of Q_{orig}, 'D.Did = T.Did', is not encapsulated in the above mappings. It is obvious that the lack of this join in the mappings and the rewritten version results in a cartesian product in the relation StuartDB.Treatment; thus, the lack of the key join affects really bad the reformulation of Q_{orig}.

Observation: *Led by the aforementioned discussion, we consider that joins on relation keys are satisfied, i.e. explicitly or implicitly present in a query rewriting, if they are present either in the mappings used for the reformulation or in the rewritten version itself. If joins on keys are not satisfied, we consider that their lack in the rewritten version affects negatively the similarity with the source query.*

In addition to the above, any other missing constraints (i.e. value constraints or joins on non-key attributes) are considered to have a bad impact on query similarity.

(3) corresponding 'select' attributes and 'where' conditions:
Suppose that the only available mapping in order to rewrite Q_{orig} is $M1'_{StuartDB\_DavisDB}$. Then the rewriting procedure would produce the query version:
$Q''_{StuartDB\_sr}$:

```
SELECT  T.Pid, T.DisDescr, T.Symptom
FROM    Treatment T
```

Again, neither the rewritten query nor the used mapping contain any form of the join 'D.Did = T.Did' of Q_{orig}. Moreover, in this case the 'select' attribute Treatment.Drug of

[3] The reader can observe that the mapping $M2'_{StuartDB\_DavisDB}$ does not map tuples 1-1; yet, this is a possible mapping and its meaning is: "tuples in DavisDB.Treatment correspond to tuples in StuartDB.Treatment where the respective attributes Drug and TreatDescr have the same value ".

Q_{orig} is not mapped in StuartDB, and, thus, it is not present in $Q''_{StuartDB\_sr}$. So, the lack of both the aforementioned join and attribute should influence negatively the similarity of the source and the rewritten query. However, the attribute Drug is not really worth of rewriting, even if this is possible, (e.g. if the mapping $M2'_{StuartDB\_DavisDB}$ is available), if the join 'D.Did = T.Did' cannot be rewritten: as we have discussed previously, the result would be a cartesian product, which in general cannot be considered a good rewriting.

The question that arises from this situation is whether the lack of these two elements should affect the query similarity in a correlated or separate way. Again, the answer to this question depends on how users think in order to form an originally posed query. If users use joins on relation keys only as an associative means for retrieving attributes from distinct relations, then the role of these joins is supportive to 'select' attributes and the impotence of their reformulation leads to the impotence of retrieving the supported attributes from the target database; these joins cannot be present alone, i.e. without the attribute(s) they support. Therefore, as far as query similarity is concerned, the lack of such rewritten joins should be correlated with the lack of the supported rewritten attributes. Nevertheless, in case we assume that users create queries without any specific logic, key joins cannot be thought of as associative to retrieved attributes, in general. Hence, their lack should be considered as not affecting query similarity.

The second assumption is more conservative than the first, because it considers that two independent query features are missing from the rewritten version. Moreover, the second assumption does not affect the query rewriting procedure, since query features can be considered separately for rewriting (Section 3); yet, in agreement with the first assumption, combinations of 'select' attributes / 'where' conditions should be spotted.

Observation: *Based on the above, we consider that the presence of each query feature is independent from the presence of the rest and they affect query similarity in a separate way.*

In the same spirit, we have to consider whether the retrieval of an attribute should be correlated with value conditions on the same attribute. For example, suppose a query similar to Q_{LuDB_sr}, where Patient.Age is a 'select' attribute:
Q''_{LuDB_sr}:

```
SELECT  P.Insurance#, D.AvgFever, P.Age
FROM    Disease D, Patients P
WHERE   D.Did = P.Did, P.Age < 13
```

Should the lack of Patient.Age in a rewriting of Q''_{LuDB_sr} be correlated with the lack of the condition 'Patient.Age $<$ 13' in the same rewriting? As in the case of joins on keys discussed previously, the answer to this question can be either 'yes' or 'no'. On one hand, the presence of the value condition in the rewritten query depends on the presence of the respective 'select' attribute, for if the second is not mapped in the target database, both of them cannot be rewritten. On the other hand, a user may create the original query with or without the value condition, meaning that in general the presence of the 'select' attribute in the original query does not depend on the presence of the respective value condition or vice versa.

Observation: *Hence, we follow the second reasoning and do not correlate 'select' attributes with respective value conditions in the calculation of query similarity.*

2.2 Query Similarity Criteria

Based on the above observations, we want to form the criteria for the assessment of query similarity. The rough outcome of the earlier discussion is that missing or additional query features, i.e. 'select' attributes or 'where' conditions should be considered decreasingly in query similarity, from a conservative point of view. We choose a conservative point of view in order to determine a correct estimation of query similarity in any context of P2P database applications.

We have to refine our observations by ordering the importance of the role of the various missing or additional query features. First, we reckon that key attributes are highly important in a relational schema since their values uniquely prescribe the values of other attributes. We think that the role of keys in queries is as important as in the schema itself, no matter if such an attribute appears in a 'select' or 'where' clause. Thus, deficient rewritings of key attributes may result in severe semantic deviations from the original query. Second, 'select' attributes represent what information the user requires actually. Thus, their lack in the rewritten query is decisively irreparable. Third, even though the lack of join conditions is a negative factor for query similarity, it results in a query version that retrieves a superset of the data that would be retrieved by a query with the rewritten joins. Furthermore, the lack of value constraints has the same effect in the query as the lack of joins. However, the lack of joins probably results in much bigger supersets of retrieved data than the lack of value constraints. Finally, the introduction of new value constraints and joins on non-key attributes is considered a deficiency [4]. Yet, new conditions produce a rewrittint that is classically contained in the source query; thus, we think of the introduction of new conditions as the weakest criterion of all.

Thus, we form the following criteria for the definition of the similarity of two query versions. The criteria are ordered according to their importance in query similarity.

1. Key attributes are rewritten, no matter what their position in the query is
2. 'select' attributes are rewritten
3. Join attributes are rewritten
4. Constrained attributes (beyond join ones) are rewritten
5. There are no new value constraints and joins on non-key attributes.

3 Query Reformulation

Having discussed how similarity should be defined on queries propagated through a P2P environment, we move next to discuss an algorithm that performs query reformulation.

We assume that peers own a query reformulation mechanism based on existing query rewriting algorithms, which enables the production of a rewritten version Q_{rewr} from the original query Q_{orig}, using the following rules:

– Q_{rewr} maintains in the 'select' clause all the 'select' attributes of Q_{orig} that can be rewritten through the available mappings. Thus, in conjunctive form, the head of Q_{rewr} is a projection of the head of Q_{orig}.

[4] This last criterion can be broken down in more criteria depending on the several observations for additional query features. Due to lack of space we summurize all of them in one.

– all the 'where' conditions of Q_{orig} that cannot be rewritten through the available mappings are ignored. The rest are rewritten and included in Q_{rewr}. In conjunctive form, Q_{orig} and Q_{rewr} have mappings between attributes of predicates even if predicates themselves cannot be mapped.

As an example of the first rule above, remember the original query posed by Dr Davis, Q_{orig}; it is not possible to rewrite it to the schema of StuartDB, because the 'select' attribute Treatment.Dosology of Q_{orig} is not mapped in StuartDB. Thus, this attribute is ignored in the classical query rewriting procedure.

As an example for the second rule above, consider the original query, Q_{orig}, augmented with a value condition:
$Q_{orig\_changed}$:

```
SELECT  V.Pid, D.DisDescr, D.Ache, T.Drug, T.Dosology
FROM    Disease D, Treatment T, Visits V
WHERE   V.Did = D.Did AND D.Did = T.Did AND T.Dosology = 5mg
```

Since the attribute Treatment.Dosology is not mapped in StuartDB, there cannot be a classically contained rewritting of $Q_{orig\_changed}$, because the condition 'T.Dosology = 5mg' cannot be rewritten. Thus, this condition is ignored.

In order to achieve the above reformulation, we propose the preprocessing of the incoming query Q_{orig} and the production of the version Q'_{orig} that contains the part of Q_{orig} that can be *best* rewritten through the available mappings. Then, Q'_{orig} is rewritten with the aforementioned classical query rewriting algorithms.

Query Preprocessing Guidelines. In order to preprocess a query and produce the input for the rewriting algorithm, we choose the mapping(s) with respect to which we will perform the preprocessing. As aforementioned, we want to choose the mappings that *best* rewrite the query. The best rewritten version is valuated with respect to the similarity criteria defined in the previous section. The mappings used for the rewriting are actually the exclusive means that provide the rewritten query features thus, the structure of the mappings reflects the rewritten query. Also, mappings are actually queries (or pairs of queries) themselves. Thereupon, we base our decision for the selection of mappings on the query similarity criteria defined in Section 2.

There is a variety of query-mapping combinations that we can come across during the mappings selection procedure. We discuss the combinations that do not match completely. The following is a categorization of the main query-mapping combinations that we can come across, where attributes of relations involved in the query are missing from the mapping. Other combinations actually fall into one or more of these categories. The categorization follows the lines of query similarity aspects.

Case A: Considering GAV mappings

1. For attributes that are not present in the 'select' clause:
   (a) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $P(x,y,\_) : -P'(x,y)$
   (b) The query is $Q(x,y) : -P(x,y,z), z =' c'$ and the mapping is $P(x,y,\_) : -P'(x,y)$
   (c) The query is $Q(x,y) : -P(x,y,z)R(z,w)$ and the mapping is $R(\_,w) : -R'(w)$
   (d) The query is $Q(x,y) : -P(x,y,z)R(z,w)$ and the mapping is $R(z,\_) : -R'(z)$
   (e) The query is $Q(x,y) : -P(x,y,z)$, the mapping is $P(x,y,z) : -P'(x,y,z), z =' c'$
   (f) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $P(x,y,z) : -P'(x,y,z)R'(z)$

2. For attributes that are present in the 'select' clause:
   (a) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $P(x,\_,z) : -P'(x,z)$

Cases 1(b) and 1(c) denote missing value constraints and joins, respectively. Cases 1(a,d) denote missing attributes from relations of the query; yet these attributes do not appear in the query itself. Cases 1(e,f) denote additional 'where' conditions.

Case B: Considering LAV mappings

1. 'select' attributes of the query are missing from the mapping.
   (a) The query is $Q(x,y) : -P(x,y,z)$ and the mapping is $Q'(x) : -P(x,y,z)$ or $Q'(x,z) : -P(x,y,z)$
2. 'where'conditions of the query are not mapped through the mapping:
   (a) The query is $Q(x,y) : -P(x,y,z), z =' c'$ and the mapping is $Q'(x,y) : -P(x,y,z)$
   (b) The query is $Q(x,y) : -P(x,y,z)R(z)$ and the considered mapping is $Q'(x,y) : -P(x,y,z)$ and there is no mapping for $R(z)$

Case 2(a) denotes a missing value constraint, whereas case 2(b) denotes a missing join. Note that both cases can refer to mappings with additional 'where' conditions

Case C: Considering GLAV mappings[5]

1. 'select' attributes of the query are missing from the mapping.
   (a) The query is $Q(x,y,z)$ and the mapping is $Q(x,y,z) : -Q'(x,y)$
2. The mapped query has additional 'select' attributes:
   (a) The query is $Q(x,y)$ and the mapping is $Q(x,y) : -Q'(x,y,z)$

We select mappings according to the criteria of Section 2.2. The lack of attributes of relations involved in the query that do not appear either in the 'select' or the 'where' clause, such as in Case A 1(a,d), is not taken into consideration: based on the mappings themselves denote that the predicates match, even if not all their attributes match.

In order to impose the above criteria we construct an associative similarity function. Specifically, for a query subgoal g and a mapping M, the associative function M_{as} quantifies the lack of non-matched attributes.

$$M_{as}(g,M) = \sum_{i=1}^{|g|} w_i \cdot a_i,\ w_i \in \{w_k, w_s, w_j, w_c\},\ a_i \in \{0,1\}$$

where $|g|$ denotes the arity of g, i.e. the total number of attributes in g involved in the query. Also, each subgoal attribute, i, has a weight, w_i that denotes if it is a key, a 'select', a join or a constrained attribute. If it has more than two such characteristics, it keeps the one that is higher in the hierarchy of the criteria. Accordingly, the weights for a key, a 'select', a join or a constrained attribute is w_k, w_s, w_j and w_c, respectively. We require that $w_k > w_s > w_j > w_c$. Finally, a_i denotes if the respective attribute is matched in the mapping ($a_i = 0$) or not ($a_i = 1$).

We use the following form that estimates the matching ability of the selected mappings:

$$Sim_g(Q,\mathcal{M}) = 1 - \frac{\sum_j Sim_{g_j} \cdot M_{as} + \sum_k w_a}{\sum_j \sum_{i=1}^{|g_j|} w_{ji}}\ w_i \in \{w_k, w_s, w_j, w_c\},\ a_i \in \{0,1\}$$

[5] We only consider the the head of queries in GLAV mappings. Refining our categorization for the bodies of these queries is future work.

Input: the query to be rewritten Q
Output: A set of mappings $\mathcal{M}$

- Step 1 Consider GLAV mappings:
  For each mapping M represented as $Q' : -Q''$ compute $M_{sim}(Q,Q')$*. Make the set $Sim_{GLAV} = \{M, M_{sim}(Q,Q')\}$
- Step 2 Consider GAV mappings:
  For each subgoal g_j of Q:
  - Make a set $Sim_{g_j} = \{M, M_{as}\}$ and initiate $M = null$[6] and $M_{as} = 0$
  - For each mapping M that matches the predicate of g_j compute $M_{as}(g_j,M)$; if $Sim_{g_j}.M_{as} < M_{as}(g_j,M)$ replace $Sim_{g_j}.M$ with M and $Sim_{g_j}.M_{as}$ with $M_{as}(g_j,M)$

  Make the set $Sim_{GAV} = \{\mathcal{M}_{\mathcal{GAV}}, Sim_g\}$, where $Sim_g(Q,\mathcal{M}) = 1 - \frac{\sum_j Sim_{g_j}.M_{as} + \sum_k w_a}{\sum_j \sum_{i=1}^{|g_j|} w_{ji}}$
- Step 3 Consider LAV mappings and produce in the same way as in Step 2 $Sim_{LAV} = \{\mathcal{M}_{\mathcal{LAV}}, Sim_g\}$
- Step 4 Compare $Sim_{GLAV}.M_{sim}$, $Sim_{GAV}.Sim_g$ and $Sim_{LAV}.Sim_g$; depending on the highest value of the three, replace $\mathcal{M}$ with one of the sets $Sim_{GLAV}.M$, $Sim_{GAV}.\mathcal{M}$, $Sim_{LAV}.\mathcal{M}$

*$M_{sim}(Q,Q')$ is a function that quantifies the semantic similarity of two queries Q and Q' based on the proposed criteria. Due to lack of space we do not present an implementation of M_{sim} in this work.

Fig. 2. Algorithm for the selection of mappings for the query rewriting

$\mathcal{M}$ is the set of selected mappings and $Sim_{g_j}.M_{as}$ is the M_{as} value for the g_j subgoal. The weight for an additional conditions is w_a ($w_c > w_a$) and $\sum w_a$ represents the total weight of all the additional conditions in the selected mappings.

The algorithm that chooses which mappings will be used in the query rewriting procedure is shown in Figure 2. Briefly, the algorithm considers three sets of mappings: GLAV, GAV and LAV. It produces one subset of each set. Each such subset contains the mappings that are most similar to the corresponding query subgoals according to the aforementioned criteria. Finally, the algorithm chooses the subset of the three which has the highest Sim_g value for the query.

4 Conclusions and Future Work

In this paper we propose the preprocessing of the query in order to reuse classical rewriting algorithms in P2P databases. We discuss query similarity aspects from a structural point of view and we propose criteria along which we can form a similarity function. We present an algorithm that selects the best mappings in order to perform query rewriting. We are currently implementing the proposed rewriting mechanism. We intend to perform exhaustive experiments on the similarity aspects and the preprocessing guidelines discussed in this paper, in order to determine their effect in realistic situations.

References

[1] F. Afrati, C. Li, and P. Mitra. Answering Queries Using Views with Arithmentic Comparisons. In *21th ACM PODS*, 2002.
[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *ICDE*, 2002.
[3] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated Ranking of Database Query Results. In *CIDR*, 2003.

[4] W. W. Chu and G. Zhang. Associative Query Answering via Query Feature Similarity. In *IIS*, 1997.
[5] W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In *SIGMOD*, 1998.
[6] N. Fuhr. A Probabilistic Framework for Vague Queries and Imprecise Information in Databases. In *VLDB*, 1990.
[7] A. Ghosh, J. Parikh, V. S. Sengar, and J. R. Haritsa. Plan Selection based on Query Clustering. In *VLDB*, 2002.
[8] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. In *IEEE Transactions on Knowledge and Data Engineering*, 2003.
[9] V. Kantere, I. Kiringa, J. Mylopoulos, A. Kementsientidis, and M. Arenas. Coordinating P2P Databases Using ECA Rules. In *DBISP2P*, 2003.
[10] W. Kießling and G. Kostner. Preference SQL - Design, Implementation, Experiences . In *VLDB*, 2002.
[11] A. Y. Levy. Answering Queries Using Views: A Survey. In *VLDB Journal*, 2001.
[12] A. Y. Levy, A. Rajaraman, and J. O. Ordille. Query-answering Algorithms for Information Agents. In *13th International Conference on Artificila Intelligence*, 1996.
[13] P. Mitra. An Algorithm for Answering Queries Efficiently Using Views. In *Australesian Database Conference*, 2001.
[14] A. Motro. VAGUE: A User Interface to Relational Databases that Permis Vague Queries. In *TOIS 6(3), 187-214*, 1988.
[15] B. Ooi, Y. Shu, K. Tan, and A. Zhou. PeerDB: A P2P-based System for Distributed Data Sharing. In *ICDE*, 2003.

Efficient Searching and Retrieval of Documents in PROSA

Vincenza Carchiolo, Michele Malgeri, Giuseppe Mangioni, and Vincenzo Nicosia

Università degli Studi di Catania
Facoltà di Ingegneria
V.le A. Doria 6
95100 – Catania

Abstract. Retrieving resources in a distributed environment is more difficult than finding data in centralised databases. In the last decade P2P system arise as new and effective distributed architectures for resource sharing, but searching in such environments could be difficult and time–consuming. In this paper we discuss efficiency of resource discovery in ***PROSA***, a self–organising P2P system heavily inspired by social networks. All routing choices in ***PROSA*** are made locally, looking only at the relevance of the next peer to each query. We show that ***PROSA*** is able to effectively answer queries for rare documents, forwarding them through the most convenient path to nodes that much probably share matching resources. This result is heavily related to the small–world structure that naturally emerges in ***PROSA***.

1 Introduction

Organisation of electronic resources and documents is of the most importance for efficient searching and retrieval. Nowadays the WWW is a (negative) example of how searching and obtaining informations from an unstructured knowledge base could really become difficult and frustrating. In the case of the World Wide Web, this problem is faced and partially resolved by centralised searching engines, such as Google, MSN–Search, Yahoo and so on, which can help users in pruning away unuseful resources during searches. But searching strategies used by web indexing engines cannot be easily adopted in a P2P environment, mainly because nodes of such a distributed system cannot be compared to web–servers. Each peer shares a small amount of resources, can join and leave the network many times in a week and usually searches and retrieve resources belonging to a small number of different topics. In the last few years many P2P structures have been proposed, in order to build a valuable and efficient distributed environment for resource sharing.

The problem is that existing P2P systems usually ask the user to choose between efficiency and usability. In fact, while DHT systems allow fast resource searching [1] [2] [3] introducing unnatural indexing models, unstructured and weakly structured P2P systems [4][5][6] usually allow users to easily express

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 298–309, 2007.

queries but have poor performance with respect to bandwidth and time consumption.

In this work we analyse retrieving performance of ***PROSA*** (P2P Resource Organisation by Social Acquaintances), a P2P system heavily inspired by social networks: joining, searching resources and building links among peers in ***PROSA*** are performed in a social way. Each peer gains a certain amount of strong links to peers which share similar resources and also maintains weak links to far away peers.

The linking phase is similar to a birth: each peer is given just a couple of weak links which can be used for query forwarding. Queries for resources are forwarded through outgoing links to other peers, in accordance with a defined "similarity" between the query and shared resources. New relationships in real social networks arise because people have similar interests, culture and knowledge. In a similar way, new links among peers in ***PROSA*** are established when a query is forwarded and successful answered, so that peers which share similar resources finally get connected together.

In this paper we focus on the ability of ***PROSA*** in answering queries with a sufficient number of results, even if a small amount of existing documents match them. Matching documents are retrieved in an efficient way, forwarding queries to a small amount of nodes using just a few "right" links, thanks to small–world structure that naturally emerges in a ***PROSA*** network.

In section 2 we give a brief formal description of involved algorithms ; section 3 reports simulation results, focused on retrieval of rare resources; in section 4 the efficiency of the query routing algorithm is discussed, while section 5 propose guidelines for future work.

2 *PROSA*: A Brief Description

As stated above, ***PROSA*** is a P2P network based on social relationships. More formally, we can model ***PROSA*** as a directed graph:

$$\boldsymbol{PROSA} = (\mathcal{P}, \mathcal{L}, P_r, Label) \tag{1}$$

$\mathcal{P}$ denotes the set of peers (i.e. vertices), $\mathcal{L}$ is the set of links $l = (s, t)$ (i.e. edges), where t is a neighbour of s. For link $l = (s, t)$, s is the source peer and t is the target peer. All links are directed.

In P2P networks the knowledge of a peer is represented by resources it shares with other peers. In ***PROSA*** the mapping $P_r : \mathcal{P} \rightarrow 2^{\mathcal{R}}$, associates peers with resources. For a given peer $s \in \mathcal{P}$, $P_r(s)$ is the set of resources hosted by peer s. Given a set of resources, we define a function $R_c : 2^{\mathcal{R}} \rightarrow \mathcal{C}$ that provides a sort of compact description of all resources. We also define a function $P_k : \mathcal{P} \rightarrow \mathcal{C}$, such that, for a given peer s, $P_k(s)$ is a compact description of the peer knowledge (*PK - Peer Knowledge*). It can also be obtained combining P_r and R_c: $P_k(s) = R_c(P_r(s))$.

Relationships among people in real social networks are usually based on similarities in interests, culture, hobbies, knowledge and so on [7][8][9]. Usually

these kind of links evolve from simple "acquaintance–links" to what we called "semantic–links". To implement this behaviour three types of links have been introduced: *Acquaintance–Link* (AL), *Temporary Semantic–Link* (TSL) and *Full Semantic–Link* (FSL). TSLs represent relationships based on a partial knowledge of a peer. They are usually stronger than ALs and weaker than FSLs.

In ***PROSA***, if a given link is a simple AL, then the source peer does not know anything about the target peer. If the link is a FSL, the source peer is aware of the kind of knowledge owned by the target peer (i.e. it knows $P_k(t)$, where $t \in \mathcal{P}$ is the target peer). Finally, if the link is a TSL, the peer does not know the full $P_k(t)$ of the linked peer; it instead has a *Temporary Peer Knowledge* (TP_k) which is based on previously received queries from the source peer. Different meanings of links are modelled by means of a labelling function $Label$: for a given link $l = (s, t) \in L$, $Label(l)$ is a vector of two elements $[e, w]$: the former is the link label and the latter is a weight used to model what the source peer knows about the target peer; this is computed as follows:

- if $e = AL \Rightarrow w = \emptyset$
- if $e = TSL \Rightarrow w = TP_k$
- if $e = FSL \Rightarrow w = P_k(t)$

In the next two sections, we give a brief description of how ***PROSA*** works. A detailed description of ***PROSA*** can be found in [10].

2.1 Peer Joining *PROSA*

The case of a node that wants to join an existing network is similar to the birth of a child. At the beginning of his life a child "knows" just a couple of people (his parents). A new peer which wants to join, just looks for n peers at random and establishes ALs to them. These links are ALs because a new peer doesn't know anything about its neighbours until he doesn't ask them for resources. This behaviour is quite easy to understand: when a baby comes to life he doesn't know anything about his parents and relatives. The ***PROSA*** peer joining procedure is described by algorithm 1.

Algorithm 1 JOIN: Peer s joining to ***PROSA***$(\mathcal{P}, \mathcal{L}, P_r, Label)$

Require: ***PROSA***$(\mathcal{P}, \mathcal{L}, P_r, Label), Peer\ s$
1: $\mathcal{RP} \leftarrow rnd(P, n)$ {Randomly selects n peers of ***PROSA*** }
2: $\mathcal{P} \leftarrow \mathcal{P} \cup s$ {Adds s to set of peers}
3: $\mathcal{L} \leftarrow \mathcal{L} \cup \{(s, t), \forall t \in \mathcal{RP}\}$ {Links s with the randomly selected peers}
4: $\forall t \in \mathcal{RP} \Rightarrow Label(p, q) \leftarrow [AL, \emptyset]$ {Sets the added links as AL}

2.2 *PROSA* Dynamics

In order to show how does ***PROSA*** work, we need to define the structure of a query message. Each query message is a quadruple:

$$Q_M = (qid, q, s, n_r) \tag{2}$$

where qid is a unique query identifier to ensure that a peer does not respond to a query more then once; q is the query, expressed according to the used knowledge model[1]; $s \in P$ is the source peer and n_r is the number of required results. ***PROSA*** dynamic behaviour is modelled by algorithm 2 and is strictly related to queries. When a user of ***PROSA*** asks for a resource on a peer s, the inquired peer s builds up a query q and specify a certain number of results he wants to obtain n_r. This is equivalent to call $ExecQuery(\boldsymbol{PROSA}, s, (qid, q, s, n_r))$.

Algorithm 2 ExecQuery: query q originating from peer s executed on peer cur

Require: $\boldsymbol{PROSA}(\mathcal{P}, \mathcal{L}, P_r, Label),\ cur \in \mathcal{P},\ q \in QM$

```
 1: Result ← ∅
 2: if cur ≠ s then
 3:    UpdateLink(PROSA, cur, s, q)
 4: end if
 5: (Result, numRes) ← ResourcesRelevance(PROSA, q, cur, n_r)
 6: if numRes = 0 then
 7:    f → SelectNextPeer(PROSA, cur, q)
 8:    if f ≠ null then
 9:       ExecQuery(PROSA, f, qm)
10:    end if
11: else
12:    SendMessage(s, cur, Result)
13:    L ← L ∪ (s, cur)
14:    Label(s, cur) ← [FSL, P_k(cur)]
15:    if numRes < n_r then
16:       {– Semantic Flooding –}
17:       for all t ∈ Neighborhood(cur) do
18:          rel → PeerRelevance(P_k(t), q)
19:          if rel > Threshold then
20:             qm ← (qid, q, s, n_r − numRes)
21:             ExecQuery(PROSA, t, qm)
22:          end if
23:       end for
24:    end if
25: end if
```

The first time $ExecQuery$ is called, cur is equal to s and this avoids the execution of instruction # 3. Following calls of $ExecQuery$, i.e. when a peer receives a query forwarded by another peer, use function $UpdateLink$, which updates the link between current peer cur and the forwarding peer $prev$, if necessary. If the requesting peer is an unknown peer, a new TSL link to that peer is added having as weight a Temporary Peer Knowledge(TP_k) based on the received query message. Note that a TP_k can be considered as a "good

[1] If knowledge is modelled by Vector Space Model, for example, q is a state vector of stemmed terms. If knowledge is modelled by ontologies, q is an ontological query, and so on.

hint" for the current peer, in order to gain links to other remote peers. It is really probable that the query would be finally answered by some other peer and that the requesting peer will eventually download some of the resources that matched it. It would be useful to record a link to that peer, just in case that kind of resources would be requested in the future by other peers. If the requesting peer is a TSL for the peer that receives the query, the corresponding TP_k is updated. If the requesting peer is a FSL, no updates are necessary.

The relevance of a query with respect to the resources hosted by a peer is evaluated calling function *ResourcesRelevance*. Two possible cases can hold:

- If none of the hosted resources has a sufficient relevance, the query has to be forwarded to another peer f, called "forwarder". This peer is selected among s neighbours by $SelectForwarder$, using the following procedure:
  - Peer s computes the relevance between query q and the weight of each links connecting itself to his neighbourhood.
  - It selects the link with the highest relevance, if any, and forward the query message to it.
  - If the peer has neither FSLs nor TSLs, i.e. it has just ALs, the query message is forwarded to one link at random.

  This procedure is described in algorithm 2, where subsequent forwards are performed by means of recursive calls to $ExecQuery$.
- If the peer hosts resources with sufficient relevance with respect to q, two sub-cases are possible:
  - The peer has sufficient relevant documents to full-fill the request. In this case a result message is sent to the requesting peer and the query is no more forwarded.
  - The peer has a certain number of relevant documents, but they are not enough to full-fill the request (i.e. they are $< n_r$). In this case a response message is sent to the requester peer, specifying the number of matching documents. The message query is forwarded to all the links in the neighbourhood whose relevance with the query is higher than a given threshold (semantic flooding). The number of matched resources is subtracted from the number of total requested documents before each forward step.

When the requesting peer receives a response message it build a new FSL to the answering peer and then presents results to the user. If the user decides to download a certain resource from another peer, the requesting peer directly contacts the peer owning that resource asking for download. If download is accepted, the resource is sent to the requesting peer.

3 Information Retrieval in PROSA

Other studies about ***PROSA*** [10] [11] revealed that it naturally evolves to a small–world network, with a really high clustering coefficient and a relatively small average path length between peers.

The main target of this work is to show that ***PROSA*** does not only has desirable topological properties, but also that resource searching can be massively improved exploiting those characteristics. The fact that all peers in ***PROSA*** are connected by a small number of hops does not guarantees anything about searching efficiency. In this section we show that searching resources in ***PROSA*** is really fast and successful, mainly because peers that share resources in the same topic usually results to be strongly connected with similar peers.

3.1 Two Words About Simulations

In order to show that ***PROSA*** can be used to efficiently share resources in a P2P environment, we developed a event-driven functional simulator written in Python. The knowledge base used for simulations is composed by scientific articles in the field of math and philosophy. Articles about math come from "Journal of American Mathematical Society"[12], "Transactions of the American Mathematical Society"[13] and "Proceedings of the American Mathematical Society"[14], for a total amount of 740 articles. On the other hand, articles in the field of philosophy come from "Journal of Social Philosophy" [15], "Journal of Political Philosophy" [16], "Philosophical Issues" [17] and "Philosophical Perspectives" [18], for a total amount of 750 articles.

The simulator uses a Vector Space [19] knowledge model for resources. Each document is represented by a state vector which contains the highest 100 TF–IDF [20] weights of terms contained into the document.

Each peer contains, on average, 20 ± 5 articles in the same topic. Nodes perform 80% of queries in the same topic of the hosted resources and the remaining 20% in the other topic. We choose to do so after some studies about queries distribution in a Gnutella P2P system [4] and with real social communities in mind, where the most part of requests for resources are focused on a really small amount of topics.

3.2 Number of Retrieved Documents

One of the most relevant quality measure of a resource searching algorithm is the number of documents retrieved by each query. In this paragraph we examine results obtained with ***PROSA***, using the query mechanism described in section 2. We also compare ***PROSA*** to other searching strategies, such as random walk and flooding.

Figure 1(a) shows a comparison of average number of retrieved documents in a ***PROSA*** network for different number of nodes, when each node performs 15 queries on average.

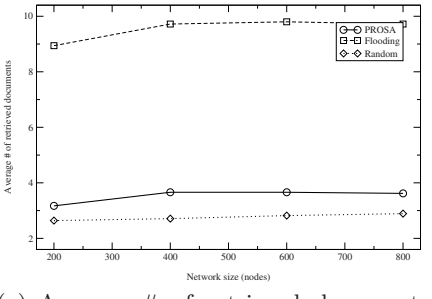
As showed in figure 1(a), the best performance is obtained by flooding, since the average number of retrieved documents per query is about 10, that is the number of documents required by each query (n_r)[2]. Nevertheless, ***PROSA*** is

[2] A query is no more forwarded if a sufficient number of documents has been retrieved, as explained in 2.

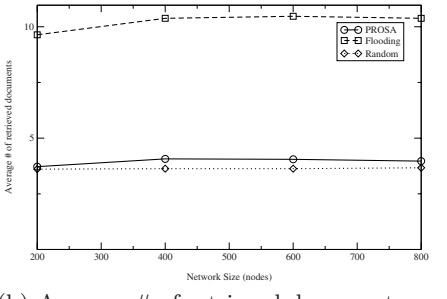
able to retrieve about 4 documents per query, on average, and this result is still better than that obtained with a random walk, which usually retrieves only 2.8 documents per query.

This suggests that the query routing algorithm, based on local link ranking, is really efficient and usually let queries "flow" in the direction of nodes that can probably answer them. We note that ***PROSA*** is able to retrieve a relatively high number of documents also if compared with a simple flooding. This is a good result, since flooding is known as being the optimal searching strategy: queries are actually forwarded to all nodes, so all existing and matching documents are retrieved, until the number of required documents has not been obtained.

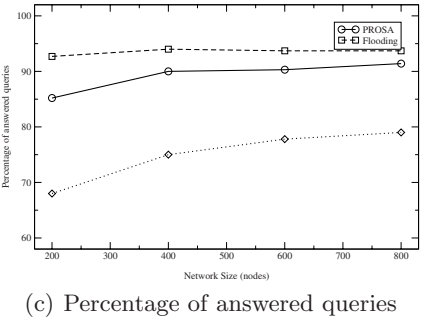
In figure 1(b) the average number of retrieved documents per successful query is reported. The best performance is once again obtained by flooding, while ***PROSA*** retrieves an average of 4.2 documents for each successful query over 10 documents required. Random walk has, once again, the worst performance.



(a) Average # of retrieved documents per query



(b) Average # of retrieved document per successful query



(c) Percentage of answered queries

Fig. 1.

Looking only at the number of retrieved documents could be misleading: it is not important to have a small amount of queries answered with a high number of documents. It is desirable having almost all feasible queries[3] answered by a

[3] A query is feasible if there exist matching documents to answer it. Otherwise it is considered unfeasible.

sufficient number of documents. Figure 1(c) shows the percentage of retrieved documents for ***PROSA***, flooding and random walk, on the same ***PROSA*** network with different network sizes. Note that in every case the average amount of unfeasible queries is around 6%.

The highest percentage of answered queries is obtained by flooding the network, since about 94% of queries have an answer. This means that practically all the queries are answered, if we except those that have no matching documents. A valuable result is obtained also by ***PROSA***: 84% to 92% of all queries are answered, while random walk usually returns result for less than 80% of issued queries [4]. The percentage of answered queries increases with network size, for all searching strategies, because all nodes have an average number of 20 documents: more nodes means more documents, i.e. an higher probability of finding matching documents.

3.3 Query Recall

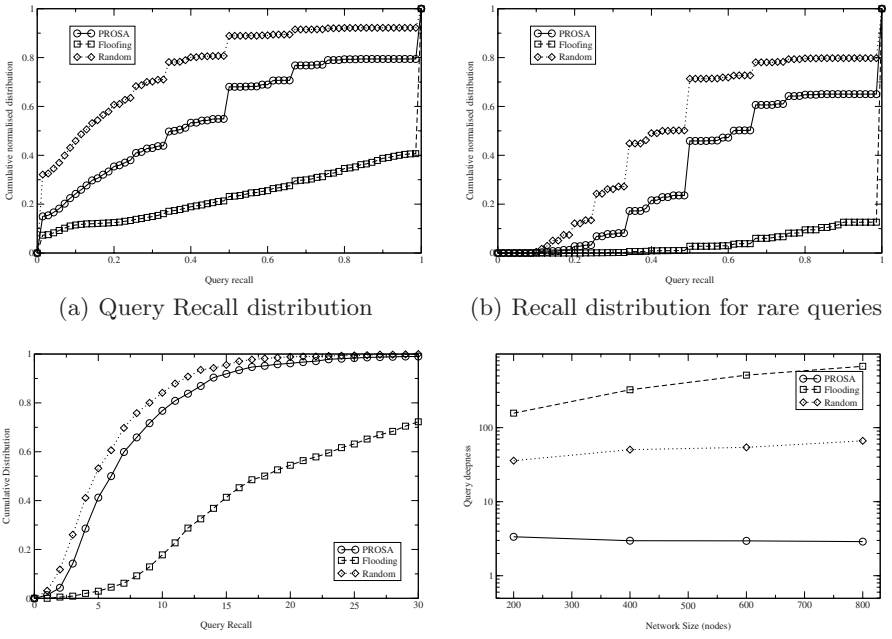
Either if it is an important parameter for a resource searching and retrieving strategy, the number of retrieved documents is not the best measure of how much documents a searching algorithm is able to retrieve. Since not all queries match the same number of documents, it is better to measure the percentage of retrieved documents over all matching documents. A valuable measure is the so–called "recall", i.e. the percentage of distinct retrieved documents over the total amount of distinct existing documents that match a query. In figure 2(a) we show the recall distribution for ***PROSA***, flooding and random walk when each node performs 15 queries on average.

The best performance is obtained, once again, flooding the network: about 60% of queries have a recall of 100%, and about 80% of queries have a recall of 50%. Searching by flooding could not return all documents because ***PROSA*** is a directed graph, and unconnected components could still exist. Also ***PROSA*** has high recall: about 20% of queries obtain all matching documents, while 45% of queries are answered with one half of the total amount of matching documents. Random walk is the worst case: about 80% of queries has a recall of less than 50% and only 8% of queries obtain all matching documents.

Recall measured as the simple percentage of retrieved document over the total amount of matching documents does not take into account the fact that in ***PROSA*** queries are requested to retrieve n_r documents and no more. This fact could practically influence the recall measure for ***PROSA*** networks, since queries are no more forwarded if a sufficient number of documents has been retrieved. On the other hand, it is important to analyse the recall in the case of "rare" queries. Note that we consider a query as being "rare" when the total number of matching documents is lower than the number of requested documents; similarly a query is considered "common" if it matches more than n_r[5].

[4] If a query eventually enters an unconnected component, it cannot be further forwarded.

[5] Reported results are relative to $n_r = 10$.



(a) Query Recall distribution

(b) Recall distribution for rare queries

(c) Recall distribution for common queries

(d) Average query deepness

Fig. 2.

Figure 2(b) shows the cumulative normalised distribution of recall for rare queries, while figure 2(c) reports the cumulative distribution for common queries.

Results reported in figure 2(b) are really interesting: ***PROSA*** answers 35% of rare queries by retrieving all matching documents, while 75% of queries retrieve at least 50% of the total amount of matching documents; less than 10% of queries obtain less than 30% of matching documents. Performance of a random walk is worse than that obtained by ***PROSA***: only 20% of queries obtain all matching documents, while more than 30% of them obtain less than 30% of matching results.

The situation is slightly different for common queries. As reported in figure 2(c), ***PROSA*** is able to retrieve at least 10 documents for 20% of issued queries and, in every case, at least one document is found for 99% of queries, and at least 3 documents for 85% of queries. We think that this behaviour is also affected by the chosen value of n_r.

In order to better understand benefits of using ***PROSA***, it is interesting to look also at other measures that could clarify some ***PROSA*** characteristics. For instance, recall results are of poor relevance without a measure of how fast answers are obtained. A feasible measure of speed could be the average query deepness, defined as the average number of "levels" a query is forwarded far away from the source node.

In figure 2(d) we show average deepness of successful queries for ***PROSA***, flooding and random walk on the same ***PROSA*** network for different number of peers.

Query deepness for ***PROSA*** is around 3 and is not heavily affected from the network size, while that of flooding and random walk is much higher (from 30 to 60 and from 120 to 600, respectively). Better results obtained by ***PROSA*** cannot be simply explained by network clustering coefficient, since all simulation are performed on the same network. We suppose that it is mainly due to the searching algorithm implemented by ***PROSA*** itself: it is able to find a convenient and efficient route to forward queries along, avoiding a large number of forwards to non–relevant nodes.

4 Energetical Considerations

An important parameter to take in account in order to quantify the efficiency of a searching strategy is the "energy" needed to forward and answer each query. In a theoretical model it is probably of no great importance how much power is needed in order to answer a query. But for real systems this is a crucial parameter. One of the main issues with unstructured P2P networks such as Gnutella [4] is that queries waste a lot of bandwidth, since a large fraction of the network is flooded and a great amount of nodes are involved in answering each query. It is possible to roughly define the average "energy" required for each query using equation 3, where N_q is the number of nodes to which the query has been forwarded and L_q is the number of links used during query routing. b and c are dimensional scaling factors.

$$E_q = b \cdot L_q + c \cdot N_q \tag{3}$$

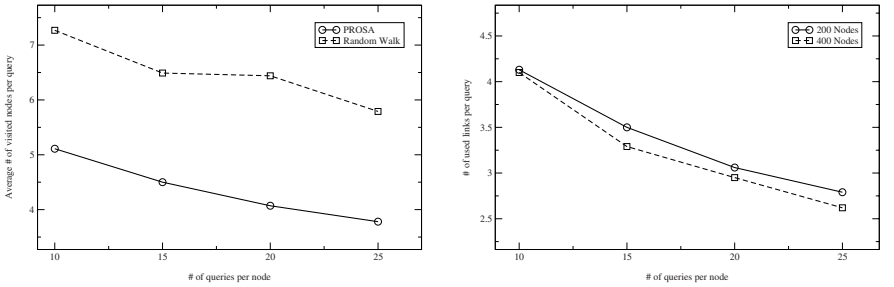
The definition given here for query energy is quite simple: it takes into account the required bandwidth, represented by the factor $b \cdot L_q$, and the computational power needed by nodes in order to process queries, represented by $c \cdot N_q$.

To estimate the amount of energy required to answer queries, we could look at the average number of nodes and the average number of links involved in each query. Note that N_q and L_q are usually different, since a node can be reached using many paths: either if it processes the query only once[6], the bandwidth wasted to forward the query to it cannot be saved.

Figure 3(a) and 3(b) show, respectively. the average number of nodes involved and the average number of links used by successful queries, both for ***PROSA*** and a simple random walk search.

Since random walk uses a higher number of nodes and a higher number of links in order to answer the same queries, it is clear that ***PROSA*** requires less energy. On the other hand, since ***PROSA*** is able to retrieve more matching documents than a random walk (as shown in section 3.2), we can state that ***PROSA*** is really efficient with respect to average "energy" required to answer queries.

[6] Requests with the same query id are ignored.



(a) Average number of visited nodes per query

(b) Average number of used links per query

Fig. 3.

5 Conclusions and Future Work

This work presented a formal description of ***PROSA***, a self–organising system for P2P resource sharing heavily inspired by social networks. Simulations show that resource searching and retrieving in ***PROSA*** is really efficient, because of the ability of peers in making good local choices that result in fast and successful global query routing. Interesting results are obtained for query recall measured on rare documents: ***PROSA*** is able to route queries for those documents directly to nodes that probably can successfully answer them. Since ***PROSA*** results to be a small–world, all nodes are reached in a few steps, avoiding to waste bandwidth and processing power. Future works include further studying ***PROSA*** in order to discover emerging structures, such as semantic groups and communities of similar peers.

References

1. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. Lecture Notes in Computer Science **2009** (2001) 46
2. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA (2000)
3. Zhao, B.Y., Kubiatowicz, J.D., Joseph, A.D.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley (2001)
4. Loo, B., Huebsch, R., Stoica, I., Hellerstein, J.: The case for a hybrid p2p search infrastructure. In: Proceedings of the 3rd Internationa Workshop on Peer–to–Peer Systems (IPTPS). (2004)
5. Zhu, Y., Yang, X., Hu, Y.: Making search efficient on gnutella-like p2p systems. In: Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International, IEEE Computer Society (2005) 56a– 56a

6. Bawa, M., Manku, G.S., Raghavan, P.: Sets: search enhanced by topic segmentation. In: SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval, New York, NY, USA, ACM Press (2003) 306–313
7. Newman, M.E.J.: The structure of scientific collaboration networks. PROC.NATL.ACAD.SCI.USA **98** (2001) 404
8. J., S.: Social Networks Analysis: A Handbook. Sage Pubblications, London (2000)
9. Albert, R., Barabasi, A.L.: Statistical mechanics of complex networks. Reviews of Modern Physics **74** (2002) 47
10. Mangioni, V.C.M.M.G., Nicosia, V.: Social behaviours applied to p2p systems: an efficent algorithm for resources organisation. 2nd International Workshop on Collaborative P2P Information Systems, COPS 2006, Manchester (2006)
11. V. Carchiolo, M. Malgeri, G.M., Nicosia, V.: Self-organisation of resources in prosa p2p network. In: Self–Managed Networks, Systems, and Services – Proceedings of Second IEEE International Workshop, SelfMan 2006, Dublin. Number 3996 in LNCS (2006) 172–174
12. Society, A.M., ed.: Journal of the American Mathemetical Society. WWW (1998-2006)
13. Society, A.M., ed.: Tnansactions of the American Mathemetical Society. WWW (1998-2006)
14. Society, A.M., ed.: Proceedings of the American Mathemetical Society. WWW (1998-2006)
15. Publishing, B., ed.: Journal of Social Philosophy. WWW (1998-2006)
16. Publishing, B., ed.: Journal of Political Philosophy. WWW (1998-2006)
17. Publishing, B., ed.: Philosophical Issues. WWW (1998-2006)
18. Publishing, B., ed.: Philosophical Perspectives. WWW (1998-2006)
19. Salton, G., Buckley, C.: Term weighting approaches in automatic text retrieval. Technical report, Ithaca, NY, USA (1987)
20. Schutze, H., Silverstein, C.: A comparison of projections for efficient document clustering. In: Prooceedings of ACM SIGIR, Philadelphia, PA (1997) 74–81

P2P Query Reformulation over Both-As-View Data Transformation Rules

Peter McBrien[1] and Alexandra Poulovassilis[2]

[1] Department of Computing, Imperial College London
pjm@doc.ic.ac.uk
[2] School of Computer Science and Information Systems, Birkbeck College, Univ. of London
ap@dcs.bbk.ac.uk

Abstract. The both-as-view (BAV) approach to data integration has the advantage of specifying mappings between schemas in a bidirectional manner, so that once a BAV mapping has been established between two schemas, queries may be exchanged in either direction between the schemas. In this paper we discuss the reformulation of queries over BAV transformation pathways, and demonstrate the use of this reformulation in two modes of query processing. In the first mode, public schemas are shared between peers and queries posed on the public schema can be reformulated into queries over any data sources that have been mapped to the public schema. In the second, queries are posed on the schema of a data source, and are reformulated into queries on another data source via any public schema to which both data sources have been mapped.

1 Introduction

In [1] we presented the **both-as-view** (**BAV**) approach to data integration, and compared it with **global-as-view** (**GAV**) and **local-as-view** (**LAV**) [2]. In BAV, schemas are mapped to each other using a sequence of schema transformations which we term a transformation **pathway**. These pathways are reversible, in that a pathway $S_x \rightarrow S_y$ from a schema S_x to a schema S_y may be used to automatically derive the pathway $S_y \rightarrow S_x$. Also, from BAV pathways it is possible to extract GAV, LAV and GLAV mapping rules [3]. The BAV approach has been implemented as part of the AutoMed data integration system (see `http://www.doc.ic.ac.uk/automed`).

One advantage of BAV is that it readily supports the evolution of global and local schemas, including the addition or removal of local schemas. An evolution of a schema S_x to S'_x is expressed as a pathway $S_x \rightarrow S'_x$, and then pathways of the form $S_x \rightarrow S_y$ may be 'redirected' to S'_x by prefixing the reverse of $S_x \rightarrow S'_x$ to derive a pathway $S'_x \rightarrow S_x \rightarrow S_y$. As we discussed in [4], this feature makes BAV well-suited to the needs of **peer-to-peer** (**P2P**) data integration, where peers may join or leave the network at any time, or may change their schemas or pathways between schemas.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 310–322, 2007.

Standard centralised data integration of data sources $S_1, S_2, \ldots$ into a global schema S_p is specified by a set of pathways $S_1 \rightarrow S_p$, $S_2 \rightarrow S_p, \ldots$ managed centrally by the data integration system. In the AutoMed P2P data integration system, each peer P_x manages the integration of a data source S_x as a pathway $S_x \rightarrow S_p$, and there is a directory service and P2P protocol that allows the peers to interact. The shared global schema is called a **public schema**, emphasising that no single peer controls the global schema but, by contrast, it is simply a publicly available schema definition that any peers may use. Note that the *same* BAV pathway specification is used to map $S_x \rightarrow S_p$ in both the centralised and the P2P systems. The directory service allows a peer to discover what public schemas S_p exist, and which peers support pathways to that public schema [5].

One contribution of this paper is that we specify how, given a pathway $S_x \rightarrow S_y$ and a query q posed on S_y, q can be reformulated using a combination of LAV and GAV techniques into a query q' posed on S_x. This is an advance on our previous work which only showed how GAV or LAV views individually could be derived from BAV pathways. A second contribution of this paper is that the P2P protocol combined with the reversibility of BAV pathways allows us to support two types of query processing:

- In **public schema querying** we simulate centralised data integration within a P2P environment: a user at a peer P_x poses a query on a public schema S_p, and P_x asks each other peer P_y supporting S_p to either (1) process the query and return the result back to P_x, or (2) send its pathway to S_p to P_x so that P_x can construct the centralised data integration model and process the query itself.
- In **data source querying** a user at a peer P_x poses a query q on data source S_x and wishes it to be reformulated into a query q' on some other data source S_y. This is achieved by using the pathway $S_x \rightarrow S_p$ to reformulate q into a query on S_p. Then P_x is able to interact with other peers supporting the public schema S_p, using the public schema querying techniques already described.

Previous work on P2P data integration in the Piazza system has used combinations of LAV and GAV rules between schemas, and a combination of GAV and LAV query processing techniques [6]. Piazza differs from our approach in that mappings must be specified directly between peers. Whilst our approach does not preclude this, we also allow mappings to be specified to a public schema, making our approach more scalable.

Other related work is [7] which uses a superpeer based network topology to provide better scalability than pure peer-to-peer networks. Routing indexes at superpeers store information about the data reachable from the peers directly connected to them, and aid in the forwarding of query requests only to relevant peers.

The need for a superpeer is avoided in the local relational model [8], where peers are directly related by a combination of a domain relation that specifies how the data types of the peers are related, together with coordination formulae that specify that if one predicate is true in one peer, then another predicate is true in another peer.

Our approach combines the respective advantages of these systems by having virtual public schemas — allowing peers to reuse the existing integration of other peers with public schemas — but having no physical superpeer nodes that may act as a bottleneck in the system — in particular, any peer can combine the integrations of other peers with public schemas in order to form direct pathways between peers for query and update processing.

In [9] global-local-as-view (GLAV) rules [10] are used to specify the constructs of each schema in terms of the constructs of some set of other peer schemas. There is no distinction between source and global schemas, and any number of GLAV rules may be specified between schemas. However, unlike BAV, [9] does not differentiate between sound, complete and exact rules, as the GLAV rules are always sound. CoDB [11] generalises this to allow sound and complete GLAV rules to be specified.

The remainder of the paper begins with a review of the BAV data integration approach in Section 2 together with details of a data integration example. We then describe in Section 3 the process of query reformulation over BAV pathways, and illustrate how it supports public schema querying. In Section 4 we discuss how to improve support for data source schema querying, where a certain degree of pathway repair may be needed in order to fully support data source schema querying.

2 Overview of BAV Data Integration

The basis of the BAV approach to data integration is a low-level **hypergraph-based data model (HDM)**. Higher-level modelling languages are specified in terms of this lower-level HDM. An HDM schema consists of a set of nodes, edges and constraints, and each modelling construct of a higher-level modelling language is specified as some combination of HDM nodes, edges and constraints. For each type of modelling construct of a modelling language (*e.g.* Table, Column, Primary Key and Foreign Key in the relational model) there are available a set of primitive schema transformations for adding such a construct to a schema, removing such a construct from a schema and, in the case of constructs with textual names, renaming such a construct. Schemas are incrementally transformed by applying to them a sequence of primitive schema transformations, each primitive transformation adding, deleting or renaming just one schema construct.

In general, schema constructs may be extensional *i.e.* have a data extent associated with them (*e.g.* Table and Column in the relational model) or may be constraints (*e.g.* Primary Key and Foreign Key in the relational model). In this paper we will restrict our discussion to the relational model, and hence extensional schema constructs consist of sets of values. The general form of a primitive transformation that adds an extensional construct c of type T to a schema S in order to generate new schema S' is $\mathsf{add}T(c, q_S)$, where q_S is a query over S specifying the extent of c in terms of the existing constructs of S. The semantics of this transformation are that $\forall x \,.\, x \in c \leftrightarrow x \in q_S$. In the AutoMed implementation of BAV, q_S is expressed in a functional **intermediate query language** (**IQL**) (see Section 2.1).

When it is not possible to specify the exact extent of the new construct c being added in terms of the existing schema constructs, the primitive transformation extend$T(c, \text{Range } q_l\ q_u)$ must be used instead of add. This adds a new construct c of type T to a schema S, generating a new schema S'. The query q_l over S states what is the minimum extent of c in S'; q_l may be the constant Void if no lower bound on the extent can be specified. The query q_u over S states what is the maximal extent of c in S', and may be the constant Any if no upper bound on the extent can be specified. For non-Void q_l therefore, $\forall x \,.\, x \in c \leftarrow x \in q_l$; and for non-Any q_u, $\forall x \,.\, x \in c \rightarrow x \in q_u$. Also, add$T(c, q_S)$ is equivalent to extend$T$ $(c, \text{Range } q_S\ q_S)$.

In a similar fashion, the transformation delete$T(c, q_s)$ when applied to schema S' generates a new schema S with the construct c of type T removed. The extent of c may be recovered using the query q_S on S, and $\forall x \,.\, x \in c \leftrightarrow x \in q_S$. Note therefore that from a transformation delete$T(c,q_S)$ used to transform schema S' to schema S we can automatically infer that add$T(c,q_S)$ transforms S to S', and vice versa. When it is not possible to specify the exact extent of the construct c being deleted from S' in terms of the remaining schema constructs, the transformation contract$T(c, \text{Range } q_l\ q_u)$ must be used instead of delete. This removes a construct c of type T from schema S' to form a new schema S. The query q_l over S states what is the minimum extent of c in S', while the query q_u over S states what is the maximal extent of c in S'. Again, q_1 may be Void and q_u may be Any. delete$T(c, q_S)$ is equivalent to contract$T(c, \text{Range } q_S\ q_S)$. Also, from contract$T(c, \text{Range } q_l\ q_u)$ used to transform schema S' to schema S we can infer that extend$T(c, \text{Range } q_l\ q_u)$ transforms S to S', and vice versa.

Finally, the transformation rename$T(c, c')$ causes a construct c of type T in a schema S to be renamed to c' in a new schema S', where $\forall x . x \in c \leftrightarrow x \in c'$. Thus, from rename$T(c, c')$ used to transform S to S' we can infer that rename$T(c', c)$ transforms S' to S.

2.1 AutoMed's IQL Query Language

IQL is a comprehensions-based functional query language [12], and such languages subsume query languages such as SQL-92 and OQL in expressiveness [13]. It supports strings e.g. 'Computer Science', booleans True and False, real numbers, integers, tuples e.g. {1,2,3}, and sets, bags and lists. There are several polymorphic primitive operators for manipulating sets, bags and lists. The operator ++ concatenates two lists, and performs bag union and set union on bags and sets, respectively. The operator flatmap applies a collection-valued function f to each element of a collection and applies ++ to the resulting collections. For sets, it is defined recursively as follows, where [] denotes the empty set and (SCons x xs) denotes a set containing an element x with xs being the rest of the set (which may be empty):

flatmap f [] = []
flatmap f (SCons x xs) =(f x) ++(flatmap f xs)

Henceforth in this paper, we confine our discussion to collections that are sets.

The operator flatmap can be used to specify **comprehensions** over sets. These are of the form $[h \mid q_1; \ldots; q_n]$ where h is an expression termed the **head** and $q_1, \ldots, q_n$ are **qualifiers**, with $n \geq 0$. Each qualifier is either a **filter** or a **generator**. A generator has syntax $p <- e$ where e is a set-valued expression and p is a **pattern** i.e. an expression involving variables and tuple constructors only. The variables of p are successively bound by iterating through e. Any variables appearing in the head, h, inherit these bindings. A filter is a boolean-valued expression, which must be satisfied by the values generated by the generators in order for these values to contribute to the final result of the comprehension. Comprehensions are a convenient high-level syntax and add no extra expressiveness to languages such as IQL since they translate into applications of flatmap. We give the translation below for a set comprehension, where Q denotes a sequence of qualifiers and [h] a set comprising a single element h:

$$
\begin{array}{ll}
[\mathsf{h} \mid \mathsf{p} <- \mathsf{e};\ \mathsf{Q}] & \equiv \mathsf{flatmap}\,(\mathsf{lambda\ p}.[\mathsf{h} \mid \mathsf{Q}])\ \mathsf{e} \\
[\mathsf{h} \mid \mathsf{e};\ \mathsf{Q}] & \equiv \mathsf{if\ e} = \mathsf{True\ then}\ [\mathsf{h} \mid \mathsf{Q}]\ \mathsf{else}\ [] \\
[\mathsf{h} \mid] & \equiv [\mathsf{h}]
\end{array}
$$

IQL supports unification of variables appearing in the patterns of generators within the same comprehension. For example, $[\{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e}\} \mid \{\mathsf{a},\mathsf{b},\mathsf{c}\} <- \mathsf{r}; \{\mathsf{d},\mathsf{c},\mathsf{e}\} <- \mathsf{s}]$ is equivalent to $[\{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e}\} \mid \{\mathsf{a},\mathsf{b},\mathsf{c}\} <- \mathsf{r}; \{\mathsf{d},\mathsf{c2},\mathsf{e}\} <- \mathsf{s}; \mathsf{c} = \mathsf{c2}]$

Several equivalences hold for these IQL operators [14], which follow from their definition and from the interpretation assigned to the Void and Any constants.

2.2 An Example

Figure 1 shows four schemas S_1, S_2, S_3, S_p. S_1, S_2, S_3 are data source schemas while S_p is what in a centralised data integration system would be called a **global schema** and in our P2P system is called a **public schema**. The semantics of the application domain are that a student with name sname may repeatedly sit the exam for a course (identified by ccode, and each having a title) over any number of semesters, and achieve an exam mark on each exam sitting. However, for all attempts of the course, the student will have the same tutor (tutors having been introduced at the start of 1994, along with a coursework mark cwmark that students can attempt only once per course). Each student studies for one degree. Each degree is identified by a dcode, has a title dname and has an associated qualification.

Schema S_p is a virtual schema modelling the application domain, omitting the information about tutors and about the qualification associated with degrees. The cwmark is shown as optional (by a '?' suffix) since it was only awarded from 1994 onwards. Schema S_1 represents a data source that holds information about courses with a ccode greater or equal to 500, and holds data in first normal form (since dname is dependent on just sname and title is dependent on just ccode). Schema S_2 represents a data source that holds information about courses with a ccode less than 500, and is also in first normal form, since it holds in tname the tutor's name (an optional attribute), which is dependent on just sname and ccode. Schema S_3 represents a data source that details students' tutors, the

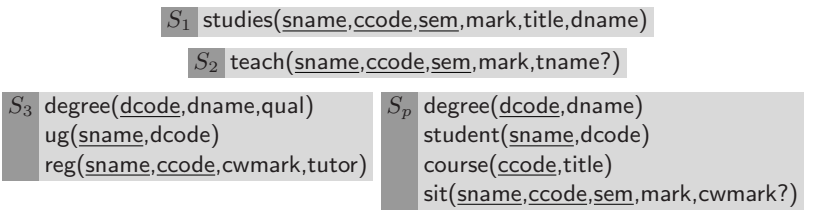


Fig. 1. Three data sources S_1, S_2, S_3, and a public schema S_p

degrees students studied, and the coursework mark students gained for courses, and is held in third normal formal.

We consider below fragments of the pathways $S_1 \rightarrow S_p$ and $S_2 \rightarrow S_p$ in order to illustrate the BAV approach and the use of IQL queries within transformations. Within $S_1 \rightarrow S_p$ it is necessary to decompose the studies table in S_1 in order to produce the separate course table that is present in S_p. Here is the fragment of that pathway:

① extendTable(⟨⟨course⟩⟩, Range ([{c} | {s, c, t} <− ⟨⟨studies⟩⟩]) Any)
② extendColumn(⟨⟨course,ccode⟩⟩, Range [{c, c} | {c} <− ⟨⟨course⟩⟩] Any)
③ extendColumn(⟨⟨course,title⟩⟩, Range ([{c, ti} | {{s, c, t}, ti} <− ⟨⟨studies,title⟩⟩]) Any)
④ contractColumn(⟨⟨studies,title⟩⟩,
Range Void [{{s, c, t}, ti} | {s, c, t} <− ⟨⟨studies⟩⟩; {c, ti} <− ⟨⟨course,title⟩⟩])

Transformation ① states that the course table in S_p contains as its set of keys at least those ccode attributes of studies in S_1 (the first argument of the Range constructor). We note here that the AutoMed representation of a relational table models the table itself by its set of primary key values, and models each attribute a of the table by the projection of the table onto the primary key attributes plus a (see [1]).

Transformations ② and ③ add the ccode and title columns to course. Again these are extend transformations with upper bound Any. The final transformation ④ removes the title attribute of the studies table and specifies the upper bound that the title attribute in S_p places on the extent of the title attribute in S_1.

The pathway $S_2 \rightarrow S_p$ needs to specify that the tutor tname has no representation in S_p, using transformation ⑤ below. The remainder of the pathway is not required for the examples that follow, and is therefore omitted from our discussion.

⑤ contractColumn(⟨⟨teach,tname⟩⟩, Range Void Any)

3 Query Reformulation over BAV Pathways

In this section, we discuss how query reformulation can be undertaken over BAV pathways. We first illustrate how BAV pathways can be used for GAV and LAV query reformulation, and hence can support GAV and LAV query processing. We then present a BAV-specific query reformulation algorithm which subsumes as special cases GAV and LAV query reformulation.

GAV query reformulation is based on query unfolding. For example, to evaluate a query q on S_p with respect to S_1, we traverse the pathway $S_p \rightarrow S_1$ (i.e. the *reverse* of the pathway $S_1 \rightarrow S_p$ described earlier) replacing each scheme in q that appears in an delete or contract transformation with the corresponding query of that transformation.

Example Query 1: To reformulate the query

$q_1 = [\{\mathsf{ti}\} \mid \{\mathsf{c}, \mathsf{ti}\} <- \langle\langle \mathsf{course}, \mathsf{title} \rangle\rangle; \mathsf{c} = 500]$

first ④ is ignored (since its reverse is an extend transformation), and then ③ unfolds ⟨⟨course,title⟩⟩ giving:

$[\{\mathsf{ti}\} \mid \{\mathsf{c}, \mathsf{ti}\} <- \mathsf{Range}([\{\mathsf{c}, \mathsf{ti}\} \mid \{\mathsf{s}, \mathsf{c}, \mathsf{t}, \mathsf{ti}\} <- \langle\langle \mathsf{studies}, \mathsf{title} \rangle\rangle]) \ \mathsf{Any}; \mathsf{c} = 500]$

Using equivalence rules for IQL [14], this simplifies to

$\mathsf{Range}[\{\mathsf{ti}\} \mid \{\mathsf{c}, \mathsf{ti}\} <- [\{\mathsf{c}, \mathsf{ti}\} \mid \{\mathsf{s}, \mathsf{c}, \mathsf{t}, \mathsf{ti}\} <- \langle\langle \mathsf{studies}, \mathsf{title} \rangle\rangle]; \mathsf{c} = 500] \ \mathsf{Any}$

and then to:

$\mathsf{Range}[\{\mathsf{ti}\} \mid \{\mathsf{s}, \mathsf{c}, \mathsf{t}, \mathsf{ti}\} <- \langle\langle \mathsf{studies}, \mathsf{title} \rangle\rangle; \mathsf{c} = 500] \ \mathsf{Any}$

Transformations ② and ① have no further effect on this query, and thus this is the transformed query that can execute on data source S_1.

As another example, consider table reg in S_3 that has sname and ccode as its key attributes. In the pathway $S_3 \rightarrow S_p$, reg is mapped to table sit of S_p that has sname, ccode and sem as its key attributes since students may (re)sit the examination part of any course once in any semester. Recall that the tutors for courses were only introduced from sem 1 of 1994. Below is the relevant fragment of the pathway $S_3 \rightarrow S_p$. We note that transformation ⑥ contains the expression Const1 s c in the head of the comprehension. Here, Const1 is an IQL **constructor** (Skolem function), used because it is not possible to derive the sem attribute of ⟨⟨sit⟩⟩ from ⟨⟨reg⟩⟩.

⑥ extendTable(⟨⟨sit⟩⟩,
  Range [{s, c, Const1 s c} | {s, c} <− ⟨⟨reg⟩⟩] Any)
⑦ extendColumn(⟨⟨sit,sname⟩⟩, Range [{{s, c, t}, s} | {s, c, t} <− ⟨⟨sit⟩⟩] Any)
⑧ extendColumn(⟨⟨sit,ccode⟩⟩, Range [{{s, c, t}, c} | {s, c, t} <− ⟨⟨sit⟩⟩] Any)
⑨ addColumn(⟨⟨sit,cwmark⟩⟩,
  [{{s, c, t}, cw} | {s, c, t} <− ⟨⟨sit⟩⟩; {{s, c}, cw} <− ⟨⟨reg,cwmark⟩⟩])
⑩ extendColumn(⟨⟨sit,sem⟩⟩, Range [{{s, c, t}, t} | {s, c, t} <− ⟨⟨sit⟩⟩] Any)
⑪ deleteColumn(⟨⟨reg,sname⟩⟩, [{{s, c}, s} | {s, c} <− ⟨⟨reg⟩⟩])
⑫ deleteColumn(⟨⟨reg,ccode⟩⟩, [{{s, c}, c} | {s, c} <− ⟨⟨reg⟩⟩])
⑬ deleteColumn(⟨⟨reg,cwmark⟩⟩, [{{s, c}, cw} | {{s, c, t}, cw} <− ⟨⟨sit,cwmark⟩⟩])
⑭ contractTable(⟨⟨reg⟩⟩, Range Void [{s, c} | {s, c, t} <− ⟨⟨sit⟩⟩; t >= '1994-1'])

There are a family of constructors Const1, Const2, ... Any expression of the form Const$i$ $e_1 \ldots e_n$ is only comparable with an expression constructed using the same constructor i.e. with an expression of the form Const$i$ $e'_1 \ldots e'_n$. Thus, an expression of the form Const$i$ $e_1 \ \ldots \ e_n \ =$ Const$i$ $e'_1 \ \ldots \ e'_n$ evaluates to True if $e_j = e'_j$ evaluates to True for all j otherwise it evaluates to False, and similarly for the other comparison operators. Any other kind of comparison of Const$i$ returns the value Null, denoting "unknown". If Null is the value of a filter in a comprehension, then the result will be a Range expression i.e. the second rule of comprehension translation in Section 2.1 becomes:

$[\mathsf{h} \mid \mathsf{e}; \mathsf{Q}] \equiv \mathsf{if}\ \mathsf{e} = \mathsf{True}\ \mathsf{then}\ [\mathsf{h} \mid \mathsf{Q}]\ \mathsf{elseif}\ \mathsf{e} = \mathsf{False}\ \mathsf{then}\ [] \ \mathsf{else}\ (\mathsf{Range}\ \mathsf{Void}\ [\mathsf{h} \mid \mathsf{Q}]\ )$

Example Query 2: Consider the following query posed on S_p:

$q_2 = [\{s, c, cw\} \mid \{\{s, c, t\}, cw\} <- \langle\langle sit,cwmark\rangle\rangle; t >= \text{'1997-1'}]$

Unfolding $\langle\langle sit,cwmark\rangle\rangle$ using ⑨ we obtain:

$[\{s, c, cw\} \mid \{s, c, t, cw\} <- [\{s, c, t, cw\} \mid \{s, c, t\} <- \langle\langle sit\rangle\rangle;$
$\{\{s, c\}, cw\} <- \langle\langle reg, cwmark\rangle\rangle]; t >= \text{'1997-1'}]$

which using IQL equivalences [14] simplifies to

$[\{s, c, cw\} \mid \{s, c, t\} <- \langle\langle sit\rangle\rangle; \{\{s, c\}, cw\} <- \langle\langle reg, cwmark\rangle\rangle; t >= \text{'1997-1'}]$

Unfolding $\langle\langle sit\rangle\rangle$ using ⑥ we obtain:

$[\{s, c, cw\} \mid \{s, c, t\} <- Range[\{s, c, Const1\ s\ c\} \mid \{s, c\} <- \langle\langle reg\rangle\rangle]\ Any;$
$\{\{s, c\}, cw\} <- \langle\langle reg, cwmark\rangle\rangle; t >= \text{'1997-1'}]$

Using IQL equivalences [14] this simplifies to

$Range[\{s, c, cw\} \mid \{s, c, t\} <- [\{s, c, Const1\ s\ c\} \mid \{s, c\} <- \langle\langle reg\rangle\rangle];$
$\{\{s, c\}, cw\} <- \langle\langle reg, cwmark\rangle\rangle; t >= \text{'1997-1'}]\ Any$

Swapping the last two qualifiers of the outer comprehension, and moving t >= '1997-1' into the inner comprehension gives:

$Range[\{s, c, cw\} \mid \{s, c, t\} <- [\{s, c, Const1\ s\ c\} \mid$
$\{s, c\} <- \langle\langle reg\rangle\rangle; (Const1\ s\ c) >= \text{'1997-1'}];$
$\{\{s, c\}, cw\} <- \langle\langle reg, cwmark\rangle\rangle]\ Any$

At run time this gives the same result as the following query, since Const1 s c >= '1997-1' evaluates to Null:

$Range\ Void\ [\{s, c, cw\} \mid \{s, c, t\} <- [\{s, c, Const1\ s\ c\} \mid$
$\{s, c\} <- \langle\langle reg\rangle\rangle]; \{\{s, c\}, cw\} <- \langle\langle reg, cwmark\rangle\rangle]$

i.e. it returns as an upper bound the student names, courses they have taken and coursework marks obtained from S_3.

Consider now the $\langle\langle studies,dname\rangle\rangle$ attribute of S_1, which corresponds in S_p to some instances of the join between $\langle\langle student,dcode\rangle\rangle$ and $\langle\langle degree,dname\rangle\rangle$. This is expressed in BAV by the following fragment of the pathway $S_1 \rightarrow S_p$:

⑮ $extendTable(\langle\langle student\rangle\rangle, Range\ [\{s\} \mid \{s, c, t\} <- \langle\langle studies\rangle\rangle]\ Any)$
⑯ $addColumn(\langle\langle student,sname\rangle\rangle, [\{s, s\} \mid \{s\} <- \langle\langle student\rangle\rangle])$
⑰ $extendColumn(\langle\langle student,dcode\rangle\rangle, Range\ Void\ Any)$
⑱ $extendTable(\langle\langle degree\rangle\rangle, Range\ [\{d\} \mid \{s, d\} <- \langle\langle student,dcode\rangle\rangle]\ Any)$
⑲ $addColumn(\langle\langle degree,dcode\rangle\rangle, [\{d, d\} \mid \{d\} <- \langle\langle degree\rangle\rangle])$
⑳ $extendColumn(\langle\langle degree,dname\rangle\rangle, Range\ [\{d, dn\} \mid \{s, d\} <- \langle\langle student,dcode\rangle\rangle;$
$\{\{s, c, t\}, dn\} <- \langle\langle studies,dname\rangle\rangle]\ Any)$
㉑ $contractColumn(\langle\langle studies,dname\rangle\rangle, Range\ Void[\{\{s, c, t\}, dn\} \mid \{s, c, t\} <- \langle\langle sit\rangle\rangle;$
$\{s, d\} <- \langle\langle student,dcode\rangle\rangle; \{d, dn\} <- \langle\langle degree,dname\rangle\rangle])$

Example Query 3: Consider the following query on S_p:

$q_3 = [\{s\} \mid \{s, d\} <- \langle\langle student,dcode\rangle\rangle; \{d, dn\} <- \langle\langle degree,dname\rangle\rangle;$
$dn = \text{'CS'}]$

Using GAV, $\langle\langle degree,dname\rangle\rangle$ would unfold using ⑳ and $\langle\langle student,dcode\rangle\rangle$ would then unfold using ⑰, obtaining:

$[\{s\} \mid \{s, d\} <- Range\ Void\ Any;$
$\{d, dn\} <- Range[\{d, dn\} \mid \{s, d\} <- Range\ Void\ Any;$
$\{\{s, c, t\}, dn\} <- \langle\langle studies, dname\rangle\rangle]\ Any; dn = \text{'CS'}]$

which simplifies to just Range Void Any, i.e. giving no answers.

However the query q_3 on S_p can yield answers using **LAV query processing**. There are two main techniques for this, the **inverse rule** algorithm [15,16] and the **bucket** algorithm [17]. For simplicity we focus here on the former. Using the inverse rule approach, the definition of a construct c by a query of the form $[h \mid Q]$ is inverted in a two-step process. First, replace each variable in Q that does not appear in h by a distinct Const$i$ with arguments the variable(s) in h. For example, (15) has two such variables, c and t which are replaced by Const2 s and Const3 s respectively; while in (21), there is one such variable d, which is replaced by Const8 s dn (see below). Next, for each generator $p<-cs$ in Q, generate a query defining cs in terms of $[p \mid h <- c; Q']$ where Q' consists of all the filters from Q. To illustrate, we list below all the inverse rules derived from the fragment (15)–(21) of the BAV pathway $S_1 \rightarrow S_p$.

(15.1) ⟨⟨studies⟩⟩ =Range Void [{s, Const2 s, Const3 s} | {s} <− ⟨⟨student⟩⟩]
(16.1) ⟨⟨student⟩⟩ = [{s} | {s, s} <− ⟨⟨student,sname⟩⟩]
(18.1) ⟨⟨student,dcode⟩⟩ =Range Void [{Const4 d, d} | {d, d} <− ⟨⟨degree,dcode⟩⟩]
(19.1) ⟨⟨degree⟩⟩ = [{d} | {d, d} <− ⟨⟨degree,dcode⟩⟩]
(20.1) ⟨⟨student,dcode⟩⟩ =Range Void [{Const5 d dn, d} | {d, dn} <− ⟨⟨degree,dname⟩⟩]
(20.2) ⟨⟨studies,dname⟩⟩ =Range Void [{{Const5 d dn, Const6 d dn, Const7 d dn}, dn} |
{d, dn} <− ⟨⟨degree,dname⟩⟩]
(21.1) ⟨⟨student,dcode⟩⟩ =Range [{s, Const8 s c t dn} | {{s, c, t}, dn}<−
⟨⟨studies,dname⟩⟩] Any
(21.2) ⟨⟨degree,dname⟩⟩ =Range [{Const8 s c t dn, dn} | {{s, c, t}, dn}<−
⟨⟨studies,dname⟩⟩] Any
(21.3) ⟨⟨sit⟩⟩ =Range [{s, c, t} | {{s, c, t}, dn} <− ⟨⟨studies,dname⟩⟩] Any

Query processing that requires to use a particular construct can now combine the direct definition of the construct within the BAV pathway with all the inverse rules for that construct derived from the BAV pathway. These definitions can be combined using a merge function defined as follows, where union and intersect are set union and set intersection:

merge (Range e1 e2) (Range e1′ e2′) =Range (union e1 e1′) (intersect e2 e2′)

Returning to our example, when a query is submitted to S_p and answers are required from S_1, the rules (15),(16), (17),(18),(19),(20),(21.1),(21.2), (21.3), can be used. In particular, for processing query q_3 above, we have:

⟨⟨student,dcode⟩⟩ =merge (17) (21.1) = (21.1) and
⟨⟨degree,dname⟩⟩ =merge (20) (21.2) = (21.2)

Substitution now for ⟨⟨student,dcode⟩⟩ and ⟨⟨degree,dname⟩⟩ in q_3 gives:

[{s} | {s, d} <− Range[{s, Const8 s c t dn} |
{{s, c, t}, dn} <− ⟨⟨studies, dname⟩⟩] Any;
{d, dn} <− Range[{Const8 s c t dn, dn} | {{s, c, t}, dn} <−
⟨⟨studies, dname⟩⟩] Any; dn = 'CS']

which simplifies to:

Range[{s} | {s, d} <− [{s, Const8 s c t dn} | {{s, c, t}, dn} <− ⟨⟨studies, dname⟩⟩];
{d, dn} <− [{Const8 s c t dn, dn} | {{s, c, t}, dn} <− ⟨⟨studies, dname⟩⟩];
dn = 'CS'] Any

which when evaluated would give the same set of answers as:

Range[{s} | {{s, c, t}, dn} <− ⟨⟨studies, dname⟩⟩; dn = 'CS'] Any

3.1 BAV Query Reformulation

Following the examples presented above, we now summarise how combined GAV and LAV query reformulation can be carried out over a BAV pathway $S_x \rightarrow S_y$, with the aim of obtaining the maximal information that would be derivable from the BAV pathway by means of GAV and LAV query processing techniques.

Suppose we wish to reformulate a query q posed on S_x to be posed with respect to S_y. (We note that, due to the reversibility of BAV pathways, from a pathway $S_x \rightarrow S_y$ it is also possible to reformulate a query q posed on S_y to be posed with respect to S_x. The process is exactly as described below except that now it is with respect to the, automatically derivable, *reverse* pathway $S_y \rightarrow S_x$. This was the scenario illustrated in the examples above, where pathways $S_x \rightarrow S_p$ were used to reformulate queries on S_p so that they could be evaluated on S_x.)

The first step is to construct a set of view definitions, $\mathcal{V}$, defining constructs in S_x in terms of constructs in S_y. This is undertaken by traversing the pathway $S_x \rightarrow S_y$, and at each transformation step t taking one of the following actions:

- if t is of the form $\mathsf{rename}(c, c')$ the rule $c = c'$ is added to $\mathcal{V}$;
- if t is of the form $\mathsf{delete}(c, q)$ or $\mathsf{contract}(c, q)$, the rule $c = q$ is added to $\mathcal{V}$;
- if t is of the form $\mathsf{add}(c, q)$, where q is a comprehension referencing schema constructs $c_1, \ldots, c_n$ in its generators, then invert the rule $c = q$ (as described above) to obtain a set of rules of the form $c_i = q_i$ for $1 \leq i \leq n$ such that the only scheme referenced in each q_i is c; add these rules to $\mathcal{V}$;
- if t is of the form $\mathsf{extend}(c, \mathsf{Range\,Void}\ q_u)$, where q_u is a comprehension as in the case of $\mathsf{add}(c, q)$, then invert the rule $c = \mathsf{Range\,Void}\ q_u$ to obtain a set of rules of the form $c_i = \mathsf{Range}\ q_i\ \mathsf{Any}$; add these rules to $\mathcal{V}$;
- if t is of the form $\mathsf{extend}(c, \mathsf{Range}\ q_l\ \mathsf{Any})$, where q_l is a comprehension as in the case of $\mathsf{add}(c, q)$, then invert the rule $c = \mathsf{Range}\ q_l\ \mathsf{Any}$ to obtain a set of rules of the form $c_j = \mathsf{Range\,Void}\ q_j$; add these rules to $\mathcal{V}$;
- if t is of the form $\mathsf{extend}(c, \mathsf{Range}\ q_l\ q_u)$, where q_l and q_u are comprehensions as in the case of $\mathsf{add}(c, q)$, then invert the rule $c = \mathsf{Range}\ q_l\ q_u$ by inverting separately q_u and q_l, as in the previous two cases, to obtain from q_u a set of rules of the form $c_i = \mathsf{Range}\ q_i\ \mathsf{Any}$ and from q_l a set of rules of the form $c_j = \mathsf{Range\,Void}\ q_j$; add these rules to $\mathcal{V}$;

We note that the worst-case complexity of constructing $\mathcal{V}$ is $O(N \times M)$ where N is the number of primitive transformations in the pathway and M is the maximum number of schema constructs appearing in comprehension expressions.

Once constructed, $\mathcal{V}$ can be used to reformulate a query q posed on S_x with respect to S_y. We term a schema construct c which appears in S_y **final** otherwise it is **non-final**. The query reformulation algorithm is as follows, where the function $NF(q)$ returns the set of non-final schemes occurring in an IQL query q:

while $NF(q) \neq \emptyset$
  for each $c \in NF(q)$
    $e :=$ Range Void Any
    for each rule $r \in \mathcal{V}$ *such that* $head(r) = c$

$$e := \mathsf{merge}\ e\ body(r)$$
$$q := [c/e]q$$

In other words, non-final constructs in q are successively replaced by their definition in $\mathcal{V}$ until there are no non-final constructs left. It is easy to see that this process terminates: Let $\mathcal{G}$ be the graph obtained from $\mathcal{V}$ by creating a node in $\mathcal{G}$ for each schema construct in the head of a rule in $\mathcal{V}$ and an arc $c \rightarrow c'$ in $\mathcal{G}$ if c' appears in a rule defining c. The acyclicity of $\mathcal{G}$ follows from the syntactic properties of BAV transformation sequences: an add or extend transformation can only add a construct that does not exist in the input schema, and the query within the transformation can only refer to constructs existing in the input schema; a delete or contract transformation can only delete a scheme that exists in the input schema and the query within the transformation can only refer to schemes existing in the output schema. By the acyclicity of $\mathcal{G}$ the query reformulation algorithm must terminate. The complexity of the query reformulation algorithm is again $O(N \times M)$.

4 Data Source Schema Query Processing

BAV pathways can in principle be used to map directly between peer schemas in a P2P data integration scenario, and the techniques we have described above can be used to reformulate queries with respect to a BAV pathway between two peer data source schemas. However, in AutoMed we also support P2P BAV data integration via **public schemas**, as already described in the Introduction. A desirable property in data integration is that the mapping between a pair of schemas S_x and S_y should form a **complete mapping**, in the sense that it identifies all possible mappings between schema objects in S_x and S_y. In our P2P framework, we can construct mappings between S_x and S_y by finding some shared or public schema S_z for which we already know the pathways $S_x \rightarrow S_z$ and $S_z \rightarrow S_y$, and form a concatenation of these two pathways to form a pathway $S_x \rightarrow S_y$. However, this pathway may not in general represent a complete mapping, since S_z might not contain a schema object to represent data associated with schema objects that appear in S_x and S_y and for which a mapping could be specified in a *direct* pathway from S_x to S_y. Suppose that SO_x is a schema object in S_x and SO_y is a schema object in S_y for which a mapping between SO_x to SO_y could be established, but that it is currently absent due to the absence of a corresponding schema object in S_z. Then the pathway $S_x \rightarrow S_z$ must contain a transformation of the form
(a) $\mathsf{contractObj_x(SO_x, Range\ Void\ Any)}$
expressing the fact that SO_x cannot be derived or represented in S_z, and similarly $S_z \rightarrow S_y$ must contain a transformation of the form
(b) $\mathsf{extendObj_y(SO_y, Range\ Void\ Any)}$
expressing the fact that SO_y cannot be derived or represented in S_z.

Hence, we can use the presence of pairs of transformations of the form of (a) and (b) to extract pairs of schema objects that might be mappable between S_x

and S_y, and feed such pairs into a **schema matching** process [18] in order to derive any mappings that exist between objects as yet unmapped in S_x and S_y. AutoMed supports a suitable schema matching tool [19], which automatically derives possible matchings between pairs of schema objects, and the transformations representing their mapping; the user is then asked to confirm or manually modify the matchings and generated transformations.

Thus, to construct a complete mapping $S_x \rightarrow S_y$ from two complete mappings $S_x \rightarrow S_z$ and $S_z \rightarrow S_y$, we can: (i) Form the set U_x of schema objects that appear in contract transformations in $S_x \rightarrow S_z$, and the set U_y of schema objects that appear in extend transformations in $S_z \rightarrow S_y$. (ii) Perform a pairwise match of objects in U_x against objects in U_y; for each positive match found, remove the transformation steps that contract/extend the matched pair of objects, and replace with the transformations that represent the match found. To illustrate, we return to our running example. Within the pathway $S_3 \rightarrow S_p$ there are two transformations:

㉒ contractColumn(⟨⟨degree,qual⟩⟩, Range Void Any)
㉓ contractColumn(⟨⟨reg,tutor⟩⟩, Range Void Any)

When deriving the pathway $S_2 \rightarrow S_3$ from $S_2 \rightarrow S_p$ (which will include transformation ⑤) and the reverse of $S_3 \rightarrow S_p$, a schema match table as follows is first formed (the filled in circles indicate that the reverse of a transformation is being used):

Data Source S_2		Data Source S_3	
Transformation	Schema Object	Transformation	Schema Object
⑤	⟨⟨teach,tname⟩⟩	㉒	⟨⟨degree,qual⟩⟩
		㉓	⟨⟨reg,tutor⟩⟩

The schema matching process should then discover that ⟨⟨teach,tname⟩⟩ and ⟨⟨reg,tutor⟩⟩ match (specifically, that they are equivalent, with the exception of the key used). Hence transformations ⑤ and ㉓ can be removed and the following transformations added to the end of $S_2 \rightarrow S_3$:

㉔ addColumn(⟨⟨reg,tutor⟩⟩, [{{s, c}, tu} | {{s, c, t}, tu} <− ⟨⟨teach,tname⟩⟩])
㉕ deleteColumn(⟨⟨teach,tname⟩⟩, [{{s, c, Const1 s c}, tu} | {{s, c}, tu} <− ⟨⟨reg⟩⟩])

5 Concluding Remarks

The BAV approach has the advantage in a P2P data integration setting of allowing bidirectional logical mappings to be specified between peers. We have discussed how these mappings can be used to support two types of query processing in a P2P data integration system, where either queries are posed on the schema of a data source at a peer or on a virtual public schema. We have shown how GAV and LAV query reformulation can be combined over BAV pathways — specifically, for a comprehensions-based query language — thus obtaining the maximal information from BAV pathways that would be derivable by means of GAV and LAV query processing techniques.

We have focused here on query processing along a single BAV pathway, which cannot generate cyclic relationships between schema objects and hence for which

query answering is decidable c.f. [20]. The extension of P2P query processing along a network of arbitrary BAV pathways is an area of ongoing work, and in particular we wish to investigate the applicability of the epistemic semantics approach of [9,21] to BAV.

References

1. McBrien, P., Poulovassilis, A.: Data integration by bi-directional schema transformation rules. In: Proc. ICDE'03, IEEE (2003) 227–238
2. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. PODS'02, ACM (2002) 233–246
3. Jasper, E., Tong, N., McBrien, P., Poulovassilis, A.: View generation and optimisation in the AutoMed data integration framework. In: Proc. Baltic DB&IS04. Volume 672 of Scientific Papers., Univ. Latvia (2004) 13–30
4. McBrien, P., Poulovassilis, A.: Defining peer-to-peer data integration using both as view rules. In: Proc. DBISP2P, at VLDB'03. (2003) 91–107
5. Bellahsène, Z., Lanzanitis, C., McBrien, P., Rizopoulos, N.: Querying distributed data in a super-peer based architecture. In: Proc. IWI2006 (in conjunction with WWW06). (2006)
6. Halevy, A.Y., Ives, Z.G., Mork, P., Tatarinov, I.: Piazza: Data management infrastructure for semantic web applications. In: Proc. WWW'03. (2003)
7. Loser, A., Nejdl, W., Wolpers, M., Siberski, W.: Information integration in schema-based peer-to-peer networks. In: Proc. CAiSE'03. LNCS, Springer (2003)
8. Bernstein, P., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: a vision. In: Proc. WebDB'02. (2002) 89–94
9. Calvanese, D., Damagio, E., De Giacomo, G., Lenzerini, M., Rosati, R.: Semantic data integration in P2P systems. In: Proc. DBISP2P, at VLDB'03. (2003)
10. Friedman, M., Levy, A., Millstein, T.: Navigational plans for data integration. In: Proc. 16th National Conference on Artificial Intelligence, AAAI (1999) 67–73
11. Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I.: Queries and updates in the coDB peer-to-peer database system. In: Proc. VLDB. (2004) 1277–1280
12. Jasper, E., Poulovassilis, A., Zamboulis, L.: Processing IQL queries and migrating data in the AutoMed toolkit. Technical Report No. 20, AutoMed (2003)
13. Buneman et al., P.: Comprehension syntax. SIGMOD Record **23**(1) (1994) 87–96
14. McBrien, P., Poulovassilis, A.: P2P query reformulation in AutoMed. Technical Report No. 33, AutoMed (2006)
15. Qian, X.: Query unfolding. In: Proc. ICDE, IEEE (1996) 48–55
16. Duschka, O., Genesereth, M.: Answering recursive queries using views. In: Proc. PODS, ACM (1997) 109–116
17. Levy, A., Rajamaran, A., Ordille, J.: Querying heterogeneous information sources using source description. In: Proc 22nd VLDB. (1996) 252–262
18. Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. VLDB Journal **10** (2001) 334–350
19. Rizopoulos, N.: Automatic discovery of semantic relationships between schema elements. In: Proc. of 6th ICEIS. (2004)
20. Halevy, A.Y., Ives, Z.G., Suciu, D., Tatarinov, I.: Schema mediation in peer data management systems. In: Proc. ICDE'03, IEEE (2003)
21. Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Logical foundations of peer-to-peer data integration. In: Proc. PODS. (2004) 241–251

RDFCube: A P2P-Based Three-Dimensional Index for Structural Joins on Distributed Triple Stores

Akiyoshi Matono, Said Mirza Pahlevi, and Isao Kojima

Grid Technology Research Center,
National Institute of Advanced Industrial Science and Technology(AIST)
{matono,mirza,kojima}@ni.aist.go.jp

Abstract. Today, RDF data/triples are scattered everywhere and their total size is rapidly increasing. Centralized RDF triple stores have limitations on both their failure tolerance and scalability. Therefore, RDF query processing in a P2P environment is an important issue. So far, several conventional P2P-based RDF triple stores have been proposed. They, however, are designed merely for triple retrieval rather than for triple join query processing. Consequently, they suffer from an unnecessary data transfer problem. This paper presents an RDF query processing technique based on a three-dimensional hash index. The triples are mapped into the index; then bit information that represents the presence or absence of triples in the index is introduced. We implemented our approach on the top of an emulated P2P environment. Evaluation results show that our approach can achieve good performance and scalability.

1 Introduction

Today, Resource Description Framework (RDF) [1] is anticipated as a foundation for representing metadata. Metadata formatted according to RDF (RDF data) are rapidly increasing and dispersing: RDF is therefore anticipated to be widely used in many fields. As a consequence, it is essential to provide efficient and scalable RDF query processing in a distributed environment.

An infrastructure for RDF-based metadata using P2P, Edutella [2], uses a Gnutella-like [3] unstructured P2P network. However, Edutella broadcasts RDF queries to the whole network. Furthermore, because each node that receives the queries must process them, an unnecessary burden is placed on the unrelated nodes.

RDFPeers [4, 5] is a distributed RDF repository to efficiently search RDF triples using Multi-Attribute Addressable Network (MAAN) [6], which extends Chord [7] to answer multi-attribute and range queries. It stores a triple by specifying the triple's subject, predicate, or object as a key. Therefore, the storage process requires three lookups. On receiving a query triple, RDFPeers uses one constant in the query triple as a key to identify a node that stores triples related to the query. In other words, RDFPeers can obtain a set of triples that are matched for a constant and bind answers to its variables.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 323–330, 2007.

The RDF is used to describe semantic relationships among scattered data. Therefore, efficient processing is necessary for structural join queries that join data that are located on different sites. Because RDFPeers is designed for RDF triple retrieval and because it is based on a DHT that supports only exact-match queries, it cannot efficiently handle a join operation. To perform a join operation RDFPeers must gather all candidate answers into a node. Consequently, the amount of unnecessary data transferred among nodes increases considerably.

Our aim is to provide an RDF query processing mechanism that can efficiently process structural join queries to deal with this issue. To this end, we propose an indexing scheme for RDFPeers that removes unrelated triples from candidate answers so that the amount of unnecessarily transferred data can be reduced.

Our scheme uses a three-dimensional hash index called *RDFCube* that is designed based on the structure of an RDF triple. This RDFCube scheme consists of a set of cubes of the same size called *cell*s, each of which contains a bit called an *existence flag* (*e-flag*) that indicates the presence or absence of triples mapped into the cell. The existence flag of the cell is set to on when an RDF triple is mapped into a cell. By checking the existence flags of cells into which candidate answer triples are mapped, we can know of the existence of the triples before actually accessing remote nodes where the candidate answer triples are stored. This information is useful for processing a join query because we can narrow down the candidate triples by AND-ing the e-flags bits and transfer only the actual present candidate triples. As a consequence, this scheme markedly reduces the amount of data that is transferred among nodes.

2 Our Proposed Approach

2.1 RDFCube: A Three-Dimensional Hash Space

To provide efficient join operations in a distributed environment, we propose a three-dimensional hash space named *RDFCube*, whose model is shown in Fig. 1. The axes in RDFCube respectively represent hash spaces for subject, predicate and object.

We assume that the range of the hash function is $[0, 2^m)$ and divide every space by 2^c. The three-dimensional space that is surrounded by the ranges

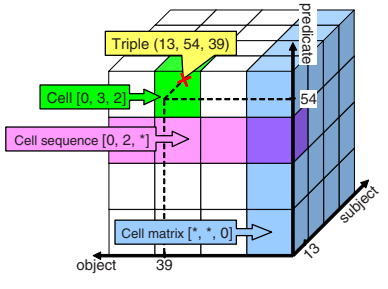


Fig. 1. A three-dimensional hash space (RDFCube)

$[2^{m-c}i, 2^{m-c}(i+1)), [2^{m-c}j, 2^{m-c}(j+1)), [2^{m-c}k, 2^{m-c}(k+1))$ is called *cell* ($0 \leq i < 2^c$, $0 \leq j < 2^c$, $0 \leq k < 2^c$) and is identified by *cell id* $[i, j, k]$.

We call a set of consecutive cells on a line parallel to an axis a *cell sequence*. A cell sequence parallel to the subject axis is identified as $[\{0, \ldots, 2^c - 1\}, j, k]$; its abbreviation is $[*, j, k]$. Cell sequences to the other axes are similar. Similarly, all cells on a plane perpendicular to an axis form a *cell matrix*. A cell matrix that is perpendicular to the object axis is identified as $[\{0, \ldots, 2^c - 1\}, \{0, \ldots, 2^c - 1\}, k]$, and its abbreviation is $[*, *, k]$. The other axes are similar. As examples, the cell $[0, 3, 2]$, the cell sequence $[0, 2, *]$, and the cell matrix $[*, *, 0]$ are shown in Fig. 1.

An RDF triple is mapped into a point of coordinates in RDFCube based on the hash values of the three elements in the triple. Every cell has a bit called an *existence flag*, which indicates the presence or absence of the triples that are mapped into the cell. We define the function that represents the bit values of the set of cells C as $flag(C)$. If C is a cell sequence, then we call the value of the function a *bit sequence*. If C is a cell matrix, then we call it a *bit matrix*. Furthermore, we define the ratio of the number of cells whose bits are set to 1 to the number of a set of cells as *1-bit ratio* for the set of cells.

An RDF query triple is also mapped into a line or plane in RDFCube based on hash values of its constants in the query triples. We call the set of cells including the line or plane where a query triple is mapped as *candidate answer cells; CA-cells*; their bits are *candidate answer bits; CA-bits*.

2.2 RDFCube Construction

We build a distributed hash table called RDFCube DHT to store existence flags in a distributed environment. The RDFCube DHT uses a cell matrix id as a key. It stores the associated bit matrix as a value (on the left side in Fig. 2), whereas RDFPeers DHT stores triples using each element of each triple as a key (on the right side in Fig. 2). Since RDFCube does not store triples, but stores bit information of e-flags, RDFCube DHT is used as an index for RDFPeers.

To set an existence flag on, we must update three bits on three bit matrixes on RDFCube DHT to 1. If the given triple is mapped into the cell $[i, j, k]$, We have to set $flag([i, j, k])$ to 1 by, at most, three lookups to RDFCube DHT. Using the cell matrix id $[i, *, *]$, $[*, j, *]$ and $[*, *, k]$ as keys, and update the bit on each of the three cell matrixes to 1. If the existence flag has already been set to 1, we do perform one lookup, because the first lookup is performed to not only set a bit to but also check the value. Therefore, Given t triples for an n-node DHT, the number of hops to construct RDFCube is, at most, $3t \log n$.

2.3 Query Processing with Join Operations

On receiving an RDF query containing join conditions, to perform it efficiently, we use RDFCube DHT to find the existence of the candidate answer triples before actually accessing the remote nodes where the triples are stored.

Query processing for the join operation shown in Algorithm 1 is as follows: i) In lines 3–6, we first get e-flags; and then ii) in line 7 perform AND operations

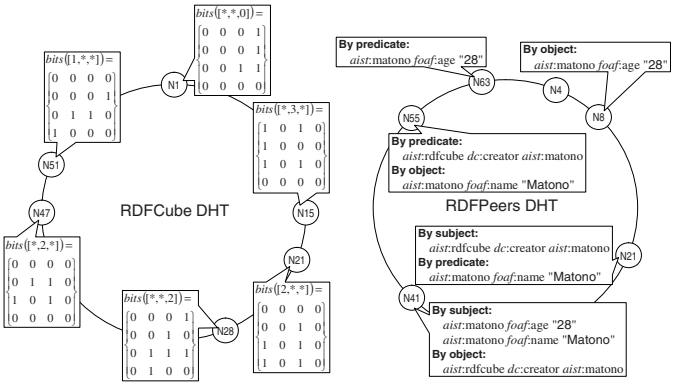


Fig. 2. Two distributed hash tables

Algorithm 1: Query Processing

Input: set of query triples Q
Output: set of triples T

1 set of triples $T \leftarrow \emptyset$
2 a set of bits using a query triple as a key is stored into map $M \leftarrow \emptyset$
/* i) Getting e-flags from RDFCube DHT. */
3 **foreach** $qt \in Q$ **do**
4 $cells \leftarrow$ get a set of cells where qt is mapped into
5 $bits \leftarrow$ get a set of bits of $cells$ by lookup to RDFCube DHT
6 store $bits$ into M using qt as key
/* ii) Construction of filters by performing AND operations. */
7 $M \leftarrow$ `constructFilter`(M, Q)
/* iii) Reducing the number of CA-triples using a filtering process. */
8 **foreach** $qt \in Q$ **do**
9 $bits \leftarrow$ get a set of bits from M using qt as keys
10 $T \leftarrow T \cup$ get triples that are matched by qt and that are filtered based on $bits$
11 $T \leftarrow$ apply T to RDFPeers query processing using Q
12 **return** T

among the e-flags; and finally iii) in lines 8–10, we access to the remote nodes where the CA-triples are stored on RDFPeers DHT, and then we filter the CA-triples using the results of the AND operations on the remote nodes. This filtering process reduces the number of CA-triples transferred among nodes. Fig. 3 illustrates an example of join operation.

i) In line 5, if *cells* is a cell matrix, by performing a lookup to RDFCube DHT using the id of the cell matrix as a key, the associated bit matrix is obtained. On the other hand, if *cells* is a cell sequence, to reduce the data that are transferred among nodes, the associated bit matrix is not returned, but the associated bit sequence that is extracted from the bit matrix is returned from RDFCube DHT.

ii) In line 7 the `constructFilter` function is called for construction of bit sequcences and/or matrixes for filtering CA-triples by performing bit operations among sets of the given CA-bits. Fig. 4 depicts an example of `constructFilter`

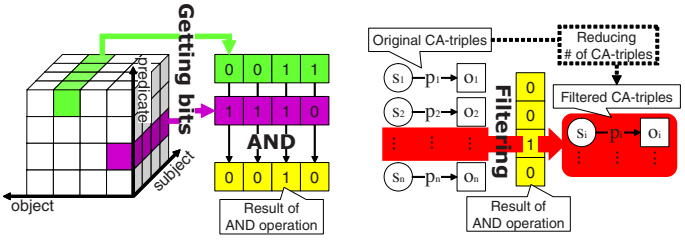


Fig. 3. An example of a join operation

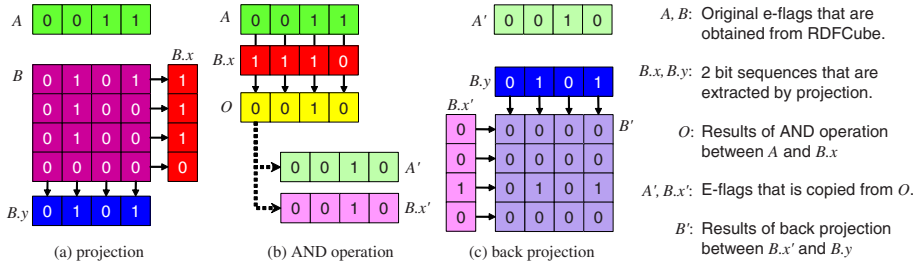


Fig. 4. Projection, AND operation, and back projection

function using a bit sequence A and a bit matrix B. This process performs the following processes: (a)*Projection*; for each e-flags A and B that are obtained from RDFCube DHT, we do nothing if the e-flags is a bit sequence, whereas, if the e-flags is a bit matrix, we extract two bit sequences $B.x$ and $B.y$ from the matrix by AND-ing among all bits in its columns and rows. (b)*AND operation*; we do an AND operation between the bit sequences A and $B.x$ for the same variable in different query triples. And then, we copy the e-flags O as the result to A' and $B.x'$. (c)*Back projection*; for each query triple, we do nothing if the original e-flags A is a bit sequence, whereas, if the original e-flags B is a bit matrix, we reconstructs a bit matrix B' from the two bit sequences $B.x'$ and $B.y$ by AND-ing between each bits of its columns and rows.

Because the 1-bit ratios of A' and B' became less than those of A and B, respectively, in Fig. 4, you can see that this process can reduce the number of CA-cells.

iii) In line 10, we access remote nodes where CA-triples are stored by performing lookups to RDFPeers DHT. For each CA-triple, we check whether the existence flag of the cell where the CA-triple is mapped is equal to 1 on remote nodes, and finally return CA-triples that satisfy the condition. This filtering process can narrow down the number of CA-triples.

The number of hops to obtain a set of triples for n-node RDFPeers DHT is, at most, $\log n$, as well as getting a set of bits from n-node RDFCube DHT. Thus, the number of hops to perform RDF query composed of m query triples is $2m \log n$.

3 Performance Evaluation

3.1 Experiment Setup

We next evaluate the performance of our approach through a series of experiments. In our experiments, we compare the native RDFPeers with RDFPeers using RDFCube, called *PEERS* and *CUBE*, respectively.

For our experimental dataset, we transform XML documents distributed by DBLP[1] into RDF data. We choose triples randomly from the original RDF data to create four data sets. The numbers of triples of the data sets are 13761, 26413, 52926, and 103076.

We use the three queries below in the experiments.

Query 1 requires a join operation among four bit sequences.

```
?x rdf:type dblp:Article.      ?x dblp:author "Jim Gray".
?x dblp:year "1998".           ?x dblp:journal "CoRR".
```

Query 2 requires a join between a bit matrix and a sequence.

```
?x dblp:series ?y.   ?y dblp:title "LNCS".
```

Query 3 requires 2 joins among 2 matrixes and a sequence.

```
?y dblp:crossref ?x.   ?x dblp:title "VLDB2004".   ?y dblp:title ?z.
```

We use an overlay construction toolkit, Overlay Weaver[2] , and emulate RDFCube and RDFPeers on top of the system. In our experiments, we use Chord [7] as a DHT protocol. All finger tables are constructed before the evaluation. The number of nodes is fixed as 100.

3.2 Experiment Results

Fig. 5 and Fig. 6 respectively shows the storing and retrieval performance of CUBE and PEERS. The horizontal axes show the numbers of triples; the vertical axes show the performance ratios of CUBE to PEERS.

In Fig. 5, when the ratio is 2, the cost for storing triples is equal to that for index construction. Index construction is costless when the ratio is 1. Therefore, we can see that the ratio of the hops is smaller than two. This means that the cost for RDFCube DHT construction is smaller than that for storing triples into RDFPeers. The reason for this is that one or three lookup(s) is/are required to set an e-flag to 1, whereas three lookups are required to store a triple into RDFPeers DHT as described in Section 2.2. For this reason, as the number of triples increases, the 1-bit ratio increases and the ratio of the number of hops decreases.

For the amount of data transferred among nodes, the ratio is very close to one, meaning that the amount of data transferred for RDFCube DHT construction is much smaller than that for storing triples into RDFPeers DHT because data

[1] http://dblp.uni-trier.de/xml/

[2] http://overlayweaver.sourceforge.net/

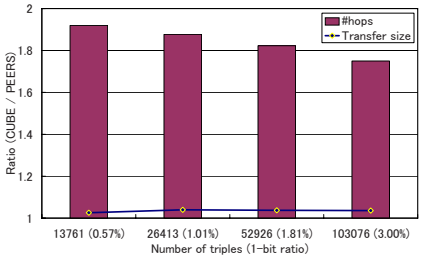


Fig. 5. The ratio of storing performance while the growth of #triples (#divisions is 128)

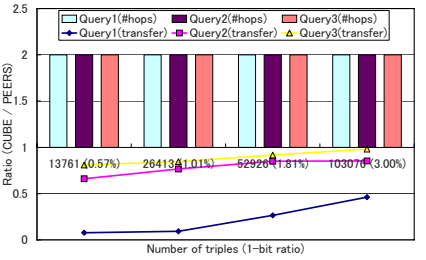


Fig. 6. The ratio of retrieval performance while the growth of #triples (#divisions is 128)

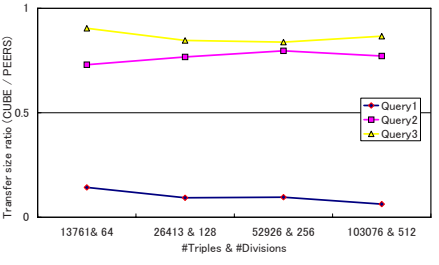


Fig. 7. Scalability results while both of #triples and #divisions increases

that are transferred for index construction (i.e. cell matrix id) are much smaller than that for storing triples (i.e. triple).

From Fig. 6, it is apparent that the number of hops on CUBE is twice as large as that on PEERS, which means that the number of hops to get triples is equal to that of to get e-flags. However, it can be said that the performance of RDFCube is not bad. That is true because the number of lookups on PEERS is small; it is less than the number of the query triples.

Regarding the amount of data transferred among nodes, we can reduce it to as little as 1/50. In Query 1, because the variable ?x occurs four times, the quality of filtering is very good. That for Query 2-3 is poorer than that for Query 1 because in Query 2 the variable ?y occurs two times and in Query 3 both of the variables ?x and ?y occur two times. From this, we can see that, as the number of query triples containing the same variables increases, the performance for executing the query is good.

Fig. 7 shows the ratio of the amount of data transferred among nodes of CUBE to PEERS while the numbers of both triples and divisions increase. From Fig. 7, it is apparent that the ratio of CUBE to PEERS remains approximately constant in all queries because the 1-bit ratio for a bit sequence associated with each variable obtained by projection remains constant. From this, we can see the scalability with respect to the ratio of transfer size reduction, when, as the number of triples increases, the number of divisions increases proportionately.

4 Conclusions

In this paper, we proposed an indexing scheme for a distributed triple stores so that it can efficiently perform structural join operations for RDF data. The scheme filters unnecessary candidate answers before actually accessing remote nodes where the answers are stored, thereby greatly reducing the number of triples transferred among nodes. The evaluation results demonstrate the scalability of our scheme. Especially, good scalability is obtainable when the 1-bit ratio is small and the query contains many variables.

Using DHT suffer from some problems; such as freshness of data and security. We should therefore consider an approach without DHT in future studies. We also think that an indexing scheme for/using RDF schema information in a distributed environment is an important issue. In addition, we shall consider triple deletion and dynamic division of RDFCube.

References

1. World Wide Web Consortium: Resource Description Framework (RDF). `http://www.w3.org/RDF/` (2004) W3C Recommendation 10 February 2004.
2. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: EDUTELLA: a P2P networking infrastructure based on RDF. In: WWW. (2002) 604–615
3. Ripeanu, M., Iamnitchi, A., Foster, I.T.: Mapping the Gnutella Network. IEEE Internet Computing **6** (2002) 50–57
4. Cai, M., Frank, M.R.: RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In Feldman, S.I., Uretsky, M., Najork, M., Wills, C.E., eds.: WWW, ACM (2004) 650–657
5. Cai, M., Frank, M.R., Yan, B., MacGregor, R.M.: A subscribable peer-to-peer RDF repository for distributed metadata management. J. Web Sem. **2** (2004) 109–130
6. Cai, M., Frank, M.R., Chen, J., Szekely, P.A.: MAAN: A Multi-Attribute Addressable Network for Grid Information Services. In Stockinger, H., ed.: GRID, IEEE Computer Society (2003) 184–191
7. Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM. (2001) 149–160

Optimal Caching for First-Order Query Load-Balancing in Decentralized Index Structures

Anwitaman Datta[1], Wolfgang Nejdl[2], and Karl Aberer[3]

[1] Nanyang Technological University (NTU), Singapore
anwitaman@ntu.edu.sg
[2] University of Hannover, Germany
nejdl@kbs.uni-hannover.de
[3] Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
karl.aberer@epfl.ch

Abstract. Balancing query-load in structured overlays is an important and mostly unattended problem, apart from heuristics to deal with hot-spots. We determine the optimal caching strategy - how many caches (dependent on relative frequency of queries) to reduce search latency and where to place these caches? Our query-adaptive replication scheme provides first-order balancing of query-load. We also elaborate the limitations of first-order balancing (ignored by all existing heuristics) though it should have been obvious given the experiences of the P2P community with storage-load balancing.This work lays the ground for our ongoing work on load-aware routing in structured overlays as a mechanism complementing caching to achieve query-load balancing.

1 Introduction

Structured overlay networks are decentralized large-scale index-structures maintained among peers for resource discovery in wide-area networks and is a fundamental building block used by peer-to-peer (P2P) applications. The basic idea is to associate the resources to unique keys, and maintain an index of these keys. Each peer is responsible for a partition of the key-space - i.e., stores the key-value pairs for the corresponding keys. The value may be the resource itself, or more generally a set of pointers to the resources.

As in any distributed system, load-balancing is very important for such decentralized index structures. If there is a centralized resource allocator with global information - on workload and resources, policies and enforcement of load-balancing would be relatively straight forward. In absence of such central coordination randomized techniques are resorted to.

Before delving into the specific problem of query-load balancing, we'll like to explain what we mean by a first-order load-balancing, and what a second order load-balancing would mean in that context. Consider that M balls are to be placed in N bins. If these balls are thrown into bins *sequentially* at uniformly

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 331–342, 2007.

randomly chosen bins, one would expect that the balls will be uniformly distributed among the bins. The uniform random bin choices essentially achieves a first-order load-balancing. It has been seen that the variance of the number of balls per bin is quite high using the uniformly random bin choices [1]. A second-order load-balancing would involve reduction of this statistical noise (variance). A simple and popular way of achieving such a second order load-balancing uses multiple choices. Thus multiple bins (say k) are chosen at random, and the least filled bin among these k bins is assigned the *next ball*. Power of two choices [2] is a special case of using such multiple choices.

In the context of structured overlay networks, there are many facets of load-balancing. The most extensively studied one is to balance the number of keys assigned to each peer. The resources may have skewed load-distribution in the name-space (e.g., distribution of filenames). In order to achieve first-order balancing under such load-skew, early overlay networks used uniform hashing to generate the keys to be indexed, and these networks came to be known as Distributed Hash Tables (DHTs) [3,4,5]. Second-order load balancing to reduce the variance of number of keys per peer was achieved in DHTs using power-of-two choices [6]. Ever since, load-balancing techniques for range-partitioned networks (with skewed distribution of keys over the key-space) have also been developed [7,8,9,10].

Dealing with skewed access load on keys has not yet been dealt with systematically. This paper takes a more objective look at the issue of balancing skewed query-load in structured overlays.

The result leads us also to interesting general observations on structured overlay networks - load balancing for query load in structured overlay networks (i.e., if replication is proportional to query frequency) is also optimal with respect to reduction of expected search latency. This result appears at a first glance obvious, but for the expert it is surprising, since for other problems of query frequency dependent replication, both for unstructured overlay networks [11] and for mobile broadcast environments [12], it has been shown that following a square-root rule for replication is optimal for latency reduction under memory constraint, and has thus often been used generally as a rule of thumb. The square-root rule is that different objects are replicated proportional to the square-root of the frequency with which they are accessed/queried.

From this we can also conclude that for structured overlay networks accidentally replication can be simultaneously optimized with respect to availability and non-uniform query loads. But more fundamentally, we answer how good is even the best possible solution relying solely on caching?

Concretely, this paper makes the following contributions:

(i) Provides a *taxonomy of replication* placement strategies for structured overlays. (ii) Determines the *optimal replication strategy* - (a) *How many replicas* need to be maintained, in order to meet the design goals of availability and load balancing with respect to arbitrary skews in query-load?; (b) *Where in the network* should these replicas be placed, in order to best utilize the resources?

(iii) In various environments - internet, content distribution networks, unstructured as well as structured overlays - caching is expected to improve performance in terms of search latency as well as handling hot-spots. We expose the *limitations of caching* techniques. (a) There is enormous storage consumption for marginal gains in latency, even for skewed access rates (like Zipf). (b) Caching proportional to access load in itself is not good enough because of statistical noise.

This raises important concerns also about the effectiveness of existing heuristics found in the structured overlay literature with respect to alleviating query-load, and highlights the need to use sophisticated load-balancing techniques.

This work makes the following assumptions: (i) The number of keys a peer is responsible for is already balanced, which is more or less achieved under various settings - range-partitioned or DHTs - by [7,6,10]. (ii) All peers have same[1] and limited storage, part of which is dedicated to store the keys it is responsible for based on its role in the index, and the rest is used in order to dynamically cache some keys in order to alleviate load imbalance and improve search latency. (iii) The routing network itself does not lead to any systematic hot-spots because of large variations in degree distribution at each peer.

2 A Taxonomy of Replication

We can distinguish six general replication strategies found in the literature on structured overlays:

Structural replication: In this replication scheme, multiple peers can be responsible for the same key space partition. This strategy can be seen as exploiting multiple identical non-replicated overlay networks superimposed on each other, where the routing choices are made randomly (or based on other considerations like proximity) from several choices of peers belonging to the same key space partitions. Such replication is called zone overloading in CAN [4] or simply replication of the tree leaves in P-Grid [7].

Constrained replication at peers with closest ID: In this scheme, keys are replicated at a globally fixed number of immediate successors of the peer responsible for a key for fault tolerance. This is the strategy that has typically been used in Chord based CFS [13] and the Chord variant DKS(n,f,k) [14]. All queries are typically routed to the primary replica, i.e., the one peer originally responsible for the key being queried, while other peers act purely as back up for fault tolerance. Because of this routing behavior, the search process does not exploit the wider availability of the resource for reducing the search cost or distribute query load. On the other hand, an advantage of this replica placement strategy is that updates are easy to perform, since all replicas can be deterministically located and there is a natural choice of a primary replica.[2]

[1] Homogeneity can be achieved by using multiple virtual peers for resource rich peers.

[2] Even if pointers instead of actual content is stored and replicated, these pointers may need to be updated, possibly with information on new copies of the actual content.

Replication along the query path: Replication can be done along the path used by a previous query. This strategy, apart from yielding increased redundancy, fault tolerance and adaptivity to queries, has the additional benefit that future queries can potentially be answered based on the cached information resulting in improved search performance.This is the strategy used in semi-structured networks like Freenet [15]. The drawbacks of this scheme are that, it does not completely exploit the structure of the network, so while the query performance improves, it is not necessarily optimal. Additionally, since the scheme is inherently heuristic and non-deterministic, the replicas can not be located in the network efficiently and exhaustively, and hence replication along query path conflicts with effective updates of replicas.

Caching at querying peers: This strategy neither exploits the structure of the overlay at all to reduce search cost or distribute load, nor are replicas easily locatable for updates.

Replication at least loaded peers: A recent load adaptive replication proposal (LAR [16]) places replicas purely based on peer load information gathered using sampling mechanisms, and can potentially create replicas at any nodes in the system. This requires modification of original DHT routing, leading to a different and more complex routing mechanism, based on disseminated information about replicas. This replication scheme fails to exploit the properties of the original structure of the network to reduce search cost and also loses the deterministic nature of the replica location in the network, which makes updates difficult. That apart, such a load-balancing strategy does not have a separation of concern between routing structure and query-load balancing and loses the simplicity and efficiency of routing in structured overlays.

Optimal placement strategy: Different topologies may require different placement strategies in order to satisfy the optimality criterion. An interesting observation is that structural replication is an optimal placement strategy independent of the topology involved, provided every item is equally replicated. However, query-adaptive replication implies different items are replicated differently, needing different placement strategies. Later we'll explore optimal query-adaptive replica placement strategies for some important classes of overlay networks like generic tree structured overlay network [17] which subsumes Hypercubes and de Bruijn networks, tree abstracted networks like P-Grid, Pastry and XOR topology based Kademlia, as well as other topologies like CAN and Chord.

3 Replication and Search Cost

In many real life applications queries are non-uniformly distributed. The standard solution approach both in an internet setting as well as in overlay networks is to cache (replicate) the queried objects in a query-adaptive manner to provide good load-balancing as well as to reduce search latency. There is a huge literature on caching in standard networks and also some work done in the context of P2P overlays. Rest of this paper will provide a generic theoretical framework for

optimal replication in structured overlays, along with simulation based evaluation of its discernable nonetheless limited effectiveness.

Definition 1. *Let $\sigma(N, r)$ be the expected cost to search a key in a structured overlay network (using any particular topology/routing mechanism) with N peers, and r replicas of the key.*

Consider that we form clusters of r peers such that there are N/r logical units, with one of these clusters consisting of the r peers storing the replicas of the concerned data item. These clusters may be considered to be connected using the same topology, as the original network of N peers. Then exploiting the self-similarity of the structured overlay networks, the expected cost of locating the particular cluster is $\sigma(N/r, 1)$, independent of the topology. Stated otherwise, we can assume r parallel networks of network population of N/r to achieve a search cost of $\sigma(N/r, 1)$ for a key replicated r times in a network of N peers.

Observation 1. *In a structured overlay network of N peers, if a data item is replicated r times, then there exists (for a wide range of structured overlays) a policy to place these replicas so that at least any one of the r replicas can be located with an expected search cost $\sigma(N, r)$ of $\sigma(N/r, 1)$.*

Of the various existing replication strategies (summarized in Section 2) for diverse overlay topologies, none is known to have better search performance, and most even do not match up with the above mentioned potential search cost reduction. In the following we use this known best achievable search cost reduction criterion to define optimality for replica placement strategies. Whether even further search cost reduction is possible in an overlay by cleverly placing the replicas and possibly even changing the routing strategy (subject to the same topology) is an interesting open question.

Definition 2. *The optimal replication placement strategy from the search cost perspective in an overlay network is the one which guarantees that $\sigma(N, r) = \sigma(N/r, 1)$ for a data item replicated r times.*

Note that when different data items need to be replicated with different frequency, for example, for query-load balancing; it may not even be possible to exactly form such clusters (or juxtaposed networks) in practice. In such cases we can only aim to determine the best replica placement in the network in order to be as close to this optimal as possible.

4 Optimal Query-Adaptivity

For the rest of this paper we consider the family of DHTs which have logarithmic search cost, i.e., $\sigma(N, r) = c\ log(N)$ for a peer population of N, where c is a routing topology dependent constant. This is typical in many structured overlays including Chord, Pastry, P-Grid and Kademlia among others. Additionally, we assume for the time being, that it is indeed possible to optimally place replicas in the network, such that for a data item d_i with r_i replicas, $\sigma(N, r_i) = c\ log(N/r_i)$.

Theorem 1. *Replication of objects proportional to frequency they are queried minimizes the expected search cost in a structured overlay with limited storage capacity, assuming replicas are placed optimally. (Placement strategy optimality as defined in Definition 2.)*

Proof: Consider that peers have the possibility to replicate a data item such that replica placement is optimal with respect to the search cost. Thus to say, search cost for any one instance of a data item replicated r times in a network of N peers is $c\ log(N/r)$.

Assume also that there are M distinct data items in total, and the access (query) probability for data item $d_i, i = 1, \ldots, M$ equals q_i and d_i has r_i replicas.

The total storage used in the system is then (with R being the average replication factor in the system)

$$\sum_{i=1}^{M} r_i = RM$$

The expected access time (in terms of messages) to access any data item is

$$T = c \sum_{i=1}^{M} q_i log(\frac{N}{r_i})$$

In order to determine the optimal allocation of replicas for a given global average replication factor R (essentially determined by the total storage capacity of the system, and the total number of distinct data items in the system) we have to solve the system of partial differential equations

$$\frac{\partial T}{\partial r_i} = 0, i = 1, \ldots, M$$

Substituting $r_M = RM - \sum_{i=1}^{M-1} r_i$ and differentiating we obtain

$$\frac{\partial T}{\partial r_i} = -q_i \frac{1}{r_i} + q_M \frac{1}{r_M} = 0$$

from which we conclude that $\frac{q_i}{r_i}$ must be constant $\forall i = 1, \ldots, M$.

Note that there are thus two dimensions of optimality for replication:

(a) Query-adaptivity determines how many replicas to maintain for individual data items to minimize expected search cost. Unlike in unstructured overlays [11], it so happens that for a large family of (logarithmic search-cost) structured overlays, this simultaneously provides a first-order query-load balancing.

(b) The placement strategy determines, for r replicas of a data item to be placed in the network, where exactly these replicas are to be placed in order to reduce search latency.

Next we explore specific placement strategy for some important groups of DHTs, particularly determining the optimal placement strategy for P-Grid.

5 Optimal Replica Placement

In recent years, many routing network topologies have been proposed. There have been recent attempts to derive abstracted, generalized models [17,18] for these diverse topologies. However, none of these models are exhaustive, and in absence of any universal abstraction, we limit the discussion to some specific, well-researched DHTs, and try to generalize our proposal when possible. In particular, we concentrate on some important classes of DHTs, and informally describe the optimal replica placement policy.

Let us first focus on tree-like abstractions [17], particularly on P-Grid [7] and XOR topology based Kademlia [19] - the generalization to other PRR [20] variants like Pastry [5] is straightforward.

In these virtual tree based access structures, a peer is responsible for all data items in its leaf node. Data items may be assigned to the leaf node based on various association mechanism, for instance prefix matching. This peer then acts as the primary replica for the data items. When a particular data item needs to be replicated, the optimal replication strategy places the replicas at the other leaf nodes which share common intermediate nodes in the tree. In effect, this is like replicating in the tree at a higher level (or reducing the depth of the search tree), since the data item will then be found at any of all the peers which split the intermediate (imaginary) node. This process can be repeated, successively propagating the replication to larger number of peers, which all share the same internal nodes in the tree abstraction. Such a placement strategy effectively leads to logically coalescing of the key space partitions. In terms of the generic overlay network scheme [17], essentially, the search space can be seen as being split among peers recursively (when joining). So a peer should replicate the content at other peers with whom it had conducted the splitting operation, such that effectively the data is available in the logically coalesced search space.

The basic idea of such a replication scheme is intuitive. Coalescing r partitions which are topologically closest (based on incoming links), effectively leads to a logical network of N/r partitions. Lookups from different clients for the same data item tend to converge to the same set of peers, such that a replica is located earlier, hence speeding up the lookup process. The N/r partitions are connected with the same topology as the original N partitions, that is to say - these networks are self-similar, hence the speeding up matches the optimality criterion in the ideal case.

Chord does not have a direct tree-mapping nor is modeled by the generic abstraction [17]. Still the routing network resembles a tree-like access structure from the perspective of individual peers. If a peer p_1 is the primary responsible peer for a data item, p_1 should replicate this data item at the closest preceding peers from which p_1 has incoming links. This information is locally available to p_1 (from the route maintenance operations). This replication scheme is a small variation of caching along query downstream which CFS already uses, and has marginal influence in terms of search cost improvement. However, by choosing the closest incoming links at a peer (network maintenance protocols require these peers to communicate periodically in any case), instead of necessarily choosing

the peers from the direction the last query arrived, the replicas are placed in a deterministic manner, based purely on locally available information. This placement strategy can also be used to improve fault tolerance in CFS [13], thus amortizing the costs of fault-tolerance and load-balancing.

6 Query Adaptive Replication

6.1 Numerical Evaluation

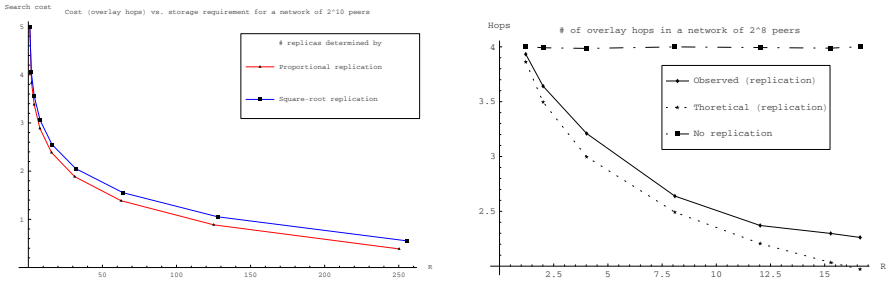
Replication proportional to the square-root of the query frequency has been shown to minimize expected search cost for unstructured P2P systems [11]. We choose this strategy as the baseline to compare with our proposal of replicating directly proportional to the query frequency. We consider a Zipf-distribution (parameter 0.8614) of queries. We consider that there are $M = 4096$ unique keys d_i that are queried with relative frequencies q_i which vary between 1 and 256. We further assume an overlay with 1024 partitions of equal size (e.g., for a tree-structured overlay, a balanced tree). That is, ignoring any structural replication, its a network of $N = 1024$ peers. We additionally assume that the keys are distributed over these partitions in such a manner that it would be possible to optimally place the replicas by exploiting only the available storage in the correct peers (determined by optimal placement criterion). So to say, this numerical analysis makes several idealizing assumptions.

Consider that there is storage capacity for an average replication factor of $R \geq 1$ in the system. Note that $R = 1$ actually means there is no replication, but only the original copy is stored. For the proportional replication case, there are thus $r_i = Max(1, \alpha_1 q_i)$ replicas and for the square-root replication strategy, there are $r_i = Max(1, \alpha_2 q_i^{0.5})$ replicas for data item d_i, where α_1 and α_2 are determined under the constraint of limited available storage in the system, i.e, $\sum_{i=1}^{M} r_i = RM$.

We numerically evaluate and show in the y-axis of Figure 1(a) the expected search cost $0.5 \sum_{i=1}^{M} q_i log_2(\frac{N}{r_i})$ for various values of overall storage capacities shown in the x-axis.

This demonstrates that for the same average storage capacity R, in structured overlay networks with logarithmic search cost (in terms of network size), proportional replication outperforms the square-root replication strategy of unstructured networks, thus highlighting a fundamental difference of the effect of replication in these two broad classes of P2P systems - structured and unstructured.

We also observe that search cost reduction consumes storage exponentially, which further shows the fundamental limitation of replication/caching to improve search latency in DHTs. Beehive [21] tries to achieve $O(1)$ lookup using replication. While it may work for small network sizes and some specific workloads, replication based latency reduction approaches in general have limited practical use in structured overlays (even for heavily skewed distributions like Zipf) because of the exponential increase in storage requirement.



(a) (b)

Fig. 1. (a) Numerical evaluation: Proportional (linear) vs. Square-root replication for Zipf (parameter 0.8614) distributed queries for various storage capacities; **(b) Simulation:** Storage space vs. search latency trade-off under Zipf-distributed (parameter 0.8614) queries

6.2 Simulations

Setup and workload: We simulated a randomized tree-structured network of 2^8 peers (specifically using the P-Grid routing topology), with the routes chosen either (i) randomly or (ii) using the power-of-two-choices to balance the in-degree.

Each peer initially held 5 unique data items, thus there were 1280 unique keys. Each peer had a capacity for R_{pot} data items (including the original), where R_{pot} was varied between 1.2 to 20. Queries with relative frequencies Zipf-distributed (parameter 0.8614) were originated at random peers chosen uniformly. Approximately 16700 queries were issued. Queried objects were replicated according to the optimal placement strategy described earlier, with one additional replica created for each query received by any current replica. In case of lack of storage space, least recently queried object (locally perceived at individual peers) was removed in order to replicate a newly queried object.

Experiment results: In Figure 1(b) we show the average number of hops required to answer the queries for different values of average replications R. First thing we noticed in the experiments was that the value of R stayed smaller than available storage space R_{pot} since replication placement is constrained by the placement optimality criterion and thus any arbitrary available space can not be used. For example, in our experiments, with $R_{pot} = 20$, only $R \approx 16$ was used. We also notice that for the storage space used, the average search cost is slightly higher than as expected from theory, even though it follows the same trend. This is because before enough replication is done, the queries can not leverage the advantage of replication and hence require more hops. The analysis in Section 4 assumed the replicas were already in place.

We show *cumulative distribution functions* in Figure 2 to summarize the load-balancing results, where two different measures of load are used: (i) the number of query messages forwarded by peers, as well as the (ii) the number of queries

actually answered by peers (possible when the peer has the corresponding key stored locally). The *cumulative distribution* plots are to be interpreted as follows: The x-axis represents the load and the y-axis the percentage of the peer population which has a load less than or equal to this specific (x-axis) load. Thus steeper ascent of the curve represent smaller variation of load among peers, while gradual ascent results from greater variation (poorer load-balance). Two sets of experiments were conducted, once with queries with relative frequency Zipf-distributed, another where all keys were queried exactly the same number of times. Queries were issued at random peers.

The fact that the curve for adaptive replication based search (in Figure 2(b) & 2(d)) is above the one without replication implies, that the adaptive replication based strategy requires fewer number of messages per peer for search. A steeper slope in Figure 2(a) shows that the deviation in the number of queries answered using the replication based strategy is lower than without replication, i.e, replication leads to better query-answering load-balance. We also notice that balancing the in-degree based on power-of-two choices (Po2C) leads to improvement in load-balance. The improvements are discernable, but limited. We attribute it to the statistical noise. The huge effect of statistical noise becomes apparent in Figure 2(d) for the experiment where all keys are queried the same number of times. In this case, the query-answering load is balanced if no replication is done, since each peer receives queries for its own keys and all keys are queried equally. However, with adaptive replication, as keys are replicated - the effect of statistical noise kicks in, thus in fact leading to load-imbalance.

This experiment where keys were queried equally was more to put in context the effect of statistical noise. Under realistic work-loads, we'll need the query-adaptive replication as a means to achieve first-order load-balancing. However, a first-order load-balancing is in itself inadequate unless complemented with a second order mechanism to reduce the variance.

7 Conclusion and Future Work

There is a sense of déjà-vu in what we observe from an objective look at first-order query-load balancing. Initial research in overlay design [3,4,5] hoped to achieve good balancing of key-distributions among peers by using uniform distribution. The effect of statistical noise [1] was recognized only later, and had to be fixed using second-order load-balancing mechanisms [6]. Still, one can find in literature that in order to deal with hot-spots, caching is proposed (which is necessary!), and presumed sufficient, which is not the case. In some sense, it is unfortunate that despite dealing with the effect of randomization for storage load balancing, the same effects of randomization for more critical resources - bandwidth and peer's answering capacity under hot-spot conditions, were totally ignored, presuming caching itself will solve the problem. One may speculate several reasons for overlooking such an important thing: (i) Initial work based on simulations could observe the imbalance of key distribution, since it accumulates over time. Bandwidth consumption is however temporary, and if only the average is measured (as has

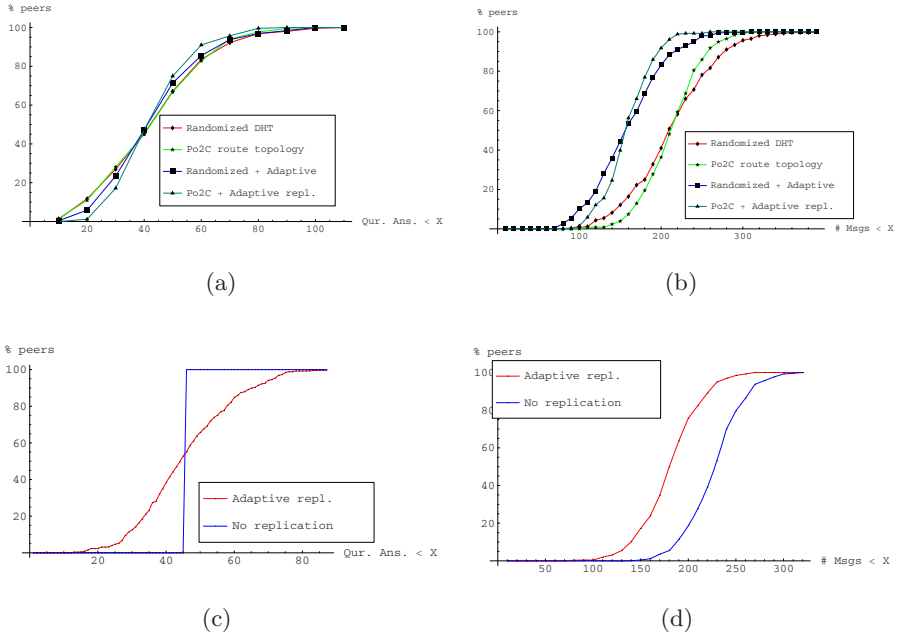


Fig. 2. Cumulative distribution of query forwarding and query answering loads at peers. **(a)** Queries answered by peers (Zipf distributed queries). **(b)** Messages forwarded by peers (Zipf distributed queries). **(c)** Queries answered by peers (Same number of queries per key). **(d)** Messages forwarded by peers (Same number of queries per key).

often been reported in most results), the imbalance goes unnoticed. (ii) It is only recently that some structured overlay implementations have matured enough to be deployed and is dealing with moderate query loads, and hence the effect of imbalance has not been observed. But as the volume of traffic in structured overlays increase, second-order balancing of query-load will become critical, since otherwise it'll cause congestion (and IP layer congestion control mechanisms won't be useful if the overlay systematically causes the congestion at end-nodes) even while other peers would have their resources under-utilized.

In that context, our work rediscovers the ghost of statistical noise. We are currently modifying greedy routing strategy as used in overlays to a load-aware routing mechanism to further improve query load balancing.

References

1. Raab, M., Steger, A.: "balls into bins" - a simple and tight analysis. In: RANDOM '98: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science, London, UK, Springer-Verlag (1998) 159–170
2. Mitzenmacher, M.: The power of two choices in randomized load balancing. IEEE Trans. Parallel Distrib. Syst. **12** (2001) 1094–1104

3. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the ACM SIGCOMM. (2001)
4. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. In: Proceedings of the ACM SIGCOMM. (2001)
5. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001)
6. Byers, J., Considine, J., Mitzenmacher, M.: Simple load balancing for distributed hash tables. In: IPTPS. (2003)
7. Aberer, K., Datta, A., Hauswirth, M., Schmidt, R.: Indexing data-oriented overlay networks. VLDB (2005)
8. Aspnes, J., Shah, G.: Skip graphs. In: SODA. (2003)
9. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. SIGCOMM Comput. Commun. Rev. **34** (2004) 353–366
10. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In: VLDB. (2004)
11. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and Replication in Unstructured Peer-to-peer Networks. In: International Conference on Supercomputing. (2002)
12. Acharya, S., Alonso, R., Franklin, M., Zdonik, S.: Broadcast disks: Data management for asymmetric communication environments. In: SIGMOD Conference. (1995)
13. Dabek, F., Kaashoek, M., Karger, D., Morris, R., Stoica, I.: Wide-area cooperative storage with CFS. In: SOSP. (2001)
14. Alima, L., El-Ansary, S., Brand, P., Haridi, S.: DKS (n, k, f): A family of low communication, scalable and fault-tolerant infrastructures for p2p applications. In: CCGrid. (2003)
15. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval sy stem. In: Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability. Number 2009 in LNCS (2001)
16. Gopalakrishnan, V., Silaghi, B., Bhattacharjee, B., Keleher, P.: Adaptive replication in peer-to-peer systems. In: ICDCS. (2004)
17. Abraham, I., Awerbuch, B., Azar, Y., Bartal, Y., Malkhi, D., Pavlov, E.: A generic scheme for building overlay networks in adversarial scenarios. In: IDPDS. (2003)
18. Ratajczak, D., Hellerstein, J.: Deconstructing DHTs. In: IBM-TR-03-042. (2003)
19. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: IPTPS. (2002)
20. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing nearby copies of replicated objects in a distributed environment. Theory of Computing Systems **32** (1999)
21. Ramasubramanian, V., Sirer, E.: Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays. In: NSDI. (2004)

On Triple Dissemination, Forward-Chaining, and Load Balancing in DHT Based RDF Stores

Dominic Battré, Felix Heine, André Höing, and Odej Kao

University of Paderborn
Paderborn Center for Parallel Computing
Fürstenallee 11, 33102 Paderborn, Germany
{battre,fh,andrehoe,okao}@uni-paderborn.de

Abstract. The Resource Description Framework provides a powerful model for structured knowledge representation that allows the inference of new knowledge. Because of the anticipated scope of semantic information available in the future, centralized databases will become incapable of handling the load. Peer-to-Peer based distributed databases offer better scalability and integration of many different data sources. In this paper we present a detailed data management strategy for a DHT based RDF store that provides reasoning, robustness, and load-balancing.

1 Introduction

The combination of the Resource Description Framework (RDF) [1] and RDF Schema [2] provides a powerful model for storing information and inferring new knowledge from this information. Using RDF/S to describe resources or services allows to discover these even in case the description does not exactly match a query because a query is formulated more generally than the resource or service description.

In RDF everything is represented by triples of the form (S, P, O). Triples can be read as sentences, and the triple components represent subject, predicate, and object respectively. Taking the example of resource description, the CPU of a cluster node can be described by a triple (Identifier for CPU, *rdf:hasType*, Identifier for Itanium processor). Identifiers are represented by URIs. The object component can contain not only URIs but literals (like strings or numbers) as well.

The real power of RDF stems from the possibility to derive new knowledge from explicit knowledge and background knowledge. Background knowledge like the fact that an Itanium processor belongs to the class of 64 bit processors allows to query all 64 bit machines and to find this particular cluster even though it is not described as a 64 bit machine explicitly.

The capacities of centralized RDF databases like Sesame [3] and Jena [4] are limited by the hardware of a single server and therefore do not scale very well. Thus, we argue that efficient distributed databases are a necessary precondition for the acceptance of the Semantic Web. Peer-to-Peer networks offer a foundation layer for such distributed databases. Current attempts are based on structured

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 343–354, 2007.

and super-peer Peer-to-Peer networks. BabelPeers [5,6], RDFPeers [7] and Atlas [8] are representatives of the first category. Edutella [9] is a representative of the latter category.

For distributed databases it is crucial to not only distribute data among the peers but also to provide data integration. A set of many independent databases does not compensate for one integrated database because the inference of knowledge and processing of queries that span multiple databases requires to connect different information sources.

In this paper we address some of the challenges encountered in designing RDF stores based on distributed hash tables (DHTs). We first contrast structured and super-peer Peer-to-Peer approaches for RDF stores in section 2. Then, we show in section 3 how reasoning and data-integration can be done even though data are distributed over the network. There we also address the issue of node-failure and churn. Section 4 presents load-balancing issues that arise in DHT based Peer-to-Peer networks and describes a strategy to address these issues with overlay trees. This strategy is compatible with thoughts on robustness and reasoning from the preceeding section. Finally, section 5 concludes the paper.

2 Related Work

Edutella [9] retains the structure of many independent RDF databases and connects these by a super-peer Peer-to-Peer network. Data remain at their original position and queries are routed to peers who may store relevant information. This has the advantage that content providers stay in possession of their data and decide whether to answer queries or to reject them. Furthermore, local databases enable easy updates of data and allow to query data sources which do not provide native RDF data. The disadvantage of this approach is that queries might need to be flooded into large parts of the network if many peers may contain relevant data. Imagine, for example, that persons and companies have local calendars stored as RDF data. Queries, which can be routed only based on the fact that a database knows a certain schema, have to be flooded through the entire network because each calendar might contain appointments of the person of interest. No hints on where to look for data are available and efficient lookups are not possible. This creates scalability issues. Furthermore, reasoning beyond the borders of local databases is very difficult to achieve. The same holds for the integration of data that span many databases. Therefore, queries cannot be evaluated just by querying local data stores.

BabelPeers [5,6], RDFPeers [7] and Atlas [8] follow a different approach based on structured Peer-to-Peer networks, in particular ring-shaped distributed hash tables (DHTs). The basic idea is to store each triple at three locations on the DHT ring determined by calculating the hash value of the subject, predicate, and object. Insert and lookup operations can be conducted in $O(\log N)$ message routing steps, where N represents the number of participants in the Peer-to-Peer network. The major advantage of this approach is that all triples with common subjects (or predicates/objects) can be looked up at one node and do not need

to be collected from all data sources spread over the network that may possibly contain triples of interest. Bloom filters and lookups for result-set sizes can reduce the search space and the number of triples transmitted over the network during query processing. Details can be found in [5]. BabelPeers is based on FreePastry an open source implementation of Pastry [10].

In the following section we present a strategy to store and disseminate triples in a way that allows (a) soft-state updates, (b) replication for node departure and exploitation of data-locality, and (c) reasoning and data integration. The last point in particular is a major advantage of DHT based approaches as data from different sources are gathered into a virtual global database. Information about individuals can be distributed arbitrarily over various databases. To our knowledge, this is the first paper to address the issue of reasoning in DHT based RDF stores.

3 RDF Storage

In our architecture, each node hosts multiple RDF databases that serve different purposes. The first of these RDF databases to be mentioned is the *local triples* database. It stores those RDF triples that originate from the particular node and is therefore comparable to the local databases of Edutella. A triple should be accessible in the network for the whole time its owner is member of the network. If several nodes contribute identical triples, a triple should be accessible as long as any node contributing this triple is member of the network.

All local triples are regularly disseminated to the nodes in the network by calculating the hash function of the subjects, predicates, and objects and sending the triples to the nodes responsible for the corresponding parts of the DHT space. These nodes store the triples in their *received triples* set. Owners may occasionally be the responsible nodes for some of the triples, so perhaps some of the triples in the *local triples* set are duplicated in the *received triples* set. However, the larger the network grows, the more unlikely is this to happen. Thus, we do not take special care of duplicates and neglect the memory consumption caused by a triple stored twice on a single node.

Unless a triple has two or even three identical components, it is disseminated to three distinct positions in the DHT. The larger the network grows, the higher is the probability that these positions are actually located on different nodes. Thus, for large networks our distribution procedure results in a tripling of the number of triples.

A third kind of triple store is responsible for a *replica* database. Pastry [10] provides means to store replica of data to the k whose IDs that are nearest to the target hash value determined by the hash function. The purpose of these replica is to support the network when nodes depart or crash and to allow queries to be routed to a replica near the query issuer. In Pastry it is likely that a lookup operation that is routed to a position in the DHT passes a replica of this position. The replica node is often closer to the query issuer than the actual target of the query and can thus intercept the message and answer the query. This exploits locality information.

The node with an ID closest to the hash value of a triple becomes *root node* of the replica set. This node is responsible for sending all triples in its *received* database to the replica nodes. In pastry these nodes are simply defined by the leaf set. The replica, ordered by their distance to the target hash-value, are said to have *rank* 1 through k.

Finally, each node hosts a database for *generated triples* that originate from forward chaining. The forward chaining rules are applied to the *received triples* databases and all generated triples are stored into the *generated triples* database. They are then disseminated like local triples to the respective nodes and replicated from there on.

Figure 1 illustrates the triples stores hosted by peers and the flow of triples. The following sections elaborate further on this topic.

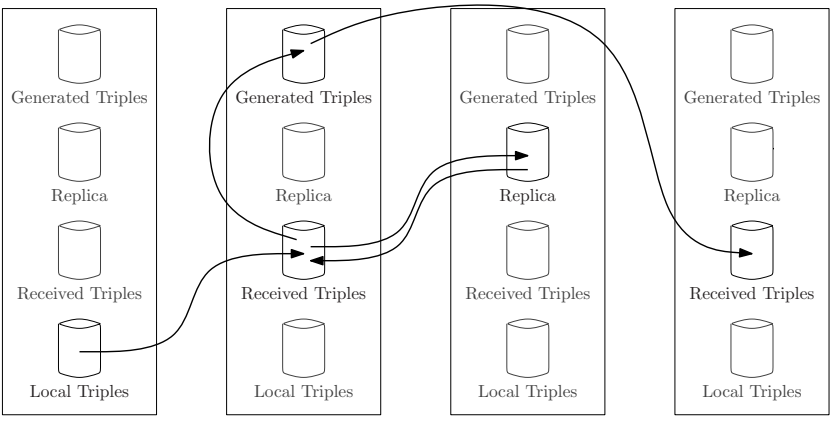


Fig. 1. Triple dissemination in DHT

3.1 Life-Cycle of Triples

The life-cycle of triples is influenced by various events. As each node contributes its local triples to the network, nodes joining and departing from the network change the knowledge base. Furthermore, nodes can of course acquire new information that needs to be distributed within the network, and delete or update triples.

Owing to the very dynamic nature of Peer-to-Peer networks and the autonomy of peers, it is not possible to guarantee that nodes depart gracefully and unregister all triples contributed by them to the network. In order to delete these obsolete triples from the *received triples* databases eventually it is either necessary for nodes to poll the triple sources whether their triples are still up to date, or to add an expiration date to triples. We have decided to follow the latter approach. Each triple has an expiration date and the owner is in charge of continuously sending update messages. This approach is known as "soft-state" in literature. The information source has to decide whether it considers itself as

stable, such that triples have rather long life-times and few refreshes, or whether it considers itself as volatile.

The life-time of triples that were generated in the reasoning process is set to be the minimum remaining life-time of the triples that fulfilled the precondition of the RDFS rule (see section 3.4). An update of triples triggers an update of the inferred triples as well.

3.2 Node Departure

A node departing or crashing does not only cause the expiry of triples but makes it also necessary that other nodes cover the area and data of the DHT that was previously occupied by the departing nodes. The DHT layer repairs the routing tables of the peers and thereby ensures that the Peer-to-Peer network remains connected. But besides correct message routing it is important that one or more nodes take over the responsibilities of the departing node. As the node storing the first replica of a triple is located in the immediate vicinity of the departing node on the DHT space (closest to the respective hash value of the triple), this node receives all following queries for the respective triple. As it stores the relevant data due to being rank 1 replica, we do not have to take special provision for failing nodes. Owing to the DHT schema used, the data from replication is automatically available to queries.

As a replica node became root node for a triple, the number of replica has decreased. The new root node is notified by the DHT layer that its leaf-set (set of nodes in the vicinity of the node regarding DHT space) has changed and sends triples to a new replica node. It is important to start this update procedure immediately as more nodes can fail until the next update of triples is issued.

3.3 Node Arrival

A new node disseminates all its local triples. But while the departure of nodes is handled almost automatically, the arrival of a new node is more complicated. At the moment a node joins the DHT, it receives queries for the covered ID space but lacks the data to answer these queries correctly. As the owner of a triple (the node having the triple in the *local triples* database) cannot notice the arrival of the new node and because it sends update messages at a low frequency, it is task of the replica nodes of the new node to provide it with all necessary data. These observe the new node as a new member in their leaf set and instantly send all triples to the new node for which it is to be considered a root node.

This strategy reduces the period of time during which a new node sends empty results enormously. If the amount of data that needs to be transferred is large, however, the transfer creates a rather long period during which the new node sends incomplete results. Because of that, the new node forwards queries to one of the replica picked at random during the transition period. The transition period ends when the new node does not receive any new triples from its replica for a certain time.

3.4 RDFS Rules

The RDF semantics document [11] describes how RDFS entailment can be seen as a set of rules which generate new RDF triples from existing ones. Several rules map a single triple to the creation of a new triple. An example for such a rule is rdfs4a where a triple (u, a, x) creates a new triple $(u, \texttt{rdf:type}, \texttt{rdfs:Resource})$. These rules are easy to evaluate with the local knowledge of peers.

Rule Name	Precondition	Generated Triple
rdfs2	$\underline{a}$, `rdfs:domain`, x u, $\underline{a}$, v	u, `rdf:type`, x
rdfs3	$\underline{a}$, `rdfs:range`, x u, $\underline{a}$, v	v, `rdf:type`, x
rdfs5	u, `rdfs:subPropertyOf`, $\underline{v}$ $\underline{v}$, `rdfs:subPropertyOf`, x	u, `rdfs:subPropertyOf`, x
rdfs7	$\underline{a}$, `rdfs:subPropertyOf`, b u, $\underline{a}$, y	u, b, y
rdfs9	$\underline{u}$, `rdfs:subClassOf`, x v, `rdf:type`, $\underline{u}$	v, `rdf:type`, x
rdfs11	u, `rdfs:subClassOf`, $\underline{v}$ $\underline{v}$, `rdfs:subClassOf`, x	u, `rdfs:subClassOf`, x

Fig. 2. RDF Schema rules

Figure 2 contains a list of those RDF Schema rules that contain two triples in the precondition. The underscored variables show that all 6 RDF Schema rules can be evaluated on nodes with only local knowledge. As triples are hashed by subject, predicate, and object, all triples contributing to the precondition of a rule can be found on the root node. At the example of rdfs7, we see that all triples with common subject a and predicate a are mapped to the same ID in the DHT space and are, therefore, stored at the same node.

4 Load Balancing

A major criticism against DHT based RDF stores is the issue of load-balancing. Owing to the nature of RDF triples, many collisions are unavoidable. It is for example obvious that the DHT will store many triples with predicate `rdf:type` as each individual has at least one type (see rule rdfs4a in [11]). As triples are hashed by subject, predicate, and object, a node responsible for the hash-value of `rdf:type` is subjected to a very high storage load. Reasoning aggravates this problem because transitive relations like `rdfs:subClassOf` can create many additional triples with predicate `rdf:type` for example. While the majority of URIs and literals will occur very rarely we expect that a few of them make up for a large proportion of all data. Figure 3 shows a figurative histogram of the occurrences of hash-values with a large peak for `rdf:type` predicates. Our goal

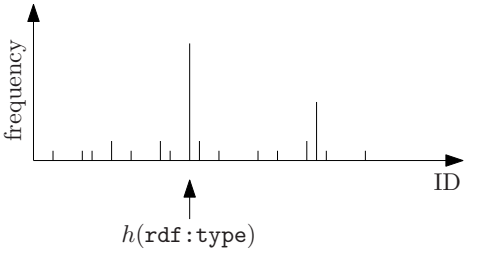


Fig. 3. Frequency distribution of hash values

is to distribute primarily the load of these peaks as we expect the IDs of small frequency to be sufficiently randomly distributed due to the hash function.

Several publications have addressed the issue of load-balancing in DHT networks already (see e.g. [12,13,14,15]). Neither of these strategies is however capable of solving the load-balancing issue in RDF stores. The first three strategies do not address the RDF specific problem of large discrete peaks due to collisions. The last strategy distributes the triples of a peak to n nodes. As n is an a priori defined constant we do not believe this to scale well enough. Furthermore, it creates an unnecessary overhead for rarely occurring values.

Cai et al. [7] address the issue of load-balancing in RDFPeers by not storing overly popular triples. This comes of course at the cost of possibly losing the correctness of the algorithm.

4.1 Overlay Tree

Load-balancing can be implemented by constructing an overlay tree over the DHT when nodes detect that they are overloaded. As the DHT range assigned to a node can change at runtime it is very difficult to store and maintain this range in an overlay tree. Instead, we build overlay trees only for discrete DHT positions like, for example, the one that stores triples with `rdf:type` predicate. In case a node detects that it is overloaded, it initiates the split of its most frequent hash-value.

While all triples colliding at a DHT position have the same hash value they can still be put into a total order by comparing the components that were not involved in the calculation of a hash value. This allows to determine a split element that divides the triples of the peak into those lexicographically smaller and those larger than the split element. One half of the triples is relocated to another node which has had little load during the overload detection phase. The other half remains on the current node. Subsequent queries are either processed by both nodes or, in case the split element allows to determine that a query regards only one half it, only by the respective node. Each half of the triples can be split again in case the respective node is still overloaded.

To understand the details of this load-balancing, one should first note, that the *replica* database is merged into the *received triples* database because their entries can be distinguished by the DHT at runtime. At the same time, we

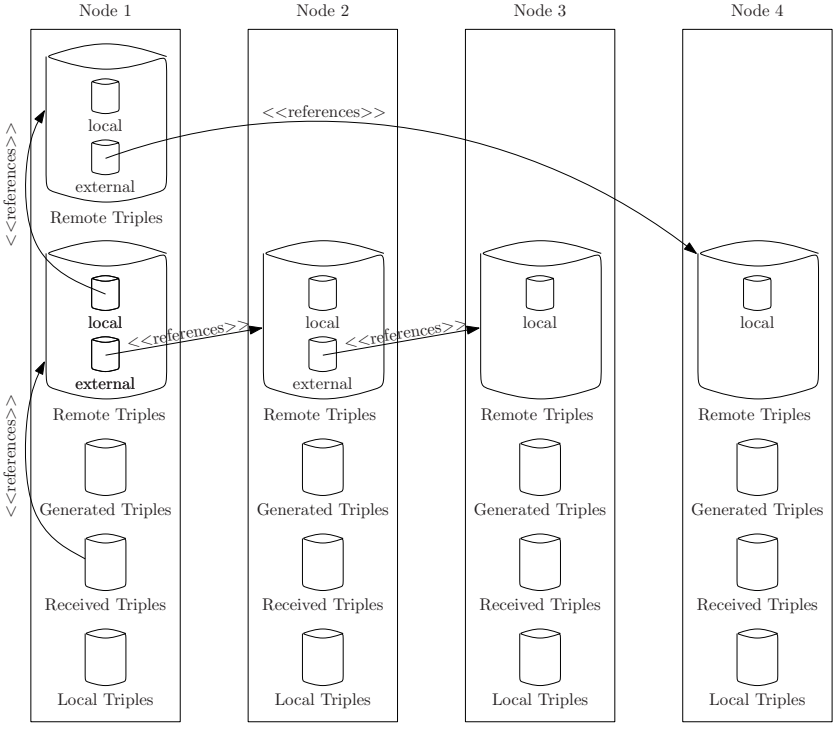


Fig. 4. Load-balancing with remote triples databases

introduce a new *remote triples* database type which is responsible for storing triples in the overlay tree structure. I. e. these databases store triples they are not necessarily responsible for according to their position in the DHT. Each node can have several of these *remote triples* databases as it can offer its capacity to several overloaded nodes.

Figure 4 illustrates a possible overlay tree. Let the four pillars represent peers 1 through 4, then the setup could be established by this history for example: First, node 1 recognized a load imbalance and replaced a frequent collision from its "normal" in-memory or persistent *received triples* database to a *remote triples* database. Half of the triples remained in the *local* part of the *remote triples* database and half of them were moved to node 2. The *external* database of node 1 was linked to the respective new *remote triples* database on node 2. At this point, node 2 had a *remote triples* database with a *local* part and no *external* part. Node 2 recognized that is was overloaded as well, and recursively split its data and moved half of the triples to node 3. At this point, node 1 recognized that it was still overloaded. Therefore, it split the *local* part of the *received triples* by creating a *remote triples* database that stores only half of the triples and links to another *remote triples* database on another node that stores the remaining triples. This effectively reduced the load of node 1 to a fourth of the initial load.

If node 1 receives a query for the ID that is stored in the *remote triple* database, it is possible that this query can be routed directly into either of the two branches. Otherwise, the query is forwarded into all branches and executed on each *local* part on the path to the leaves. The query module on the node who submitted the initial query assembles the results such that it is handled completely transparently for the upper layers. For this to work, each *local* part of a *remote triples* database that processes a query includes a field in the result set that indicates whether the query has been forwarded to an *external remote triples* database and in this case to which one. This allows the initial query submitter to delay further processing until all result sets have arrived.

This approach reduces the storage and query load of nodes by shifting some of the triples to other nodes and thereby distributing the load. As references can be stored as IP addresses of the target nodes, forwarding queries within the tree does not consume expensive DHT routing but can be done via direct communication. A query reaches its destination in $\mathcal{O}(\log N + d)$ steps, where N denotes the number of nodes in the DHT network and d denotes the depth of the tree.

4.2 Compensation of Node Failure

If a node in the overlay tree crashes, a whole branch of the tree vanishes. This makes load-balancing with the kind of overlay tree described rather fragile. For that reason, we need to do further modifications to the replication strategy and introduce a replicated overlay tree as depicted in figure 5. All nodes of the tree are of course members of the DHT ring but are connected at the same time to form a virtual tree.

We see that each node (represented by a white circle) in the depicted overlay tree is replicated twice (represented by gray circles). A parent knows its children and their replica, as well as its replica. Hence, if a node fails, a replica node can take its position and replicate data for a new backup node.

A difficult question is to decide whether a split of a node shall be initiated because this decision involves the current load of a root node and its replica if either of them being overloaded can create a bottle neck. For this reason we change the query routing strategy such that the first replica to receive a query does not intercept and process the query any more. Instead, a query is routed to the root node which then performs load-balancing among itself and the replica. Only if all of them are overloaded, a split is initiated. Figure 6 depicts this query routing with load-balancing. The very first root node decides not to process the query itself but to forward it to its rank 2 replica. This realizes that both of its children cover triples that are important to the result of the query and sends each of them a copy of the request message (solid arrow). At the same time it processes the query on local data and sends a result message to the node issuing the query (dashed arrow). The same happens recursively in the children.

As replica have to contain identical *remote triples* databases they share common split elements. If a split is initiated by a root node (whether it is the one at the root or at another position in the tree), it propagates a split element to

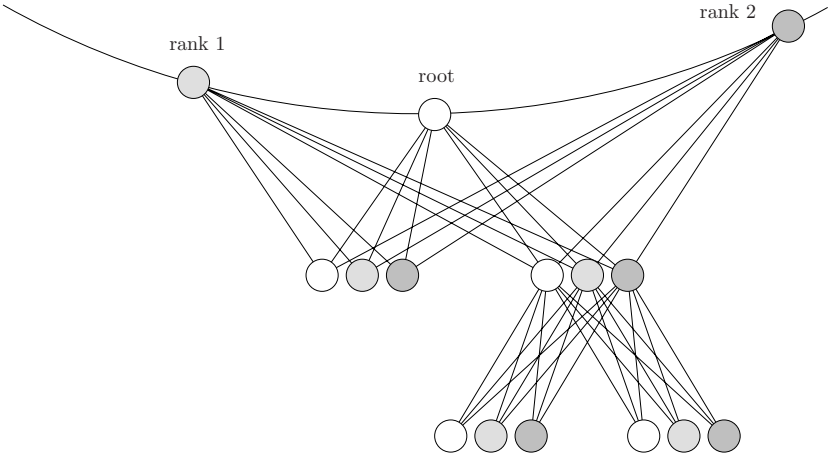


Fig. 5. Replicated overlay tree

the replica. Root node and replica look for possible children and communicate the children's addresses to each other. Then, they move triples to the children. Triples are not deleted until their arrival has been confirmed. That guarantees that queries can always be answered correctly.

4.3 Soft-State Updates

This overlay tree requires little modifications to soft-state updates and RDFS reasoning. Soft-state updates are rather easy to implement. An update message can be compared to the split element of a *remote triples* database and executed either on its *local* part or forwarded to the *external* part. If an update message addresses the *local* part it gets propagated to the replica nodes.

4.4 RDFS Rules

The necessary modifications for RDFS reasoning are not that obvious. While RDFS rules based on a single triple in the precondition remain trivial we have to pay attention to those rules with two precondition triples because the current strategy does not guarantee that two triples with common value (e.g. u in rdfs9) are located on the same node in the overlay tree. We see however, that each rule of figure 2 contains at least one rule with an RDFS URI in the predicate. These triples describe schema knowledge. As we anticipate that nodes store much less schema knowledge than actual data, it is possible to propagate the schema knowledge to all nodes of the overlay tree (i.e. flood the tree with the schema knowledge), while each actual triple remains to be stored on exactly one node (and its replica). Note that the propagated schema knowledge does not comprise the schema knowledge of the entire network but only that fraction which has

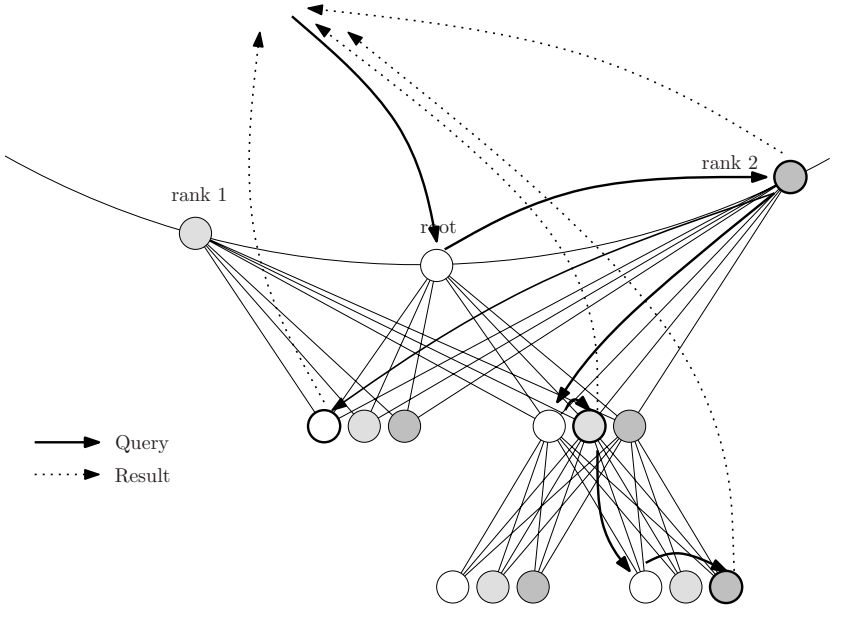


Fig. 6. Query routing in an overlay tree

a subject or object whose hash-value falls into the range of the root node of the overlay tree. With the strategy described, it is possible to combine correct RDFS reasoning, on the one hand, with the necessary load-balancing, on the other hand.

5 Conclusion

In order to address RDF Schema reasoning, which we consider necessary to exploit the real power of RDF, we have presented a data management strategy for *forward-chaining*. We furthermore addressed the problem of *robustness* to node failure and churn and presented a strategy to handle *load imbalances* that necessarily arise from the data distribution schema. While these are different dimensions of Peer-to-Peer based RDF stores, we have shown that our strategy allows to integrate them into a single system. Thereby, this paper complements our work published in [5] that focuses on intelligent and efficient query evaluation. It discusses technical issues such as reasoning and load balancing in detail that have not been addressed by other Peer-to-Peer based RDF stores so far (see [7,8]).

References

1. Manola, F., Miller, E.: RDF Primer. http://www.w3.org/TR/rdf-primer (2004)
2. Brickley, D., Guha, R.V.: RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema (2004)

3. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In: ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web, London, UK, Springer-Verlag (2002) 54–68
4. Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D.: Efficient RDF Storage and Retrieval in Jena2. In: Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases. (2003) 131–150
5. Heine, F.: Scalable P2P based RDF Querying. In: InfoScale '06: Proceedings of the 1st international conference on Scalable information systems, New York, NY, USA, ACM Press (2006) 17
6. Battré, D., Heine, F., Kao, O.: Top k RDF Query Evaluation in Structured P2P Networks. In Nagel, W., Walter, W., Lehner, W., eds.: Euro-Par 2006 Parallel Processing: 12th International Euro-Par Co nference. Volume 4128 of LNCS., Springer-Verlag (2006) 995–1004
7. Cai, M., Frank, M., Pan, B., MacGregor, R.: A Subscribable Peer-to-Peer RDF Repository for Distributed Metadata Management. Journal of Web Semantics: Science, Services and Agents on the World Wide Web **2** (2004) 109–130
8. Koubarakis, M., Miliaraki, I., Kaoudi, Z., Magiridou, M., Papadakis-Pesaresi, A.: Semantic Grid Resource Discovery using DHTs in Atlas. In: 3rd GGF Semantic Grid Workshop. (2006)
9. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmér, M., Risch, T.: EDUTELLA: a P2P networking infrastructure based on RDF. In: WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA. (2002) 604–615
10. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. Lecture Notes in Computer Science **2218** (2001) 329+
11. Hayes, P.: RDF Semantics. http://www.w3.org/TR/rdf-mt (2004)
12. Zhu, Y., Hu, Y.: Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems. IEEE Transactions on Parallel and Distributed Systems **16** (2005) 349–361
13. Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load Balancing in Structured P2P Systems. In: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03), Springer (2003)
14. Karger, D.R., Ruhl, M.: Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems. In: SPAA '04: Proceedings of the sixteenth annual ACM Symposium on Parallelism in Algorithms and Architectures, New York, NY, USA, ACM Press (2004) 36–43
15. Byers, J., Considine, J., Mitzenmacher, M.: Simple Load Balancing for Distributed Hash Tables. In: 2nd International Workshop on Peer-to-Peer Systems (IPTPS 03), Springer (2003) 31–35

Priority Based Load Balancing in a Self-interested P2P Network

Xuan Zhou and Wolfgang Nejdl

L3S Research Center
Expo Plaza 1, 30539 Hanover, Germany
{zhou,nejdl}@l3s.de

Abstract. A fundamental issue in P2P networks is that of distributing workload in a balanced way to optimize performance. Unfortunately, optimal load balance is difficult to realize, although it is easy to compute, because participants of P2P networks are usually self-interested and seek to maximize their individual utility without considering system-wide utility. In this paper, we study the influence of selfish behaviors on the load balance in P2P networks, and propose a priority based load balancing scheme to help a P2P network achieve better performance. The scheme is simple and effective, and can be easily used in a P2P environment. Besides presenting the theoretical foundation of our scheme, we also address its major implementation issues and conduct extensive experiments to verify its effectiveness and practicality.

1 Introduction

A peer-to-peer network is a network that relies on the collaboration of its participants, who pool their resources to benefit everyone. A fundamental issue in a P2P network is that of distributing workload in a balanced way to optimize its performance. In particular, given a number of peers who are offering the same services, the network needs to find an assignment of jobs to these peers such that the expected response time to these jobs is minimized. While a balanced workload distribution can always be found in principle, it is difficult to be realized in a P2P network where each participant is self-interested and rational, and seeks to minimize the response time to its own jobs without considering the efficiency of the whole network.

An example of the load balancing problem in a self-interested P2P network is illustrated in figure 1. Suppose peers A and B are providing the same service in a P2P network, and a number of peers wish to send n jobs to A or B to be processed. Suppose the response time for A to finish a job is always one minute, no matter how many jobs are arriving at A simultaneously. This can be represented by a function of response time, i.e. $R(x) = 1$, where x denotes the job arrival rate to A. In contrast, the response time for B to finish a job increases linearly with its job arrival rate, and its function of response time is $R(x) = \frac{x}{n}$. If the other peers are selfish and rational, they will choose to send their jobs to B, as it should minimize the response time to their own jobs. In the end, all

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 355–367, 2007.

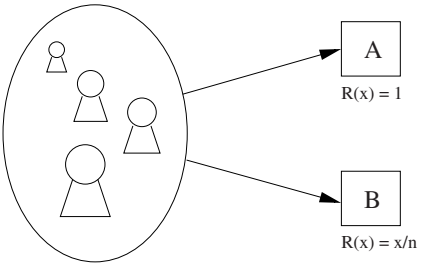


Fig. 1. An Example of Load Unbalance

n jobs are jammed at B, and the average response time to these jobs ends up at one minute. From the system-wide perspective, this is surely not the optimal solution: if $\frac{n}{2}$ jobs are submitted to A and the other $\frac{n}{2}$ jobs are submitted to B, the average response time could be as small as $\frac{3}{4}$ minute. Unfortunately, no peer is willing to sacrifice by sending his jobs to A. In economics, this phenomenon is known as "social dilemma", which describes the situation where all participants seek to maximize their individual utilities, but as a result get utilities which are worse than what they could achieve by cooperation.

Selfish behavior in large-scale computer networks has been extensively studied in recent years. Investigated areas include network routing [1,2,3], resource allocation [4], quality of service [5] and others [6]. Most of this work reveals that the cost of selfish behavior is usually not ignorable. Therefore, a number of strategies, such as pricing [7] and Stackelberg routing [8], have been proposed to cope with selfish behavior and help a network achieve better performance. However, none of these strategies can be directly applied to a P2P network, where centralized control is very difficult to establish. The investigation on *uncoordinated load balancing* by Subhash Suri et al [9,10] is the only work addressing the influence of selfish behavior on load balance in P2P networks. In their work they theoretically analyze the influence of selfish behaviors on load balance and give a worst-case bound to the performance loss caused by selfish behavior. However, no proposal is given for controlling selfish behavior to help a P2P network achieve better load balance.

In this paper, we introduce a priority based load balancing scheme for a self-interested P2P network. In this scheme, each peer divides its jobs into two classes, and assigns one class a higher priority than the other. By adjusting the proportion between the two classes of jobs, the network can optimize its workload distribution. The scheme is simple, effective, and suitable for P2P environments. We theoretically prove that this scheme can help a P2P network achieve optimal load balance. We also address the various issues of implementing this scheme to a real P2P network, and conduct extensive experiments to demonstrate its effectiveness.

The remainder of this paper is organized as follows. Section 2 defines the problem of load balancing in a self-interested P2P network, and discuss the possible solutions. Section 3 introduces our priority based load balancing scheme, and addresses the issues of implementing the scheme in real P2P networks. Section 4

experimentally evaluates the effectiveness and practicality of our scheme. Finally, we conclude this paper in section 5.

2 Problem Definition

We will use the term "uncoordinated load balancing" from [9] to describe the problem of balancing workload among selfish peers. This section presents a model of uncoordinated load balancing and discusses the possible approaches to achieve load balance in a self-interested P2P network.

2.1 The Model

Consider a P2P network, where n peers are providing the same kind of service, and the other peers periodically request and receive services from the n peers. We call these n peers *servers*, and denote them by $S_0, S_1, ..., S_{n-1}$. We call all the other peers in the network *clients*. Each server S_i would display different efficiency in processing its received jobs. This efficiency can be expressed by a load-dependent function of response time, which is denoted by $R_i(x)$, where x is the job arrival rate (load) on the server. In this paper, we assume that $R_i(x)$ is always nondecreasing and convex, as it is regarded true in common real-world systems [3]. Suppose the total job arrival rate to the n servers is λ. Then, the triple (n, R, λ) forms an instance of the uncoordinated load balancing problem. A solution to this problem is an assignment of the total job arrival rate (load) over the n servers, which is denoted by $a = (\lambda_0, \lambda_1, ..., \lambda_{n-1})$, where λ_i is the job arrival rate to S_i and $\lambda_0 + \lambda_1 + ... + \lambda_{n-1} = \lambda$. The cost $C(a)$ of a solution a is the total response time of processing the λ jobs. That is,

$$C(a) = \sum_{i=0}^{n-1} R_i(\lambda_i)\lambda_i$$

When the clients in the P2P network are all selfish, they seek to minimize the response time to their own jobs, without regard to the overall cost of the network. As a result, each job will be sent to the peer that would incur the shortest response time. In the end, the assignment of the job arrival rate λ will end in a *Nash equilibrium*, where no job can improve its response time by switching to a different server. In this Nash equilibrium, the servers that are assigned a positive workload will display equal response time. This can be formally presented as follows.

Lemma 1. A load assignment $a = (\lambda_0, \lambda_1, ..., \lambda_{n-1})$ for instance (n, R, λ) is at Nash equilibrium if and only if for every $\lambda_i > 0$, $R_i(\lambda_i) \leq R_j(\lambda_j)$. ■

According to Roughgarden's results on *selfish routing* [3], if $R_i(.)$ is nondecreasing, there is a unique Nash equilibrium of load assignment for every instance (n, R, λ).

The solution at the Nash equilibrium is not necessarily the optimal solution that has the minimal cost $C(a)$. The optimal solution is indeed the solution to the following nonlinear program (NLP) [11].

$$\begin{aligned} min: C(a) &= \textstyle\sum_{i=0}^{n-1} R_i(\lambda_i)\lambda_i \\ subject\ to: &\textstyle\sum_{i=0}^{n-1} \lambda_i = \lambda, \\ &\lambda_i > 0 \end{aligned}$$

It can be deduced (see [3]) that, given an instance (n, R, λ), where $R(.)$ is convex, the optimal solution should satisfy the following conditions.

Lemma 2. Suppose $R_i^*(x) = \frac{d}{dx}(R_i(x)x)$. A load assignment $a = (\lambda_0, \lambda_1, ..., \lambda_{n-1})$ is optimal if and only if for every $\lambda_i > 0$, $R_i^*(\lambda_i) \leq R_j^*(\lambda_j)$. ■

In other words, $C(a)$ reaches its minimum when the marginal benefit $(R_i^*(\lambda_i))$ of decreasing the load on server S_i is at most the marginal cost $(R_j^*(\lambda_j))$ of increasing the load on any other peer S_j.

We can apply the model to the example in figure 1. (Suppose n denotes the total job arrival rate instead of the number of arriving jobs.) Through Lemma 1, we find that the load assignment on Servers A and B at the Nash equilibrium is $(\lambda_A = 0, \lambda_B = n)$, and the resulting cost is $C(a) = n$. Through Lemma 2, we find that the optimal load assignment is $(\lambda_A = \frac{n}{2}, \lambda_B = \frac{n}{2})$, and its cost is $C(a) = \frac{3}{4}n$.

The maximal ratio between the cost of Nash equilibrium solution and the cost of the optimal solution is called the *price of anarchy*. Some earlier work [3] has shown that selfish behavior can have significant influence on the system-wide performance.

If we consider the optimal load assignment as the balanced one, then the load assignment at the Nash equilibrium would be unbalanced. The objective of our research is to design a mechanism for the P2P network that could affect the clients' selfish behaviors, so that the resulting load assignment will be more balanced.

2.2 Possible Solutions

To achieve load balance in a P2P network, obviously we cannot count on the selfish clients to collaboratively distribute their workload in a balanced way. However, we can resort to the cooperativeness of servers. First, the servers have no incentive to disturb the workload distribution, as they benefit nothing from it. Second, as users of a P2P network, they can have improved utility if the network works more efficiently. Therefore, we regard that the servers of a P2P network are usually benign and willing to take action to balance the workload.

Under this assumption, a possible approach to load balance is to calculate the proportion of workload on each server in the optimal load assignment, and restrict each server from taking more workload than that proportion. However, as the workload in a P2P network is changing frequently, it is impractical to

estimate the optimal load assignment in real time. If the workload is overestimated, the approach will not work. If it is underestimated, some jobs would be starved severely. Therefore, this approach is not an appropriate solution.

In this paper, we propose a priority based load balancing approach, which achieves optimal load balance by assigning priorities to workload.

3 Priority Based Load Balancing

Section 3.1 introduces the priority based load balancing scheme. Sections 3.2 and 3.3 give its theoretical foundation. Section 3.4 presents implementation issues.

3.1 The Scheme

In our scheme, each server divides the received jobs into two classes – *1st class* and *2nd class*. The jobs in the 1st class have strictly higher priority than the jobs in the 2nd class, in the sense that a server will always first process the 1st class jobs prior to 2nd class jobs. To differentiate between the two classes of jobs, a server selects a number of clients in the network and assigns each of them a *1st class quota*, which specifies the number of 1st class jobs it can submit to the server per time unit. Later on, the jobs arriving from these clients within their 1st class quotas will be regarded by the server as the 1st class jobs, and the other jobs will fall into the 2nd class.

Applying this priority based approach to our model, the job arrival rate λ_i on each server S_i becomes the sum of the arrival rates of the two classes of jobs, which can be represented by $\lambda_i = \lambda_i^1 + \lambda_i^2$, where λ_i^1 denotes the arrival rate of the 1st class jobs and λ_i^2 denotes the arrival rate of the 2nd class jobs. As we will show subsequently, by setting appropriate 1st class quotas on each server, we can guarantee that, for all the servers, the response time of any 1st class jobs is always shorter than that of 2nd class jobs. Therefore, whenever a client is issuing jobs, it will first use up its 1st class quota, so as to minimize the response time. Hence, λ_i^1 is directly determined by the amount of the 1st class quota S_i assigns to its clients. At the same time, each client will also seek to minimize the response time to its 2nd class jobs. As a result, the assignment of λ_i^2 will end in a Nash equilibrium, where no 2nd class job can improve its response time by switching to a different server. In this scheme, the final load assignment $\lambda_i^1 + \lambda_i^2$ on each server can be adjusted by adjusting the 1st class quotas. The basic idea of our priority based load balancing is to appropriately set the 1st class quotas on each server to optimize the load distribution. In the next section, we prove that we can achieve the optimal load balance through this scheme.

3.2 Finding Optimal Solution

Suppose the original function of response time on server S_i is $R_i(\lambda_i)$, where λ_i is the job arrival rate to the server, i.e. $\lambda_i = \lambda_i^1 + \lambda_i^2$. As the 1st class jobs have strictly higher priority than the 2nd class jobs, the response time for the 1st class

jobs will not be affected by the 2nd class jobs present. Therefore, the response time for the 1st class jobs on server S_i will be $R_i(\lambda_i^1)$ no matter what λ_i^2 is. At the same time, the 2nd class jobs will bear all additional overhead incurred by the co-existence of the 1st and 2nd classes of jobs. Suppose the response time for the 2nd class jobs is R_i^2, this can be represented by the following equation.

$$R_i(\lambda_i^1 + \lambda_i^2) \times (\lambda_i^1 + \lambda_i^2) = R_i(\lambda_i^1)\lambda_i^1 + R_i^2\lambda_i^2$$

Solving this equation, we get the response time function for the 2nd class jobs as

$$R_i^2(\lambda_i^1, \lambda_i^2) = \frac{R_i(\lambda_i^1 + \lambda_i^2) \times (\lambda_i^1 + \lambda_i^2) - R_i(\lambda_i^1)\lambda_i^1}{\lambda_i^2}.$$

As we have stated earlier, the arrival rates of the 1st class jobs $(\lambda_0^1, \lambda_1^1, ..., \lambda_{n-1}^1)$ are determined by the 1st class quotas issued by the servers. Meanwhile, each server will seek to minimize the response time to their 2nd class jobs, so that the arrival rates of the 2nd class jobs $(\lambda_0^2, \lambda_1^2, ..., \lambda_{n-1}^2)$ will end in a Nash Equilibrium, which satisfies the following criterion.

Lemma 3. Given an instance (n, R, λ) and the distribution of the 1st class jobs $(\lambda_0^1, \lambda_1^1, ..., \lambda_{n-1}^1)$, a assignment of the 2nd class jobs $(\lambda_0^2, \lambda_1^2, ..., \lambda_{n-1}^2)$ is at Nash equilibrium if and only if for every $\lambda_i^2 > 0$, $R_i^2(\lambda_i^1, \lambda_i^2) \leq R_j^2(\lambda_j^1, \lambda_j^2)$. ■

The optimal load assignment is the assignment that incurs the smallest overall cost $C(a) = \sum_{i=0}^{n-1} R_i(\lambda_i^1 + \lambda_i^2)(\lambda_i^1 + \lambda_i^2)$. Following the spirit of Lemma 2, the criterion of the optimal solution is:

Lemma 4. Suppose $R_i^*(x) = \frac{d}{dx}(R_i(x)x)$. A load assignment $a = (\lambda_0^1 + \lambda_0^2, \lambda_1^1 + \lambda_0^2, ..., \lambda_{n-1}^1 + \lambda_0^2)$ is optimal if and only if for every $\lambda_i^1 + \lambda_i^2 > 0$, $R_i^*(\lambda_i^1 + \lambda_i^2) \leq R_j^*(\lambda_j^1 + \lambda_j^2)$. ■

The objective of our scheme is to find a assignment of the 1st class jobs $(\lambda_0^1, \lambda_1^1, ..., \lambda_{n-1}^1)$, such that the load assignment at the Nash Equilibrium is exactly the optimal load assignment. The objective can be formally represented as the following problem:

$$\begin{array}{ll} \textit{find} & \lambda_0^1, \lambda_1^1, ..., \lambda_{n-1}^1 \\ \textit{that satisfy} & 1.\ \sum_{i=0}^{n-1}(\lambda_i^1 + \lambda_i^2) = \lambda \\ & 2.\ \lambda_i^1 \geq 0 \ \text{ and } \ \lambda_i^2 \geq 0 \\ & 3.\ R_i^2(\lambda_i^1, \lambda_i^2) \leq R_j^2(\lambda_j^1, \lambda_j^2) \quad \text{if} \quad \lambda_i^2 > 0 \\ & 4.\ R_i^*(\lambda_i^1 + \lambda_i^2) \leq R_j^*(\lambda_j^1 + \lambda_j^2) \text{ if } \lambda_i^1 + \lambda_i^2 > 0 \end{array} \tag{1}$$

Figure 2 plots the curves of the two functions $R_i(x)x$ and $R_j(x)x$, where $R_i(x)$ and $R_j(x)$ are the response time functions of S_i and S_j respectively. Four angles, which are determined by the job arrival rates λ_i^1, λ_i^2, λ_j^1 and λ_j^2, are drawn out in figure 2. $\angle a_i$ is the slope angle of the line connecting $\left(\lambda_i^1, R_i(\lambda_i^1) \times \lambda_i^1\right)$ and $\left(\lambda_i^1 + \lambda_i^2, R_i(\lambda_i^1 + \lambda_i^2) \times (\lambda_i^1 + \lambda_i^2)\right)$. $\angle a_j$ is the slope angle of the line connecting $\left(\lambda_j^1, R_j(\lambda_j^1) \times \lambda_j^1\right)$ and $\left(\lambda_j^1 + \lambda_j^2, R_i(\lambda_j^1 + \lambda_j^2) \times (\lambda_j^1 + \lambda_j^2)\right)$. $\angle b_i$ is the slope angle

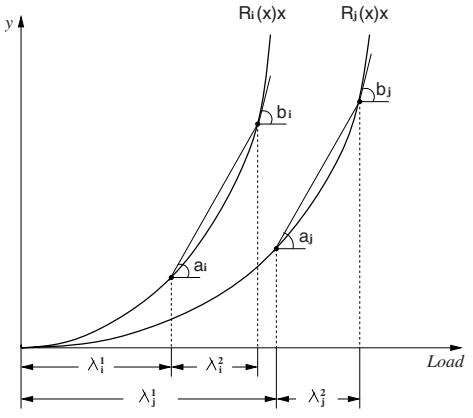


Fig. 2. Problem Interpretation

of $R_i(x)x$'s tangent at point $\left(\lambda_i^1+\lambda_i^2, R_i(\lambda_i^1+\lambda_i^2)\times(\lambda_i^1+\lambda_i^2)\right)$. $\angle b_j$ is the slope angle of $R_j(x)x$'s tangent at point $\left(\lambda_j^1+\lambda_j^2, R_i(\lambda_j^1+\lambda_j^2)\times(\lambda_j^1+\lambda_j^2)\right)$. As $\tan(a_i) = R_i^2(\lambda_i^1,\lambda_i^2)$ and $\tan(a_j) = R_j^2(\lambda_j^1,\lambda_j^2)$, condition 3 could be represented as $\angle a_i \leq \angle a_j$. As $\tan(b_i) = R_i^*(\lambda_i^1+\lambda_i^2)$ and $\tan(b_j) = R_j^*(\lambda_j^1+\lambda_j^2)$, condition 4 could be represented as $\angle b_i \leq \angle b_j$. Then, the above problem could be interpreted as:

$$\begin{array}{lll} \textit{find} & \lambda_0^1, \lambda_1^1, ..., \lambda_{n-1}^1 & \\ \textit{that satisfy} & 1.\ \sum_{i=0}^{n-1}(\lambda_i^1+\lambda_i^2) = \lambda & \\ & 2.\ \lambda_i^1 \geq 0 \ \text{ and } \ \lambda_i^2 \geq 0 & \\ & 3.\ \angle a_i \leq \angle a_j & \text{if} \quad \lambda_i^2 > 0 \\ & 4.\ \angle b_i \leq \angle b_j & \text{if } \lambda_i^1+\lambda_i^2 > 0 \end{array} \tag{2}$$

To solve this problem, we proceed as follows:

1. *Determine* $\lambda_i = \lambda_i^1 + \lambda_i^2$*:* Find a load assignment $a = (\lambda_0, \lambda_1, ..., \lambda_{n-1})$ that satisfies: (a) $\sum_0^{n-1}\lambda_i = \lambda$; (b) $\lambda_i \geq 0$ for all i; (c) $\angle b_i \leq \angle b_j$ if $\lambda_i > 0$.
2. *Determine* $\angle a_i$*:* Select the minimum $\angle a$ that satisfies: (a) $\angle a < \angle b_i$ for all i, (This guarantees that $\lambda_i^2 > 0$); (b) If $\lambda_i > 0$, then $\tan a > R_i(\lambda_i)$. Then, set $\angle a_i = \angle a$ for all $\lambda_i > 0$.
3. *Determine* λ_i^1*:* If $\lambda_i > 0$, λ_i^1 is uniquely determined by $\angle a_i$ and λ_i; if $\lambda_i = 0$, set $\lambda_i^1 = 0$.

As function $R_i(x)$ is nondecreasing and convex, function $R_i(x)x$ must be nondecreasing and convex too. Then, we can prove that the above three steps can always find a solution to Problem (2), as well as Problem (1). With the resulting $\lambda_0^1, \lambda_1^1, ..., \lambda_{n-1}^1$, we can ensure that the load assignment at the Nash Equilibrium is exactly the optimal load assignment. Then the 1st class quotas on each server can be set accordingly to enforce the $\lambda_0^1, \lambda_1^1, ..., \lambda_{n-1}^1$, so that the P2P network can achieve optimal load balance.

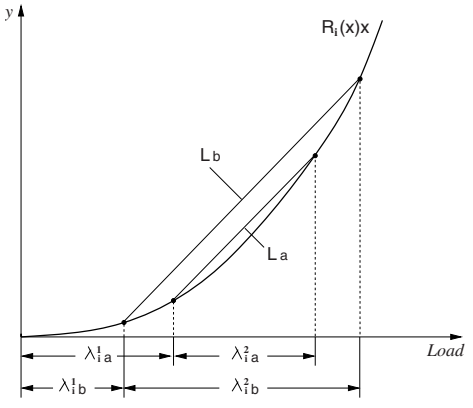


Fig. 3. Tuning Strategy

In addition, step 2 guarantees that if $\lambda_i^1 > 0$ then $\lambda_i^2 > 0$. That is to say, if a server receives 1st class jobs, it will certainly also receive 2nd class jobs. As all the 2nd class jobs have the same response time at the Nash equilibrium, we can guarantee that the response time of any 1st class job is shorter than that of the 2nd class jobs. This justifies our previous assumption that a client would always first use up its 1st class quota in order to minimize the received response time.

3.3 Tuning Method

While we have theoretically proved that our scheme can achieve optimal load balance, it is still difficult to realize the scheme in a practical P2P system. On the one hand, it is infeasible to accurately estimate the workload in a P2P network in real time. On the other hand, it is very costly to obtain the response time functions of all peers. Both make it difficult to compute the optimal load assignment. Thus, a more practical strategy of load balancing can be composed of two steps. First, when a P2P network starts or when a new server joins the network, the server contacts some other servers to initialize its 1st class quota. Second, at running time each server continually observes the network and tunes its workload, to draw the system-wide load assignment closer to the optimal one.

During the tuning step, each peer can increase or decrease its workload by decreasing or increasing the 1st class quota assigned to its clients. This is illustrated in figure 3, which plots the curve of function $R_i(x)x$, where $R_i(x)$ is the response time function of S_i. ($R_i(x)x$ is always nondecreasing and convex.) The two points $(\lambda_i^1, R_i(\lambda_i^1) \times \lambda_i^1)$ and $(\lambda_i^1 + \lambda_i^2, R_i(\lambda_i^1 + \lambda_i^2) \times (\lambda_i^1 + \lambda_i^2))$ on the curve are connected by a line L. The slope angle of L is actually $\angle a_i$ in figure 2. According to the condition of the Nash equilibrium in Lemma 3, the slope of L should be the same for every $R_i(x)x$, as long as $\lambda_i^2 > 0$. For this reason, if we decrease the arrival rate of the 1st class jobs on S_i from λ_{ia}^1 to λ_{ib}^1, line L will move parallel from L_a to L_b. Then, the total job arrival rate will be increased

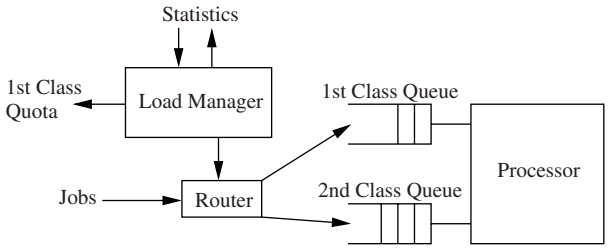


Fig. 4. Architecture

from $\lambda_{ia}^1 + \lambda_{ia}^2$ to $\lambda_{ib}^1 + \lambda_{ib}^1$. This reveals the following intuitive principles of load assignment tuning:

1. When a server decreases its 1st class quota, it shortens the response time of its 2nd class jobs $R_i^2(\lambda_i^1, \lambda_i^2)$, so that the server will attract more workload $\lambda_i^1 + \lambda_i^2$;
2. When a server increases its 1st class quota, it enlarges the response time of its 2nd class jobs $R_i^2(\lambda_i^1, \lambda_i^2)$, so that the server will get less workload $\lambda_i^1 + \lambda_i^2$.

Such tuning is limited by an upper bound and a lower bound. Namely, when $\lambda_i^1 = 0$, the server will receive the maximum workload; when $\lambda_i^1 = \lambda_i$, the server will receive the minimum workload. In practice, a server can probe the status of some other peers in the network to assess whether itself is overloaded or under-loaded, and tune its workload accordingly.

3.4 Implementation Issues

Figure 4 shows the architecture of a server in a P2P network. When a job is sent to the server, a router decides whether the job is in the 1st class or in the 2nd class, and routes the job to the corresponding queue. The processor always first processes the jobs in the 1st class queue. Only when the 1st class queue is empty, it turns to process the jobs in the 2nd class queue. To prevent starvation, some 2nd class jobs will be upgraded to the 1st class if their waiting time exceed a certain threshold. However, the system should ensure that the efficiency of processing the 1st class of jobs is least affected by the 2nd class jobs.

The load manager is the key component responsible for monitoring and adjusting the workload of the server. It continuously collects statistic information to learn the workload distribution in the network, and correspondingly adjusts the workload of its host server to help improve the load balance of the network. Meanwhile, it also monitors the status of its host server, and reports the information to the network.

In order to achieve load balance, a server needs to assign a proper amount of 1st class quota to its clients. During the quota assignment, the server issues quota certificates to some randomly selected clients. Each quota certificate contains the identity of the target client, the time of expiration, and the quota indicating the

number of 1st class jobs the client can submit in a unit time. To protect its integrity, each certificate is signed by the server's private key. When a client submits a 1st class job to the server, it attaches the quota certificate to the job, so that the router can classify it into the 1st class. If the 1st class jobs from a client exceed its quota, the extra jobs will be classified into the 2nd class.

Initializing 1st Class Quota: When a P2P network is started or when a new server joins the network, the server needs to correctly initialize its 1st class quota, so as to attract an appropriate amount of workload. As it is infeasible to accurately compute the optimal load assignment of a P2P network, the initial 1st class quota is estimated through sampling, which can performed through the following steps:

1. Choose a set of sample servers, and compute the optimal load assignment over the sample servers through the process introduced in Section 3.2.
2. Estimate the optimal workload on the new server based on the load assignment over the sample servers.
3. The initial 1st class quota of the new server can be estimated through its optimal workload and the load distribution over the sample servers.

Starting from this initial quota, the server can later tune its workload to further improve the load balance of the network.

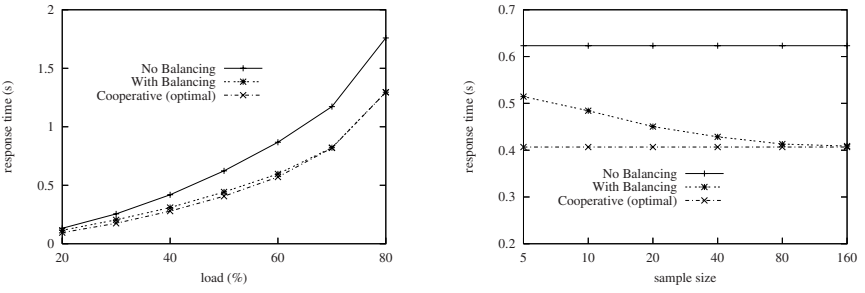
Load Assignment Tuning: In load assignment tuning, a server probes the status of other servers to learn whether itself is overloaded or under-loaded, and increases or decreases its 1st class quota correspondingly. The information obtained in each probe will include the response time function and the current workload of each visited server. A server can conduct tuning through the following steps.

1. Calculate the local optimal load assignment over the group of servers that comprise all the probed servers and itself.
2. Compare its current workload against the optimal one obtained in step 1.
3. Tune its 1st class quota through the method introduced in section 3.3, such that its workload reaches the optimal workload.

Intuitively, when each server keeps tuning their 1st class quotas, the load assignment in the P2P network will gradually converge to the optimal solution. This will be experimentally verified in the next section.

4 Experimental Evaluation

We evaluate the proposed load balancing scheme through simulation. We simulate a P2P network that is composed by millions of participants, in which 1000 participants are offering the same service and are considered as servers. The other participants can select any server to submit their jobs. The response time function on each server follows that of M/M/1 queueing system. Namely, it is



(a) varying load (b) varying sample size

Fig. 5. Effectiveness of Initial Load Assignment

in the form $\frac{1}{\mu-\lambda}$, where μ denotes the service rate of a server and λ denotes the job arrival rate to the server. The distribution of the service rates μ on the 1000 servers follows the Pareto distribution [12]. Due to limited space, only a part of the experiment results are reported.

4.1 Initial Load Assignment

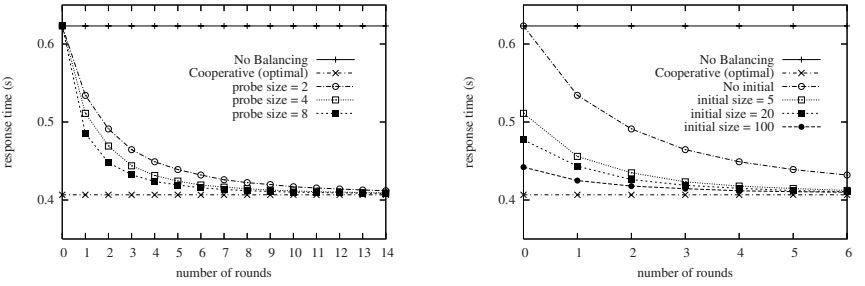
The goal of the first set of experiments is to study how much the initial load assignment of our load balancing scheme could improve the performance of the selfish P2P network. The process of initialization follows that in section 3.4.

In the first experiment, we fix the sample size at 20 servers. We vary the workload on the whole network from 20% to 80%, and measure the average response time resulted by the initial load assignment. We also compare this response time against that of the selfish network without using our scheme and that of the cooperative network where optimal load assignment is achieved. The results are shown in figure 5 (a). As expected, the initial load assignment by our load balancing scheme could remarkably improve the performance of the selfish P2P network (by 10% to 30%).

In the second experiment, we fix the workload of the network at 50%, and vary the sample size from 5 servers to 160 servers. Then we measure the average response time resulting from the initial load assignment. Figure 5 (b) shows the results. As expected, a larger sample could enable the servers to obtain a more complete overview of the P2P network, so that the servers could estimate their 1st class quotas more accurately and achieve better load balance. When the sample increases to 100 servers, the initial load assignment almost reaches the optimal load assignment.

4.2 Load Assignment Tuning

This set of experiments is intended to study how the load assignment tuning of our scheme could improve the load balance of the simulated P2P network.



(a) Without Initialization (b) With Initialization

Fig. 6. Effectiveness of Load Assignment Tuning

When conducting tuning, a server periodically probes some other servers to assess whether itself is overloaded or under-loaded, and tunes its 1st class quota correspondingly.

In the first experiment, we do not use the initial load assignment, but set all the 1st class quotas to 0. Then, we conduct load assignment tuning on each server by repeating the process in section 3.4. The tuning is performed round after round. In each round, each server tunes its workload once. After each round, we set the load assignment to the Nash equilibrium defined by Lemma 3, and measure the average response time to accomplish a job. The probe size of the tuning varies from 2 servers to 8 servers. Figure 6 (a) plots the varying of average response time when the tuning proceeds. We can see that the load assignment obviously converges to the optimal one as the tuning going on. In addition, a larger probe size amplifies the effects of tuning – with the probe size of 8 servers, the convergence could be twice as fast as with probe size of 2 servers.

In the second experiment, we study the effects of combining initial load assignment and tuning. We fix the probe size of tuning at 2 servers, and vary the sample size of the initial load assignment from 5 servers to 100 servers. Then, we measure the average response time after each round of tuning. The results are shown in figure 6 (b). As expected, when the initial load assignment is used, the load assignment could converge much faster to the optimal load assignment. The improvement could be further magnified by using a larger sample size for initial load assignment.

5 Conclusion

In this paper, we proposed a priority based load balancing scheme for a self-interested P2P network, in which each participant is selfish and seeks to minimize its individual response time without considering the system-wide performance. We theoretically proved the correctness of our scheme, and conducted experiments to show that our scheme helps P2P networks to achieve better load balance in an efficient way.

References

1. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: 16th STACS. (1999) 404–413
2. Czumaj, A., Vocking, B.: Tight bounds for worstcase equilibria. In: 13th SODA. (2002) 413–420
3. Roughgarden, T., Tardos, E.: How bad is selfish routing. Journal of ACM **49** (2002) 235–259
4. Lazar, A., Orda, A., Pendarakis, D.: Virtual path bandwidth allocation in multi-user networks. IEEE/ACM Transactions on Networking **5** (1997) 861–871
5. Yaiche, H., Mazumdar, R., Rosenberg, C.: A game theoretic framework for bandwidth allocation and pricing of elastic connections in broadband networks: theory and algorithms. IEEE/ACM Transactions on Networking **8** (2000) 667–678
6. Altman, E., Boulogne, T., Azouzi, R., Jimenez, T.: A survey on networking games in telecommunications. Computers and Operations Research **33** (2006) 286–311
7. Cole, R., Dodis, Y., Roughgarden, T.: Pricing network edges for heterogeneous selfish user. In: 35th STOC. (2003) 521–530
8. Roughgarden, T.: Stackelberg scheduling strategies. SIAM Journal of Computing **33** (2003) 332–350
9. Suri, S., Toth, C., Zhou, Y.: Uncoordinated load balancing and congestion games in p2p systems. In: 3rd IPTPS. (2004) 123–130
10. Suri, S., Toth, C.D., Zhou, Y.: Selfish load balancing and atomic congestion games. In: SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures. (2004) 188–195
11. Peressini, A.L., Sullivan, F.E., J. J. Uhl, J.: The Mathematics of Nonlinear Programming. Springer-Verlag (1993)
12. Adamic, L.A.: (Zipf, power-laws, and pareto - a ranking tutorial) http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html.

A Self-organized P2P Network for an Efficient and Secure Content Location and Download

J.P. Muñoz-Gea, J. Malgosa-Sanahuja, P. Manzanares-Lopez,
J.C. Sanchez-Aarnoutse, and J. Garcia-Haro

Department of Information Technologies and Communications
Polytechnic University of Cartagena
Campus Muralla del Mar s/n, E-30202, Cartagena, Spain
{juanp.gea, josem.malgosa, pilar.manzanares, juanc.sanchez,
joang.haro}@upct.es

Abstract. In this paper, we propose a hybrid and self-organized P2P overlay network. Unstructured networks have easy mechanisms (human readable form) to search for a content, but the location process does not take advantage of the distributed system nature. Conversely, structured networks can efficiently (completely distributed system) locate items, but the searching process is not user friendly. We suggest the usage of a structured network for managing a hierarchical unstructured network, where some peers take the role of *SuperPeers* periodically. Therefore, although the users perform the search in an unstructured way, the location process is performed in a structured way. Moreover, the network is self-organized dynamically. This self-organized nature gives the network the necessary reliability in front external attacks. Some aspects of the proposed system are evaluated analytically and with simulations.

1 Introduction

There are two types of P2P overlay networks: structured and unstructured. The technical meaning of structured is that the P2P overlay network topology is tightly controlled. Such structured P2P systems have a property that consistently assigns uniform random *NodeIds* to the set of peers into a large space of identifiers. With this identifier, the overlay network places the terminal in a specific position into a graph. Typical representatives of this approach are Chord [1] and Pastry [2]. In contrast, unstructured P2P overlay networks are ad-hoc in nature, and do not have the possibility of being unified under a common platform. Gnutella [3] for instance offers a non-hierarchical P2P approach. However, the most popular unstructured networks organize peers in a random graph in a hierarchical manner, by means of special nodes called *rendezvous* points, with network management functions. As typical representatives of this design direction the architectures of eDonkey [4] and KaZaa [5] can be considered.

In structured networks every content has an identifier, called data *Key*. This identifier is mapped to a peer, and its associated content is placed at this specified location. This structured graph enables efficient discovery of data items, using

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 368–375, 2007.

the given *Keys*, and is much easier to reorganize when changes occur (terminals registering and leaving). Consequently, the overlay network is more scalable and robust. However, in this simple form, this class of systems does not support complex queries. They only support exact-match lookups: one needs to know the exact *Key* of a data item to locate the node responsible for storing that item. On the other hand, unstructured non-hierarchical architectures support complex queries, but the flooding mechanism used for querying makes these networks unscalables. In unstructured hierarchical architectures *rendezvous* nodes store the content list of the nodes placed around them. The search requests are sent to the *rendezvous* node, and this evaluates the query locally on its own knowledge. If the content is not located in the *rendezvous*, most of the available networks also use flooding search on the *rendezvous* nodes to query content stored by overlay peers.

In this work we have designed a hybrid P2P system that shares the advantages of both types of networks. The searching process uses an unstructured hierarchical architecture in order to support complex queries, but the nodes of the system are organized in a structured network that makes the system more scalable and robust. The structured organization of the network enables the dynamical definition of the *SuperPeers*. Moreover, when a content is located, the downloading process can be performed hop-by-hop or peer-to-peer, giving the user an extra security (anonymity) issue.

The remainder of the paper is organized as follows: Section 2 describes the system proposed in this paper in detail. Section 3 shows the simulation and analytical results. Section 4 presents several related works and finally, Section 5 concludes the paper.

2 System Description

Our proposal tries to define a hybrid P2P system able to search in an unstructured way (supporting complex queries) while all the nodes are immersed in a structured overlay network. All the nodes are assigned automatically into different sub-groups. Every sub-group is managed by a level-1 *SuperPeer*, which is the node with the best performances in terms of CPU, bandwidth and reliability, among the sub-group's members. The level-1 *SuperPeers* are also grouped into higher level sub-groups, managed by level-2 *SuperPeers*. This process recursively continues up to the highest level sub-group.

All the level-1 *SuperPeers* maintain a database that registers the content list of all the nodes in their sub-groups. The rest of the *SuperPeers* do not need to maintain this information, since they can request it to their sub-group's members when needed. In order to search for content, the user's node will send the search parameters to its level-1 *SuperPeer*, and this one will return information about who has the content in its sub-group. If the search fails, the level-1 *SuperPeer* will send the request to its associated higher level *SuperPeer*. If it is necessary, this process will continue up to the highest level *SuperPeer*.

2.1 Initializing and Joining the Network

Every node obtains its *NodeId* applying a well-known hash function (MD5 or SHA-1) to its MAC or IP address. In order to join the structured overlay network, every node uses an underlying service capable of obtaining the IP address of an existing node in the same network. The new node contacts the existing node in order to link the structured network, and also obtains its *SubgroupId* that identifies the sub-group to which this node belongs. The new node will use this identifier to try to join that sub-group.

2.2 Joining a Sub-group

Next, the node must look for its corresponding *SuperPeer*. It uses the structured network to locate the node whose *NodeId* fits with the *SubgroupId* previously obtained. This node knows the IP address of the sub-group's *SuperPeer* (see Section 2.3). The new node contacts that *SuperPeer*, and request to join its sub-group. Each sub-group has a maximum number of nodes. If there is room, the node sends its content list to it.

2.3 *SuperPeer* Assignment

The previous process fails when the node tries to join a full sub-group. In this case, a new (randomly generated) *SubgroupId* is assigned to the node by the *SuperPeer*, and it becomes the *SuperPeer* of the new sub-group. In order to be reachable by the rest of the community, the new *SuperPeer* uses the structured network to insert its own IP address in the node whose *NodeId* fits whit the new *SubgroupId*. If this node has already one value, the generated *SubgroupId* is repeated. In this case, the node tries to join that sub-group as a regular node.

Every *SuperPeer* can only generate an additional *SubgroupId*, and they must keep this identifier in order to answer all future requests. Therefore, any joining request will be retransmitted from one sub-group to another until a sub-group with room (or without any additional sub-group identifier) is reached.

Just when the second level-1 sub-group is created, it also becomes necessary to create a new level-2 sub-group. The *SuperPeer* responsible for creating this second level-1 sub-group becomes the *SuperPeer* in the new level-2 sub-group. The level-2 sub-groups also have a maximum number of nodes, and therefore the sub-group creation process continues in a similar way up to the highest level of hierarchy. With this algorithm, only the set of level-1 *SuperPeer* is candidate for being *SuperPeer* in the rest of levels.

2.4 *SuperPeers* Maintenance

When a new node finds its *SuperPeer*, it notifies its available resources of bandwidth, memory and CPU. The *SuperPeer* controls the nodes that are linked to its sub-group and it forms and ordered list of future *SuperPeer* candidates. In order to supply more security to our system against centralized attacks, every *SuperPeer* node will play this role only for a fixed period of time, and then it will become a regular node.

2.5 Registering the Shared Files

As it was mentioned previously, all the level-1 *SuperPeers* maintain a database that registers the content list of all the nodes in their sub-groups. This database associates every content with the *NodeId* of the owner node. In this way, an intruder with access to one of the system databases cannot obtain the IP addresses of the nodes that are using the system. Every time a node downloads content, the *NodeIds* of both source and destination nodes are notified to the level-1 *SuperPeer*. In file-sharing applications, this feature may be used to reduce the downloading time by enabling chunking techniques.

2.6 Searching for a Content

As it was mentioned previously, in order to search for content, the user's node will send the search parameters to its level-1 *SuperPeer*, and this one will return information about who has the content in its sub-group. If the search fails, the level-1 *SuperPeer* will send the request to its associated higher level *SuperPeer*. If it is necessary, this process will continue up to the highest level *SuperPeer*. The higher level *SuperPeers* do not maintain content databases, they request information to their subgroup's members when needed. The search requests sent by the higher level *SuperPeers* to their associated lower level *SuperPeers* have a similar effect to a request sent by a regular node to its level-1 *SuperPeer*.

To provide some degree of anonymity, the level-1 *SuperPeers* can be configured to provide not only the *NodeId* of the owner node, but also an additional set of *NodeId*. This feature can be used to enable hop-by-hop download, instead of peer-to-peer. In this way, a spy never knows who the owner of the content is.

2.7 Additional Reliability

The system proposed in this work offers two additional kinds of reliability: reliability in front of unexpected nodes shutdown (fails) and reliability in front of internal attacks. If a node unsuccessfully tries to connect with its *SuperPeer*, maybe it was turned-off. In this case, the node looks for the node whose *NodeId* fits with its *SubgroupId* and checks the registered IP address. If the IP address still corresponds to the IP address of the failed *SuperPeer*, the node inserts its own IP address in this register and becomes the new *SuperPeer* of the corresponding sub-group. This way, when the rest of nodes detect that the previous *SuperPeer* is off, they will be able to obtain the IP address of the new *SuperPeer*.

In order to provide reliability in front of internal attacks, the system offers a reputation mechanism able to locate and knockout the malicious nodes. When a user downloads a corrupt content from a node, he informs its corresponding level-1 *SuperPeer* of this fact, sending a special message (*vote*) that contains the *NodeId* of the node. Every level-1 *SuperPeer* keeps a database that registers every *NodeId* and the corresponding number of *votes* associated. When the number of *votes* exceeds a predefined threshold, the level-1 *SuperPeer* registers that *NodeId* as malicious in the previous database. Our proposal manage this database in the same way that content databases.

3 Performance Analysis

In this section, we evaluate our proposed system by simulation. We have assumed that the number of available contents is finite. Given an initial content distribution among all the nodes, in each simulation step all the nodes issue a request to find and download a new content. This assumption strongly reduces the simulation time without distorting the system evaluation. Furthermore, when testing the figures of merit, the downloading process is completely irrelevant.

In order to give a better insight into the system inertia, the contents are sometimes classified into three types based on their degree of interest from the user point of view: very interesting, interesting and of little interest. In any case, in every simulation the contents are uniformly distributed among all the nodes.

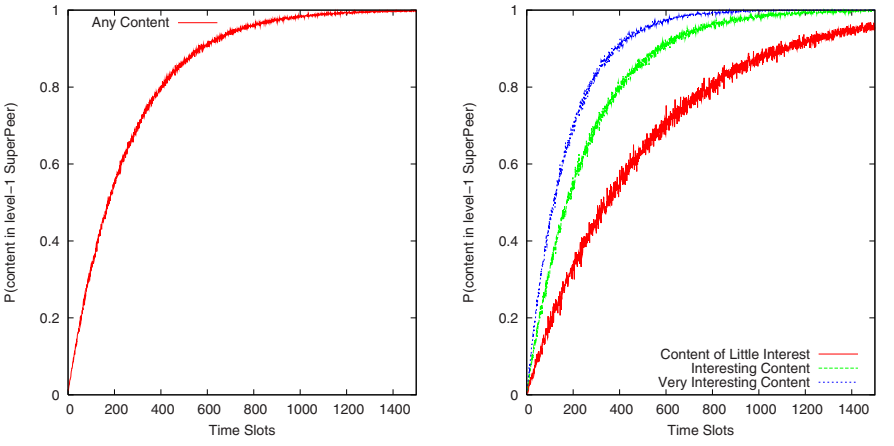
Our simulation shows three interesting parameters. The first parameter is the time evolution of the probability of locating a content in the same requester's sub-group. The second is the time evolution of the average number of *SuperPeers* consulted until content location. Finally, to evaluate the system performances it is also interesting to represent the time evolution of the number of contents known by *SuperPeers*.

The simulation scenario is formed by 12,500 different contents uniformly distributed among 6,250 nodes. The sub-group sizes are set to 50 and 5 nodes for the first and the rest of the hierarchy respectively. Therefore, there are 125 level-1 *SuperPeers*, 25 level-2, 5 level-3 and one level-4.

Fig. 1 shows the time evolution of the probability that a content is in the same requester's sub-group. Fig. 1.(a) shows this probability for any kind of content and 1.(b) for classified contents. The probability is calculated as the ratio between the number of requests successfully answered at the same sub-group and the total number of requests. It can be observed that this probability grows and converges to a value of one. In order to explain this result, note that each time a search request is resolved by the intervention of another sub-group, at the end of the download process the content is also registered in the requester's *SuperPeer*, increasing the value of this probability. This behavior indicates that in our architecture, the more a popular content is downloaded, the more equally distributed it is among all the sub-groups. This is also the reason why the probability converges to unit more quickly for the most interesting contents (see Fig. 1.(b)). With this behavior, the risk that a higher level *SuperPeer* becomes overload in steady state is minimal.

Fig. 2.(a) shows the time evolution of the average number of *SuperPeers* consulted until a content is located. This parameter is calculated averaging the numbers of *SuperPeer* consulted for each node. It is observed that the number of consulted *SuperPeers* quickly decreases, converging to unit in a short period of time (indicating that the content is in the same requester's sub-group).

Obviously, the two previous variables are strongly related. We have developed an analytical study that establishes the relation between this pair of parameters. We define p as the probability that a content is in the same requester's sub-group. We suppose that the network has L different levels. We also suppose that, irrespective of the number of nodes at the first level, there are exactly



(a) Any content (b) Classified contents

Fig. 1. Time evolution of the probability of locating a content in the same requester's sub-group

N nodes per sub-group in the rest of the levels. If we define S as the average numbers of consulted *SuperPeers* until a content is located, then

$$S = \sum_{n=1}^{L} nP(n) \tag{1}$$

where *P(n)* is the probability of consulting n *SuperPeers*. It can be defined as,

$$P(n=1) = p \qquad P(n=L) = (1-p)^x \tag{2}$$

$$P(n) = (1-p)^x \sum_{j=1}^{y} \binom{y}{j} p^j (1-p)^{y-j} \quad 1 < n < L \tag{3}$$

where x and y are defined as

$$x = N^{n-2} \qquad y = (N-1)x \tag{4}$$

Note that Equation 3 can be simplified using the binomial theorem, and therefore,

$$P(n) = (1-p)^x[1-(1-p)^y] \quad 1 < n < L \tag{5}$$

This analytical study has been verified by means of simulation. It is important to see that when p tends to unit (which is the expected value for popular contents), the number of *SuperPeer* involved in the searching process (S) also tends to unit. This result analytically demonstrates that the most popular contents are -sooner or later- widely spread around the network.

Finally, Fig. 2.(b) shows the time evolution of the amount of contents known by *SuperPeers* at different levels. As can be seen, the level-4 *SuperPeer* is the

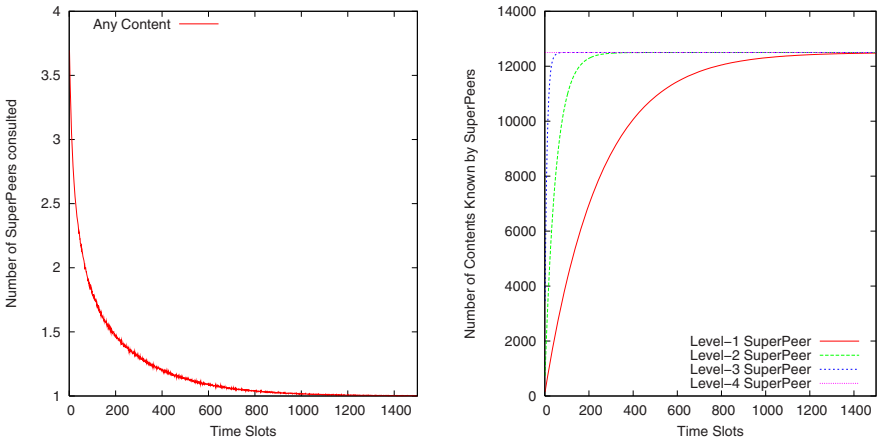


Fig. 2. (a) Time evolution of the average number of *SuperPeers* consulted until content location. (b) Time evolution of the number of contents known by *SuperPeers*.

only node with a global knowledge. The rest of the *SuperPeers* quickly reach the maximum value (12,500), which means that, in this system, the content distribution procedures spread around all the sub-groups the most popular files. Let us recall that, in fact, only the level-1 databases are really needed.

4 Related Works

In this section it is briefly summarized other contributions related with this work. The first one is JXTA [6]. In JXTA peers are self-organized into groups, and every group has associated a *rendezvous* node. The queries are propagated only among *rendezvous*, which are organized into a hybrid network that combines the use of a loosely-consistent DHT with a limited-range rendezvous walker. Compared with our proposal, JXTA has a high flooding load since the *rendezvous* are not organized into hierarchical levels.

In SHARK [7], each node is assigned to a Group of Interest according to the objects it stores. When a node initiates a query, a *SuperPeer* is responsible of address this query to the corresponding Group of Interest. Then the query is flooded towards all the members of the group. This hybrid solution allows some degree of human readable form lookups, but does not report any issue about reliability in front of external attacks.

OceanStore [8] is a P2P storage system built on top of Tapestry. It also employs an additional probabilistic mechanism based on attenuated *Bloom Filters*, resulting in a hybrid solution. When the fast probabilistic algorithm fails to provide the requested results, OceanStore activates Tapestry routing mechanism to forward the request to the final destination. The system replicates the objects

to provide durability against attacks. However, in our opinion, our solution guarantees a great level of reliability.

5 Conclusions

In this paper a hybrid and self-organized P2P overlay network is presented. From the user point of view, the hybrid nature of the system (structured and unstructured) allows the searching process to be written in human readable form, without losing the distributed nature of the P2P systems. The network is also self-organized because the *SuperPeers* are automatically selected.

Nodes which are responsible for being *SuperPeers* change their role periodically. Morevoer, our proposal is prone to distribute the most popular contents around the network. Finally, to improve the system performances, some additional features like hop-by-hop download and chunking can be easily implemented.

Acknowledgements

This work has been supported by the Spanish Researh Council under project ARPaq (TEC2004-05622-C04-02/TCM) and with funds of DG Technological Innovation and Information Society of Industry and Environment Council of the Regional Government of Murcia and with funds ERDF of the European Union.

References

1. Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H. Chord: A Scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 SIGCOMM Conference. (2001) 149 - 160
2. Rowstron, A., and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001) 329 - 350
3. Gnutella. http://www.gnutella.com. (2006)
4. eDonkey2000. http://www.edonkey2000.com. (2006)
5. Leibowitz, N., Ripeanu, M., and Wierzbicki, A. Deconstructing the KaZaa Network. In Proceedings of the 3rd IEEE Workshop on Internet Applications. (2003) 112 - 119
6. Traversat, B., Arora, A., Abdelaziz, A., Duigou, A., Haywood, C., Hugly, J.-C., Pouyoul, E., and Yeager, B. Project JXTA 2.0 Super-Peer Virtual Network. http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf. (2003)
7. Mischke, J., and Stiller, B. Rich and Scalable Peer-to-Peer Search with SHARK. In Proceedings of the 5th International Workshop on Active Middleware Services. (2003) 112 - 122
8. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., and Weatherspoon, H. OceanStore: an Architecture for Global-scale Persistent Storage. In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems. (2000) 190 - 201

Query Coordination for Distributed Data Sharing in P2P Networks

Maybin Muyeba and M. Sulaiman Khan

School of Computing, Liverpool Hope University, UK
muyebam@hope.ac.uk, m_sulaiman78@hyahoo.com

Abstract. Organisations often store information about the same entity objects or features in different formats. Accessing and integrating this distributed information can be a difficult task because of schema differences and database platform issues. In this paper, a discussion on query coordination in schema conflicting databases of Peer-to-Peer (P2P) systems is presented. The coordination mechanism with a global query to resolve the issue is proposed. The coordination mechanism is done without peers knowing each others schemas by translating the global query into the local query according to the under lying database schema while a wrapper is used to deal with database platform issues. The paper simulates a real-life application and shows that schema resolution by a query coordination mechanism in P2P systems is effective and minimises most of the complexities encountered by schema integration systems.

1 Introduction

Organisations often store information about the same entity objects or features in different formats. It is usually the case in distributed databases that data integration of such systems is bottlenecked, among other things, by varying schemas and platform issues. Thus accessing and integrating the distributed data, besides analyzing it, is a challenge. Schema integration [1] by a global schema is possible but not so desirable. A coordination mechanism to deal with the above mentioned issues for large scale distributed, heterogeneous and autonomous systems (P2P) is presented, characterized by a query system using JXTA technology. JXTA is a java technology that has desirable peer functionalities such as provision of independent IP naming space address, platform independence and handles firewalls etc. To query or even analyse data without knowing typical peer schemas and database platforms requires an intelligent coordination strategy. A query is first sent to the nodes of relevant domains where it is translated into the local query according to the under lying database schema using metadata schema descriptions. Domain here refers to any database application in the system and relevant domains refer to groups or classes of sub-domains with similar schematic data, to be defined later. Analysis of such systems is not an easy task and leads to two major issues:

For different platforms, its not easy to access it by simple traditional database techniques used in central processing,

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 376–384, 2007.

For schematically inconsistent databases, sometimes it is usually not feasible to integrate the data for further analysis or reporting.

Thus data must be processed by distributed algorithms suitable to this kind of computing environment.

Considering issue two above, for example, an insurance company may want to access the medical history of their customers from a GPs (General Practitioners) clinic database stored with different names and types. Thus, a query coordination mechanism in a P2P setting with model-theoretic semantics is introduced with examples. We assume the terms "node" and "peer" mean the same thing.

Data management issues are important in P2P systems and later, their solution useful in overall data analysis [2] . In [3], a local relational model (LRM) replaces the concept of a global schema in the conceptual view thus allowing semantic interoperability without a global schema by using coordination formulas. Another Local Relational Model (LRM) is described in [4], the coDB system which uses JXTA. Each node is queried via a coordination rule from its neighbours for data migration purposes. This means neighbouring nodes need to be aware and maintain up-to-date rule coordination information, which includes cyclic rules used to synchronise between database data. Piazza [5] uses "semantic mediation" between peers and allow expressing both GAV (Global as View) and LAV (Local as View) styles to map between peer schemas. Other systems [6] use coordination rules and rule bases respectively. This motivates our study in this paper i.e. an easy to use query mechanism to coordinate global queries to relevant peer groups without the complexity of a rule-base and knowledge sharing between peers, as in related works.

The paper is organised as follows: section 2 gives the problem definition; section 3 shows a coordination engine architecture, section 4 presents an example application; section 5 gives experimental results and section 6 concludes the paper.

2 Problem Definition

In this section, we present a problem definition and the theoretic-model semantics of our approach.

We organise peers in peer groups and denote each query between peers as a relevant criteria for determining peer equivalence as follows: number of peers P, databases D for each peer, a set S of similar attribute names (or keys, to be defined later) from all databases and a general descriptor g for mapping all attribute values to their similar terms S. For given attributes, we can express the set S as

$$S = \{A_1, A_2, .., A_k\} \tag{1}$$

where $S \subseteq \{A_1, A_2, .., A_m$ for some $k > 0$. Note that $S \subseteq g$ as g may contain metadata of the attribute values in S, stored in a property file F. This file consists, in simple terms, a series of (key, value) pairs, where key can be any

value from set g. P2P global queries are locally evaluated according to these local schema mappings. Thus F has mappings g onto S as:

$$S \rightarrow g \tag{2}$$

The P2P network is in fact a graphing problem $G =< V, E >$, where V is a vertex and E is the edge. Each peer P_i has database D_i, where a query Q_i is undirected to any node (i.e. a broadcast) can form the edge of the query graph G, so G is redefined as $G =< P, D, Q >$, where P is the set of peers, Q is a family of queries and subset of $P \times D$ between nodes. A graph relation $\sim$ formed is, by deduction, reflexive, symmetric and transitive and equivalence classes $C_i's$ are formed from peer relations of similar domains. A set of nodes P_i will be "coordination active" if they belong to an equivalent class . This is useful and reduces the costly global communication. In our case, we use the term "neighbour" relatively to mean those nodes that can answer a given query either transitively or symmetrically, where reflexity is an intuitive case. The approach also avoids returning many empty results from non-equivalent peers. An optimal way to generate equivalence classes $C = \{C_1, .., C_s\}$ is to implement an intelligent dynamic algorithm that forms such classes, which is our approach. We are aware that in a large network with a large number of attributes, the number of property files grow large linearly. This is still scalable to within reasonable size for most applications. Our P2P framework based on the problem definition above is illustrated as follows:

Definition 1 (P2P system). A peer-to-peer (P2P) system is a graph $G =< P, D, Q >$, where P is a set of nodes $P \subseteq \{P_1, P_2, .., P_n$ with a local property file,F_i, $1 \leq i \leq n$, D is the set of corresponding databases $D = \{D_1, .., D_n\}$ and Q is a set of queries $Q = \{Q_1, .., Q_k\} \subseteq D \times P$, for some n and $k > 0$.

Definition 2 (Peer Node). A peer P_i is a self-contained node with local schema LS_i derived from D_i and local property file F_i, $LS_i \subseteq F_i$. Due to security restrictions of the local schema, $F_i \subset LS_i$, if peer attribute information is not shareable.

Definition 3 (General Query). A general query q_i is an undirected broadcast query among m nodes, $m =\mid P \mid$, $q_i \subset P \times D$. Nodes may not be partitioned into equivalent classes or relevant domains or that the translation of query attributes into their local schemas has not yet occurred, so the equivalence relation defined above still holds. However, note that for peers belonging to relevant domains, which are part of the whole P2P system generally, this also holds under general terms of this definition. The difference is the number of nodes that respond and the type of returned acknowledgement information and query results. This leads to definition 4.

Definition 4 (Domain Query). A domain relevant query q_j is an undirected query among m nodes,$m \leq \mid P \mid$, $q_i \subset P \times D$ where these nodes satisfy equivalence relations. It is easy to see that $q_j \subseteq q_i D$ and that q_i translates to q_j using a query interpreter of the prototype system. Queries in definition 4 only apply to peers belonging to equivalent domain classes.

Definition 5 (Peer Class Relations). A peer class relation is a collection of peers $\{\bigcup P_i\}$ that belong to a particular domain or peer group and exhibit an equivalence relation amongst them.

Definition 6 (Property Schema File). A property schema file F_i consists of a collection of (key, value) pairs per peer, where "value" is a schema attribute and "key" can assume different global (general) or metadata names for that attribute value. Therefore each local property schema file can use any chosen "key" name(s) but the attribute value remains the same for that local schema. These property files (Database Schema-DBS) in our system are directly accessible to the peer manager.

Definition 7 (Naming and Key differences). An attribute value has a naming difference if there exists another similar attribute e.g. phone, telephone, contactNo. According to definition 6 above, possible (key, value) pairs could be (contact, telephone), (phone, telephone) etc. Note also that these pair definitioins could be cyclic across the P2P system. Key differences are trivial. Other differences like attribute types are dealt with implicitly e.g. age and DOB are integer and string types.

3 Coordination Engine Architecture

One aspect of P2P networks is their varied platforms and in this case their dynamic variation in schemas. The JXTA technology provides a set of protocols to develop P2P applications such as peer discovering each other and self-organizing into peer groups (definition 5), advertising and discovering network services, peer communication and monitoring. Overall, it offers interoperability enabling peers providing various P2P services to locate and communicate with each other, platform independence and is ubiquitous.

In the node architecture, a wrapper approach is used to handle database platform varieties and in case of schema evolution, there will be no need to make any change in the wrapper or program to update the obsolete schema. This technique also minimizes the amount of processing the peer manager and the wrapper do. The nodes communicate by passing messages through pipes containing all the relevant information regarding the peer that generated it and its domain. The session ends after the sending node gets all replies.

Table 1 shows a node domain directory maintained by each peer to keep track of all peers in the network.

The architecture and process of coordination as shown in figure 1 is abstracted by a few components. A query received by a node is immediately sent to the query detector to determine its type and then the query interpreter translates to a local query according to the query type and the nodes DBMS using a wrapper. The Peer Manager checks whether a query can be forwarded to other nodes by determining equivalence classes, and if so, propagates it to other nodes and the recipients nodes information are sent back to the originating node.

The system optimally deals with global queries by using equivalence classes of particular domains, which keep track of peers and their domains. Our system

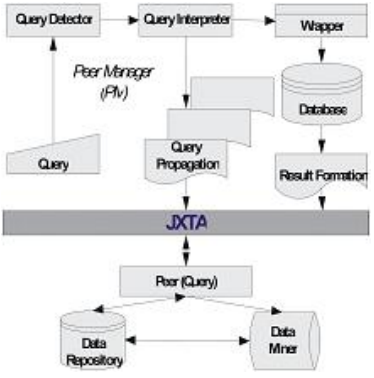


Fig. 1. P2P Coordination Architecture

Table 1. Node Domain Directory

Node	Domain
Clinic 1	Medical
Hospital 1	Medical
Insurance 1	Insurance
Clinic 2	Medical
Hospital 2	Medical
Insurance 2	Insurance
.....	.....

monitors newer peers and subscribe them to appropriate classes or domains dynamically by analyzing their schemas and eventually their appropriate domains.

4 Application Example

Assume the following schemas in some domains:

Database Hospital (H)
Patient (HospitalID, PatientName, Sex, DOB, HIN, Telephone, PatientRecord)
Treatment (TreatmentID, HospitalID, Date, Disease, TreatmentDesc, PhysicianID, HIN)
Medication (HospitalID, DrugID, Dosage, StartDate, EndDate)
Database GP Clinic (C)
Patient (HIN, FName, MName, LName, Phone, Gender, DOB, PatientRecord)
Visit (HIN, Date, Purpose, Outcome)
Prescription (HIN, Disease, MedicineID, Dose, Quantity, Date)
Database Insurance Company (I)
Customer (CustID, FName, LName, PhoneNum, Sex, DOB, RegistrationDate)
Policy (PolicyID, CustID, StartDate, EndDate, PolicyType, Premium)
Claims (ClaimNo, CustID, PolicyID, ClaimType, ClaimDescription)

A global query is translated on each node according to its underlying database schema to overcome schema conflicts as follows:

"SELECT name from medical" is translated on hospital peer as:

"SELECT p.PatientName as Name from patient p" and on clinic peer as:

"SELECT(p.FName+" "+p.MName+" "+p.LName) as Name from patient p".

The general descriptor "name" in the global query is translated to similar values described in property files as "p.PatientName","p. Name", "p.MName", "p.LName" according to the database schema etc. This is referred to as *naming differences* (definition 7). For example

PatientName, Name, FName+MName+LName ← Full Name

Sex, Gender ← Sex

Telephone, Phone, Mobile ← Contact

Age, DOB ← Age

where full name can be any metadata or attribute in the query etc. Again note that attribute names and keys could be defined cyclically i.e. one could mean the other and viceversa. Similarly, the same entity object (customer or insurer or patient etc) has different identities such as:

CustID, HIN, PatientID, HID← Unique Identifiers

Let us consider, for example, a data mining problem where an insurance company wants to access its customers medical records and cluster them w.r.t their age whilst suffering from a certain disease in order to readjust their premiums.

The insurance company needs to access medical domain nodes (medical equivalence class) that contain the required information, here nodes H and C. To get the desired results, node I first generates a general but domain specific query using simple basic notations, which in this particular case would be:

Q1: Select Name, Age, HIN, Sex from medical where Illness= "diabetes";

Q1 is a general but medical domain specific query and not built on actual fields and table names, and so unable to execute on clinic and hospital databases in its form. Node H converts Q1 into the naming format understandable by the hospital database and any platform differences are resolved by a wrapper algorithm. This is done as follows:

Q1 on H: Select p.PatientName, p. (dateddiff(month, p.DOB,GETDATE())/12) as Age, p.HIN, p.Sex from Patient p,Treatment t where p.HIN=t.HIN and t.Disease="*diabetes*";

Node H will execute Q1, prepare the results and send back to node I. If there is some node present to which query is not forwarded yet, it will forward it to that node and add extra information about that node with the results sent to node I. The same process will be repeated on node C. Node I then determines the type of information received. If its a result then the data is extracted and sent to the data repository where it will be used later for possibly further analysis or even application of a data mining algorithm. The node domain directory is then updated if the message contains any information to be added or updated and the query session is then terminated.

5 Experimental Results

The experimental P2P network set up consisted of 15 desktop computers with Intel Pentium 4, 1.6 MHz processor and 256MB of RAM, and all the machines running Windows XP on a network transfer rate of 100Mbps.

Table 2. Statistical Attribute Information

Normal Attributes	**Naming Differences**
Name	4
Gender	2
Age	2
Contact	3
Medical	2
Illness	2
Prescription	2
PatientRecord	2
Key Attributes	**Key Differences**
HIN	2
PID	3
Domain ID	3

Table 2 shows, in our case, similar term groupings of normal schema names (Nor. Attrs) and differences in naming the attributes (Nam. Diff.) and their primary keys (Key Diff.) in local schemas.

The physical P2P network layout, with up to 15 nodes, is shown in Figure 2. Synthetic data used for implementation purpose is generated through a synthetic random data generator. Each peer consist a database with three tables containing 10000 records in each table (30K in total) with different schemas as shown in section 5. Three database engines, MS Access, MySQL and MSSQL Server, have been used for platform heterogeneity purposes.

For experimental purpose all the machines and the network were fully dedicated.

Figure 3 show an exponential increase in time as the number of nodes increase. The result indicate the complexity involved in converting global broadcast queries to local queries as the number of nodes and attributes increase. There are minor differences when increasing the number of attributes with a constant node number. It is apparent that query translation into similar terms of the local schema affects execution time by some factor given constant node sizes. The system shows reasonable scalability for the few nodes with varying attribute schemas. For queries with many such variations under the same conditions, exponential time is expected. Similarly, figure 4 shows query completion time against data size with increasing node results and consequently raising communication overheads. Notice also that huge attribute naming differences and the number of attributes in a query can degrade performance.

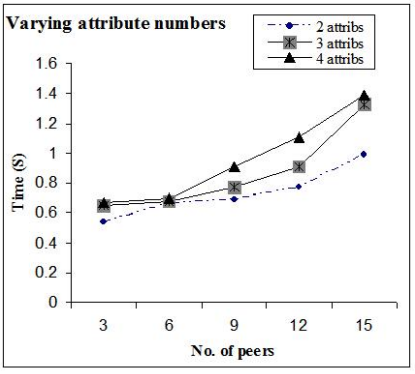


Fig. 2. Query Completion Time

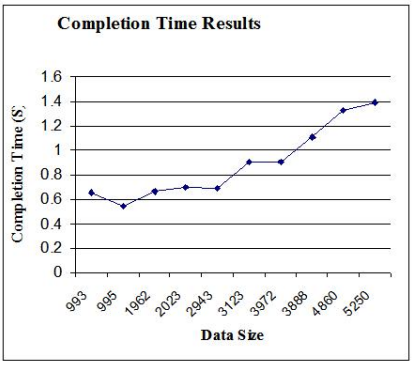


Fig. 3. Completion time (Results)

Analysis algorithms in practice, could benefit from our overall approach, for example if an insurance company wanted to re-evaluate customer premiums by clustering customer data based on age groups prone to particular diseases. Our proposed approach is therefore extensible to other domains where data sharing is inevitable.

6 Conclusion

The paper proposes a mechanism for query coordination in P2P systems that share data on different platforms with varying schemas. Issues relating to resolving schema conflicts were presented and benefits of such an approach include schema independence for global queries among peers, capability of directing queries to specific peer groups and dynamic schema updates in P2P environments. The system shows fair scalability given the limited number of nodes, attributes and peer databases with different schemas used in our experiments. The system works well in a distributed and heterogeneous DBMS environment and does not need global synchronisation or schema integration.

Further work will investigate developing further ontological frameworks for naming schema differences and also similarity relations between entity objects.

References

1. Batini, C., Lenzerini, M., Navathe, S. B.: A comprehensive analysis of methodologies for database schema integration. ACM Computing Surveys, (1986),18(4), 322-364
2. Wolff, R., Schuster, A.: Association Rule Mining in Peer-to-Peer Systems. IEEE Transactions Systems, Man and Cybernetics, (2004) 34(6) 2426-2438
3. Bernstein, P. A., Giunchiglia, F., Kementsietsidis, Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: A Vision. International Workshop on the Web and Databases, WebDB, (2002)

4. Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I.: Queries and updates in the coDB peer-to-peer database system. Proceedings of the 31st VLDB Conference, Toronto, Canada, (2004) 1291-1294
5. Halevy, A. Y., Ives, Z., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I.: The Piazza Peer Data Management System. IEEE Transactions on Knowledge and Data Engineering, 16(7), (2004) 787-798
6. Roshelova, A.: Building Database Coordination in P2P system using ECA rules. Technical Report DIT, Informatics and Telecommunication, (2004) University of Trento

A Comparative Study of Pub/Sub Methods in Structured P2P Networks

Matthias Bender, Sebastian Michel, Sebastian Parkitny, and Gerhard Weikum

Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
{mbender,smichel,parkitny,weikum}@mpi-inf.mpg.de

Abstract. Methods for publish/subscribe applications over P2P networks have been a research issue for a long time. Many approaches have been developed and evaluated, but typically each based on different assumptions, which makes their mutual comparison very difficult if not impossible. We identify two design patterns that can be used to implement publish/subscribe applications over structured P2P networks and provide an analytical analysis of their complexity. Based on a characterization of different real-world usage scenarios we present evidence as to which approach is preferable for certain application classes. Finally, we present simulation results that support our analysis.

1 Introduction

Peer-to-Peer system have been a hot research topic for years now, but only recently there have been some success stories of actually deploying legal P2P-based applications on a large-scale basis, such as BitTorrent or Skype. We feel that this is partly due to the fact that the P2P paradigm has been applied to all kinds of popular (web) application scenarios, even though they mostly did not necessarily expose any natural P2P-like usage scenario in practice.

One of the few applications that naturally fit with a fully decentralized setting are publish/subscribe (pub/sub) applications. We bring forward the following two archetypical pub/sub scenarios that we will characterize and refer to throughout the paper. Consider a *User-to-user scenario* in which users want to share community knowledge, e.g., on computer troubleshooting, or discuss recent events in Blogs or Wikis. In this scenario, all users are interested in (i.e., subscribe to) a certain subset of new content, and become publishers themselves at a low rate, publishing a new article or adding a comment to an existing Blog entry. For example, imagine a publish/subscribe application where subscribers want to be notified when any participating Blog publishes a new article that contains the term P2P. In the *Publisher-to-user scenario*, a smaller set of publishers (e.g., news agencies like Reuters or AP) publish new content at a much higher rate, and a large number of users (typically distinct from the set of publishers) is interested in monitoring a developing news story, staying up-to-date

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 385–396, 2007.

on a competing business, or getting the latest tabs on a celebrity of the favorite sports team.

In spite of exposing some very distinctive characteristics, both sample scenarios expose a system model of fully distributed and autonomous information providers and information consumers, which is well-suited for a P2P-style organization. Indeed, numerous approaches on how to efficiently design publish/subscribe in a fully distributed manner have independently emerged from the P2P research community. Unfortunately, the sheer number of proposals and their often hard-to-compare assumptions regarding the underlying network structure and usage patterns render the comparison a troublesome task. Typically, the approaches are only evaluated for a very specific set of system parameters, and not compared to other existing approaches for different settings.

This paper's contribution is threefold. First, it identifies and introduces two design patterns that can be used to implement pub/sub applications over structured P2P networks. Second, the paper provides an analytical characterization of the complexity of both approaches and provides guidelines regarding which approach might be preferable for which usage patterns. Third, we present measurements that back up both our analytical results and our suggestions for design guidelines, also in the presence of network dynamics. We explicitly model network dynamics, as peers constantly enter and leave the system. Note that this paper does not compare any concrete prototypes, but instead focuses on evaluating the design approaches for various system parameters.

2 Related Work

Distributed hash tables, DHTs, (such as Chord [15], CAN [12], Pastry [13], or P-Grid [1]) have emerged as the preferred family of structured architectures for overlay P2P networks. The main advantage of DHTs compared to unstructured P2P networks stems from the performance guarantees that they can offer regarding the routing efficiency and ultimately the network scalability, even in the presence of high network dynamics (such as high rates of node arrivals/departures and failures/recoveries). P2P data networks over structured and/or unstructured networks has been a hot research topic for years [6,9,4,17,11,10].

More recently there have been numerous proposals for distributed pub/sub systems [19,2,7,16,14,18,8]. Most of them can be classified into three categories: topic based systems, content based systems, and hybrid solutions. Topic based systems usually consider that users subscribe to a publisher that regularly publishes documents of a certain topic (e.g. mailing lists), whereas in content based systems users subscribe to publishers that do not have a particular topic but employ a, usually, term based filtering to distinguish between relevant and non-relevant documents w.r.t. the users' interests. Hybrid systems basically consider subscriptions to topics but allow for a term based filtering. The first step reduces the communication overhead compared to the term-subscriptions case whereas the second design choice drastically reduces the number of actually shipped documents, i.e. preventing the delivery of irrelevant documents to the end user. [3]

reasons on the difference between information filtering and information retrieval that can be interpreted as P2P pub/sub versus P2P retrieval.

3 System Model

We consider a network of nodes $N = \{n_1, ..., n_k\}$. Each node can take one or both of the following roles: *Subscribers* $S = \{s_1, ..., s_m\}$ express their interest in selected newly-published content, *Publishers* $P = \{p_1, ..., p_l\}$ regularly generate new content. Subscribers issue subscriptions. $sub_{i,j}$ denotes the j-th subscription of s_i, i.e., each subscriber can have more than one subscription. The average number of subscriptions per subscriber is denoted sub_{avg}. Each $sub_{i,j}$ is a set of terms t from a domain T, i.e., $sub_{i,j} \subset T$. $\overline{sub}$ denotes the average number of terms per subscription. Publishers generate content in form of documents $d \in D \subset 2^T$, i.e., documents are a set of terms $t \in T$. Publishers issue their new content at a certain publishing rate r, measured in new documents per time interval. The notation is summarized in Table 1.

Table 1. Notation

$s_i \in S$, s_i	Subscriber from set of all subscribers
$p_i \in P$	Publisher from set of all publishers
$n_i \in N$	Node in the network
$t \in T$	Term from term domain
$sub_{i,j}$	j-th subscription of s_i
sub_{avg}	average number of subscriptions per subscriber
$\overline{sub}$	average number of terms per subscription
$d, \|d\|$	document, length of document

3.1 Discussion

For this work, we assume subscribers to express their interests in form of terms, i.e., they are interested in all new content that contains all subscription terms. We feel that more sophisticated means of subscriptions are a high burden for non-expert users, which make up a large fraction of users in a large-scale real-world network. Note that the ratio between publishers and subscribers and also the average number of subscriptions per subscriber highly depends on the targeted application scenario. While in the User-to-user scenario, typically all subscribing users also play the role of publishers (typically at low rates), the Publisher-to-user scenario with a limited set of news agencies as publishers exposes a small number of publishers that publish at high rates. The ratio between subscribers and publishers highly influences the optimal design pattern of the system; an extreme ratio could render either approach infeasible.

This work focuses on structured P2P architectures, such as distributed hash tables (DHTs), as a network overlay. Not limiting ourselves to a particular DHT implementation, we only assume a basic *nodeID* $\leftarrow$ *lookup(k)* functionality that

maps a key k to the node identified by *nodeID* currently responsible for that key. DHTs can straightforwardly be used to construct conceptually global, but physically distributed directories. This paper focuses on structured overlays because we strongly believe that gossiping in unstructured networks inevitably leads to scalability and performance issues, in particular for application classes which aim at efficiently locating specific information such as publish/subscribe. Our study does not include publish/subscribe approaches that let publishers circulate their documents through the whole network and that make subscribers "pick" all content that they are actually interested in, as we feel that distributing content to all peers even though they might not be interested in it exposes scalability issues already for medium-sized networks. In this context, we also point out that our analytical model and our measurements do not consider the actual document dissemination, but only the matchmaking between the subscriptions in the network and newly-published content. While we will sketch possible ways of efficiently disseminating the documents at a later stage for both design patterns, our evaluation ends as soon as the set of all matching subscribers for a new content is identified.

4 Design Patterns

The common basis for the following two design patterns to implement pub/sub functionality is the presence of a conceptually global, but physically distributed directory, built on top of all nodes in the network, i.e., subscribers and publishers alike. We use the term *directory peer* to refer to a node, stressing its participation in the distributed directory. The directory maps *keys* in form of features (e.g., terms or topics) to appropriate *values* describing the key value, e.g., in form of statistical aggregations. Such a directory can straight-forwardly be implemented on top of any DHT, offering a basic *nodeID* $\leftarrow$ *lookup(key)* method, as follows: Each node that wants to learn (or add to) the statistics for a certain key issues the corresponding *lookup(key)* request to retrieve the peer that is currently responsible for maintaining the value for that key. In a point-to-point fashion (not stressing the directory), the user can subsequently contact that peer to retrieve or add the desired information.

We strongly believe that storing (pointers to) individual *documents* in the distributed directory is not feasible, as (e.g., for Blogs in the Publisher-to-user scenario) the number of publications increases rapidly. We think it is unavoidable to abstract and aggregate the information, yielding a light-weight system that offers scalability to support an a-priori unlimited number of nodes. The following subsections describe two design patterns to implement pub/sub functionality, storing different metadata in the distributed directory.

4.1 Store-Sub

Most of the existing approaches of implementing a publish/subscribe infrastructure over structured P2P networks follow the general paradigm that we subsequently refer to as **Store-Sub**: The subscribers store their subscriptions in a conceptually global, but physically distributed directory implemented on top

of the DHT. When publishers publish new content, they retrieve all applicable subscriptions from the directory, which is feasible because publishers can inverse the subscription process to find all subscriptions that match their new content. Perhaps the most obvious way to actually implement this for each $sub_{i,j}$ is to let each s_i send a message for each term t occurring in at least one of its subscription to the directory peer identified by $lookup(t)$ and to attach $sub_{i,j}$ completely. A publisher that wants to publish new content has to retrieve these lists from the appropriate directory peers for each term t of its new content in order to identify all subscribers that have issued a subscription that is matched by the content.

The bottlenecks of this approach are obvious: Publishers need to retrieve a large number of potentially huge lists of subscribers to find all appropriate subscribers when they publish new content, because they have to retrieve the subscriptions for all terms that occur in the content, in order to make sure not to miss any matching subscribers. One typical way to tackle this issue is to reduce the number of features that a publisher needs to check in order to find all appropriate subscribers. For example, this can be achieved by mapping all terms to a much smaller number of topics[1]. Subscribers send their subscription to the directory peer responsible for the appropriate topic and attach a more expressive subscription (e.g., the complete set of subscription terms or XQuery expressions). In this case, publishers only need to check for potential subscribers at a much smaller subset of directory peers. While the most obvious way of eventually matching all such attached subscriptions might be for the publisher to retrieve the list of all subscribers for the appropriate topic(s) of a new content item and to perform the matchmaking locally, this may require to transfer an excessive number of (eventually non-matching) subscriptions from the directory peer(s) back to the publishers. Another possibility is, thus, for the publisher to transfer the actual document to the applicable directory peer(s) and let them perform the matchmaking locally, without transferring the lists of subscriptions. Subsequently, the directory peer can either start the document dissemination itself immediately or return the (much smaller) list of matching subscribers back to the publisher. Figure 1 illustrates an example of the Store-Sub infrastructure on top of a DHT. $N17$, $N45$, and $N76$ are directory nodes responsible for topics sports, politics, comedy, and music. The mapping from topics to directory nodes is given by the DHT's *lookup()* method that, e.g., maps the topic "sport" to the directory node with nodeID 17. $N17$, for example, has already received subscriptions from $S42$ regarding the combination of terms *(worldcup, 2006, germany, opening)* and is currently receiving another subscription from $S14$. Directory peers store incoming subscriptions in a subscriber list. All publishers that generate content about the topic "sport" will turn to $N17$ to identify all matching subscribers, either by retrieving the complete lists of subscribers for all relevant topics (and subsequent local filtering) or by sending the document itself to the directory node, which will return an appropriately prefiltered subscriber list (or start the document dissemination itself).

[1] Building such a topic hierarchy is a well-addressed research problem and out of scope for this paper.

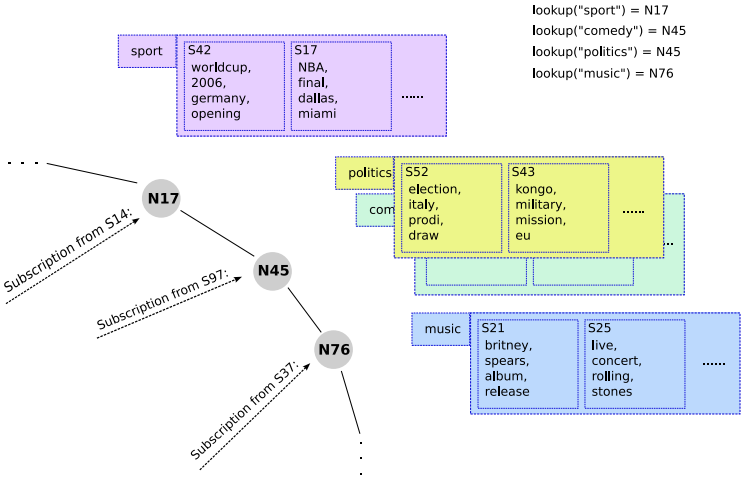


Fig. 1. Store-Sub Architecture

4.2 Store-Pub

Another approach towards implementing a publish/subscribe infrastructure over structured P2P networks is a design pattern which we will subsequently refer to as **Store-Pub**: Each publisher p_i announces its existence together with some statistical profile $prof_i$ in the distributed directory. $prof_i$ describes the content that P_i has previously published (or, potentially the expected and forecasted behavior in the near future). The distributed directory again partitions the feature (term, topic, ...) space, so that subscribers can access this (regularly updated) data to find potentially promising publishers for their information needs, and register directly with selected publisher(s).

To our knowledge, this approach has not been used so far to actually implement a publish/subscribe infrastructure. When comparing it to the Store-Sub approach, the following advantages (+) and disadvantages (-) can be identified:

+ Subscribers have full control over which publishers to contact and, e.g., prefer reputable publishers.
+ Subscribers can finetune the amount of content they receive by adapting the number of publishers they register with.
+ Subscribers do not expose information to the public that may be used for social reengineering.
+ Subscribers can subscribe to particular publishers, not being overwhelmed by content from all publishers.
- Publishers need to announce profiles. Depending on the number of publishers, that may lead to extensive network resource consumption.
- Subscribers base their decision on publishers' profiles describing the past. Subscribers may miss prospective publishers if they have not published relevant content before. If a new story arises, they cannot find any matching publishers until publishers have refreshed their profiles.

In other words, Store-Sub is an *exact* approach to pub/sub, i.e., a subscriber will in principle not miss any content that matches its subscriptions. On the other hand, Store-Pub is an *approximate* approach, there is no guarantee that a subscriber actually receives all content that matches its subscriptions, because he may not have selected the appropriate publisher based on its profile or the publisher did not yet have an appropriate profile at the time of subscription (e.g., for a developing news story). To overcome the last issue, we assume that subscribers revisit the directory on a regular basis to check whether other or better publishers have arisen to match their subscriptions, i.e., they conceptually repeat their subscription process regularly.

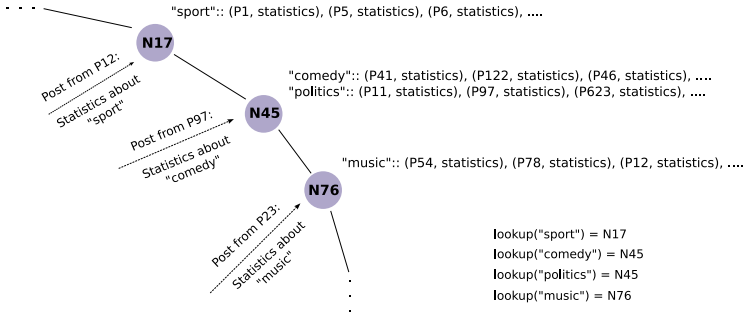


Fig. 2. Store-Pub Architecture

Figure 2 illustrates an example of the Store-Pub infrastructure on top of a DHT. $N17$, $N45$, and $N76$ are again directory nodes responsible for topics sports, politics, comedy and music. This time, they maintain profile information describing publishers P. The mapping from features to directory nodes is given by the DHT's *lookup()* method that, e.g., maps "sport" to the directory node with nodeID 17[2]. Subscribers interested in the soccer worldcup can turn to $N17$ and identify publishers at their discretion. Registering their subscriptions is a point-to-point communication with publishers, which store the subscriptions locally.

5 Complexity Analysis

For the upcoming complexity analysis, we assume a distributed directory on top of a DHT network. As messages to an arbitrary remote node of an $|N|$-node network require an expected $log_2(|N|)$ message hops in most popular DHT architectures, we use this factor for each message sent to actually reach its destination. Regarding the notation, readers might want to refer back to Table 1.

[2] Note that it is due to the random assignment of topics/terms to peers that the statistics for "comedy" and "politics" are stored on the same peers. This does not reflect the authors' opinion ...

5.1 Store-Sub

The messaging complexity of Store-Sub consists of two ingredients. First, subscribers joining the network need to issue their subscriptions to the distributed directory Second, when publishing new content, a publisher needs to retrieve candidate subscriptions from the directory. Note again that we do not model the actual document dissemination, which is an orthogonal task as soon as all matching subscriptions are identified. The number of messages necessary to dispatch all subscriptions of new subscribers S_{new} depends on the size of the network N, the average number of subscriptions sub_{avg}, and the number of directory nodes f_s that each subscription has to be sent to. Depending on the actual implementation, f_s can be as high as $\overline{sub}$ if the subscription needs to be sent to the directory nodes for each subscription term, or as low as 1, if the subscription only needs to be sent to a single topic directory peer (cf. Section 4.1). The number of messages caused by publishers publishing new content depends on the number of publishers $|P|$, the *rate* at which they publish new content, the number of directory peers f_p that they need to retrieve subscriptions from (which, analogously, can be as high as the number of terms in a new document or as low as 1, if each document can be mapped to exactly one topic) and the total size of the network $|N|$. Table 2 summarizes the complexity of Store-Sub.

Table 2. Complexity Store-Sub

	Complexity
Send subscriptions (subscribers)	$O\left(\lvert S_{new}\rvert * sub_{avg} * f_s * \log(N)\right)$
Retrieve subscriptions (publishers)	$O\left(\lvert P\rvert * rate * f_p * \log(N)\right)$

5.2 Store-Pub

The messaging complexity of Store-Pub again consists of two ingredients. First, publishers need to announce their profiles. Second, subscribers need to retrieve the profiles from the directory to identify promising publishers. As the publisher's profiles can only describe previous behavior of the publishers (or, at best, forecast the future based on this previous behavior), we assume that not only new publishers P_{new} entering the system have to distribute their profiles, but also existing publishers $|P|$ have to update their profiles at regular intervals. Since the profiles are feature-(i.e., term- or topic-)specific (i.e., publishers may want to issue their profile w.r.t. each available feature), the number of messages necessary to distribute the profiles depends on the size of the feature space $|F|$, and the size of the network $|N|$. Analogously, we expect subscribers to regularly re-check whether new publishers have become promising sources for their subscriptions, so they regularly retrieve the appropriate profiles from the directory. The messages necessary for this purpose depend on the number of subscribers, S_{new} and S, respectively, the average number of subscriptions per subscriber sub_{avg}, the number of directory nodes f_s that carry profiles relevant to a subscription, and the size of the network $|N|$. Table 3 summarizes the complexity of Store-Pub.

Table 3. Complexity Store-Pub

	Complexity
Send profiles (publishers)	$O\left((\lvert P_{new}\rvert + \frac{\lvert P\rvert}{interval}) * F * \log(N)\right)$
Retrieve profiles (subscribers)	$O\left((\lvert S_{new}\rvert + \frac{\lvert S\rvert}{interval}) * sub_{avg} * f_s * \log(N)\right)$

5.3 Discussion

For Store-Sub, which of the two contributing message types is responsible for the majority of the traffic highly depends on the system parameters. If the network volatility is high, i.e., many new subscribers S_{new} enter the system per time interval, the messages necessary to announce their subscriptions may exceed the traffic caused by the publishers generating content at low rates. Analogously, which of the two contributing message types is responsible for the majority of the traffic in Store-Pub also highly depends on the system parameters, in particular the degree of network dynamics and the average number of subscriptions per subscriber. This dependency is illustrated in Figures 3 and 4. The number of messages necessary to retrieve the subscriptions by the publishers in Store-Sub clearly dominates the messages necessary to store the subscriptions in the directory as the publishing rate increases. The number of messages to retrieve the profiles in Store-Pub dominates the number of messages to actually store the profiles if the average number of subscriptions per subscriber increases (or, analogously, if the average length of subscription increases).

Fig. 3. Store-Sub message types

Fig. 4. Store-Pub message types

6 Experiments

Our experimental contribution is threefold. First, we support our analytical model of the previous section with actual simulations to verify the validity of our cost formulas. The simulation results almost exactly match the numbers forecasted by our analysis; we do not show the corresponding figures as they do not offer any insights. Second, we provide evidence based on our analytical results that, depending on the usage scenario, either one of Store-Sub or Store-Pub is

the method of choice for efficiently implementing a scalable pub/sub application, as they are sensitive to different system parameters. Third, we conducted more elaborate simulations with user and document models in order to measure the actual message and traffic counts for concrete usage scenarios. The numbers back up our analysis that the resource consumption is well below reasonable limits if the implementation method of choice is in line with the expected usage pattern.

6.1 Analytical Results

For the upcoming analytical results we fix the following system parameters (cf. Table 1): $|P| = 100$, $|S| = 100,000$, $sub_{avg} = 3$, $\overline{sub} = 5$, $|T| = 100,000$, $S_{new} = 10$, $|d| = log(|T|) \sim 16$.

Figure 5 shows the sensitivity of both Store-Sub and Store-Pub to changes in the publishing rate, i.e., the amount of new content that each publisher publishes per time interval. While Store-Pub is not sensitive to this parameter, the number of messages in the Store-Sub approach increase as the publishing rate increases, as the publishers have to retrieve data from the distributed directory more often. Figures 7 and 8 show the total number of messages that were generated per round, where one round corresponds to one time interval (as explained for Store-Pub). It can be seen that Store-Sub generally has a larger variation in the number of messages, as the randomness introduced by new subscribers entering the system with a varying number of subscriptions is larger than for Store-Pub, where most of the traffic is introduced by the publishers refreshing their profiles (which is constant over time). Additionally, Store-Sub, as expected performs well at a lower publishing rate, while Store-Pub is immune against changes in the publishing rate.

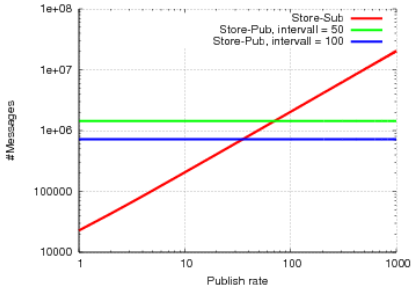


Fig. 5. Sensitivity to Publishing Rate

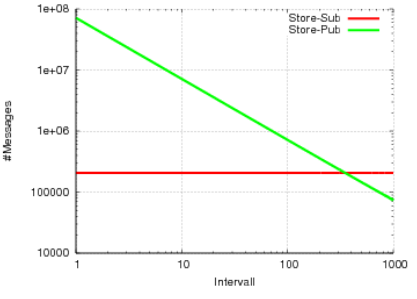


Fig. 6. Sensitivity to refresh time interval

6.2 Simulations

We have implemented a discrete event simulator mimicking the behavior of Store-Sub and Store-Pub. For this purpose, 10 publishers synthetically generate 100,000 documents using a Zipf-distribution over 100,000 terms. To achieve thematically distinct peers, we shift the terms by 20, i.e., p_i starts at t_{i*20} as

its most frequent term to generate its documents. For each publisher, the first 50,000 documents were used as "seeds" to generate the publisher's profiles; the remaining 50,000 were used sequentially whenever a publisher publishes new content. The simulation starts at 5.000 subscribers ($sub_{avg} = 3$; $\overline{sub} = 5$); the average number of subscribers does not change, as expected in the Publisher-to-user scenario. At each round, a random set of 1-10 subscribers leave the system (without any notice to the system), while another 1-10 new subscribers join the system. Figures 7 and 8 show the total number of messages that were generated per round, where one round corresponds to one time interval (as explained for Store-Pub). It can be seen that Store-Sub generally has a larger variation in the number of messages, as the randomness introduced by new subscribers entering the system with a varying number of subscriptions is larger than for Store-Pub, where most of the traffic is introduced by the publishers refreshing their profiles (which is constant over time). Additionally, Store-Sub, as expected performs well at a lower publishing rate, while Store-Pub is immune against changes in the publishing rate.

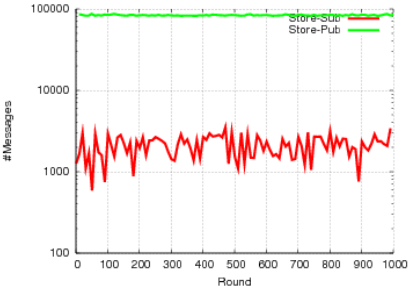


Fig. 7. Publishing rate 1 per round

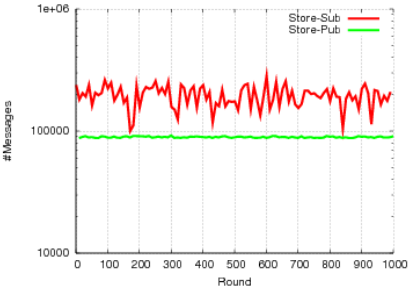


Fig. 8. Publishing rate 100 per round

7 Conclusion and Future Work

We have introduced two general design patterns, Store-Sub and Store-Pub, to implement pub/sub functionality on top of a structured P2P network. While Store-Sub has frequently been the basis for P2P pub/sub prototype system, we are not aware of any prototype implementing the principles of Store-Pub. One key insight of this work is that there is no "one-fits-all" pub/sub approach, but that the optimal design pattern highly depends on a large number of system parameters, such as the expected ratio between subscribers and publishers or the rate at which publishers generate new content. While Store-Sub seems well-suited for a User-to-user scenario where publishers generate content at lower rates, Store-Pub seems attractive for a Publisher-to-user scenario where a small number of publishers generates content at high rates. We will implement both design patterns on top of our Minerva [5] architecture. For Store-Pub, one particularly interesting field of research is how to forecast the future behavior of a publisher based on its previously published content.

References

1. K. Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS*, 2001.
2. I. Aekaterinidis and P. Triantafillou. Internet scale string attribute publish/subscribe data networks. In *CIKM*, 2005.
3. N. J. Belkin and W. B. Croft. Information filtering and information retrieval: Two sides of the same coin? *Commun. ACM*, 35(12), 1992.
4. M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Improving collection selection with overlap awareness in p2p search engines. In *SIGIR*, 2005.
5. M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. Minerva: Collaborative p2p search. In *VLDB*, 2005.
6. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. Planetp: Using gossiping to build content addressable peer-to-peer information sharing communities. In *HPDC*, 2003.
7. A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi. Meghdoot: Content-based publish/subscribe over p2p networks. In *Middleware*, 2004.
8. S. Idreos, M. Koubarakis, and C. Tryfonopoulos. P2p-diet: An extensible p2p service that unifies ad-hoc and continuous querying in super-peer networks. In *SIGMOD Conference*, 2004.
9. G. Koloniari and E. Pitoura. Content-based routing of path queries in peer-to-peer systems. In *EDBT*, 2004.
10. J. Lu and J. Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *ECIR*, 2005.
11. I. Podnar, M. Rajman, T. Luu, F. Klemm, and K. Aberer. Beyond term indexing: A p2p framework for web information retrieval. In *Informatica, Special Issue on Specialised Web Search.*, 2006.
12. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM 2001*. 2001.
13. A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.
14. A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *NGC*, 2001.
15. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM 2001*.
16. D. Tam, R. Azimi, and H.-A. Jacobsen. Building content-based publish/subscribe systems with distributed hash tables. In *DBISP2P*, 2003.
17. C. Tang and S. Dwarkadas. Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *NSDI*, 2004.
18. W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS*, 2003.
19. C. Tryfonopoulos, S. Idreos, and M. Koubarakis. Publish/subscribe functionality in ir environments using structured overlay networks. In *SIGIR*, 2005.

Answering Constrained k-NN Queries in Unstructured P2P Systems

Bin Wang[1], Xiaochun Yang[1,*], Guoren Wang[1,**], Lei Chen[2], Sean X. Wang[3], Xuemin Lin[4], and Ge Yu[1]

[1] School of Information Sciences and Engineering, Northeastern University, China
{binwang,yangxc,wanggr,yuge}@mail.neu.edu.cn
[2] Department of Computer Science, Hong Kong University of Science and Technology
leichen@cs.ust.hk
[3] Department of Computer Science, University of Vermont, USA
xywang@cs.uvm.edu
[4] School of Computer Sci. and Eng., The University of New South Wales, Australia
lxue@cse.unsw.edu.au

Abstract. The processing of k-NN queries has been studied extensively both in a centralized computing environment and in a structured P2P environment. However, the problem over an unstructured P2P system is not well studied despite of their popularity. Communication-efficient processing of k-NN queries in such an environment is a unique challenge due to the distribution, dynamics and large scale of the system. In this paper, we investigate the problem of efficiently computing k-NN queries over unstructured P2P systems. We first propose a location-based domination model to determine a search space. We then present two types of probing strategies, radius-convergence and radius-expanding. A comprehensive performance study demonstrates that our techniques are efficient and scalable.

1 Introduction

Due to their importance in many applications in a variety of domains, k-NN queries have been extensively studied [8,13]. An often used mechanism that provides much needed retrieval efficiency is a centralized index. However, for k-NN queries in a distributed environment, especially unstructured P2P environments, centralized indexing is not a practical solution. The following example shows the reason that new techniques are needed and hence motivate the work of this paper.

Consider a tsunami alarm system for a certain area, e.g., the bay area of Indonesia. Detection of a tsunami in many cases need data from areas that go across multiple nations. Assume the nations establish a logical cooperative

* Supported by National Natural Science Foundation of China No. 60503036 and Fok YingTong Education Foundation No. 104027.
** Supported by National Natural Science Foundation of China No. 60473074.

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 397–405, 2007.

network, where for any two nations, they have either direct cooperative relationship, represented as logical neighbors in the network, or indirect cooperative relationship if their logical neighbors can cooperate. Each nation autonomously maintains a set of sensors to monitor her own sea area and can request data from her cooperative nations. Note that it is not necessary that two logical neighbors are geographically bordered. Therefore, those autonomous nations need to cooperate to answer a distributed k-NN query efficiently. In this application, global indexing is not available for the k-NN query, either, since data values may change dynamically. Furthermore, it is often not possible for all the nations to follow some organization rules (hashing functions) to arrange or store the data as required by a structured P2P system.

Compared to the distributed k-NN problems on P2P [6,7], a constrained k-NN search with value predicates in unstructured P2P systems has posed the following unique challenges: (i) Topology mismatch between the P2P logical overlay network and physical underlying network; and (ii) Data in each peer is maintained autonomously in unstructured P2P systems.

We address the above problems and make the following contributions. (i) We propose a new framework for processing k-NN queries in unstructured P2P systems, (ii) we give a novel filtering mechanism to reduce the communication cost and effectively terminate our search, and (iii) we also give detailed complexity analysis of our algorithms.

The rest of the paper is organized as follows. The formal definition of constraint k-NN queries and a proposed filter model, called domination model, to efficiently prune peers are given in Section 2. Section 3 provides our techniques searching k-NN queries with least communication cost. Experimental results and performance studies are discussed in Section 4. In Section 5, we discuss related work. Finally, we conclude in Section 6.

2 Problem Definition

We assume a set of logically connected, cooperative peers, each covering a spatial region that does not overlap with the spatial regions covered by all other peers. We use a non-directed graph $G = (P, E)$ to model the logical connections, where P is a set of vertices representing the peers and E a set of edges expressing the logical connections between the peers. For a peer $p \in P$, we use $R(p)$ to denote the spatial region it covers and $D(p)$ to denote data set maintained by p. We also assume that each peer has pre-knowledge of spatial regions covered by all other peers. This assumption is reasonable since it is easy for a peer to collect data from different peers and derive this knowledge in an incremental manner.

Each item in $D(p)$ maintained by p has two kinds of attributes, namely location attributes and non-location attributes. Hence, we denote each data item d in $D(p)$ as a pair $\langle\ \boldsymbol{v_l}(d),\ \boldsymbol{v_n}(d)\ \rangle$, where $\boldsymbol{v_l}(d)$ ($\boldsymbol{v_n}(d)$, resp.) is a value vector of the location (non-location resp.) attributes of d. For each d in $D(p)$,

$\boldsymbol{v_l}(d)$ must be contained in the region $R(p)$. A k-NN query q is composed of two parts, location value and non-location value predicate, denoted as $\langle\ \boldsymbol{v_l}(q), q_c\ \rangle$, respectively. A value predicate returns, when applied to a non-location value vector, true or false. We define the distance between q and data d as following:

Definition 1. *Given a query q and data d, let the distance between q and d be*

$$dist(q,d) = \begin{cases} dist(\boldsymbol{v_l}(q), \boldsymbol{v_l}(d)) & \text{if } q_c(\boldsymbol{v_n}(d)) = true \\ \infty & otherwise \end{cases}$$

where dist can be any of the L_p-norm.

Given a query q and a peer p, we use $minDist(q,p)$ ($maxDist(q,p)$, resp.) to express the minimum (maximum, resp.) distance between q and all the points in $R(p)$.

Formal Problem Statement: Assume we have a set of logically connected peers $p_1, \ldots, p_n$, each of which manages a set of data items, denoted $D(p_i)$, with locations of each data item in $D(p_i)$ being in the spatial region $R(p_i)$. Assume further that $R(p_i) \cap R(p_j) = \emptyset$ for all $i \neq j$. Given a continuous query q issued by a peer p_i, continually search k data items $\{d_1, \ldots, d_k\}$ among the data items in $D(p_1) \cup \cdots \cup D(p_n)$ such that there does not exist any data item d satisfying the condition $dist(q,d) \leq dist(q,d_h)$ for some $1 \leq h \leq k$. We aim to minimize total communication costs.

The intuition of the domination model comes from the following simple facts: If we know an upperbound r^u of the distance from the query point q to the k^{th} data item in the query results, then a peer p' does not need further probing (for possible answers) if $r^u \leq minDist(q,p')$. Similarly, if we know that a peer p provides the k^{th} item in the answer set, then a peer p' does not need further probing if $maxDist(q,p) \leq minDist(q,p')$. These facts form the basis for our optimized search algorithms, which are described in Section 4.

Definition 2. *(Dominate relationship) Given a query q, and two peers p_1 and p_2, if $maxDist(q,p_1) \leq minDist(q,p_2)$, we then say that p_1* dominates *p_2, denoted $p_1 \prec_q p_2$, or $p_1 \prec p_2$ when q is understood. For two groups of peers P_1 and P_2, if for each peer $p \in P_1$ and each peer $p' \in P_2$, $p \prec p'$, we say P_1 dominates P_2, denoted $P_1 \prec P_2$.*

3 Pruning Candidate Peers

We propose two probing strategies, namely the *radius-convergence strategy* and the *radius-expanding strategy*. The radius-convergence strategy shrinks the probe radius gradually until all peers in the probe circle have been probed, while the radius-expanding strategy gradually expands the probe radius.

3.1 The Radius-Convergence Approach

The first strategy that we introduce for direct probing is radius-convergence strategy, which keeps shrinking the proving radius when more peers are probed and closer data are found. For each probe, it keeps the new k nearest data and let the new probe radius r_k be the Euclidean distance between q and the new k^{th} data. This procedure repeats until all peers overlapping the circle with the (shrinking) r_k have been probed.

Ideally, we prefer to find a "better" peer that can get a close probe radius in the first a few times probe. Seeking "better" peers to start with can be considered as a rank aggregation (RA) problem [5], which searching top candidates from a set of sorted rank list. We propose an algorithm to choose best-k peers in the probe circle and put emphasis on the minimum distance.

Our *RA-based approach* is shown in Algorithm 1. Line 1, in Algorithm 1, normalizes the attribute values $\langle min, max, plen\rangle$ for minimum distance, maximum distance, and the logical paths of each peer to the range of [0,1], so that they have the same weight for estimating a peer. Line 2 sorts all peers according to minimum distance attribute in the ascending order. Lines 3-12 choose the top-k peers with the smallest grades. Compare it with the random-based approach, RA-based approach can decrease probe radius quickly, that is, probes peers with higher probabilities to provide closer answers.

Algorithm 1: RA-based algorithm

Input: $P = \{p_1, ..., p_h\}$, each $p_i = \langle id, min, max, plen\rangle$, $k(\leq h)$, geographical weight w_g, logical weight w_l

Output: a set of top-k peers

```
1: normalize P to P';                    // p.min, p.max, p.plen ∈ [0, 1]
2: l1=sort P' according to min;  List result = ∅;   count = 0;   λ = 0;
3: for i = 1; i ≤ |P|; i++ do
4:    λ=wg×((l1+i).min+(l1+i).max)+wl×(l1+i).plen;
5:    r = last element in result;
6:    if λ ≤ wg× (r.min+r.max)+wl× r.plen then
7:       insert {(l1+i).id} to result;   count++;
8:       if count==k then  return result;
9:       else  insert (l1+i) in candidate in ascending order;
10:      end if
11:   end if
12: end for
13: return result;
```

Note that, even some peers can provide k closest data, we are not able to prune out all the rest peers whose minimal distances to q are less than the new query radius r_k. We then choose another top-k peers, using the same procedure, in the remaining peers, and probing these peers to get another set of k'-closest satisfying data ($k' \leq k$). We choose the first k data among these two sets of returned satisfying data and shrink r_k. The above procedure repeats until all peers in the shrunk probe circle have been probed.

3.2 The Radius-Expanding Approach

We propose a *partition-based approach* (PA in short) to gradually grow the probing radius. Ideally, we hope to find a small set of peers P_1, such that if P_1 can answer the k closest answers, then all other peers can be safety pruned out. For peers in each group, we build up a steiner tree using the local filtering mechanism. Since messages are forwarded through edges of the constructed tree, we hope to find and merge groups (such as P_1 and P_2) such that the summation of edges of corresponding steiner trees T_1 and T_2 is minimal. In fact, this property has been proved and stated in the following Theorem.

Theorem 1. *Given an undirected graph $G(P, E)$ and two set of target nodes $P_1 \subseteq P$ and $P_2 \subseteq P$, the summation of edges of the two trees $T(P_1)$ and $T(P_2)$ is not less than the edge of $T(P_1 + P_2)$, i.e. we have the following property*

$$|T(P_1)| + |T(P_2)| \geq |T(P_1 + P_2)|.$$

The basic task for the PA is the following. Given an undirected graph $G = (P, E)$, a set of terminal nodes $P' \subseteq P$, and subsets $P_1 \prec \ldots \prec P_h$, $P_i \subseteq P'$ $(1 \leq i \leq h)$, find an optimal partition of $P' = \{C_1, C_2\}$, such that neither C_1 nor C_2 is empty, each P_i can only belong to C_1 or C_2 (but not both), and the sum of number of edges in steiner trees for $\{C_1, C_2\}$ is minimal. PA combines geographical dominate relationship and logical steiner tree together. We use Algorithm 2 to greedily approximately build two steiner trees.

Algorithm 2: PA

Input: k, undirected graph G, source node s, and terminal nodes $P' = \{P_1, ..., P_h\}$
Output: two steiner trees T_1 and T_2

1: **for** each P_i between P_1 and P_k **do**
2: ST_i = greedyST(G, $\{s\} \cup P_i$);
3: **end for**
4: find two minimal trees, assuming they are for P_u and P_v;
5: $T_1 = ST_u$; $RT_1 = P_u$; $T_2 = ST_v$; $RT_2 = P_v$;
6: **for** each peer P_i in $P' - RT_1 - RT_2$ **do**
7: pick the a minimal tree ST_i for P_i;
8: T_1' = greedyST($T_1 \cup T_i$, $\{s\} \cup RT_1 \cup P_i$); T_2' = greedyST($T_2 \cup T_i$, $\{s\} \cup RT_2 \cup P_i$);
9: **if** $edge(T_1') \leq edge(T_2')$ **then** $T_1 = T_1'$; $RT_1 = RT_1 \cup P_i$;
10: **else** $T_2 = T_2'$; $RT_2 = RT_2 \cup P_i$;
11: **end if**
12: **end for**

PA classifies peers into two groups C_1 and C_2 corresponding to the two steiner trees T_1 and T_1, respectively. Peers in C_1 can be probed together. Note that, groups in $C_2 = C - C_1$ may contain a group P_j dominates groups in C_1. If the first round probe gets k' $(\leq k)$ closest answers, then using the locations of these k' answers, we can know k_1 $(\leq k')$ out of k' answers are returned by P_1. Then it classifies $P' - C_1$ into two classes and request $k - k_1$ to C_1. PA iterative repeats the above procedures until k closest answers are returned.

4 Experimental Study

In this section, we built a peer-to-peer simulator to evaluate the performance of our proposed system over large-scale networks. To evaluate the cost of query processing, we tested the network with different number of peers N from 200 to 1800. Each peer contains a set of data from 1K to 200K with two dimensional location values and one non-location value. We generated two datasets. Dataset 1 conforms to uniform distribution and dataset 2 conforms to normal distribution. All approaches were implemented in C++ and run on Intel XEON(TM) 3.2GHz dual-CPU with 2G RAM on Windows 2003 Server. For each setting, we tested an algorithms by running it 10 times to compute the average result.

Comparison of different k-NN searches. We compared the communication cost (represented as # of messages) of getting k-NN results using different search strategies. We ran 10 different k-NN queries whose locations and predicate values are randomly specified. Besides flooding search, we tested two radius-convergence approaches and two radius-expanding approaches. Radius-convergence approaches include random search and RA-based search, whereas radius-expanding approaches include exhausted search every dominate groups (EX in short) and Partition-based search (PA in short). We used two datasets with uniform and normal distributions, respectively. In addition, we also compared three straightforward approaches according to the ranking of the minimal and the maximal geographical distances and the shortest path in logical graph, respectively.

Fig. 1 shows the conveying messages of these algorithms using two datasets conforming to uniform and normal distributions. We let the number of peers vary between 500 and 2500, and ran 10 different 100-NN queries. Figs. 1(a) and (b) compare five approaches using dataset 1. It shows that flooding search costs the most conveying messages, whereas PA uses the least conveying messages. Our proposed two approaches RA and PA are all independent on the number of peers. RA is the second best approach. Fig. 1(c) shows the similar result using dataset 2. Figs. 1(b) and (d) show the number of conveying messages when fixed the number of peers to 1000 and varied k from 20 to 100. They all show that both RA and PA outperform than the other approaches, and PA always use least messages to get k-NN nearest results.

Comparison of filtering capabilities. We tested the capabilities of filtering peers using PA and RA approaches on dataset 1. We first changed the number of k values from 20 to 70. Fig. 1(e) shows the filtering capability decreased when increasing k values. RA has less filtering capability than PA. The reason is RA is a radius-convergence approach that gradually shrink query radius. It keeps probing at least k peers for each iteration until all peers in the shrinking query radius have been probed. Whereas, PA is a radius-expanding approach that when k closest data is met, it can stop. Fig. 1(f) shows the filter capabilities when varying the number of peer groups. It shows that PA has better filter capability than RA. When the peer group number increased to 16, the capability of PA climbs to a peak value 52%, whereas RA has no peak value. It proved that the

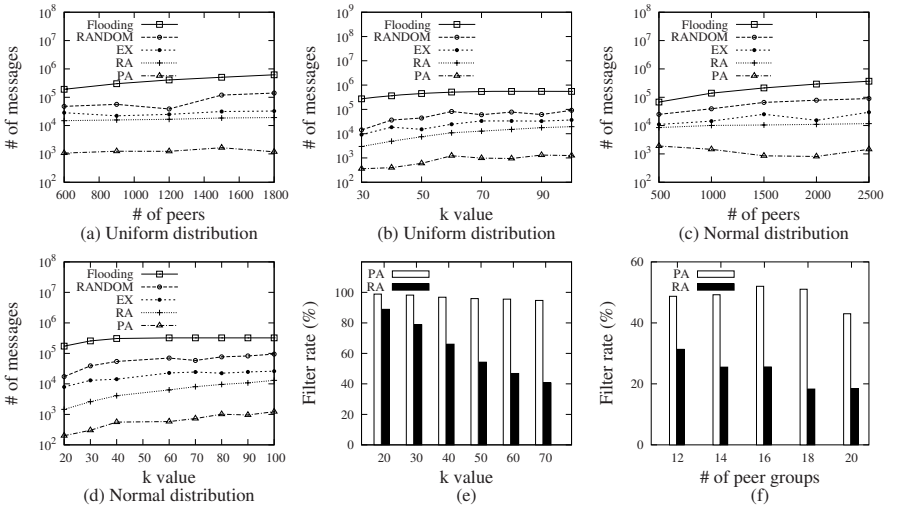


Fig. 1. Comparison of different k-NN searches

filter capability of PA depends on the chosen number of groups in all dominate group set, whereas RA does not since RA only considers the first k ranked peers using geographical distances and logical graphs.

5 Related Work

As far as we aware, there is no similar work has been on the proposed problem setting. The most closed work to ours fall in to two categories: continuous k-NN searches over spatial database and k-NN queries over structured P2P systems. We will review the work in this two directions and explain the difference between our work and theirs. There are many work have been proposed for continuous k-NN queries over moving objects in spatial database domain. Most of these works focus on reducing the number of updates to the indexes. In order to achieve this, the trajectories of moving objects are modeled by some linear functions, thus a R-tree can be built use time as a function [12,16]. Compared to these work, our problem setting is for distrusted environment and we do not assume existence of a centralized index. Moreover, we have the logical communication cost as a constraint to the k-NN queries.

k-NN queries over P2P systems can be classified into search over unstructured P2P and structured P2P. For structured P2P systems, data allocation strategies are important for k-NN search. Distrusted Hashing Table (DHT) is often used to allocate data, such as CAN [9], Chord [15], Pastry [11], and Tapestry [17], which use uniform hash functions and achieve good load balance. However, these hashing functions destroys data locality (data that are similar should be allocate near to each other in the space). Complicated queries such as k-NN have to rely

on multi-cast or additional indexes. Some locality-preserving data allocation approach are also proposed, such systems include P-Grid [2], P-Ring [4], Baton [7], Vbi-tree [6], and Mercury [3]. The basic idea of these approaches is to keep data locality over the attribute as much as possible. For unstructured P2P systems, very few work have been done so far. Gnutella [1] using flooding techniques to do k-NN search. Compared to these work, our work focus on continuous constrained k-NN search over unstructured P2P system. Besides finding the k-NN, we have to guarantee that searched value satisfying the value predicate specified in the query. Furthermore, in contrary to traditional setting of P2P systems that each peer maintains static data, a peer in our system maintains dynamic data.

6 Conclusion

This paper has investigated the new problem of processing the constrained k-NN queries over unstructured P2P systems, and proposed two approaches to efficiently filter peers in the search space. The experimental results on the two synthetic datasets have shown that (i) the algorithms proposed outperform most the other heuristic algorithms, (ii) the novel adaptive histogram can save more communication cost, and (iii) our technique can efficiently process continuous k-NN queries in a distributed, and large scale environment.

References

1. Gnutella. http://www.gnutella.com/.
2. K. Aberer, P. C. Mauroux, and et al. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003.
3. A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proc. of the 2004 ACM SIGCOMM*.
4. A. Crainiceanu, P. Linga, and et al. P-ring: an index structure for peer-to-peer systems. Technical report, TR2004-1946, Cornell University, 2004.
5. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.
6. H. V. Jagadish, B. C. Ooi, and et al. Vbi-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In *ICDE*, pages 34-45, 2006.
7. H. V. Jagadish, B. C. Ooi, and Q. H. Vu. Baton: a balanced tree structure for peer-to-peer networks. In *VLDB*, pages 661–672, 2005.
8. F. Korn, N. Sidiropoulos, and et al. Fast nearest neighbor search in medical image databases. In *VLDB*, pages 215–226, 1996.
9. S. Ratnasamy, P. Francis, and et al. A scalable content addressable network. In *In ACM SIGCOMM*, pages 161–172, 2001.
10. G. Robinsy and A. Zelikovskyz. Improved steiner tree approximation in graphs.
11. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *In IFIP/ACM Distributed Systems Platforms*, pages 329–350, 2001.
12. S. Saltenis, C. S. Jensen, and et al. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331-342, 2000.

13. T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. pages 154–165, 1998.
14. A. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *ICDE*, 2006.
15. I. Stoica, R. Morris, and et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, pages 149–160, 2001.
16. Y. Tao and D. Papadias. MV3R-tree: A spatio-temporal access method for timestamp and interval queries. In *VLDB*, pages 431–440, 2001.
17. B. Y. Zhao and J. Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, UC Berkeley, 2001.

Scalable IPv4/IPv6 Transition: A Peer-to-Peer Based Approach

Jun Bi, Xiaoxiang Leng, and Jianping Wu

Network Research Center, Tsinghua University
Beijing 100084, China
junbi@tsinghua.edu.cn

Abstract. This paper presents a scalable and robust IPv4/IPv6 transition method utilizing Peer-to-peer technology. The gateways of IPvX (IPv4 or IPv6) islands automatically set up an IPvX P2P network upon IPvY (IPv6 or IPv4) network to exchange and maintain IPv4/IPv6 transition information. IPvX in IPvY tunnels between IPvX islands can be established automatically according to the IPv4/IPv6 transition information.

1 Introduction

IPv6 has been recognized as a protocol for next generation Internet. Due to the long term transition from IPv4 to IPv6, IPv4/IPv6 transition is an important research topic and a bunch of transition methods had been proposed. During the transition period, there will be many IPvX (IPv4 or IPv6) islands in IPvY (IPv6 or IPv4) could. An IPvX island in IPvY clouds wants to communicate with other IPvX islands, and also wants to communicate with IPvX backbone [1]. According to the latest IPv4/IPv6 transition requirements issued by IETF 6trans working group, there is no ideal approach to handle this scenario.

A MP-BGP based mechanism has just been presented by IETF softwire WG [2] to connect isolated IPvX networks together over IPvY backbone. In this method, MP-BGP is extended to propagate TEP (Tunnel End Point) information inside the IPvY backbone and set up full mesh tunnels between border routers of isolated IPvX networks. However, this method can only be used on the border routers (AFBR) of some large area IPvX networks with MP-BGP support, and can not be widely deployed on the edge gateways of isolated IPvX islands. Furthermore, the BGP peers for one router are configured manually and cannot change automatically with the change of network environment.

This paper presents a Peer-to-Peer based IPv4/IPv6 transition method. The P2P network is set up automatically to handle the control plane information and flood IPv4/IPv6 address transition table. In the data plane, the IPv6-in-IPv4 or IPv4-in-IPv6 tunnel between the source gateway and destination gateway is established according to the IPv4/IPv6 transition table. Compared with existing methods, the proposed method can handle all transition requirements: (1) Automatic; (2) High performance; (3) Without using specific IPv6 address such

G. Moro et al. (Eds.): DBISP2P 2005/2006, LNCS 4125, pp. 406–416, 2007.

as 6to4 addresses. The method can be implemented in IPv4/IPv6 dual-stack routers or linux boxes that are used as gateways between IPvY clouds and IPvX islands.

The mechanism and protocol design of the proposed method will be presented in Section 2 and Section 3. Section 4 analyzes the reliability P2P network. Section 5 introduces the prototype and experiments and Section 6 concludes the paper.

2 Mechanism

In [3], a DHT-based P2P network is used to forward the IPv6 data packets. However DHT is not suitable for the longest matching in packet forwarding. The method proposed in this paper doesn't rely on structural P2P network.

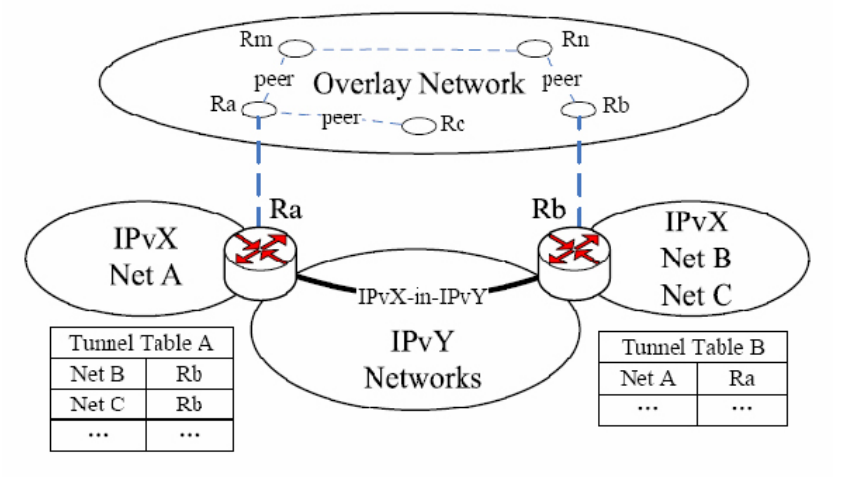


Fig. 1. The IPv4/IPv6 transition architecture

The architecture of the proposed method is shown in Figure 1. Ra, Rb, etc. are edge gateways of IPvX islands. A P2P network is maintained by them to share TEP (Tunnel End Point) information. With the TEP information, shortcut tunnels are established on each gateway for transferring data packets between these islands. The mechanism is described as below:

1. A node of the P2P overlay network is a gateway of IPvX subnet or IPvX backbone
2. There is a registration server, when a node joins the P2P network, it communicate with the register server and receives the node list with Register/Replay packets.

3. The new node select three neighbor nodes from the node list then set up adjacencies with 3 neighbors with Hello, Request, Update and Ack packets.

4. The adjacency is kept alive by exchange Hello packets every 1 second.

5. The Update packets that contain local IPv4/IPv6 address transition table are flooded all over the P2P network, so that every node knows the destination IPvX network and its gateway's IPvY address.

6. When an IPvX host in an IPvX island sends an IPvX packet to an IPvX destination, the source gateway searches the gateway's IPvY address according to destination IPvX address form transition table. Then encapsulates the IPvX packet into IPvY packet and sends our to the destination gateway.

7. When the destination gateway receives the IPvX in IPvY packet, it reads the IPvX packet and sends it to the destination IPvX host.

8. When a node wants to quit the P2P network, it sends out an Update packet to remove the IPv4/IPv6 transition table.

9. When a node figures out its neighbor is dead, it selects a new neighbor from the node list, and keep the total number of its neighbor at least 3.

The registration server is a normal P2P node and is kept update by flooding mechanism. When a node quits or is dead, the registration server removes this node form the node list.

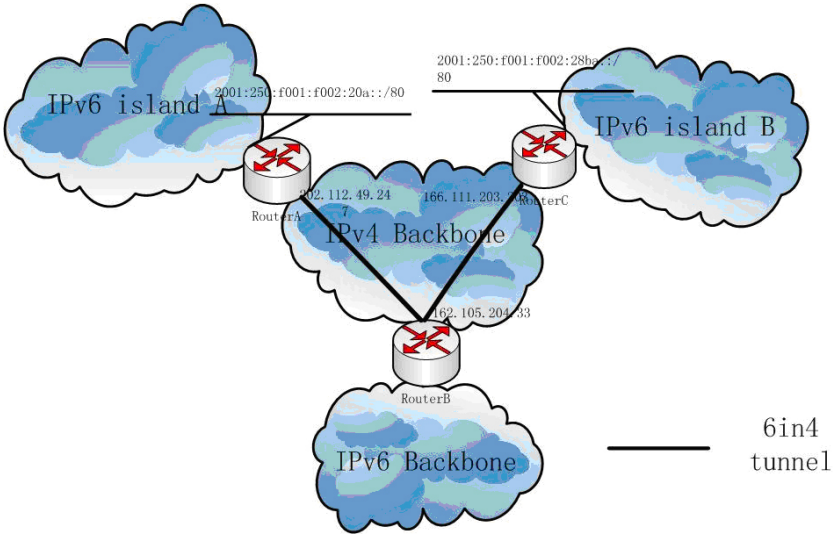


Fig. 2. An example of P2P based IPv6 over IPv4 network

Figure 2 shows an example of P2P based IPv6 over IPv4 Network. The proposed method is high performance with lower cost, because: (1) It separates the data plane and control plane. The control plane only handles flooding of transition table. (2) The control and data plane mechanism are fully automatic, no

configuration is required after a node is assigned to join the P2P network. (3) The IPv6 island can be assigned with any IPv6 address (compared with specific 6to4 addresses), therefore after the transition period, it does not need to be reconfigured to directly connect the IPv6 backbone. (4) The recovery mechanism is highly reliable (will be discussed in section 4).

3 Protocol Design

We designed six types of control packets: Hello, Request, Update, Ack, Register, Reply. The control packets are transported using UDP port 2005.

1. The Hello packet is designed for adjacency setup and neighbor keeping-alive.
2. The Request packet is designed for transition table request.
3. The Update packet is designed for transition table flooding.
4. The Ack packet is designed for acknowledgement of update packets.
5. The Register packet is designed for a new node to request joining the P2P network.
6. The Reply packet is designed for a registration server to answer the registration request, by replying IPvY address list of all the current P2P nodes.

The state machine of Registration is shown in Figure 3.

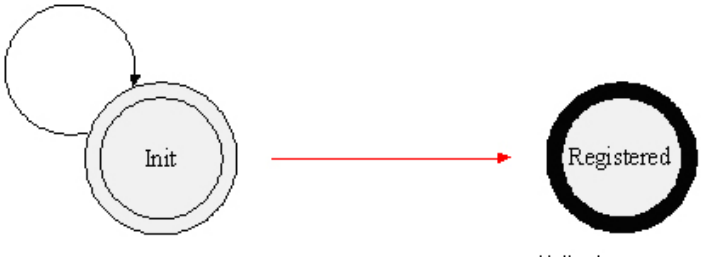


Fig. 3. The state machine of registration

When the node sends out Registration packet, it keeps Init state. After it receives Reply packet, the node transits to Registered state.

The neighbor state machine is shown in Figure 4.

In the None state:

1. When a type 0 Hello is received, it sends out type 1 Hello and transits to OneWay state.
2. When a type 1 Hello is received, it sends out Request and transits to TwoWay state.

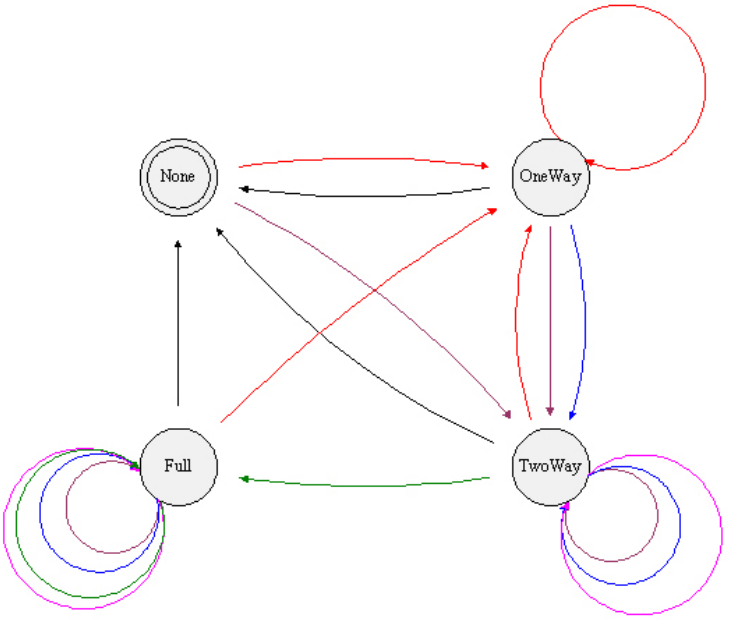


Fig. 4. The neighbor state machine

In the OneWay state:

1. When a type 0 Hello is received, it sends out type 1 Hello and keeps OneWay state.
2. When a type 1 Hello is received, it sends out Request and transits to TwoWay state.
3. When a Request is received, it sends out Update and transits to TwoWay state.

In the TwoWay state:

1. When a type 0 Hello is received, it sends out type 1 Hello and keeps OneWay state.
2. When a type 1 Hello is received, it sends out Request and keeps TwoWay state.
3. When a Request is received, it sends out Update and keeps TwoWay state.
4. When an Update is received, it sends out Ack and Update, then keeps TwoWay state.
5. When an Ack is received, it transits to None state.

In the Full state:

1. When a type 0 Hello is received, it sends out type 1 Hello and keeps OneWay state.
2. When a type 1 Hello is received, it keeps Full state.

3. When a Request is received, it sends out Update and keeps Full state.
4. When an Update is received, it sends out Ack and Update, then keeps Full state.
5. When an Ack is received, it keeps Full state.

4 Analysis of P2P Network Connectivity

Since there is no special structure maintained with the P2P network used in the proposed method, the quit or failure of some nodes may cause the P2P network to split into multiple disconnected subnets. In this section, the connectivity of the P2P network will be analyzed. It's hoped to find an effective way to keep the P2P network connected.

Paul Baran did some research on distributed communication networks which consist of unreliable nodes and unreliable links. From the viewpoints of Node Destruction and Link Destruction, his simulation and analysis shows that, with a certain node degree, the distributed network can provide a reliable communication service under a given probability of node failure and/or link failure [4]. Afterward, the situation of Link Destruction is proved by the theorem about sharp threshold in a random graph that: if G is a k-connected graph of n vertices and the edge probability p(n) satisfies $p(n) \geqslant c\log(n/k)$ for a large enough absolute constant $c > 0$, then a.s. the random sub-graph Gp is connected [5].

Furthermore, the reliability analysis of the random multicast protocol Gossip shows the sharp result that if there are n nodes, and each node gossips to logn+k other nodes on average, then the probability that everyone gets the message converges to $\exp(\exp(-\lambda))$ [1].

This research gives us inspiration in analyzing the connectivity of the P2P network. The major difference between physical network and P2P overlay network is that in a P2P network, the neighbor nodes for a node are not fixed. When a node finds one of its neighbors unreachable, it can establish a new neighborhood with another live node. Thus the probability of disconnection in a P2P network may be much lower than physical network.

As described in Section 2, a simple Random Recovery Scheme is used to avoid the disconnection of the P2P network by setting a minimal degree d for each node and randomly choosing new neighbors for recovery. The effectiveness of this recovery scheme will be analyzed by simulation. The feasible scope of the variable d to keep the P2P network connected will be discussed. When simulating, the following two kinds of failures are considered:

(1) Single Node Destruction (SND). Each node in the P2P network is destroyed with a certain probability. The probability for each node is independent of the probabilities for other nodes.

(2) Single Virtual Link Destruction (SVLD). Each virtual link between two overlay neighbors is destructed with a certain independent probability. An example of SVLD would be transport timeout or de-convergence of routing after some physical failure.

The situation of Single Node Destruction, Single Virtual Link Destruction and the combination of SND and SVLD will be simulated in the following parts. The steps of simulation are as follows:

(1) Randomly build a N-node network where the nodes join the network gradually with minimal degree d;
(2) Destroy each node or virtual link independently with a certain probability;
(3) Rebuild the network with the Random Recovery Scheme;
(4) Check the connectivity of the rebuilt network;
(5) Loop step (1) - (4) for 100,000 times to get the probability of disconnection.

4.1 Single Node Destruction (SND)

In this situation, unlike the physical network, along with the increasing probability of SND, the probability of disconnection of the P2P network can be restricted to a low level with the Random Recovery Scheme.

Figure 5(a) shows the simulation result of SND with node number N=10000. The probability of disconnection stays less than 0.2 when d is 2. And this probability decreases to less than 10-5 when d $\geqslant$ 3, whatever the probability of SND is. In another word, the probability of disconnection of this P2P network is close to 1.0 as long as d is equal to or greater than 3.

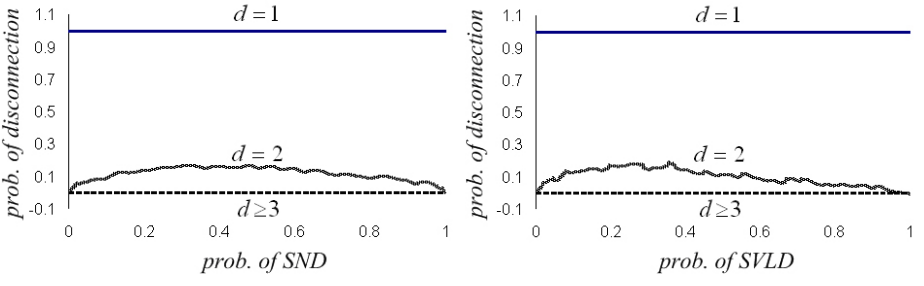


Fig. 5. Simulation results of SND or SVLD, N=10000: (a) SND (b)SVLD

4.2 Single Virtual Link Destruction (SVLD)

Similar to SND, when the virtual links destructed separately, the Random Recovery Scheme also can prevent disconnection of the P2P network effectively. As shown in Figure 5(b), the probability of connection of the P2P network is close to 1.0 when d $\geqslant$ 3.

4.3 SND+SVLD

As shown in Figure 6(a) and Figure 6(b), in the situation of both SND and SVLD, similarly, the P2P network can keep connected when d $\geqslant$ 3.

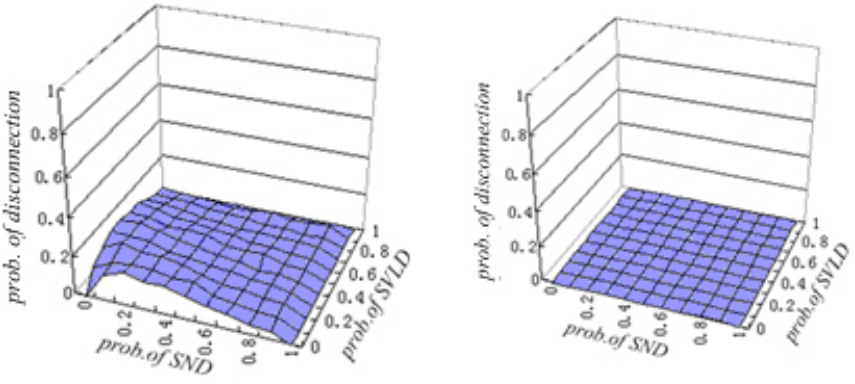


Fig. 6. Simulation results of SND + SVLD, N=10000): (a) d=2 (b) d ⩾ 3

4.4 Influence of Node Number N

A threshold C is defined to indicate the minimal value of d while the disconnection probability of the P2P network keeps less than 10-5. In other words, when dC, the P2P network can keep connected whatever the probability of SND or SVLD. With node number N from 1 to 10000, the relationship between C and N is shown in Figure 7. With increasing N, C grows quickly to 3 and stabilizes.

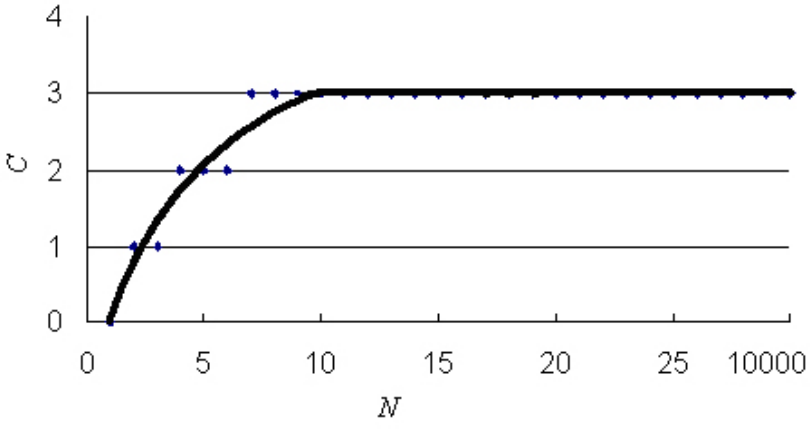


Fig. 7. Relationship between C and N

4.5 Summary

From the analysis above, when the node number is less than 10000, the P2P network used in the proposed method can keep connected as long as the minimal degree of each node d is equal to or more than 3. For some limitations of simulation environment, the situation with node number greater than 10000 is

difficult to simulate. From the analysis above, we can infer that when d is equal to or more than 3, the P2P network can keep connected no matter what the node number is.

5 Prototype and Experiments

5.1 The Prototype

Up to now, the architecture and protocol details of the proposed method has been designed already. A Linux-based prototype system has also been developed. As shown in Figure 8, our implementation on the edge gateways of IPvX islands has four functional modules:

1. P2P System;
2. RT Control;
3. Tunnel Table;
4. Virtual Interface.

The P2P System module processes configuration commands that define the information of tunnel end point of local islands edge gateways and gets the TEP information of other islands from an overlay P2P network.

The RT Control module receives the TEP information from P2P System, forms the Tunnel Table, and inserts the forwarding information into the Routing Table of the gateway to assign the direct tunnels with higher route precedence

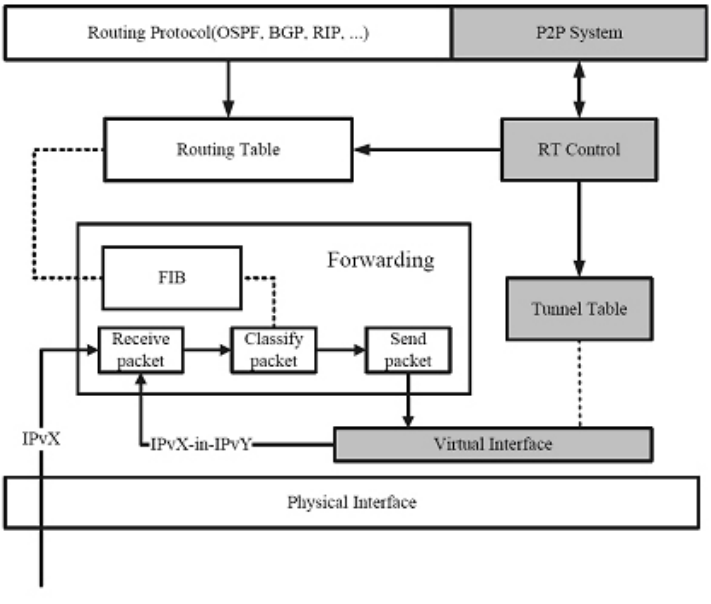


Fig. 8. The prototype system

than other routes and to make the traffic to the destination islands all forwarded to the Virtual Interface.

The module Tunnel Table holds the information of direct tunnels between IPvX islands, and point out the IPvY address of the destination gateway for the Virtual Interface when packet forwarding.

When the Virtual Interface receives a data packet to a destination IPvX island, it looks up the Tunnel Table to find out the IPvY address of the remote island gateway, encapsulates this packet with an IPvY header, and sends it back to the forwarding scheme of local gateway.

5.2 The Experiments

With this prototype system, the proposed method has already been deployed on four isolated IPv4 island gateways inside CERNET2 (a nationwide IPv6 backbone in China). These four islands all have some inner IPv4 users and services (i.e., web, ftp, media steaming, etc), and are connected with the same relay gateway through IPv6-in-IPv4 tunnels to access native IPv4 network. Both the island and relay gateways each has a 100Mbps Fast Ethernet link with the IPv6 network. We have the following two experiments to validate the effect of the proposed method from the viewpoint of reducing the burden of relay gateway and increasing the access performance of isolated islands.

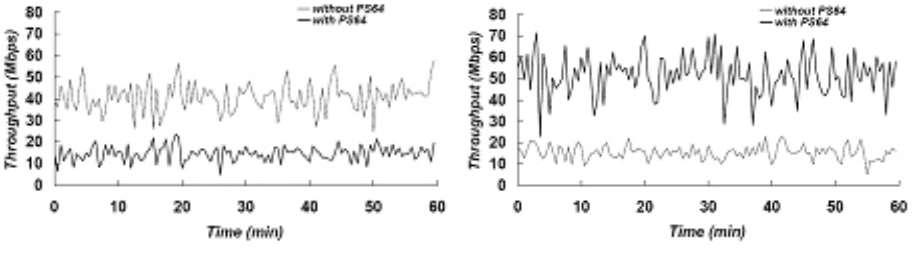


Fig. 9. Experiment results: (a) Effect on reducing the burden of relay (b) Effect on increasing access performance

In the first experiment, we limit the bandwidth of IPv4 island gateways to 10Mbps. This makes the bottleneck of this system not the relay but the island edge gateways. That means no matter whether the proposed method is used, the throughput of island gateways will be kept only 10Mbps. In this way, we can clearly notice the effect of the proposed method on reducing the burden of relay gateway.

We note the throughput of relay gateway with/without the proposed method used on island gateways for one hour. As shown in Figure 9(a), the result, with an average bit rate about 40Mbps without the proposed method while only about 15Mbps with the proposed method, shows that the proposed method can greatly reduce the reliance on the relay. Besides, from this experiment, we can see that only 37.5% traffic is outside the scope of IPv4 islands.

In the second experiment, we focus on the throughput of IPv4 island gateways with no limit at all. The bit rate of one island gateway in one hour is shown in Figure 9(b). From this result, we can see that the throughput changes from about 16Mbps to 58Mbps in average when the proposed method is in use. In other words, the proposed method can effectively increase the access performance of isolated islands.

6 Conclusions

This paper presents a new IPv4/IPv6 transition method based on Peer-to-peer technology. This method can be used to provide scalable and reliable IPvX (IPv4 or IPv6) service for IPvX islands inside IPvY (IPv6 or IPv4) cloud. The proposed method has the following advantages, compared with current methods:

1. Robustness: Our experience shows that if a node keeps its neighbor number more than three, then our neighbor recovery mechanism can always keep the P2P network connectivity.
2. Decentralization: The resignation server is also a member of P2P node, and being updated by flooding mechanism. Therefore, the P2P network doesn't has a potential bottleneck.
3. Scalable: The simulation shows that the number of nodes can be scale from 10 to 10000.
4. High performance: The IPvX in IPvY tunnels are automatically established between endpoint gateways and the data are transported and handled distributed by endpoint gateways.

The current design and implementation depends on IPvX-in-IPvY tunneling technology. In future, more work will be done to extend the propose method to support different type of encapsulation, such as GRE, IPvX-in-UDP-in-IPvY, etc.

References

1. Lind, M., Ksinant, V., Park, S., Baudot, A., Savola, P.: Scenarios and Analysis for Introducing IPv6 into ISP Networks. Request for Comments 4029. (2005).
2. IETF Softwires Working Group: http://www.ietf.org/html.charters/softwire - charter.html.
3. Zhou, L., Renesse, R.: P6P: A Peer-to-Peer Approach to Internet Infrastructure. IPTPS. (2004).
4. Baran P.: On Distributed Communication Networks. IEEE Transactions on Communication Systems. (March 1964).
5. Krivelevich, M., Sudakov, B., Vu., V.: A sharp threshold for network reliability. Combinatorics, Probability and Computing. (2002).
6. Kermarrec, A., Massoulie L.: Probabilistic reliable dissemination in large-scale systems. IEEE Transactions on Parallel and Distributed Systems. (March 2003).

Author Index