DB-Enabled Peers for Managing Distributed Data

Beng Chin Ooi, Yanfeng Shu, and Kian Lee Tan

Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
{ooibc,shuyanfe,tankl}@comp.nus.edu.sg

Abstract. Peer-to-peer (P2P) computing is the sharing of computer resources, services and information by direct negotiation and exchange between autonomous and heterogeneous systems. An alternative approach to distributed and parallel computing, known as Grid Computing, has also emerged, with a similar intent of scaling the system performance and availability by sharing resources. Like P2P computing, Grid Computing has been popularized by the need for resource sharing and consequently, it rides on existing underlying organizational structure. In this paper, we compare P2P and Grid computing to highlight some of their differences. We then examine the issues of P2P distributed data sharing systems, and how database applications can ride on P2P technology. We use our BestPeer project, which is an on-going peer-based data management system, as an example to illustrate what P2P computing can do for database management.

1 Introduction

Peer-to-peer (P2P) technology, also called peer computing, is an emerging paradigm that is now viewed as a potential technology that could re-architect distributed architectures (e.g., the Internet). In a P2P distributed system, a large number of nodes on the edge (e.g., PCs connected to the Internet) can potentially be pooled together to share their resources, information and services. These nodes, which can both consume as well as provide data and/or services, may join and leave the P2P network at any time, resulting in a truly dynamic and ad-hoc environment. The distributed nature of such a design provides exciting opportunities for new killer applications to be developed.

The concept of P2P is not new. The pervasiveness of the Internet and the publicity gained as a result of music-sharing have caused researchers and application developers to realize the untapped resources, both in terms of computer technology and information. Edge devices such as personal computers are connecting to each other directly, forming special interest groups and collaborating to become a large search engine over information maintained locally, virtual clusters and file systems. Indeed, over the last few years, we have seen many systems

being developed and deployed; e.g., Freenet [5], Gnutella [6], Napster [15], ICQ [10], Seti@home [20] and LOCKSS [14].

The initial thrusts on the use of P2P platform were mainly social. Applications such as ICQ [10] and Napster [15], enable their users to create online communities that are self-organizing, dynamic and yet collaborative. The empowerment of users, freedom of choice and ease of migration, form the main driving force for the initial wide acceptance of P2P computing. To deploy P2P in a business organization, the accesses and dynamism can be constrained as data and resource sharing may be compartmentalized and restricted based on the roles users play. While most researchers and businessmen realize the potential of P2P, it remains a difficult task to find a business model for exploiting P2P fully. Consequently, various forms of P2P architectures have emerged and will evolve and adapt over time to find a natural fit for different application domains. One such success story is the deployment of the paradigm of edge-services in content search, where it has been exploited in pushing data closer to the users for faster delivery and solving the network and server bottle-neck problems.

From database perspective, most of these P2P systems are limited in several ways. First, they provide only file level sharing (i.e., sharing of the entirety of a file) and lack object/data management capabilities and support for content-based search. Second, they are limited in extensibility and flexibility. As such, there is no easy and rapid ways to extend their applications quickly to fulfill new users needs. Third, a node's peers are typically statically defined, without taking system and network traffic into consideration. The limitations are due to the lack of database's richness in semantics and relationships, and adaptability based on data size and access patterns.

Fueled by the need for sharing remote resources and datasets, and pooling computing resources for large-scale resource intensive data analysis, Grid Computing [3, 4] has concurrently emerged as an alternative approach to distributed and parallel computing. Like P2P computing, Grid technology exploits shared resources and rides on existing underlying organizational structure. While they are similar in philosophy, they differ in several ways. In the next section, we provide some comparison between P2P and Grid computing, and argue that while Grid computing will continue to play an important role in specialized applications, the P2P technology has its own merits and offers more research challenges in view of the scale and unstability of the network. In the Section 3, we examine the issues of P2P distributed data sharing systems, and how database applications can ride on P2P technology. In Section 4, we present our BestPeer [16] project and its features, which is an on-going peer-based data management system, as an illustration of what P2P computing can do for database management. We conclude in Section 5.

2 P2P vs Grid Computing

An alternative approach known as Grid Computing to distributed and parallel computing has also emerged, with a similar intent of scaling the system

performance and availability by sharing resources. Like P2P computing, Grid Computing has been popularized by the need for resource sharing and consequently, it rides on existing underlying organizational structure. However, there are differences that distinguish the two as of today.

First, the grid network involves higher-end resources as compared to edge level devices in the P2P network. While the former requires large amount of money to be pumped in, the latter can tap into existing resources that are idling and hence require less upfront cost commitment.

Second, the participants in the Grid network are organizations which agree in good faith to share resources with a good degree of trust, accountability and common understanding; membership can be rather exclusive and hence the number of participants is usually not large. The common platform for sharing is usually clusters that have been demonstrated to be cost effective to super-computing, and together they provide an enormous amount of aggregated computing resources. In contrast, the participants of the P2P network are mainly end-users and the platform of sharing is mainly individual Personal Computer (PC). However, due to the mass appeal, the network grows in a much faster rate and may scale up to thousands of nodes. Because of the loose integration, it is more difficult and critical to manage trust, accountability and security.

Third, the Grid network is very much well structured and stable. As a result, resource discovery is less of an issue. On the contrary, P2P network is very unstable - nodes can join and leave the network anytime. This complicates the design of resource discovery mechanisms. Nodes that leave the network may mean some directories may be temporarily "unavailable".

Fourth, Grid computing can exploit traditional distributed query processing techniques and ensure that answers are complete. In contrast, nodes in the P2P network containing data may not be connected at the time of query, answers are likely to be incomplete.

Finally, computational grids are largely set up in anticipation of resource intensive applications, e.g. BioGrid for bioinformatics. However, to date, there has been no reported successful story. On the other hand, "killer" applications have surfaced in P2P naturally. For example, the Napster [15] MP3 music file sharing applications served over 20 million users by mid-2000 before it was finally shut down. As another example, the SETIhome [20] program has accumulated over 500,000 years of CPU time through more than 2 million users!

In summary, we believe Grid computing will continue to play an important role in specialized applications, although architecturely, Grid computing could be considered a special case of P2P computing, where each participating node has a larger capacity and collaboration is more constrained and organized. Notwithstanding, we believe P2P technology is more "user friendly" in the sense that it allows users (particularly those at the edges) to share their resources and information easily and freely. P2P also offers more research challenges in view of the scale and unstability of the network.

3 How Has P2P Been Used For Databases?

While database technologies are mature, P2P promises the power of autonomous and distributed resource sharing. Though these two have their different research focus, researchers are now trying to seek out possible ways to integrate the two to leverage on their respective strengths. We shall review some of these efforts here.

[7] is the first paper that discusses data management issues in P2P environment from a database research perspective. Its focus, however, is largely on *what database technologies can do for P2P applications.* Though a preliminary architecture for peer data management (Piazza) is described in [7], little is discussed about how peers cooperate. Different from [7], PeerOLAP [11] sought to address the problem in a different way - it looks at *what P2P technologies can do for database applications.* Essentially, PeerOLAP is still a client/server system, however, the cooperation among clients (peers) is explored: all data within clients is shared together.

In a P2P system, each peer maintains its data independently. In order to support semantically meaningful search, some kind of understanding among peers is required. In PeerDB[18, 17], we adopted IR-based approach to mediate peer schemas with the help of a global thesaurus. In [19], a data model(LRM) is designed for P2P systems with domain relations and coordination formulas to describe relationships between two peer databases. Data Mapping [13] is a simplified implementation of this model.

Due to the autonomy of the peer nodes and the databases each maintains, data integration naturally becomes an important research area. However, unlike traditional data integration systems, where a global schema is assumed with a few data sources, a P2P system cannot simply assume a global schema, due to its high dynamicity and large number of data sources. Nevertheless, it may be possible to compose mediators, having some mediators defined in terms of other mediators and data sources, thus achieving system extensibility. This is the main thrust of [8] and [12]. While [8] focuses on how queries are reformulated with views, [12] focuses on selective view expansion as opposed to full view expansion which may be prohibitively expensive.

4 Peer-Based Data Management in The BestPeer Project

The BestPeer project started in 1999, and an extensible P2P platform, the BestPeer platform, was developed fully in Java [16]. We have built several applications on top of the BestPeer platform, including BuddyWeb, PeerOLAP, PeerIS, PeerCQ and PeerDB. We shall focus here on the PeerDB system that is a prototype P2P application that provides database capabilities. This system has been developed at the National University of Singapore in collaboration with Fudan University, and is being enhanced with more features and applications.

PeerDB [17] is an instance of DBMS application over the BestPeer platform. The concept behind PeerDB is similar to the analogy of publishing personal web

sites, except that it is now applied to personal databases. Unlike personal web sites which are usually hosted together in a central web server, personal databases are stored in the person's own PC. In addition, it is increasingly common for people to keep their data in common personal DBMS like MySql, and MSAccess. Therefore, a PeerDB node allows an user to index and publish his/her personal database for other peers to query.

4.1 The PeerDB Network

In the PeerDB network, there are two types of entities: a large number of nodes (PeerDB nodes), and a relatively fewer number of location independent global names lookup (LIGLO) servers. Each participating node runs the PeerDB (Java-based) software that enables it to communicate or share resources with any other nodes in the PeerDB network, thus realizing a P2P distributed data management and sharing environment. Each node is essentially a data management system and retains its autonomy: it determines its degree of participation - which relations to share with other nodes, amount of resources to share, and access control.

A LIGLO server is a "super" peer with fixed IP address, and is used to uniquely recognize nodes whose IP addresses may change frequently. Thus, a node's peer whose IP address may be different at different time remain uniquely recognizable. To avoid the server from being a bottleneck, we adopted a distributed approach where several LIGLO servers exist in the PeerDB network to cater to the nodes.

Like existing P2P systems, each PeerDB node maintains addresses of a set of nodes that it can directly reached. We shall refer to these nodes as *neighbors* or *directly connected peers* of the node. Each PeerDB node also maintains metadata of objects/services provided by its neighbors. If a request can be satisfied locally at a node, it is done; if it can be satisfied by some of its neighbors, it is routed to them; otherwise, the request is routed to all neighbors, which in turn may route it to their neighbors, and so on. Answers, however, are returned directly to the node that initiates the query without passing through the search path.

PeerDB exploits BestPeer's ability to reconfigure the network to keep promising peers in close proximity based on some criterion. Currently, PeerDB supports the following reconfiguration policies, the last strategy is newly designed:

1. **MaxCount**. MaxCount maximizes the number of objects a node can obtain from its directly connected peers. It works in two steps. First, the node sorts the peers based on the number of answers (or bytes) they returned. Those that return more answers are ranked higher, and ties are arbitrarily broken. The assumption here is that a peer that returns more answer is likely to be useful for future queries. Second, the k peers with the highest values are retained as the k directly connected peers, where k is a system parameter that can be set by a participating node.
2. **MinHops**. MinHops, implicitly exploits collaboration with peers by minimizing the number of Hops. It requires that peers piggyback with their

answers the value of Hops. This will indicate how far the peers are from the initiator of the request. More importantly, this information provides an indication on what one can access from one's indirect peers. The rationale is as follows: If one can get the answers through one's not-too-distant peers (with small Hops value), then it may not be necessary to keep those nodes (that provide the answer) as one's immediate peers; it is better to keep nodes that are further away so that all answer can be obtained with the minimal number of Hops. Thus, this policy simply orders peers based on the number of Hops, and picks those with the larger Hops values as the immediate peers. In the event of ties, the one with the larger number of answer is preferred.

3. **TempLoc.** TempLoc is a temporal locality based strategy that favors nodes that have most recently provided answers. It uses the notion of stack distance to measure the temporal locality. The idea works as follow. Consider a stack that stores all the peers that return results. For each peer that returns answers, move the peer to the top of the stack, and push the existing peers down. The temporal locality of a peer is thus determined by its depth in the stack. The top k peers in the stack are retained as the k directly connected peers, where k is a system parameter that can be set by the node.

4.2 Architecture of PeerDB

Figure 1 illustrates the internals of a PeerDB node. There are essentially four components that are loosely integrated. The first component is a data management system that facilitates storage, manipulation and retrieval of the data at the node. We note that the interface of the data management system is essentially an SQL query facility. Thus, the system can be used on its own as a stand alone DBMS outside of PeerDB.

For each relation that is created (through the PeerDB interface), the associated meta-data (schema, keywords, etc) are stored in a **Local Dictionary**. There is also an **Export Dictionary** that reflects the meta-data of objects that are sharable to other nodes. Thus, only objects that are exported can be accessed by other nodes in the network. We note that the meta-data associated with the Export Dictionary is a subset of those found in the Local Dictionary, and the distinction here is a logical one.

The second component is a database agent system called DBAgent. DBAgent provides the environment for mobile agents to operate on. Each PeerDB node has a *master agent* that manages the query of the user. In particular, it will clone and dispatch *worker agents* to neighboring nodes, receive answers and present them to the user. It also monitors the statistics and manages the network reconfiguration policies.

The third component is a cache manager. We are dealing with caching remote meta-data and data in secondary storage, and the cache manager determines the caching and replacement policies.

The last component is the user interface. This provides a user-friendly environment for user to submit their queries, to maintain their sharable objects,

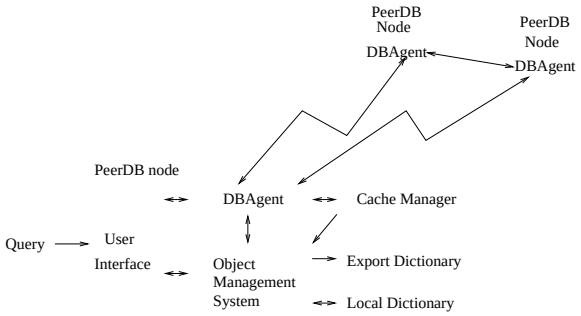


Fig. 1. PeerDB node architecture

and to insert/delete objects. In particular, users search for data using SQL-like queries.

4.3 Key Features of PeerDB

PeerDB has two main distinguishing features. First, it allows users to query data in a SQL-like interface without knowing the schema of data in other nodes. To address this issue, we adopt an Information Retrieval (IR) based approach. For each relation that is created by the user, meta-data are maintained for each relation name and attributes. These are essentially keywords/descriptions provided by the users upon creation of the table, and serve as a kind of synonymous names. DBAgents are sent out to the peers to find out potential matches and bring the corresponding meta-data back. By matching keywords from the meta-data of the relations, PeerDB is able to locate relations that are potentially similar to the query relations.

Second, in PeerDB, we adopt a two-phase agent-assisted query processing strategy. In the first phase, the relation matching strategy (as described in the first feature) is applied to locate potential relations. These relations (meta-data, database name, and location) are then returned to the query node for two purposes. One, it allows user to select the more relevant relations. This is to minimize information overload when data may be syntactically the same (having the same keywords) but semantically different. That is, different schemas are mediated. Moreover, this can minimize transmitting data that are not useful to the user, and hence better utilizes the network bandwidth. Two, it allows the node to update its statistics to facilitate future search process. Phase two begins after the user has selected the desired relations. In phase two, the queries will be directed to the nodes containing the selected relations, and the answers are finally returned (and cached). The two phases are completely assisted by agents. In fact, it is the agents that are sent out to the peers, and it is the agent that interacts with the DBMS. Moreover, a query may be rewritten into another form by the DBAgent (e.g., a query on a single relation may be rewritten into a join query involving multiple relations).

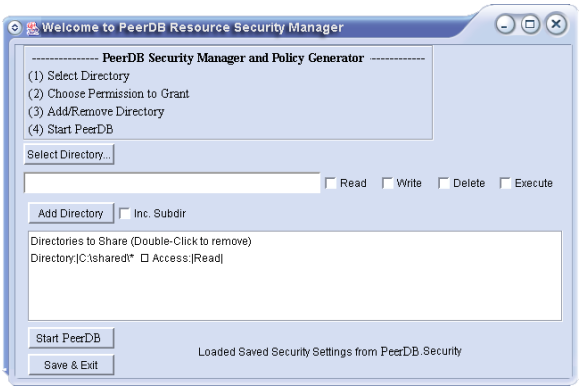


Fig. 2. Interface for specifying security policy

4.4 Security in PeerDB

Due to the security risks posed by such a potentially powerful platform, PeerDB has also been enhanced to provide a secure access to a node's computing resources and databases. Recall that each node comprises two types of data, private data and sharable data. Nodes can only access data that are sharable. This is enforced by a security policy that restricts applications to user-specified locations established during platform initialization. Figure 2 shows the interface to realize this. Communications between nodes have also been provided with 128 bit encryption to protect the sensitive data from being eavesdropped and viewed as they travel through the BestPeer network.

4.5 Implementation of PeerDB

PeerDB is fully implemented in Java. In our current implementation, we use MySQL as the underlying database management systems. Users need to enter keywords/descriptions and the columns of the table when they want to share the table to others. An example is shown in in Figure 3. In the figure, the table `studentcourses` has keywords `student` and `course`. Similarly, the attribute `CourseID` has keywords `course code` and `course number`.

The metadata is very useful as the incoming agent would determine whether the table is relevant to the query according to it. Our agent could also carry the relevant metadata from the query originating host to destination host to compare the similarity of the two metadata, and bring the set of possible relevant results back. Once the results arrive at the query originating host, the user would need to select the one he/she believe is relevant to his query, and perform the query.

Users will also need to maintain the relationship among those shared tables. i.e, they need to specify which tables can be joined and on what attributes. This is to facilitate better accuracy and more results. When an agent reaches the host, it not only searches for single tables, but also checks whether those "joinable" tables could be relevant to the query.

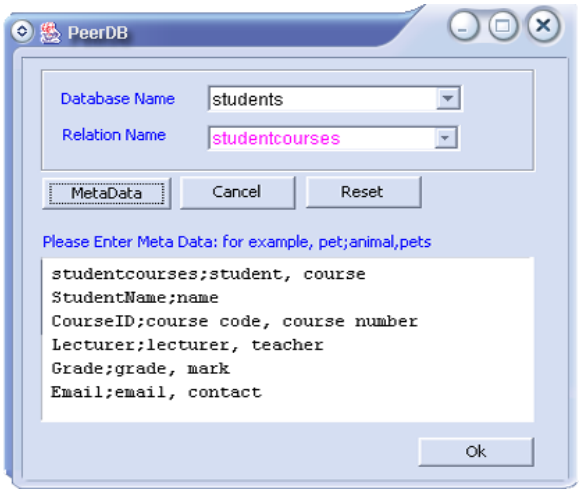


Fig. 3. A sample metadata for a PeerDB relation

When a user submits a query to our system, we will first check whether there is any cached information about the relevant tables. If yes, we will query those tables first. Our system will dispatch the agent to find more tables if the user is not satisfied with the number of results.

PeerDB maintains two thresholds input by the users, threshold1 and threshold2, to help find relevant tables. Our agent will carry the two thresholds with it; and when it arrives at a particular node, it will ask for the metadata of all the shared tables. For each shared table, it will then compute the similarity, sim1, between the two tables using the two set of metadata. If the sim1 is larger than threshold1, it will indicate the table as potentially relevant. It also computes the similarity, sim2, between the original metadata and the metadata of the "joinable table". If sim2 is larger than threshold2 and the sum of sim1 and sim2 is larger than threshold1, it will also indicate the joined table as potentially relevant. Note that in searching for relevant "joinable tables", threshold1 is for improving accuracy and threshold2 is for improving efficiency. The keywords for table name is assigned a higher weight.

After the computation, it will bring the metadata of those potential relevant tables to the query originating host. When the result is returned back, the user would need to indicate which table is relevant to his query and perform the query. The system automatically constructs a new query and directly sends the query to the destination node without dispatching the agent. We note that the reconstructed query may replace the names of the attributes and/or drop some attributes that are not relevant to the original query. PeerDB currently supports queries that join relations from multiple nodes. The user could also choose to save the metadata of the relevant tables (in the form of view), so that a subsequent search on the same tables do not require any agents to be dispatched. The view

can then be propagated to other peers and maintained based on the statistics on how often it is being used.

Figure 4 shows the PeerDB query interfaces - the first figure shows the query interface; the second figure shows a sample where the user specifies that Table Students and Table Courses (both tables are results from the relation matching strategies) should be joined on the field Students.courseID and Courses.courseID; and the last figure displays the answer tuples from a selected relation.

4.6 Joining Tuples based on 'Contents'

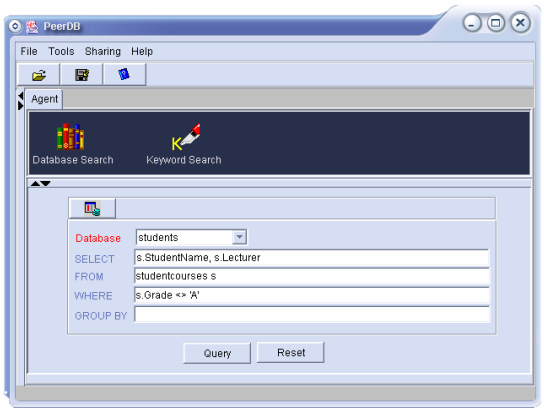
Query-by-keywords is the standard information retrieval mechanism to search for text documents. Recently, keyword search has also been proposed for searching centralized relational databases(eg. DBXplore [1], Discover[9] and BANKS[2]). In general, it is technically very challenging to query a database using keywords due to the semantics of keywords. In PeerDB, an on-going work is to provide keyword search in a peer-based data management system setting. Unlike centralized systems, the key challenges are the autonomy of peers, the lack of a global schema and the dynamics of the peer connectivity. To this end, we introduce the notion of peer-to-peer join that combines tuples from relations that contain certain keywords in the query. Compared to SQL queries, which can be only posed according to peers' local schema, keyword-based queries efface differences among peers to some extent, providing an integrated interface for the users. In addition, it is worthwhile to note that keyword-based queries in our system are answered in a semantically meaningful way, which is quite different from IR experiences.

Each peer maintains its data in a relational database. When a query is processed, the peer will search its database and return tuples that may include all or subsets of keywords in the query. Our focus is not on how keywords are searched in local relational databases, but on how partial and incomplete information from different peers can be combined and integrated. Within each peer, local keyword searching is an indispensable and important component.

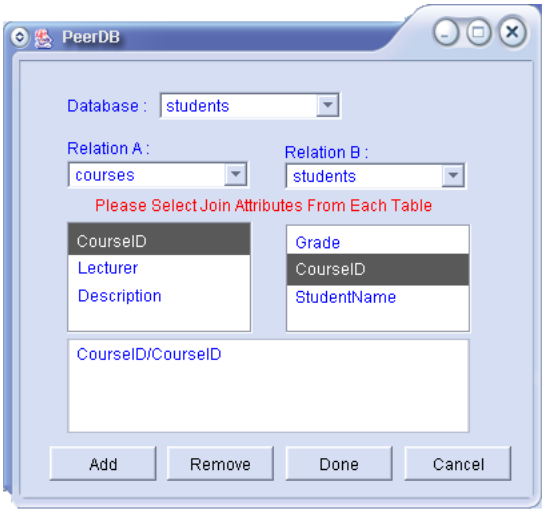
Peer-to-Peer Join is a join operation that combines two (or more) relations from two (or more) peers based on the semantics of keywords and syntax of join operation of relational database systems. Each relation either contains at least one unique keyword in the query or acts as a connector that is necessary for joining other relations, but includes no query keywords by itself. To facilitate efficient peer-to-peer join processing, various network reconfiguration and query processing strategies are designed. Hueristics are formulated to select peers to improve the 'completeness' of the answer while returning the answer as quickly as possible to the user.

5 Conclusion

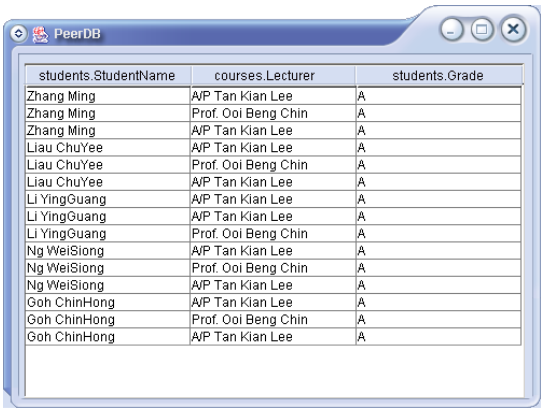
In this paper, we have relooked at P2P technology, and compare it against grid computing. We believe that P2P technology offers interesting research problems.



(a) Query submission.



(b) Specifying joins.



students.StudentName	courses.Lecturer	students.Grade
Zhang Ming	A/P Tan Kian Lee	A
Zhang Ming	Prof. Ooi Beng Chin	A
Zhang Ming	A/P Tan Kian Lee	A
Liau ChuYee	A/P Tan Kian Lee	A
Liau ChuYee	Prof. Ooi Beng Chin	A
Liau ChuYee	A/P Tan Kian Lee	A
Li YingGuang	A/P Tan Kian Lee	A
Li YingGuang	A/P Tan Kian Lee	A
Li YingGuang	Prof. Ooi Beng Chin	A
Ng WeiSiong	A/P Tan Kian Lee	A
Ng WeiSiong	Prof. Ooi Beng Chin	A
Ng WeiSiong	A/P Tan Kian Lee	A
Goh ChinHong	A/P Tan Kian Lee	A
Goh ChinHong	Prof. Ooi Beng Chin	A
Goh ChinHong	A/P Tan Kian Lee	A

(c) Answer tuples.

Fig. 4. Query interface.

We also presented how P2P technology has been applied to support data management, and described our current on-going work on building a peer-based data management system.

References

1. S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA, April 2002.
2. G. Bhalotia, C. Nakhe, A. Hulgeri, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA, April 2002.
3. I. Foster. The grid: A new infrastructure for 21st century science. In *Physics Today*, 2002.
4. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
5. Freenet Home Page. *http://freenet.sourceforge.com/*.
6. Gnutella Development Home Page. *http://gnutella.wego.com/*.
7. S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer. In *WebDB*, 2001.
8. A. Y. Halevy, Z. G. Ives, and D. Suciu. Schema mediation in peer data management systems. In *Proceedings of the 19th International Conference on Data Engineering*, 2003.
9. V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB'2002*, 2002.
10. ICQ Home Page. *http://www.icq.com/*.
11. P. Kalnis, W.S. Ng, B.C. Ooi, D. Papadias, and K.L. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *ACM SIGMOD 2002*, 2002.
12. T. Katchaounov. Query processing in self-profiling composable peer-to-peer mediator databases. In *Proc. EDBT Ph.D. Workshop 2002*, 2002.
13. A. Kementsietsidis, M. Arenas, and R. Miller. Data mapping in peer-to-peer systems. In *Proceedings of the 19th International Conference on Data Engineering*, 2003 (Poster Paper).
14. LOCKSS Home Page. *http://lockss.stanford.edu/*.
15. Napster Home Page. *http://www.napster.com/*.
16. W. S. Ng, B. C. Ooi, and K. L. Tan. BestPeer: A self-configurable peer-to-peer system. In *Proceedings of the 18th International Conference on Data Engineering*, San Jose, CA, April 2002 (Poster Paper).
17. W. S. Ng, B. C. Ooi, K. L. Tan, and A. Zhou. PeerDB: A p2p-based system for distributed data sharing. In *Proceedings of the 19th International Conference on Data Engineering*, Bangalore, India, March 2003.
18. B.C. Ooi, K.L. Tan, A.Y. Zhou, C.H. Goh, Y.G. Li, C.Y. Liau, B. Ling, W.S. Ng, Y. Shu, X.Y. Wang, and M. Zhang. PeerDB: Peering into personal databases. In *Proceedings of ACM SIGMOD Intl. Conf. on Management of Data*, San Diego, June 2003.
19. A. B. Philip, G. Fausto, K. Anastasios, M. John, S. Luciano, and Z. Ilya. Data management for peer-to-peer computing: A vision. In *WebDB Workshop*, 2002.
20. SETI@home Home Page. *http://setiathome.ssl.berkely.edu/*.