Relational Data Sharing in Peer-based Data Management Systems

Beng Chin Ooi Yanfeng Shu 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

Data sharing in current P2P systems is very much restricted to file-system-like capabilities. In this paper, we present the strategies that we have adopted in our BestPeer project to support more fine-grained data sharing, especially, relational data sharing, in a P2P context. First, we look at some of the issues in designing a peer-based data management system, and discuss some possible solutions to address these issues. Second, we present the design of our first prototype system, PeerDB, and report our experience with it. Finally, we discuss our current extensions to PeerDB to support keyword-based queries.

1 Introduction

Peer-to-Peer (P2P) technologies have recently received much attention from academia and industries due to many benefits they offer. In a typical P2P system, a large number of nodes can potentially be pooled together to share their resources, information and services, while keeping themselves fully autonomous. Though many different uses for P2P have been identified, such as distributed computation (SETI@Home[16]), Instant messaging (ICQ [8]), data sharing remains the most dominant P2P application on the internet.

Nevertheless, in current data sharing P2P systems, only file-system-like capabilities are provided while the semantics of data is largely ignored. For example, in Gnutella, queries are restricted to strings that can be contained in a filename and directory path, that is, only simple value searches on file names are supported. DHT-based systems ([17], [14], [15], [18]) do no better: only *exact* key match is supported. From data management perspective, such kind of data sharing is very limited.

More importantly, we believe that most PC users store their data in common personal DBMS such as MySQL and MSAccess. For example, our experience with genomic scientists find that they store large amount of data (e.g., gene expression) in their PCs. As another example, many doctors and specialists also store their patient information in PC databases. Peer-based data management provides a solution for these users to share their data to their colleagues. Moreover, relational data has rich structure and semantics, and queries are at a finer granularity. To share these data, current P2P systems must be database-enabled.

In this paper, we present our on-going work in our BestPeer project [3] to support relational data sharing. The BestPeer project was initiated in 2000 to study how P2P technologies can be employed for distributed data management. Peer-based data management offers challenges that go beyond that of traditional heterogenous databases. We shall first present some design issues, and possible solutions. We will then present PeerDB, a prototype peer-based data management that we have developed. PeerDB employs an IR approach to discover matching relations, and agents to assist in query processing. Finally, we introduce our current work on *Keyword Join* to facilitate user-friendly keyword queries in PeerDB.

The rest of this paper is organized as follows. In the next section, we discuss the design issues for peer-based data management. In Section 3, we review the BestPeer platform on which PeerDB is based. Sections 4 presents PeerDB. In Section 5, we discuss our approach to support keyword search. And finally, we conclude in Section 6.

2 Design Issues for Peer-Based Data Management

While a peer-based data management system can be seen as a distributed and heterogeneous database system, the scale of the system and its dynamism as nodes join and leave the network offer several major challenges.

- First, there is no predefined global schema. With each node joining and leaving the network at anytime, assuming a global schema in such a dynamic environment is apparently not practical, scalable and extensible. One possible approach is to perform "mapping" on-the-fly during querying. In other words, when a query is issued, a schema-matching routing is performed to search for nodes with matching schema [11]. An alternative solution is to allow each node to predetermine its mappings between its schema and its neighbors' before querying(in this case, neighbors share common semantic content) [13, 1, 5, 9]. By transitive relationship, new mappings can be learned and thus nodes containing relevant data can potentially be reached. However, these solutions are far from satisfaction, difficulties lie in how the mappings can be effectively described, built and maintained.

- Second, realizing efficient query processing becomes more difficult. Initial response time is expected to be high as relevant data have to be identified before any optimization and query processing can be performed. To address this, adaptive query processing techniques [6] must be employed. Optimization may be centralized at the query node. In this case, partial optimization on a subgraph of a query graph may have to be performed as soon as some relevant data is identified. Alternatively, distributed optimization can also be deployed to allow nodes with relevant data to perform optimization based on their available knowledge. Ideas such as mutant query plans may be valuable here also [12].

- Third, much information redundancy exists in the network, which inevitably brings about data and computation redundancy. Unfortunately, information redundancy cannot be avoided unless some control over data placement is taken (e.g. DHT systems). Computational redundancy arises when distributed query processing and/or optimization are employed, since each node may only have partial knowledge, and it is possible that multiple nodes compute different subgraphs of a query graph with significant overlap. Such redundancy needs to be reduced as much as possible. This calls for query processing nodes to interact and exchange information.

- Finally, the notions of correctness and completeness of query results cannot be used in their pure meaning as in traditional database systems: query results now largely depends on transient network organization and semantic mappings already established. More research need to be done in this area.

3 BestPeer - Underlying P2P Architecture

PeerDB is built on top of BestPeer [10], a generic P2P platform that can be used to develop P2P applications easily and efficiently. The network consists of two types of entities: a large number of computers (nodes), and a relatively fewer number of *location independent global names lookup* (LIGLO) servers, which are mainly used to help peers to recognize each other regardless of the change of IP addresses. Each peer has a globally unique identifier.

BestPeer has several features that distinguish itself from existing P2P systems. Here, we only list two of them that affect upper-level query processing. First, mobile agents technology is combined into the system. Since agents can carry both code and data, they can effectively perform any kind of functions. With agents performing operations at the peers' sites, the network bandwidth can be better utilized. Second, a peer in the network can dynamically reconfigure itself by keeping "best" peers that benefit it most as its neighbors, based on a simple assumption: peers that benefit it most for a query are likely to continue so for subsequent queries. Over time, peers that benefit each other will cluster together, and thus queries can always be answered by nearby neighbors with high probability.

4 PeerDB

In this section, we present PeerDB [11], a prototype peer-based data management system that we have developed. We shall present our approaches to address the various issues and discuss some preliminary results.

4.1 Architecture of a PeerDB Node

Figure 1 illustrates the architecture of a PeerDB node. The system consists of three layers, namely the P2P layer that provides P2P capabilities (e.g., facilitates exchange of data and resource discovery), the agent layer that exploits agents as the workhorse, and the object management layer (which is also the application layer) that provides the data storage and processing capabilities.

At the data management layer, there are essentially four components that are loosely integrated. The first component is an object management system that facilitates storage, manipulation and retrieval of the data at the node. We note that the interface of the object 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 relations that are

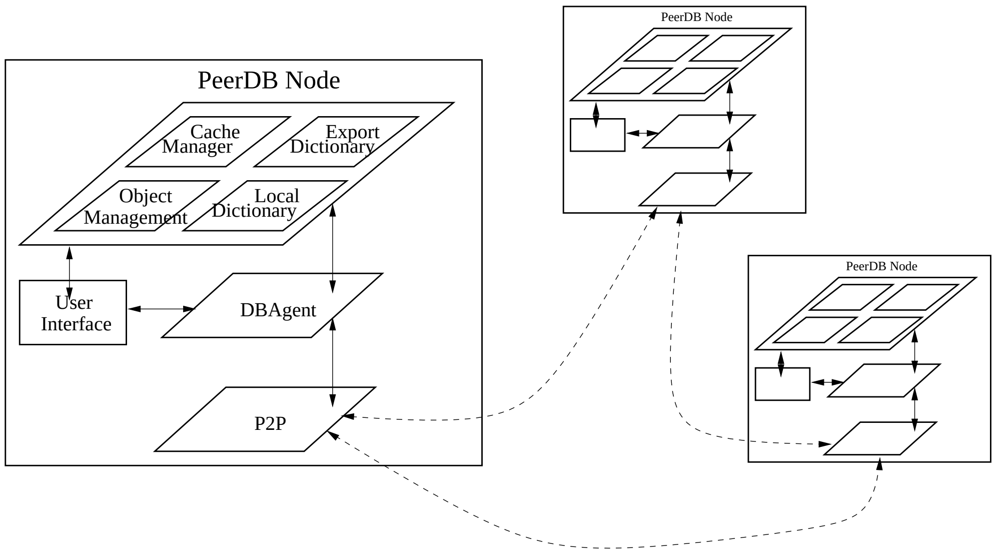

Figure 1. PeerDB architecture

sharable to other nodes. Thus, only relations 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 last 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.

At the agent layer, we have a database agent system called DBAgent that 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.

There is also an user interface subsystem. This provides a user-friendly environment for user to submit their queries, to maintain their sharable relations, and to insert/delete relations/tuples. In particular, users search for data using SQL-like queries.

4.2 Schema Mapping

In PeerDB, an Information Retrieval (IR) based approach is employed for schema mapping. 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 of relation names and attributes. 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.

Though this approach has little originality in itself, we use it first in P2P context, with the consideration of its simplicity and that in most scientific data sharing communities, there always exist some generally acknowledged terms or names among peers.

We illustrate the strategy with an example. Suppose we have four peers that share genomic data. Peer P1 defines a relation Kinases(SeqID, length, proteinSeq). Peer P2 defines a relation Protein(SeqNo, len, sequence). Peer P3 defines two relations ProteinKLen(ID, seqLength) and ProteinKSeq(ID, sequence). Peer P4 defines a relation Protein(name, char). Figure 2 shows the keywords defined for these relations by the various peers. Suppose the user at peer P1 (he knows his own schema but not the schema of other peers) issues the following query to look for kinases sequences that are longer than 30 base pairs:

```
SELECT SeqId, proteinSeq
FROM Kinases
WHERE length > 30;
```

Now, since one of the keywords for Kinases (relation name) is protein, and protein is also a keyword for P2's relation `Protein` and P3's relations `ProteinKLen` and `ProteinKSeq`, these relations match the query relation. Similarly, we find that the attributes `SeqID`, `proteinSeq` and `length` all have matching keywords in P2 and P3. For P3, we note that the query may have to be turned into a join query when evaluated there. For P4, we only have a match in relation name but not in the attributes. Thus, P4 will be ranked lower than P2 and P3. Semantically, we note that P2's data are not actually those that P1 is interested in (since they are not Kinases data). As such, it is important to have the meta-data and additional information returned to the users before fetching the data.

Peer	Names	Keywords
P1	Kinases	protein, human
	SeqID	key, identifier, ID
	length	length
	proteinSeq	sequence, protein sequence
	Protein	protein, annexin, zebrafish
P2	SeqNo	number, identifier
	len	length
	sequence	sequence
	ProteinKLen	protein, kinases, length
	ID	number, identifier
P3	seqLength	length
	ProteinKSeq	protein, sequence
	ID	number, identifier
	sequence	sequence
	Protein	protein, kinases, annexin, . . .
P4	name	name
	char	characteristics, features, functions

Figure 2. Keywords for the relations/attributes names.

4.3 Agent Assisted Query Processing

In PeerDB, we adopt a two-phase agent-assisted query processing strategy. In the first phase, the relation matching strategy (as described above) 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).

4.4 Preliminary Results

To evaluate PeerDB's performance, we conducted several sets of experiments (see [11] for the details). We summarize our findings here.

- IR based approach is effective. We used the standard IR measures, *Precision* and *Recall*, as performance metrics to measure the approach's effectiveness on relation matching. Our results show that when the threshold value is large (resulting in a large number of relations accepted as matching), *Recall* is low because of the large number of irrelevant relations that share common keywords. However, *Precision* is high showing that most of the retrieved relations are indeed relevant to the query. This result is consistent with typical IR applications, and demonstrated the effectiveness of our approach.

- Self-configuration is important. With the ability to reconfigure the network, relevant nodes will move "closer", and thus queries will always be directed to the more promising node first. One reconfiguration policy employed in PeerDB is to favor peers that have most provided answers recently. In our experiments, we study two versions of PeerDB: one with the reconfiguration feature turned on, the other with the feature turned off. Our results show that the ability to reconfigure is important for performance improvement and can lead to shorter initial response time and larger number of answers being returned within a given time.

- Caching is helpful for reducing response time. PeerDB supports caching of answers returned from remote peers in order to reduce the response time for subsequent answers. For every relation that the user retrieved, we cache the answers. Though Caching raises many complicated issues, we did some controlled experiments to evaluate the effect of caching on the performance by varying the storage capacity of each peer. We observe that as the storage capacity of each node increases, the response time decreases. This is expected as more tuples can be found in local and neighboring peers. Meanwhile, by caching previous query results, duplicate work and data movement can be avoided.

5 Work-In-Progress: Keyword Search Through Keyword Join

Our on-going work is to further extend PeerDB's capabilities. In particular, we would like to support keyword search (as most users would not be familiar with SQL) in peer-based data management systems (This is different from previous works ([2], [4], [7]) where keywords are searched over centralized databases). We need to address the search space issue: the search space is very large - every single relation in the P2P network is a possible candidate, and these relations can potentially be combined in different ways to produce answer tuples.

5.1 Keyword Join: The Issues

We introduce the concept of a keyword join to allow us to combine relations based on the common keywords across relations. For example, consider a relation on publication, PUB(title, author), and another on affiliations, AFF(name, affiliation, url). Given a query with keywords `peer-to-peer` and `beng chin` and `url`, the results return not only Beng Chin's P2P publications, but also his url. These two tables can be combined to return the resultant answers through the author's name (which is `beng chin`). However, the problem is complicated because the relationship between PUB.author and AFF.name may not have been defined (especially if the tables are hosted by different peers), and hence the two tables cannot be joined (in the traditional sense). By introducing keyword join, two tables become *dynamically joinable* on the attributes that share the common keyword.

However, for keyword joins to be practical and useful, there are several challenges to be overcome:

- First, it must be semantically meaningful to combine on the attributes containing the keyword. It is clearly possible that some unrelated attributes can be combined (for example, selling price and cost price of items). To determine automatically (without human intervention) whether two attributes can be combined in a meaningful way is a very difficult problem.

- Second, it is computationally expensive to consider all combinations. For example, given n common keywords, we may end up with n^2 combinations to examine. This calls for mechanisms to prune away combinations that are not meaningful quickly.

- Third, even for attributes that can be combined meaningfully, the values may be different. Using the same example, the name Beng Chin Ooi in PUB may be listed as Ooi Beng Chin in AFF. As another example, the term protein may be used in one attribute, but kinases may be used in another. This implies that keyword join is a form of similarity join, and needs a similarity metric to determine the degree of similarity between two values.

To address the first two problems, we can exploit two sources of semantics that are available. First, we propose to maintain meta-data on attributes of a relation. Techniques from automatic semantic integration can be applied here to compare attributes based on their meta-data. For example, two attributes with significant overlap in their data values are more likely to be semantically similar. Second, the domain of the attributes can also provide some insights, e.g., attributes cannot be joined if their domains are not compatible. Moreover, the schema matching scheme in PeerDB can also be applied here. The last issue can be addressed by tapping into techniques from similarity matching in text databases, e.g., the use of WordNet and/or thesaurus to consider synonyms and faciliate retrieval based on semantics. We are currently working on these.

5.2 A Brute-Force Query Processing Strategy

A straightforward strategy to evaluate a keyword-based query in a P2P environment is shown in Figure 3. The algorithm comprises several steps. In the first step, the query is broadcast to neighboring nodes. Here, neighbors are nodes within TTL hops for some TTL ≥ 1. A remote node within TTL hops away will perform a local search for matching keywords.For local keyword search, we adopt a DISCOVER[7]-like method. Each peer maintains an inverted index for keyword search. After receiving a query, the peer will first look up the index to see whether some or all of the keywords in the query are contained in its database. It is possible that the keywords may appear in multiple local tables, in which case these tables will have to be joined to produce the answer tuples (according to the local database schema). Unlike DISCOVER or other keyword search schemes, we do not return the tuples. Instead, we return the schema and meta-data of top few ranked relation(s).

The results returned from remote nodes fall into two categories. In the first category, tuples at remote nodes contain all the keywords. In this case, the relations are displayed to the users for selection. Those selected relations will then be retrieved from the remote nodes. In the second category, each remote node can only contribute tuples that contain some, but not all, of the keywords. In this case, the query node will determine how these relations can be combined, and ranked them accordingly. We note that this step is computationally expensive. Moreover, many of the relations returned by neighboring nodes may not be related and relevant. There are two ways in which relations can be combined. If there is a known relationship (e.g., key-foreign-key relationship) on some attributes, then this should be used to combine the relations; otherwise, a keyword join can be used. Note also that relations are only combined if combining them can potentially increase leading to answers with all keywords. The meta-data will be used here to facilitate the semantic integration.

We are currently exploring how to distribute the processing across multiple peers (instead of the query node).

```
Query node

1.  Broadcast query with a specified TTL
2.  Obtain relations and meta-data from remote peers
    // those containing all keywords
3.  for all relations that can produce answer tuples
        containing all keywords,
4.          return to users to select
5.          for those selected relations, the answer tuples are
                retrieved from the corresponding remote peers
    // those missing some keywords
6.  for each relation $R$
7.      for each relation $S$
8.          if joinable(R,S) or keywordJoinable(R,S)
9.              RS = join(R,S)
10.             if RS contains all keywords,
11.                 goto 3
12.             else
13.                 return to pool

Remote node
1.  if keyword query,
2.      perform a local search
3.      Return matching relations and meta-data
4.  else // retrieval query
5.      access the relations
6.      return the answer tuples containing the keywords
```

Figure 3. A brute-force strategy.

6 Conclusion

We have examined the problem of relational data sharing in P2P data management systems, and highlighted some query processing issues. We have also presented our solutions to these problems, and discuss our on-going work to support keyword search. We are currently evaluating the proposed keyword search strategy. We plan to explore caching as well as optimization and query processing issues to minimize redundant information and computation in the next phase of our research.

References

[1] K. Aberer, P. Cudre-Mauroux, and M. Hauswirth. A framework for semantic gossiping. In *SIGMOD RECORD, Special Section on Semantic Web and Data Management, Vol. 31, Nr. 4*, 2002.

[2] 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.

[3] BestPeer. *http://xena1.ddns.nus.edu.sg/p2p/*.

[4] 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.

[5] 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.

[6] J. Hellerstein, M. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Engineering*, 23(2), 2000.

[7] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB'2002*, 2002.

[8] ICQ Home Page. *http://www.icq.com/*.

[9] A. Kementsietsidis, M. Arenas, and R. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *ACM SIGMOD International Conference on Management of Data*, 2003.

[10] 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).

[11] 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*, 2003.

[12] V. Papadimos and D. Maier. Mutated query plans. *Information and Software Technology*, 44(4):197–206, 2002.

[13] 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.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, 2001.

[15] 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.

[16] SETI@home Home Page. *http://setiathome.ssl.berkely.edu/*.

[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 Special Interest Group on Data Communication (SIGCOMM)*, 2001.

[18] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. In *UC Berkeley Technical Report,UCB/CSD-01-1141*, 2001.