PeerDB: A P2P-based System for Distributed Data Sharing

Wee Siong Ng[1] Beng Chin Ooi[1] Kian-Lee Tan[1] Aoying Zhou[2]
[1]Dept Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
[2]Dept Computer Science and Engineering
Fudan University
Shanghai, China, 200433

Abstract

In this paper, we present the design and evaluation of PeerDB, a peer-to-peer (P2P) distributed data sharing system. PeerDB distinguishes itself from existing P2P systems in several ways. First, it is a full-fledge data management system that supports fine-grain content-based searching. Second, it facilitates sharing of data without shared schema. Third, it combines the power of mobile agents into P2P systems to perform operations at peers' sites. Fourth, PeerDB network is self-configurable, i.e., a node can dynamically optimize the set of peers that it can communicate directly with based on some optimization criterion. By keeping peers that provide most information or services in close proximity (i.e, direct communication), the network bandwidth can be better utilized and system performance can be optimized. We implemented and evaluated PeerDB on a cluster of 32 Pentium II PCs each running a Java-based storage manager. Our experimental results show that PeerDB can effectively exploit P2P technologies for distributed data sharing.

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 (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.

Many domain specific P2P systems have already been deployed [7], e.g., Freenet [8], Gnutella [9], Napster [20], ICQ [13], Seti@home [23] and LOCKSS [17]. However, 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. Unlike existing work on distributed data management, data are shared without predefined schema! Second, they are limited in extensibility and flexibility.

As such, there is no easy and rapid ways to extend their applications quickly to fulfil new users needs. Third, a node's peers are typically statically defined.

In this paper, we present PeerDB, a P2P-based system for distributed data sharing. PeerDB has several distinguishing features. First, each participating node is a full fledge object management system that supports content-based search. Second, in PeerDB, users can share data without a shared global schema. Third, PeerDB adopts mobile agents to assist in query processing. Since agents can perform operations at the peers' sites, the network bandwidth is better utilized. More importantly, agents can be coded to perform a wide variety of tasks, making it easy to extend the capabilities of a PeerDB node. For example, an agent may further manipulate the data retrieved from a node to ship back only summarized data, or filter away uninterested objects. Finally, PeerDB supports mechanisms to dynamically keep promising (or best) peers in close proximity based on some criterion. For example, peers that are most frequently accessed are directly communicable while nodes that are less frequently accessed can be reached through peers. This significantly reduces the response time to queries.

We implemented PeerDB, a prototype P2P distributed object management system that incorporates all the above features. To evaluate PeerDB, we propose a systematic methodology for evaluating P2P systems. Our methodology considers both efficiency and effectiveness (quality of answers) of P2P systems. We conducted our experiments on a cluster of 32 Pentium II PCs each running a Java-based storage manager [10]. Our experimental results show the effectiveness of PeerDB for distributed data sharing.

The rest of this paper is organized as follows. In the next section, we discuss what a P2P distributed data management system is. Section 3 gives a quick glimse of BestPeer, a P2P platform in which PeerDB is based upon. In Section 4, we present the PeerDB design and its features. Section 5 presents an extensive experimental study to evaluate PeerDB. In Section 6, we review some related works, and finally, we conclude in Section 7 with directions for future work.

2 P2P Distributed Data Management: What is it?

As noted in the introduction, practically all existing P2P systems are designed to support sharing of data at a coarse granularity (e.g., files, documents). In this section, we first distinguish between P2P systems and distributed database systems. We then "define" P2P distributed data management by looking at three examples (due to space constraints) of how

P2P technology can be employed for distributed database applications. This will also serve to motivate the need for database technology in P2P systems.

2.1 P2P vs Distributed Database Systems

There are several features that distinguish P2P systems from distributed database systems (DDBS).

1. In P2P systems, nodes can join and leave the network anytime. In DDBS, nodes are added to and removed from the network in a controlled manner, i.e., when there is a need for growth or retirement.

2. In P2P systems, there is no predetermined (global) schema among nodes. Queries are largely based on keywords. There are several reasons for this. First, most of the current applications do not require a fixed schema. Second, as nodes can join and leave the network at anytime, a fixed schema does not reflect the actual information that may be available at a single time. In DDBS, nodes are typically stable and have some knowledge of a shared schema.

3. In P2P systems, users do not expect "complete" answers. By "completeness", we mean all answers that satisfy a query. Since some nodes that contain answers may not be connected, the answers are incomplete. In fact, answers cannot even be ranked. In DDBS, one expects and can actually retrieve the complete set of answers.

4. In P2P systems, content location is typically by "word-of-mouth", i.e., a node routes its query to its neighboring nodes, and so on. In DDBS, the exact location to direct the query is typically known.

Based on the above points, we do not consider data integration systems to be P2P distributed data management systems (even if each node has the capabilities to act as middleware and server).

2.2 P2P Distributed Data Management Systems Applications

Instead of formalizing the concept of P2P distributed data management systems, we show with sample applications on what such systems may be like.

Health Care

In a hospital, each specialist has a group of patients that are solely under his care. While some patient data are stored in a centralized server of the hospital (e.g., name, address, etc), other data (e.g., X-rays, prescription, allergy to drugs, history, reaction to drugs, etc) are typically managed by the specialist on his personal PC. For most of these patients, the specialist is willing to share their data, but there are always some cases that he is unwilling to share for different reasons (e.g., part of his research program on a new drug, etc). By making the sharable patient data available to other specialists, it allows them to look for other patients who may have similar symptoms as their own patients, and hence can help them in making better decisions on the treatment (e.g., drugs to prescribe, reactions to look out for, etc).

Here, we can deploy a P2P distributed management system: (1) any specialist can join/leave the network; (2) the answers need not be complete (i.e., missing data from some specialists is not critical), (3) nodes have to search for content as in P2P systems, (4) the schema defined by each specialist may be different, (5) there is a need for data management, and (6) each specialist has something to share and is also interested in others' data.

Genomic Data

The discovery of new proteins necessitates complex analysis in order to determine their functions and classifications. The main technique that scientists use in determining this information has two phases. The first phase involves searching known protein databases for proteins that "match" the unknown protein. The second phase involves analyzing the functions and classifications of the similar proteins in an attempt to infer commonalities with the new protein. While there are several known servers on genomic data (e.g., GenBank, SWISS-PROT and EMBL), there are many more data that are produced each day in the many laboratories all over the world. These scientists create their own local databases of their newly discovered proteins and results, and are willing to share their findings to the world! Clearly, this is an application for P2P distributed data management systems for the same reasons as the heath care application.

Data Caching

In the above two examples, each participant is *actively* involved in the process of consuming and supplying data. P2P distributed data management can also be deployed in *passive* nodes: nodes that are used to share resources (storage or computational power) on data that they

may or may not be interested. Caching results from earlier queries is one such example - a node may have issued a query to some server (e.g., a data warehouse), the results of the query can be cached on the node (or some other neighboring nodes). In this way, another node that requests for data that overlap the query result can potentially obtain partial answers quickly from this node, and the remainder from the original server. This also lightens the load on the original server. Indeed, Kalnis et. al. have shown how distributed caching can be deployed in P2P environments to speed up OLAP queries [14].

3 BestPeer: An Adaptive Platform for P2P Applications

PeerDB is a database application that is implemented on top of BestPeer [1, 21]. We shall briefly review the features of BestPeer in this section.

BestPeer is a generic P2P system designed to serve as a platform on which P2P applications can be developed 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. Each participating node runs the BestPeer (Java-based) software and will be able to communicate or share resources with any other nodes (i.e., peers) in the BestPeer network.

BestPeer has several features that distinguish itself from existing P2P systems. First, BestPeer, to our knowledge, is the first system to integrate two powerful technologies: mobile agents and P2P technologies. While P2P technology provides resource sharing capabilities amongst nodes, mobile agents technology further extends the functionalities. In particular, since agents can carry both code and data, they can effectively perform any kind of functions. With mobile agents, BestPeer not only provides files and raw data, it also provides processed information (e.g., summaries). More importantly, the use of agents allows BestPeer nodes to collect information (e.g., what files/content are sharable, statistics, etc.) on the entire BestPeer network, and this can be done offline. This allows a node to be better equipped to determine who should be its directly connected peers or who can provide it better service.

Second, BestPeer not only facilitates a finer granularity of data sharing where partial content of a file may be shared, it also shares computational power. The requester sends his/her request for a file together with an algorithm (executable code) that operates on the file. In other words, the requester performs the filtering task at the provider's end! This feature has several advantages: (a) it allows filtering to be performed where the provider's end does not provide the capability (e.g,, the owner does not provide an active object); (b) it

allows individual requester to filter the content according to what (s)he desires (e.g., different requesters may be interested analyze stock data differently); (c) it facilitates extensibility - new algorithm or program can be used without affecting other parts of the system! (d) existing non-distributed objects can be easily extended for use by a P2P application by leveraging on the support provided by BestPeer; (e) it optimizes network bandwidth utilization as only the necessary data is transmitted to the requester.

Third, a node in the BestPeer network can dynamically reconfigure itself by keeping peers that benefit it most (subject to individual node's definition of 'most benefit'). The rationale is based on a simple assumption: peers that benefit a node most for a query are also likely to provide the greatest gain for subsequent queries. Thus, BestPeer will always try to make a direct connection to these nodes that have highest priority. In this way, promising peers are first traversed before the less promising ones. BestPeer currently supports two default reconfiguration strategies. The first strategy, MaxCount, maximizes the number of objects a node can obtain from its directly connected peers. The second strategy, MinHops, implicitly exploits collaboration with peers by minimizing the number of hops.

Finally, BestPeer introduces a Location-Independent Global Names Lookup Server (LIGLO) to provide each node with a unique global identity. In this way, nodes that may have different IP address can be "recognized" as a single unique entity. LIGLO is a node that has a fixed IP and running Location-Independent Global Names Lookup Server software. It provides two main functions: generate a BestPeer Global Identity (BPID) for a peer and maintain peer's current status, such as the current IP address and whether the peer is currently online or offline (if this information is available).

Due to the security risks posed by such a potentially powerful platform, the basic BestPeer platform provides a secure access to a node's computing resources. 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. 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 Peering Up for Distributed Data Sharing

In this section, we will present PeerDB, a prototype P2P-based system for distributed data sharing. PeerDB's P2P-enabling technologies are provided by BestPeer [1, 21]. However, it

extended BestPeer in the following ways. First, data in each node is managed by a database system. In other words, PeerDB is a network of database-enabled nodes. Second, data can be shared without a global schema. Third, query processing is assisted by mobile agents. Fourth, each node can reconfigure itself based on some optimization criterion from the answers returned. We shall discuss these features here.

4.1 Architecture of a PeerDB Node

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 have used the StorM storage manager [10] as our storage server. StorM is an extensible storage server implemented in its entirety in Java [11]. Thus, the system can be used on its own as a stand alone DBMS outside of PeerDB. We note that the interface of the data management system is essentially an SQL query facility. For each relation that is created, 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 (as the actual implementation minimizes redundancy). We shall defer the discussion on how the Export Dictionary will be used, and the details on the meta-data when we addressed the query processing strategy.

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 shall defer the discussion of the cache manager to a later subsection. Here, it suffices for us to know that we are dealing with caching remote data in secondary storage, and the cache manager determines the caching/replacement policy.

The last component is the user interface. This provides a user-friendly environment for user to submit their queries, to maintain their sharable objects, and to insert/delete objects. In particular, users search for data using SQL-like queries.

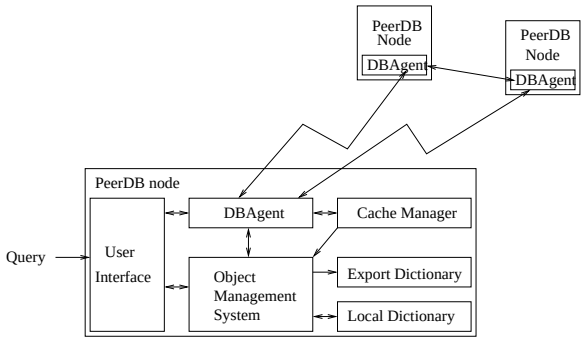


Figure 1: PeerDB node architecture

4.2 Sharing Data Without Shared Schema

One of the main objectives of PeerDB is to allow users to manage their (private and sharable) data using a database management system (DBMS). However, as noted, there is no predetermined and uniform schema that nodes share. Unless users interact with one another somehow, we can expect data to be defined differently by different users even if they may have interests in data from a common domain. For example, in naming a relation, a genome scientist may call his set of protein database by the protein name (e.g, kinases, annexin) while another may name them after the species (e.g, mouse, human, zebrafish). Similarly, at the attribute level, one scientist may call the length of sequences as `length` while another will use the term `len`. A more complicated problem would be for one to create a single "universal" schema, while another may "normalize" his database to multiple tables. Thus, it is difficult to locate data if the traditional method of exact matching of relation names/attributes is used.

To address this issue, we adopt an Information Retrieval (IR) [5] 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 provided by the users upon creation of the table, and serve as a kind of synonymous names[1]. Continuing with our examples, for a table of Kinases proteins, while the relation name may be `Kinases`, the keyword `protein` will be useful during searching. Similarly, two users defining length of a sequence as `len` and `length` respectively will likely to have the common keyword `length`. In this way, potentially relevant data can be determined using the following *relation-matching* strategy:

- Consider a query (R, A, C) where R is the set of relations A is the set of target attributes, and C is the set of conditions.[2] Let V denote all attributes that appear in

[1] One can think of this as a miniature thesaurus.
[2] This corresponds to a simple SPJ query in SQL.

A and C.

- R is searched against keywords for relation names; and V is searched against keywords for attribute names. Note that this search involves finding for matching keywords of R against keywords for other relations; the same holds for attributes. The result of this search process will be a list of relations whose relation name keywords match R (or their keywords) and/or attribute keywords match attribute names in V (or their keywords).

- The set of relations that potentially contain answers for the query are those that are ranked above a certain threshold value. For simplicity, given a database relation D and a query relation R, we use the total number of query terms (i.e., $|R \cup V|$ where $|R|$ denotes the number of terms in R) that are matched by keywords of D as the ranking metric. A relation with a larger number of matching terms is ranked higher. A relation is considered a potential relation if the number of matching terms is greater than the threshold.

With the above strategy to locate matching relations, we note that we can share data without explicit sharing of schema. This flexibility is also an important distinction between PeerDB and existing distributed DBMS. Note that the relations and meta-data will be returned to the user first, who will then decide the data that is of interests (see the next section on query processing strategies).

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 keyword 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. So, if we require the threshold value to be more than 1, then peer P4 will not be accessed. 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
P2	Protein	protein, annexin, zebrafish
	SeqNo	number, identifier
	len	length
	sequence	sequence
P3	ProteinKLen	protein, kinases, length
	ID	number, identifier
	seqLength	length
	ProteinKSeq	protein, sequence
	ID	number, identifier
	sequence	sequence
P4	Protein	protein, kinases, annexin, ...
	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 query processing strategy. In the first phase, the relation matching strategy is applied to locate potential relations. These relations are then returned to the query node for two purposes. First, 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. Moreover, this can minimize transmitting data that are not useful to the user, and hence better utilizes the network bandwidth. Second, 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.

PeerDB's query processing is 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). To elaborate on the query processing strategy, we shall distinguish two types of queries: *local* query and *remote* query. A query is local to a node if it is initiated there, and remote otherwise.

4.3.1 Processing Local Query

When a user issues a query (SQL-like selection query), a master agent will be created to oversee the evaluation of the query. The following operations are performed by the agent:

Phase I

- The agent "parses" the query to extract the list of relations and attributes names.
- The relation matching strategy is applied on the local dictionary. Promising relations can then be returned to the user immediately.
- At the same time, the master agent will clone *relation matching agents* and dispatch them to all neighbors of the node. Besides the query, the agent also carries with it two other information: (a) IP address of the node that initiates the query; (b) TTL (Time-to-live). The former is needed to allow remote nodes to return answers directly to the query node. The latter serves similar purpose as that in networking environment, i.e., to indicate the lifetime of an agent. This allows the process of cloning and forwarding to keep on going until the agent lifetime is expired.
- The master agent will wait for the answers (relations schema) from remote nodes. Upon receiving any answers, they will be returned to the users for selection.
- For peers that return multiple relations, we return the individual relations (if their scores on the number of matching keywords exceed the threshold) as well as combinations of relations that are related (e.g., has a key-foreign key relationship). Referring to our earlier example, P3 will produce three answers: `proteinKLen`, `proteinKSeq`, and `proteinKLen` $\bowtie$ `proteinKSeq` are returned.

Phase II

- For each relation selected by the user, the master agent will clone a *data retrieval agent.* for that relation. One of the first tasks of the agent is to reformulate the query so that it matches the relation name and attributes at the target node. Clearly, it is possible

that some attributes may be dropped because the target relation has no such matching attributes. For combination of relations, the data retrieval agent will also rewrite the orginal query into a join query involving the combination of relations.

- If the target relations are found locally, the worker agent will submit a reformulated SQL query to the DBMS to retrieve the data. The data is then returned to the agent, formulated for output and returned to the user.

- If the target relations are on a remote node, then the worker agent will be dispatched with the query node's IP address. Answers will be returned directly from the remote host to the master agent who will then formulate the answers and return to the user.

We note that the two phases are only logical. In fact, as soon as relations are returned, they are shown to the user, and the user can start selecting relations; and as soon as a relation (or combination of relations) is selected, the agent is sent out to retrieve the data. In this way, answers are returned progressively (without long waiting time). Moreover, users could be viewing answers (data) while there may be other agents still searching the PeerDB network for candidate relations.

4.3.2 Processing Remote Query

As mentioned, essentially, for a remote query, it is an agent that arrives at the node.

Phase I: Relation matching agent

- If the agent has not visited the node previously, the TTL value is reduced by one.

- The agent will search the **export dictionary**. Promising relations are then returned to the query node at the IP address provided by the agent.

- If TTL > 0, the agent will clone more relation matching agents and dispatch them to the neighbors of the current node; otherwise, the agent will be dropped.

Phase II: Data Retrieval Agent

- The agent will formulate an SQL query and submit it to the DBMS.

- Once the answers are retrieved, they are returned to the query node directly.

- The agent may then be dropped.

4.4 Monitoring Statistics

One of the tasks of the master agent is to perform the reconfiguration of the network based on a reconfiguration policy selected by the user. The master agent monitors two types of statistics. The first is the relation information obtained from the first phase of the query processing strategy. In particular, the keywords of selected relations may be "exchanged" to update the meta-data. The second is the number of answer objects obtained from the selected relations. This can be used to determine the nodes to be connected directly.

PeerDB also extended BestPeer with a temporal locality based reconfiguration policy 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.

4.5 Cache Management

PeerDB supports caching of answers returned from remote nodes in order to reduce the response time for subsequent answers. For every relation that the user retrieved (in phase II of the query processing strategy), we cache the answers. Caching raises many complicated issues. We looked at three of them here. First, the cached copy may be outdated. To handle this, PeerDB only keeps the answers for a fixed period of time, after which the cache is invalidated. Second, since storage space is limited, we adopt a LRU replacement policy: whenever we run out of disk space, we replace the cache that is least recently used. Finally, in a P2P environment, many PeerDB nodes may be caching the same data. As such, a search may give rise to multiple "copies" of the same data. While this is a semantic issue that is to be left to the user, we attempt to minimize the effort as follows. For each cached relation, We also maintain the information on the BPID of the source node (recall that each node has a unique identifier BPID provided by BestPeer technology). When a node is not the source of a relation, its response to a search will also include the BPID of the source node. All relations, except one, with the same keywords from the same source node will be pruned away during phase I of query processing.

5 A Performance Study

In this section, we report an extensive performance study conducted to evaluate PeerDB. We first compare PeerDB against the Client/Server (CS) Architecture. The basic difference between the two models is that in a P2P model the interacting processes can be a client, server or both while in a CS model one process assumes the role of a service provider while the other assumes the role of a service consumer. Our CS model has some flavors of P2P in that a node can be both a client and a server. However, like CS model, the server must return its result to the client - as such the results must be returned along the query path. We study two versions of PeerDB - a static PeerDB where the reconfigurable feature is turned off, and a dynamic PeerDB with the reconfiguration feature turned on. We compare both schemes with the CS architecture. This allows us to see the benefits of the reconfiguration scheme. We shall denote these two schemes as PDMS and PDMR respectively. We also compare PeerDB with pure message-passing based protocol and agent-based protocol. The objective in this experiment is to show the cost and effect of using agents in PeerDB. Finally, we evaluate the efficiency of self-reconfigurable network with a static network. We used the Gnutella [9] protocol as a representative static network candidate. Before we look at the experiments and findings, we shall propose an evaluation methodology for P2P-based systems.

5.1 Evaluation Methodology

Any system has to be evaluated based on its efficiency and effectiveness. The former deals with the performance issue, while the latter deals with the quality of the answers. Unlike existing distributed systems, there is no clear criteria on how P2P systems should be evaluated. Like Internet search engine, the answers to queries depend on the peers that are searched, which may not include every peer in the P2P network. In addition, every query may involve different peers (since peers change over time)!

For purpose of evaluation, a controlled environment is necessary. We propose that the following three scenarios be evaluated. First, different schemes should be evaluated based on a fixed set of nodes. This can be useful for a set of nodes that exploit P2P technology to facilitate collaboration, i.e., it is essentially a traditional distributed environment where all nodes participate in answering a query. Here, we can study how different P2P protocols or reconfiguration strategies perform.

Second, in a P2P network, the rate at which answers are returned are important. This is because the users have no idea of which peers will be providing the answers to his/her

queries, and how many peers will be searched. A long initial waiting time is not likely to be acceptable to the users.

Third, the quality and quantity of the answers returned are important measures too. A node may return answers quickly, but it may return only very few answers or answers that are not very relevant to the query. While quality is based on the semantics of the query, quantity of answers is easy to obtain and use as a performance metrics.

5.2 Experimental Setup

The experimental environment consists of 32 PCs with Intel Pentium 200MHz processor and 64M of RAM, and all the PCs are running on WinNT4.0 operating system. The physical network layout is shown in Figure 3.

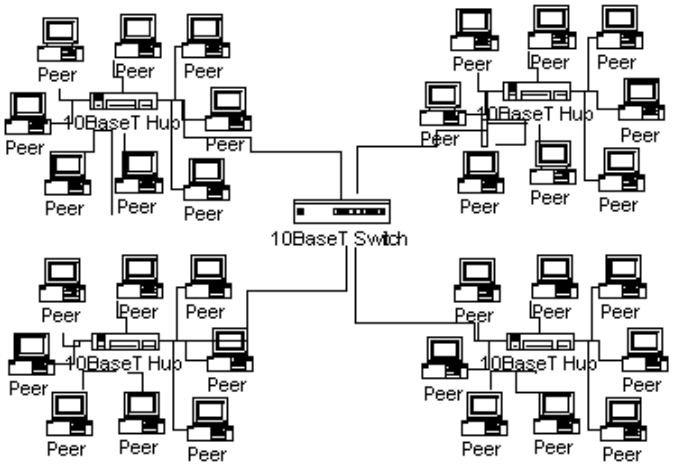


Figure 3: Experimental environment.

There are a total of 10,000 objects, each of which is 10 KB. These data are randomly assigned to nodes, such that each node holds 1,000 objects.

In practice, we expect users to be interested in part of the entire datasets only. There will always be some data that are of no interests to them, and will never be accessed by them. For example, in the case of Napster, while there are always classical music being shared, some user who prefers contemporary music may just dislike them totally. In our experiments, we try to model this by dividing the queries of each user as follows: (a) $x\%$ of queries are directed at 'hot' data in the entire dataset. These hot data are also frequently accessed by other users; (b) $y\%$ of queries are directed at $z\%$ of the cold data. This model the case that individual user may have their own taste on cold data. (c) the remaining queries are directed at the remaining cold data. As default, we set x to be 80%, y to be 15%, and z to be 20%.

Moreover, 20% of the data are hot, and 80% of them are cold.

The experiments were conducted when the machines and the network were fully dedicated. Moreover, each node is "warmed up" to fill out its local storage before we start to collect results on the experiments. The results presented correspond to the average of at least three different executions. The variance across different executions was not significant.

5.3 Effect of Storage Capacity on Caching

In the first set of experiments, we compare PeerDB with the CS architecture by varying the storage capacity of each peer. We define the storage ratio to be the size of the storage size of a node to the size of the objects stored at the central server. Figure 4 shows the effect of storage ratio on the response time. First, we observe that as the storage ratio increases, both methods' response time decreases. This is expected as more objects can be found in local and neighboring peers. This also clearly illustrates the benefits of sharing storage resources. Second, we note that PeerDB outperforms the CS model. This is expected as the CS model requires the answers to be returned via the search path.

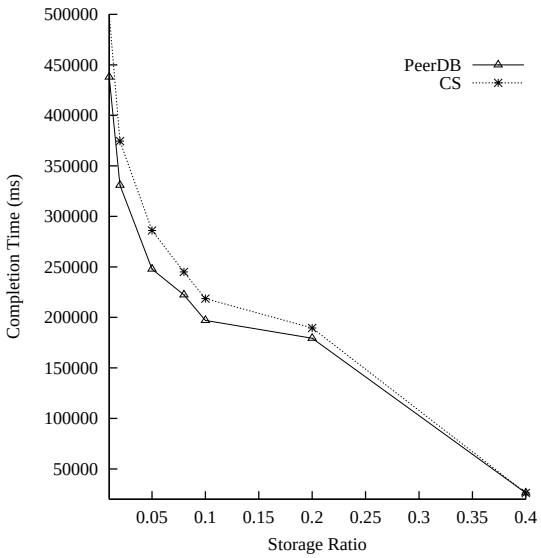


Figure 4: Effect of storage capacity.

5.4 PeerDB vs CS

5.4.1 On Initial Response Time

In this experiment, we evaluate the performance of PDMS, PDMR and CS on the rate at which answers are returned. The number of nodes is fixed at 32. A search query is issued four times, and the average time at which nodes respond are noted. Figure 5 shows the results

of the experiment. In the figure, the point (K, T) indicates that K nodes have responded after T time units. We note that it is possible that under different schemes, different nodes respond at different time and with different answers. We shall defer this discussion to the next experiment.

As shown in the figure, PDMR is still the best scheme, outperforming PDMS by virtue of its ability to reconfigure the network. It is able to reach out to more promising nodes directly - after each query, PDMR will reconfigure itself so that the next query can be directed to the more promising nodes first. We note that, except for the first few nodes, CS returns answers much slower than PDMR/PDMS - as it only returns answers along the path that the query has been transmitted.

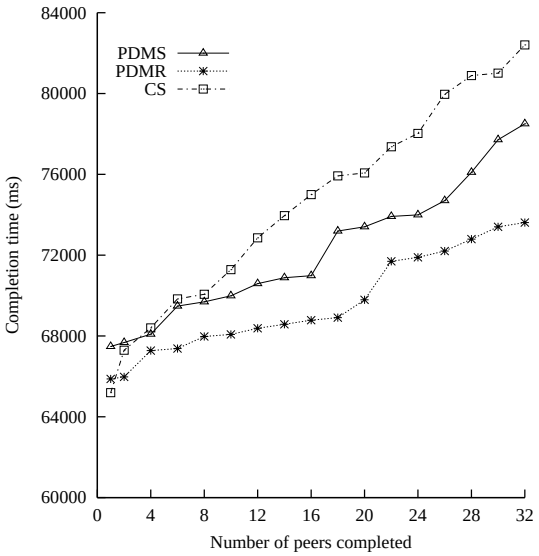
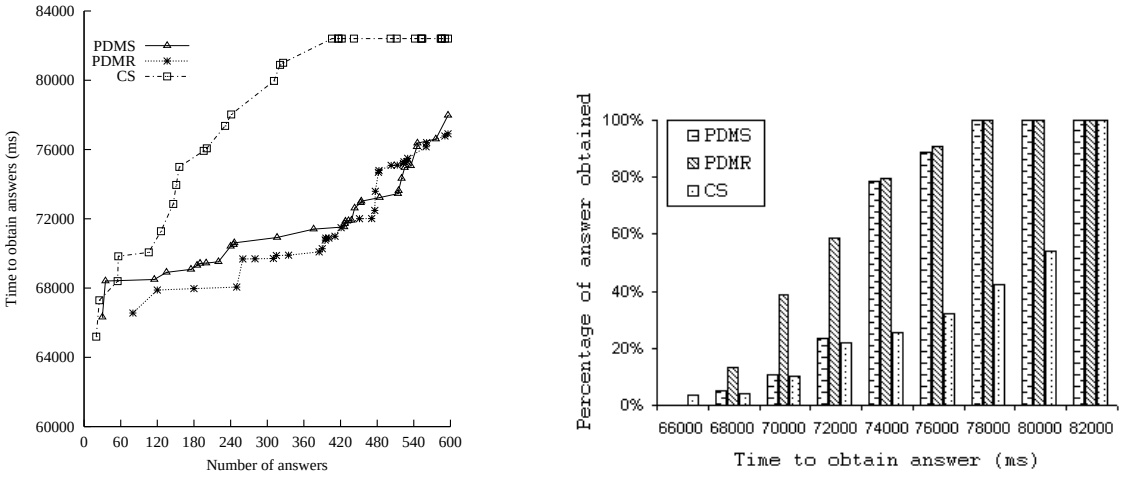


Figure 5: Rate at which answers are returned.

5.4.2 On Quantity of Answers

Having a fast initial response time is not good enough. It is possible that nodes that return answers first provide very few answers. For the earlier experiments that study the initial response time, we also keep track of the number of answers that are provided by each node. Figure 6(a) shows a plot of the result. As shown, it is clear that CS returns the first few answers much faster than PDMS and PDMR. This is expected since the first few directly connected nodes that receive the query can return their answers immediately. For PDMS/PDMR, the overhead of the code-shipping strategy results in a longer initial response time performance. However, as more answers are returned, PDMS/PDMR are superior over CS, demonstrating the superiority of P2P technologies over traditional CS models. We also



(a) First search query.

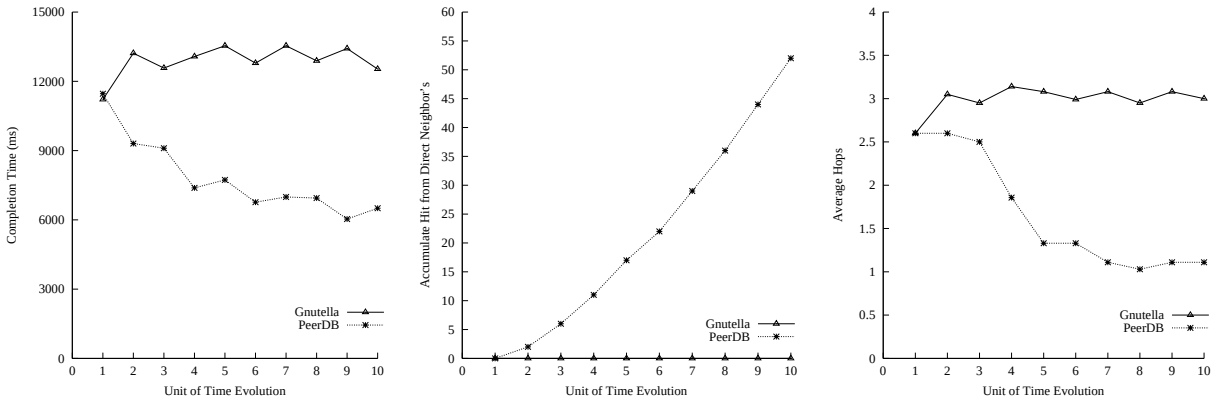
(b) % answers returned.

Figure 6: Number of answers returned.

note that PDMR is generally better than PDMS. In Figure 6(b), we can further confirm the effectiveness of PDMS/PDMR over CS. By the time PDMS and PDMR have received all the answers (100%), CS has only returned about 40% of the answers. As observed earlier, PDMR can generate more answers quickly by virtue of its ability to keep more relevant peers "closer".

5.5 PeerDB vs Gnutella

FURI [2] is a Gnutella protocol-compatible Java program that can participate in the Gnutella network. It is a full version program with a GUI interface that can perform most of the tasks of a Gnutella servant. In this experiment, we shall compare Gnutella with PDMR. We note that Gnutella essentially adopts a similar approach as PDMS, i.e., a node has a fixed set of peers and there is no dynamic adjustment of the set of peers one is directly connected to. Here, we shall denote PDMR as PeerDB. In this experiment, in order to simulate the distribution of shared objects in a real application, we divided nodes into different categories that share different number of objects. The distribution on number of objects shared is based on the study reported in [22], i.e., around 25% of the nodes in the Gnutella network share less than 10 objects, e.g., mp3 files, about 20% share 10 to 50 objects, 20% share 50 to 100 objects, 18% share 100 to 500 objects. Only 12% share 500 to 1000 objects and finally less than 5% share more than 1000 objects. Based on the statistic, we divided the 32 PCs into the following objects distribution. We have two nodes that share 1500 files, four nodes share 750



(a) Completion time

(b) Accumulated hits from direct Neighbors

(c) Average hop.

Figure 7: PeerDB vs. Gnutella.

files each, six nodes share 250 files, six nodes share 450 files, six nodes share 25 files and eight share 5 files. We assumed the whole universe has a total of 10000 files/objects. All objects stored in each node follow the 80/20 rules, where 80% of the files stored belong to the most popular 20% of the universe's objects. We issued 100 queries and the completion of every 10 queries is denoted as one time unit. At each time unit, PeerDB's network reconfiguration will be performed. The completion time is thus determined by the time when all the answers arrived at each time unit, i.e., 10 queries. Figure 7 shows the results of the experiments.

In Figure 7(a), we observe that Gnutella generally maintains a constant completion time. On the first unit of time, we find that for PeerDB, the completion time is nearly similar to the Gnutella for the same query. This is because for the first search, PeerDB has exactly the same network structure and same direct neighbor as Gnutella has. After the first unit of time, PeerDB self-reconfigured its network to reflect the usefulness of peers based on the pass statistic it obtained at the previous 10 queries. As shown in Figure 7(b) and Figure 7(c), the number of hit from its direct peers and average hops that are required for queries reduced significantly. Therefore, the performance of PeerDB for the subsequent units of time is superior compared to the Gnutella protocol.

5.6 Benefits of Agent-based Querying

In this experiment, we would like to study how much can be gained by using an agent-assisted query processing strategy. Here, we assume that the query requires some functions that is not supported by the DBMS. As such, the operation cannot be pushed down to the DBMS.

Instead, the data have to be first retrieved, and the operation performed on the data before the answers to the query can be obtained.

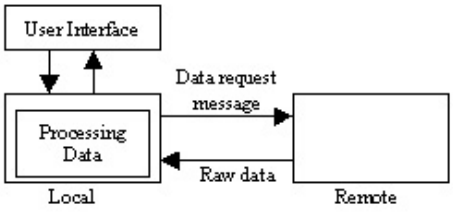
5.6.1 Single Remote Execution

In this experiment we show the cost and effect of using pure message passing protocol and agent based protocol in P2P environment. Here, we assume the query requires only one remote access. The whole process is divided into three phases (see Figure 8): sending message (message-passing protocol) or agent (agent-based protocol) to remote host, remote host processes the request, and remote host returns the result to the originator. The answer size is set to be 0.1% of the whole data set. The difference between message based and agent-based protocol is that message-based protocol is a data-shipping strategy, i.e., remote data are transferred to the query node to be processed there. On the other hand, an agent-based protocol is a code-shipping strategy that carries the processing code to the remote site and performs remote execution. Only answers produced by the agent will be returned. The total response time including the cost of data transfer, i.e., message, code and data, and processing time. In Figure 9, we observe that the completion time of the message-based protocol increases exponentially when the data size increased. The overhead of the data-shipping results in a longer response time performance. As a result, when the number of data to be transferred across the network increases, the mobile agent-based protocol is superior.

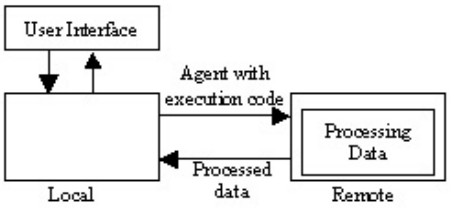
5.6.2 Multiple Remote Executions

Most of the network applications (client-server based or P2P based) require more than one communication with another node in order to complete each transaction. Therefore, in this experiment, we looked into the messages overhead in pure message-based protocol vs. agent-based protocol. In this experiment, multiple remote executions are required in order to answer a query (see Figure 10). Essentially, the query requests for multiple objects from a remote site; however, the query only knows the object to retrieve after the earlier requests is completed. Thus, we have a chain of queries and computations. In our test, each of the object requested will cause 5MB of data transfer across the network.

The result on multiple-communication transaction is shown in Figure 11. Clealry, the agent-based approach is superior. Under the message-passing protocol, the query is transmitted, and the data is returned to the node to be processed on the node. This has to be done before subsequent operations can be issued. On the other hand, in the mobile agent approach, all operations can be performed at remote node once the code is transmitted. Once the agent



(a) Message-Based Protocol



(b) Agent-Based Protocol

Figure 8: Message and agent based systems

is constructed at the remote site, thy can interact with the remote node directly until the final result is obtained. Therefore, it optimizes the network resources and bandwidth.

6 Related Works

Agent-based systems have been studied in the literature [18, 6, 15, 16]. However, these technologies provide support for agent collaboration and communication but lack support for peer-to-peer technology. Development of P2P applications based on these platforms would require a longer development effort, which would be costly.

As mentioned, P2P technologies have been deployed in many applications [20, 9, 8, 19, 3, 21]. However, these systems provide coarse granularity of sharing without capabilities for DBMS supoort. In addition, they cannot be easily extended to meet users changing needs. Moreover, their network are statically defined and lack of multi-granularity of data access.

More recently, the database community has begun to exploit P2P technologies for database applications [12, 21, 24]. In [12], data placement issues were addressed. In [24], the class of "hybrid" P2P systems where some functionality is still centralized is studied. In particular, an analytical model to describe the system performance is developed, and validated against actual hybrid P2P systems. Different architectures such as chained architecture, full replication architecture, hash architecture and unchained architecture were compared. In [14],

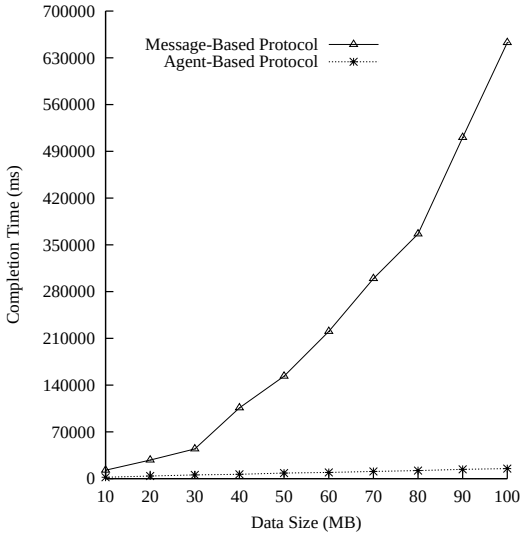


Figure 9: Completion time vs. data size

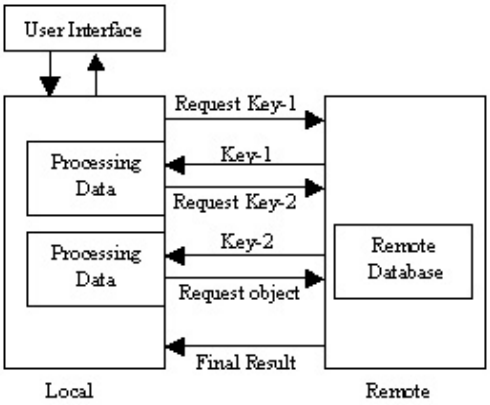
caching of OLAP queries is addresed in the context of a P2P network.

7 Conclusion

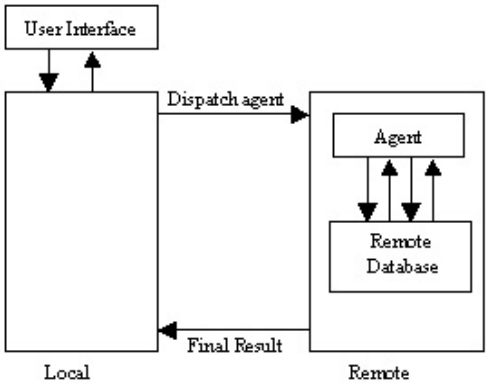
In this paper, we have presented a P2P-based distributed data sharing system called PeerDB. PeerDB has several nice features. First, it employs a data management system. Moreover, it facilitates data sharing without predetermined shared schema. Second, because its query processing capabilities is assisted by mobile agents, thus it provides easy extensibility to existing systems. Third, it provides a mechanism to reconfigure a nodes' peers based on some optimization criterion. Our extensive experimental studies show that PeerDB is a promising system for distributed processing. We plan to extend this work in two directions. First, we plan to make a node more intelligent by allowing it to determine at runtime which strategy to adopt - code-shipping or data-shipping. Second, we have focused on looking for "similar" schema. More recently, keyword-based search engine for relational databases has been developed [4]. We plan to see how such features can be integrated to facilitate keyword-based search in PeerDB.

Acknowledgements

Wee-Siong Ng and Kian-Lee Tan are partially supported by the NSTB/MOE research grant RP960668.

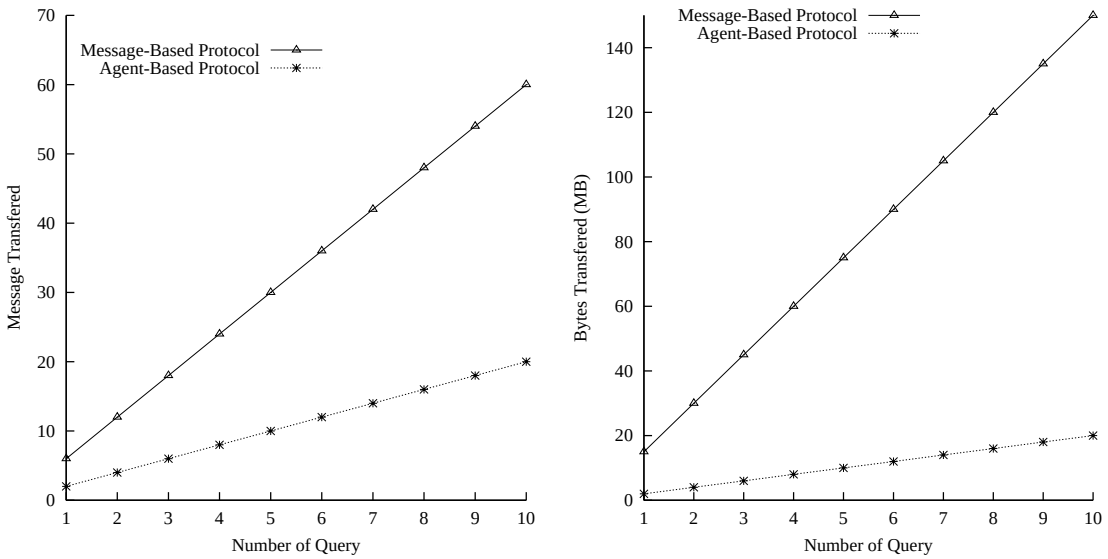


(a) Message-Based Protocol



(b) Agent-Based Protocol

Figure 10: Message-Based and Agent-Based System Setup Diagram



(a) Accumulate Message Transferred

(b) Accumulate Bytes Transferred

Figure 11: Message Overhead in Message-Based and Agent-Based Protocol

References

[1] Bestpeer. In *http://xena1.ddns.comp.nus.edu.sg/p2p/*.

[2] FURI. In *http://www.jps.net/williamw/furi*.

[3] M. Waldman A.D.R. and Cranor L.F. Publius. A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceedings of the 9th USENIX Security Symposium*, pages 59–72, 2000.

[4] 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.

[5] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison-Wesley, 1999.

[6] A. Castillo, M. Kawaguchi, N. Paciorek, and D. Wong. Concordia as enabling technology for cooperative information gathering. In *Proceedings of the 31th Annual Hawaii International Conference on System Sciences 1998 (HICSS31)*, 1998.

[7] A. Oram (Editor). *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.

[8] Freenet Home Page. *http://freenet.sourceforge.com/*.

[9] Gnutella Development Home Page. *http://gnutella.wego.com/*.

[10] C. L. Goh, S. Bressan, B. C. Ooi, and M. Anirban. Storm: A 100% java persistent storage manager. In *OOPSLA Workshop on Java and Object*, 1999.

[11] C.L. Goh, S. Bressan, B.C. Ooi, and K.L. Tan. Integrating replacement policies in StorM: An extensible approach. In *SIGMOD 2000 (Demo)*, 2000.

[12] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer. In *WebDB*, 2001.

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

[14] 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.

[15] G. Karjoth, D.B. Lange, and M. Oshima. A security model for aglets. *IEEE Internet Computing*, 1(4), 1997.

[16] N. Karnik and A. Tripathi. Agent server architecture for the ajanta mbile-agent systems. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, 1998.

[17] LOCKSS Home Page. *http://lockss.stanford.edu/*.

[18] Mitsubishi Electric. Concordia: An infrastructure for collaborating mobile agents. In *Proceedings of the 1st International Workshop on Mobile Agents (MA '97)*, April 1997.

[19] R. Dingledine D. Molnar and M. J. Freedman. The free haven project: Distributed anonymous storage service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, 2000.

[20] Napster Home Page. *http://www.napster.com/*.

[21] 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).

[22] S.Saroiu P.K.Gummadi S.D.Gribble. A measurement study of peer-to-peer file sharing systems. In *Technical Report UW-CSE-01-06-02*, Department of Computer Science and Engineering University of Washinton Seattle WA, July 2001.

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

[24] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems. In *VLDB'2001*, 2001.