BestPeer: A Self-Configurable Peer-to-Peer System

Wee Siong Ng *Beng Chin Ooi* *Kian-Lee Tan*
Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
email: {ooibc,ngws,tankl}@comp.nus.edu.sg

Abstract

In a peer-to-peer (P2P) distributed system, nodes of equivalent capabilities and responsibilities pool their resources together to share information and services. However, most of the existing P2P systems such as Napster and Gnutella provide a coarse granularity of information sharing (i.e., file level) that ignore the content of the file. Moreover, a node's peers are typically statically defined. In addition, there is no support for nodes that are connected intermittently with temporary network addresses. In this paper, we present BestPeer, a prototype P2P system that we have implemented at the National University of Singapore. BestPeer is unique in several ways. First, it combines the power of mobile agents into P2P systems to perform operations at peers' sites. This facilitates content-based searching easily. Second, it 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. Third, BestPeer provides a location independent global names lookup server to identify peers with dynamic (or unpredictable) IP addresses. In this way, several peers can always collaborate (or share resources) even if their IP addresses may be different at different occasions. We evaluated BestPeer on a cluster of 32 Pentium II PCs each running a Java-based storage manager. Our experimental results show that BestPeer provides excellent performance compared to traditional non-configurable models. Further experimental study reveals its superiority over Gnutella's protocol.

Keyword: Peer-to-peer, mobile agent, neighbors, dynamic reconfiguration, replacement policy.

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). It is a network architecture in which all participating computers (or nodes) have equivalent capabilities and responsibilities. P2P technology enables the direct exchange of resources and services between nodes by eliminating the need for centralized servers. 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. For example, Freenet [4], and Gnutella [5] enable users to share any digital files (e.g., music files, video, images); Napster [14] allows sharing of (mp3) music files; ICQ [8] facilitates exchanges of personal messages; Seti@home [15] makes computing cycles of participants available; and LOCKSS [12] pools storage resources to archive document collections.

However, most of the existing 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 support for content-based search. Second, they lack extensibility and flexibility. As such, there is no easy and rapid ways to expend their applications quickly to fulfil new users needs. Third, a node's peers are typically statically defined. Fourth, current P2P systems either rely on a DNS server to resolve domain names or deploy a centralized server to maintain globally unique names [8]. For the former, since a domain name server's entries usually refer to permanent IP addresses, this reduces the participation of nodes with variable connectivity and temporary network addresses in the activities of peers. For the latter, the server may become a bottleneck. Moreover, like all centralized approaches, it is not scalable.

In this paper, we present our solutions to the above problems. First, we integrate mobile agent and P2P technologies. 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 P2P system. For example, while an agent may search for files based on file names, another may perform a content-based search on the file. Second, we incorporate a mechanism 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. Third, we introduce a location independent global names lookup server (LIGLO) 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 being a bottleneck, we adopted a distributed approach where several LIGLO exists in the BestPeer network.

We implemented BestPeer, a prototype P2P system that incorporates all the above features. To evaluate BestPeer, we propose a systematic methodology for evaluating P2P systems. Our methodology considers both efficiency and effectiveness (quality fo answers) of P2P systems. We conducted our experiments on a cluster of 32 Pentium II PCs each running a Java-based storage manager [6]. Our experimental results show that BestPeer provides excellent performance compared to traditional non-configurable models. We also evaluated BestPeer against the protocol of Gnutella. Our study shows that BestPeer is superior over Gnutella.

The rest of this paper is organized as follows. In the next section, we shall present an overview of the BestPeer network. Section 3 describes several features of BestPeer that was incorporated to overcome the limitations of existing P2P systems. In Section 4, we report an extensive experimental study to evaluate BestPeer. Section 5 gives a review of related works, and finally, we conclude in Section 6 with directions for future work.

2 The BestPeer Network

BestPeer is a generic P2P system designed to serve as a platform on which P2P applications can be developed easily and efficiently. Figure 1 illustrates a BestPeer network. 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. Each node comprises two types of data: private data and sharable data. Nodes can only access peers' data that are sharable. Using Figure 1 as an example, Peer A can *directly connect*[1] to Peer B to obtain its sharable data, while it can only reach Peer C via Peer B. However, in BestPeer, data are downloaded out-of-network, i.e., a direct connection between Peer A and Peer C is established in order to perform the data transfer (without having to go through Peer B). In addition, no messages need to be transmitted from the peer back to the query initiator along the query path.

We shall delay the discussion on the LIGLO servers to a later section. It suffices to say here that they are used to uniquely identify nodes whose IP addresses may change as a result of frequent connection to and disconnection from the BestPeer network. Through the LIGLO servers, a node knows exactly who its peer is; otherwise, the same peer with a different IP address each time it joins the network may be considered as a 'new' participant. Strictly speaking, if a node does not care about the identity of its peers, then, it need not use the service of LIGLO servers.

The BestPeer software essentially provides each node with an environment in which (mobile) agents can reside and perform their tasks. This makes the system highly extensible and powerful.

Now, consider a node (not a registered member of BestPeer) who would like to become a participant of BestPeer. The process is as follows:

- The node registers with a LIGLO server. This is similar to a user registering to a mail server in Internet environment.
- The LIGLO server will issue the node with a global and unique identifier, which we shall refer to as BPID (BestPeer ID). This BPID serves to uniquely recognize this node regardless of its current IP address. BPID is essentially a (LIGLOID, NodeID) pair where LIGLOID is the IP address of the LIGLO server and NodeID is a unique id for the node assigned by the LIGLO server.
- At the same time, the LIGLO server will also send a list of (BPID, IP) pairs that the node can communicate directly, i.e., direct peers of the node. Here, the ith BPID value is the

[1] Note that this is only a logical 'connection'.

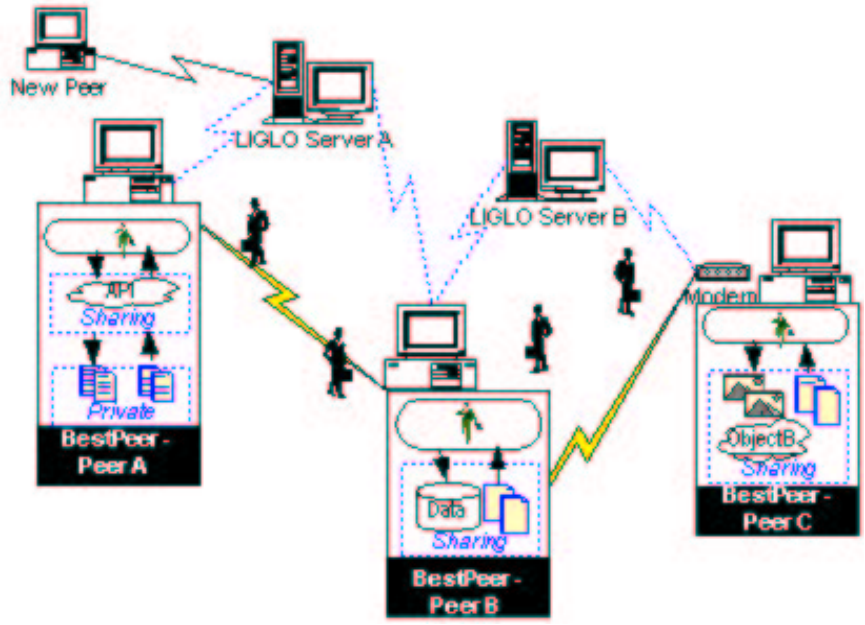


Figure 1: BestPeer network

identifier of the ith peer, and the corresponding IP value is the current IP address of this peer. We note that since the peer is not obliged to inform LIGLO of its disconnection, the IP address may not be a valid one. In BestPeer, LIGLO will periodically check the validity of its registered participants' IP addresses.

- The node is now a participant of BestPeer and is ready to communicate with any peers (without going through LIGLO anymore).

On the other hand, for a participating node who wants to rejoin the BestPeer network after disconnecting, the process is as follows:

- The node will send its IP address to its LIGLO. This allows its LIGLO to update its IP address if it has changed.
- For each peer of the node, say p, it will send p's BPID to its (i.e., p) registered LIGLO server. Recall that p's registered LIGLO can be obtained from p's BPID.
- p's registered LIGLO server will reply with the IP address of p if its is currently connected to the network; otherwise, it will indicate that p is now offline. This is necessary for the node to know its peers' new IP addresses if they have been changed. We note that p is not obliged to inform its LIGLO server that it will be (or is) disconnected. As such, the information may not be accurate anyway.

- The node has rejoined the BestPeer network, and is ready to communicate with its peers.

We note that this process is not necessary for a participating node who rejoins the BestPeer network (except to inform the LIGLO server of its new IP address). It can simply communicate with its existing peers. Should the IP addresses of some peers be invalid (i.e., they may have changed their IP addresses or disconnected), then it can simply replace those peers by new peers that it encounters (based on some criterion).

Once a node is connected to the BestPeer network, it is ready to share its resources as well as has access to other nodes' sharable resources. A node essentially broadcasts its query to its directly connected peers, and its peers will broadcast the message to their peers, and so on. Any nodes with matching results will respond to the initiating node directly. In BestPeer, there are two modes in which a node can have access to data from other nodes:

1. In the first mode, nodes with matching answers will return the answers directly. This method can provide fast answers but may result in overloading and poor bandwidth utilization especially if significant amount of data are not desirable (e.g., too much overlap, files too large, etc).

2. In the second mode, nodes with matching answers will only indicate that they have the information, e.g, by returning the file name, etc. The initiating node will then send a further message to some, if not all, of these nodes to obtain the desired information. This mode provide better resource utilization at the expense of a delayed request. Since there is a delay, and the request is initiated by the source of the query, it is possible that the target node may have removed the desired content or updated it during the period of delay.

3 Features of BestPeer

In designing BestPeer, we sought to overcome the limitations of existing P2P systems. As such, BestPeer was designed with several features that distinguish itself:

1. BestPeer combines the power of agent technology and P2P technology into a single system.

2. BestPeer not only facilitates a finer granularity of data sharing where partial content of a file may be shared, it also shares computational power.

3. BestPeer facilitates dynamic reconfiguration of BestPeer network so that a node is always directly connected to peers that provide the best service (based on some optimization criterion such as providing the most number of answers or providing answers most of the time).

4. BestPeer adopts a distributed approach to minimize bottlenecks of servers acting as LIGLO.

In this section, we shall discuss these features in greater details.

3.1 Integration of Mobile Agents and P2P Technologies

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, but processed and meaningful information. For example, in BestPeer, an agent can be sent to a peer with the data file to "digest" its content and to generate reports for the requester.

In BestPeer, we have implemented our own Java-based agent system instead of using existing systems (e.g., [11]). Like existing systems, both the agent and its class have to be present for the agent to resume execution at the destination engine. Thus, if the class is not already at the destination node, the class has to be transmitted also. For the moment, we have adopted a purely "code-shipping" strategy where a node will always perform its operation at the destination node (where the data reside). This is a reasonable approach as it exploits parallelism by enabling all peers to operate on their data simultaneously; otherwise, the node will become a bottleneck.

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.

Traditionally, mobile search agents perform search operations by moving itself to the site containing the target information and executing a given task. The agent's path is pre-defined. The agent's programmers have to know where the agent need to go and where the next destination is after the task at a site is completed. Another problem with the traditional agent approach is that when a host has more than one directly connected host, the agent's developers have to decide which path to follow and then keep track of it. When the network grows more complicated, searching through the network becomes a nightmare.

BestPeer adopts a different strategy. It solves these problems by providing a simple interface to search all directly and indirectly connected hosts. An agent's path is transparent to the agent's developer. An agent is sent to all connected host automatically without statically defining the mobile agent's path. An agent will be cloned and sent to all connected hosts in parallel. The process of cloning and forwarding will keep on going until the agent lifetime is expired. The lifetime of an agent is determined by Time-to-live (TTL) and Hops variables. It is similar to

other packet approach using in networking environment. Once received an incoming agent, if the agent is not expired (if TTL > 0), remote host will decrease the TTL values of an agent before sending it to any other host that it is directly connected to. Hops variable will be increased at the same time too. The redundant use of TTL and Hops together is to enable hosts to drop any incoming agent that already has a copy on the site.

3.2 Resource Sharing

The notion of sharing is one of the main factors that fueled the growth of the Internet. Most P2P applications permit sharing of static files such as mp3 audio files, text files and image files. BestPeer supports sharing of static digital files, *active objects* that facilitates finer granularity of data sharing (and hence access control), as well as computational power. For uniformity, all requests for these resources are performed with agents.

3.2.1 Static files

In BestPeer, any kind of files in digital format can be traded in its entirety including text file, word document, images, music files, movie files, executable code (programs, software), and so on.

3.2.2 Active Objects

However, in many applications, different users may have different access rights to the content of a file. While one may be allowed to see the entire content of the file, another may be denied access to some sensitive information. To support finer granularity of data sharing, BestPeer employs the concept of an *active object*. In active objects, two types of elements are defined: data elements and active elements. A data element describes the content of an object; an active element, on the other hand, contains the name of an *active node* that operates on the object to generate the content. Essentially, an active node is a 'black box' (i.e.,an executable code) that receives and sends messages and interacts with the outside through an interface. Depending on the access right of the requester, the active node returns the appropriate content. Using the same illustration, for a person who should be denied sensitive information, the active node will scan the input file, filter away the sensitive information and return the non-sensitive portion to the requester. It is the responsibility of the owner to ensure the correctness of the active object (i.e., that sensitive information should only be accessed by those with the proper access rights).

3.2.3 Computational Power

BestPeer also facilitates sharing of computational power for requests to local files as follows. 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. First, 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). Second, 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). This is in contrast with the use of active objects where the owner defines what to filter. Third, it facilitates extensibility - new algorithm or program can be used without affecting other parts of the system! Fourth, existing non-distributed objects can be easily extended for use by a P2P application by leveraging on the support provided by BestPeer. Finally, it optimizes network bandwidth utilization as only the necessary data is transmitted to the requester.

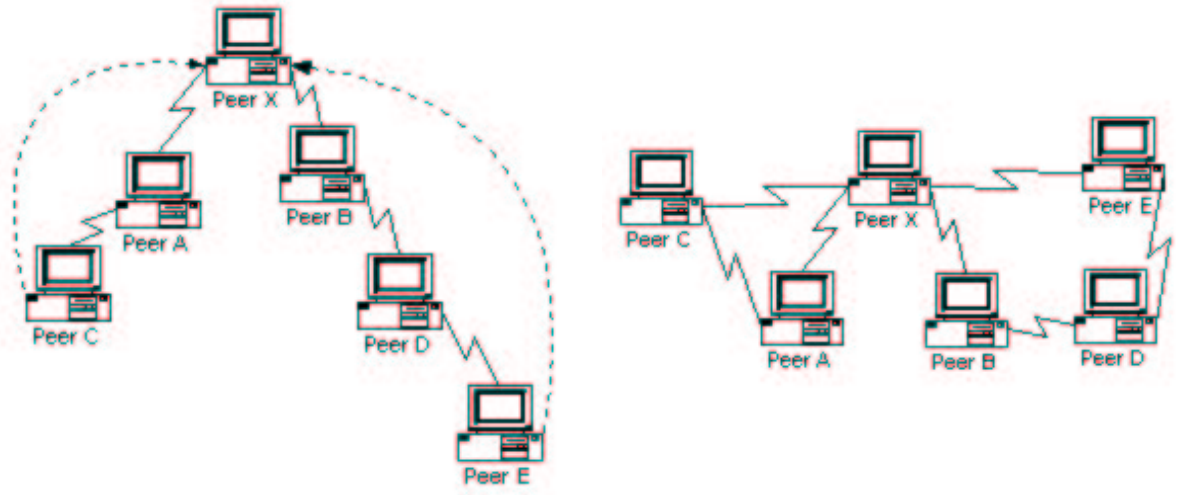
This feature is easily realized by the integration of mobile agents into P2P framework. Agents that carry code can be dispatched to the data provider.

3.3 Reconfigurable BestPeer Network

Existing P2P systems either adopt a static peer network where a node always has the same set of peers or allows users to manually determine the peers of a node (that does not change automatically during runtime).

BestPeer takes a different approach – 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. Every BestPeer node has its own control over the maximum number of direct peers it can have. Figure 2 illustrates an example of BestPeer's reconfigurable feature. In Figure 2(a), Peer X is the base node that initiates a request. Here, Peer X initially has two directly connected peers - Peers A and B. However, only Peer C and Peer E contain objects that match Peer X's current query. Peer X can then obtain the results from Peer E and Peer C directly. At the same time, Peer X determines that Peer C and Peer E are not its direct peers and they benefit it most. As such, Peer X will keep these two peers as its directly connected peers (assuming Peer X can keep at least 4 directly connected peers), resulting in the new network layout shown in Figure 2(b).

Our approach is to keep promising peers as close as possible with no (or little) information exchange between peers. This is to keep the nodes as autonomous as possible. Moreover, since nodes can redefine the number of direct peers it would like to have and implement their own reconfiguration strategies, any tight form of "collaboration" will be complicated to realize and



(a) Before reconfiguration.

(b) After reconfiguration.

Figure 2: Example on BestPeer's Reconfigurable Feature.

maintain. In BestPeer, two default reconfiguration strategies have been designed and deployed.

The first strategy, MaxCount, maximizes the number of objects a node can obtain from its directly connected peers. It works as follows:

- The node sorts the peers based on the number of answers (or bytes) they returned.[2] Those that return more answers are ranked higher, and ties are arbitrarily broken. The assumption here is that a peer that returns more answers can potentially satisfy future queries.
- 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.

We note that this strategy only keeps track of the k directly connected peers, without any knowledge about these peers' direct peers.

The second strategy, 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 answers can be obtained with the minimal

[2] We note that many different criteria can be defined. However, their usefulness are domain dependent. We believe a simple strategy like MaxCount should suffice to cover a wide range of applications.

number of hops. Thus, this strategy simply orders peers based on the number of hops, and pick those with the larger hops values as the immediate peers. In the event of ties, the one with the larger number of answers is preferred.

3.4 Location-Independent Global Names Lookup Server

In P2P systems, since nodes can join and leave the network at any time, their IP addresses may be different each time. As such, under DNS, a participating node is effectively treated as a different peer whenever its IP address is different. However, for some applications, recognizing a node (even if the IP address may change each time it is connected to the network) is important. For example, a set of nodes may agree to be peers to collaborate in performing some tasks. As another example, a node may particularly be interested in monitoring the updates of a set of peers. These cannot be realized with DNS alone. To facilitate identification of a single node that may have different IP addresses at different occasion, each participating node can be assigned a unique BPID, and a centralized server keeps track of the (BPID,IPaddress) pair whenever a node is connected. In this way, one can always be certain of its peers and their "new" IP addresses.

BestPeer adopted such an approach - it introduces a Location-Independent Global Names Lookup Server (LIGLO). 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). As mentioned, BPID is a unique identifier for a peer. Unlike the centralized approach that is used in systems like ICQ [8], where only one server has the control to maintain the consistency of defined names, there is no limit on the number of LIGLO servers that can run in one BestPeer network. Each LIGLO needs only to maintain its members' name uniquely.

Most of the centralized name servers have to be powerful machines because they have to handle huge number of requests. On the contrary, a LIGLO sever can limit the number of members that it will handle based on its capability. When the limit is reached, a LIGLO server can reject any new inquiry on assigning BPID in order to preserve the efficiency for the existing members. The node has to seek for another LIGLO for registration. Once a node is registered with the BPID, it has to inform LIGLO each time it is connected to the BestPeer network by submitting its current IP to LIGLO. This information can be used by other nodes who may want to uniquely track a node whose IP addresses may change.

The use of a distributed LIGLO services has the following advantages:

1. No single point failure - LIGLO is a distributed name server; therefore they do not have any single point failure problem. For example, if a peer A registered with LIGLO A finds

that LIGLO A is down, it can still communicate with other peers that it has. In addition, other peers that registered with other LIGLO server will not be affected at all. This is in contrast with centralized name server approach (such as ICQ server) where a failure at the centralized server means that all peers will lose their connection.

2. Unlimited name resources - One of the problems with centralized name server is that all the names must be uniquely defined. For example, if somebody has registered the domain name of "www.mydomainname.com", then that is the one and only one. In LIGLO, on the other hand, a name is unique only with respect to its own server. Two nodes can register to two different servers and be assigned the same name as long as this name has never been previously registered.

3. Scalability. A LIGLO server can be added easily into the network without affecting any of the existing network environments.

4. Support temporary network addresses as the norm - LIGLO defines its own protocol-specific addresses, BPID, to replace the dynamic IPs. This BPID is fixed and can be used in place of dynamic or fixed IPs. Therefore, there is no difference between nodes that have DNS entries and those that do not have. All of them now have the same ability to hosting data and net-facing applications locally.

4 A Performance Study

We implemented the BestPeer software with the features discussed in the previous sections. Any node that installs the BestPeer software and register with a predetermined set of LIGLO servers can participate in the BestPeer network. In this section, we report an extensive performance study conducted to evaluate BestPeer. We compare BestPeer against Single-Thread Client/Server (CS) Architecture and Multi-Thread CS Architecture in different network layout topologies. The basic difference between CS and P2P 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 also compare BestPeer's protocol against Gnutella's. We studied two versions of BestPeer - a static BestPeer where the reconfiguration feature is turned off, and a a dynamic BestPeer with the reconfiguration feature turned on. This allows us to see the benefits of the reconfiguration scheme. We shall denote these two schemes as BPS and BPR respectively. Before we look at the experiments and findings, we shall propose a evaluation methodology for P2P systems.

4.1 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.

4.2 Experimental Setup

The experimental environment consists of 32 PCs with Intel Pentium 200MHz processor and 64M of RAM. Nine of the PC running on WinNT4.0 operating system, 22 running on Window98 and 1 running on Window Millennium. The physical network layout is shown in Figure 3. The experiments were conducted when the machines and the network were fully dedicated and the results presented correspond to the average of at least three different executions. The variance across different executions was not significant.

In the experiment, there is a set of nodes in the network and each of these nodes has a local copy of StorM object [2]. StorM is a 100% Java persistent storage manager. Data to be shared are stored in StorM. For our study, each node stores 1000 objects in StorM to be shared, and these objects are accessible via StorM's API. For simplicity, we have set all objects to be of the same size - 1K bytes. Moreover, there is no replication, i.e., there is only one copy of an object in the BestPeer network used in our study.

We implemented a StorM agent, that takes as input a query from the user (in the form

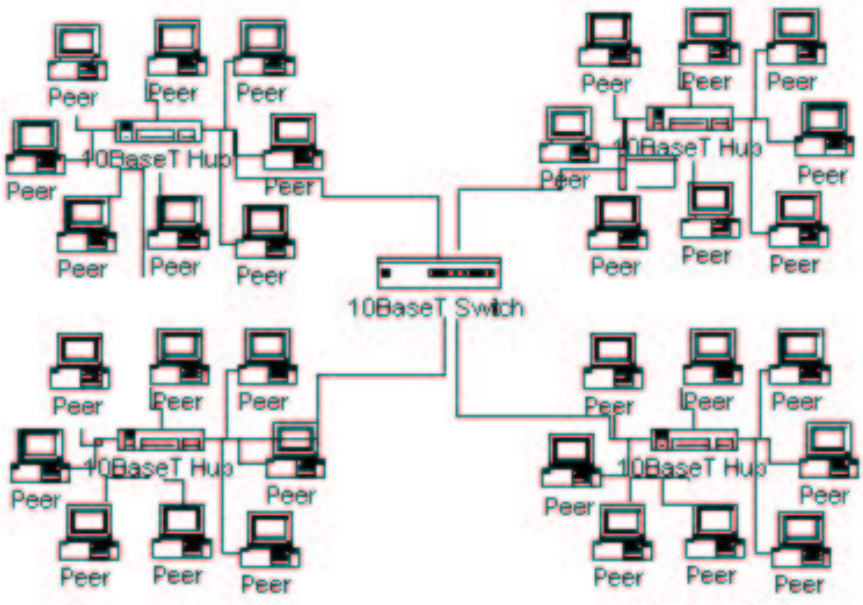


Figure 3: Experimental environment.

of a keyword), and then search through the entire BestPeer network. The goal is to find the occurrences of objects in StorM of each node that match the query. The whole search process of StorM agent operates as follows:

1. Send a StorM agent. The base node sends an agent to its directly connnected peers.

2. Executing the StorM agent. All the peers that receive the incoming agent will prepares a new thread of execution for the agent.

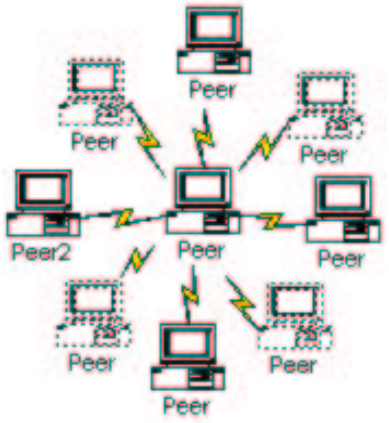
3. Interact with StorM object. The agent makes a comparison for each object stored in the Shared-StorM database with its query. All the matched results are stored in a temporally array.

4. Send the result back. The result is sent back to the base node.

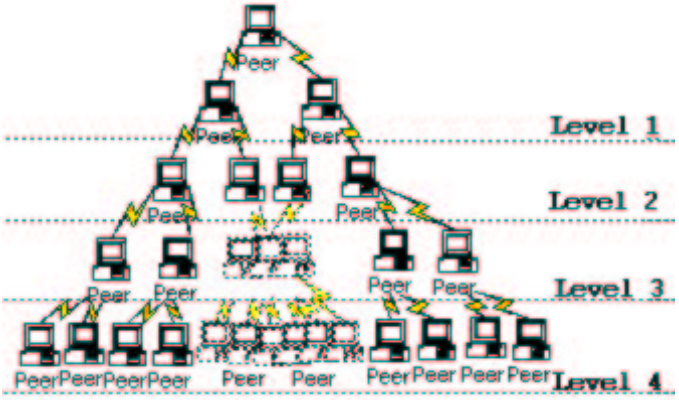
We also incorporated the GZIP data-compression algorithm in the current implementation of BestPeer. All the agent and messages used for communications between every nodes or peers are in a compressed data representation. Compression and un-compression are performed automatically by BestPeer platform and are transparent to the software developers.

4.3 On Different Network Topology

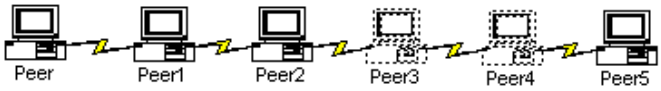
We first begin by evaluating BestPeer on different logical network topology - namely the Star, Line and Tree structures as shown in Figure 4. In the Star structure (see Figure 4(a)), the central node is the base node that initiates the search query, and all other nodes are directly connected to the base node. In the Tree structure (see Figure 4(b)), the root node is the base node that initiates the search request. Each node in the Tree structure, except for the leaf nodes, has k directly connected peers. In the Line structure (see Figure 4(c)), all nodes have two peers, except for the end nodes which have only one peer. Here, we used the left most node as the base node that initiates the search query.



(a) Star topology.



(b) Tree topology.



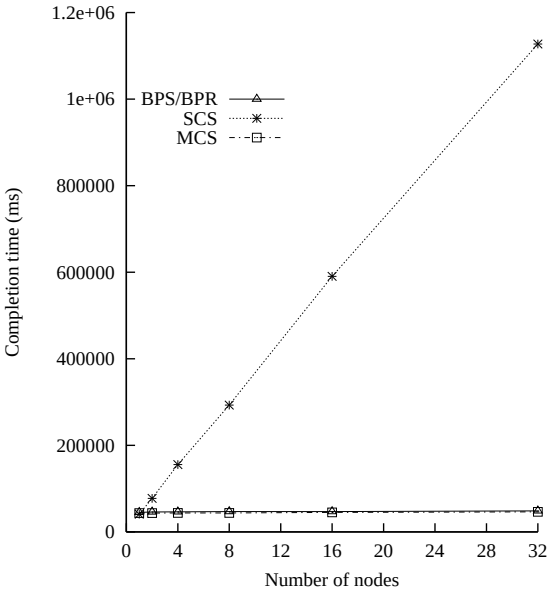
(c) Line topology.

Figure 4: Different network topologies used in the experiment.

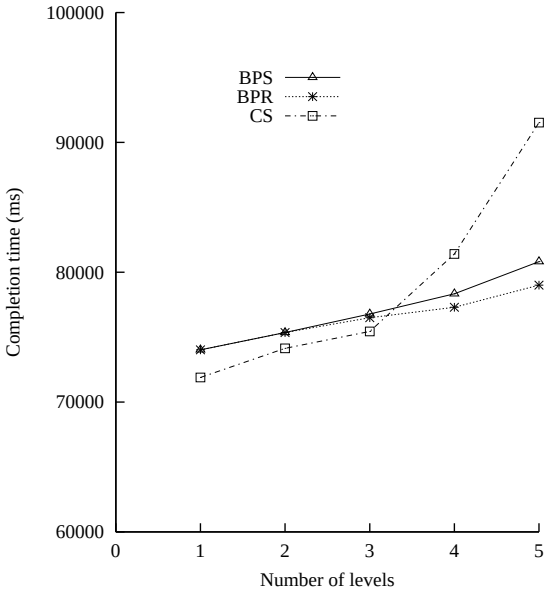
In this experiment, we vary the number of nodes from 1 to 32. We run each scheme several times and used the completion time as the performance metrics. The completion time is taken to be the time when all answers from all nodes have been received. Figure 5 shows the results.

On Star Topology

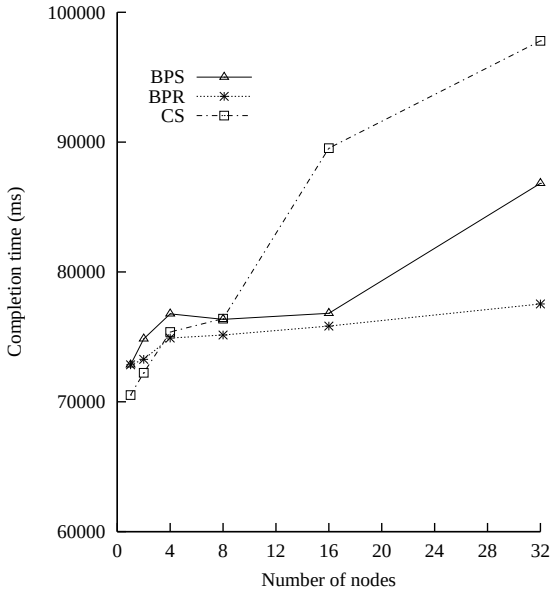
Figure 5(a) shows the results for Star topology. First, we note that Static BestPeer (BPS) and Reconfigurable BestPeer (BPR) show similar performance. This is because under the Star



(a) Star topology.



(b) Tree topology.



(c) Line topology.

Figure 5: On network topologies.

topology, there is no difference between the two schemes. As shown in the results, when we increase the size of the network, the Single-Thread CS (denoted SCS) performs worse than the other models. This is because SCS can only handle one connection at any moment - it has to complete the first operation before switching to the second node for another operation. We also note that both MCS and BP-based schemes outperform SCS significantly. This is so because these schemes exploit parallelism by simultaneously handling multiple connections and transmitting multiple queries to all peers. We also observe that MCS is slightly better than BPS/BPR but the gain is not significant enough to be visible. We shall explain this further when we look at the results for Tree and Line topologies. Since SCS performs poorly, we shall not discussed it further. For all subsequent experiments, we shall use MCS only, and for simplicity, we shall denote it as CS.

On Tree Topology

Figure 5(b) shows the result on Tree topology. We note that in this experiment, we used only 48 nodes instead of 63 for level 5. We make several interesting observations. First, we note that CS can outperform BPS and BPR (as noted in the earlier experiment). This is expected as BPS and BPR are essentially code-shipping strategies - not only do they need to transmit the code/agent to the peers, they must also incur the overhead of reconstructing the agent at the peer site.[3] On the other hand, under CS, it is simply transmitting a query, and the algorithm at the server performs the task there. As a result, when the number of levels is 1 (which means all peers are directly connected as in the Star network), CS is superior. However, as the number of levels increases, CS begans to degenerate. This is because CS requires the data to be returned along the path at which the request is sent. For BPS and BPR, the answers are returned directly back to the query node.

Comparing BPR and BPS, it is clear that BPR outperforms BPS by virtue of the fact that BPR is able to reconfigure itself resulting in a more optimal network structure. BPS, on the other hand, must always pass through the same set of nodes regardless of their service quality.

On Line Topology

The results on Line topology (see Figure 5(c)) show similar behavior to that of the Star structure. Essentially, the various schemes have the same relative performance for the same results as that for Tree topology, i.e., BPR is the best and BPR outperforms CS for most cases (except when

[3]There are two possible implementations for CS. In the first implementation, a server who acts as a client will consolidate all answers from its servers before returning the answers to its clients. In the second implementation, a server acting as a client will return any answers that its servers may return through it immediately. We adopted the second implementation in this work.

the number of nodes is very small).

4.4 On Initial Response Time

In this experiment, we evaluate the performance of BPR, BPS and CS on the rate at which answers are returned. The number of nodes is fixed at 32, and we used the Tree topology. A search query is issued four times, and the average time at which nodes respond are noted. Figure 6 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, BPR is still the best scheme, outperforming BPS by virtue of its ability to reconfigure the network. It is able to reach out to more promising nodes directly - after each query, BPR 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 BPR/BPS - as it only returns answers along the path that the query has been transmitted.

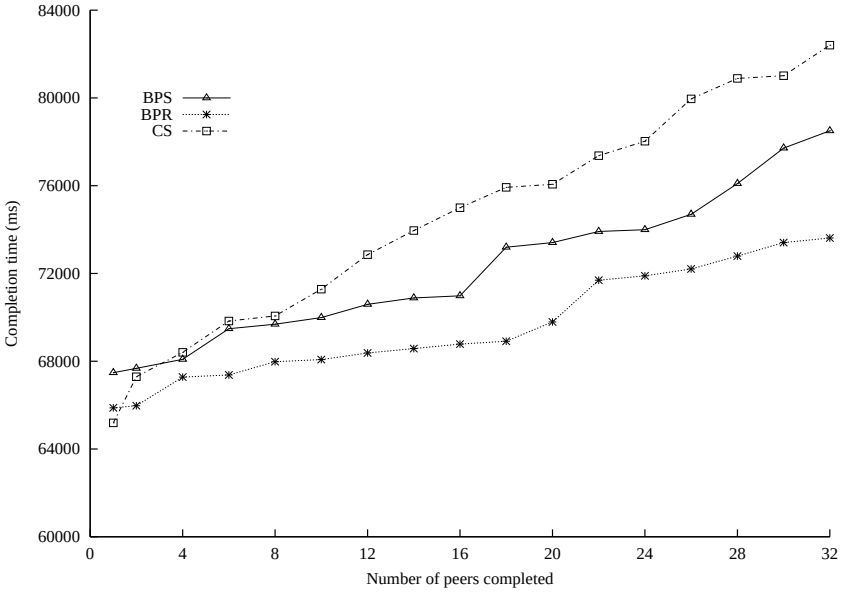
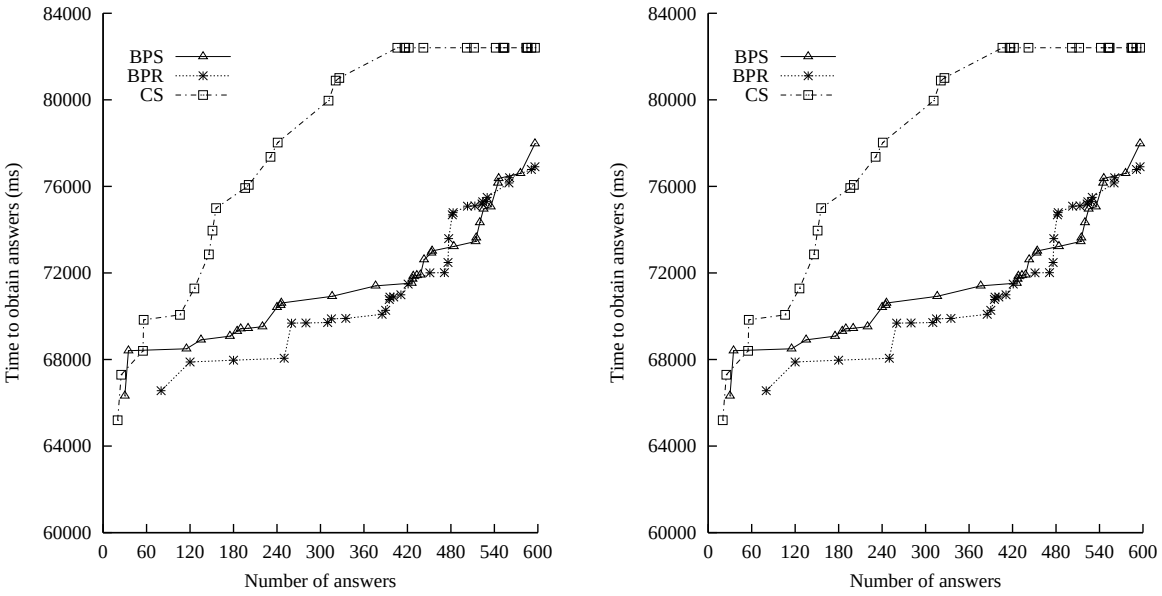


Figure 6: Rate at which answers are returned.

4.5 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 7

shows a plot of the result. As shown, it is clear that CS returns the first few answers much faster than BPS and BPR. This is expected since the first few directly connected nodes that receive the query can return their answers immediately. For BPS/BPR, the overhead of the code-shipping strategy results in a longer initial response time performance. However, as more answers are returned, BPS/BPR are superior over CS, demonstrating the superiority of P2P technologies over traditional CS models. We also note that BPR is generally better than BPS.



(a) First search query.

(b) Last search query.

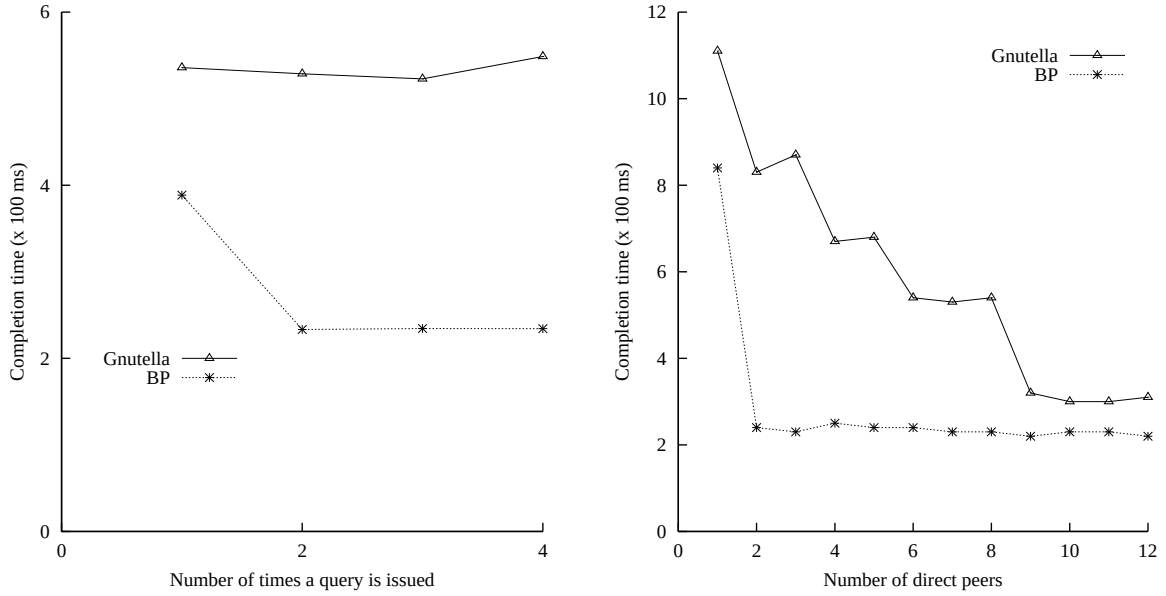
Figure 7: Number of answers returned.

4.6 Comparison of BestPeer and Gnutella

FURI [1] 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 BPR (denoted BP here). We note that Gnutella essentially adopts a similar approach as BPS, 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. In this experiment, each node has 1000 sharable text files (since the source we obtained from from [1] can only evaluate keyword search on text files). We also restrict the answers to come from only a few nodes. The completion time is thus determined by the time when all the answers arrived. We repeated a single search query four times during an experiment, and several experiments were conducted to obtain an average result. Figure 8 shows the results of the experiments.

In Figure 8(a), each node has up to 8 directly connected peers, and show the results for each run of a query. We observe that Gnutella is essentially not affected by the number of times the query is run since it employs the same search path each time. On the other hand, we find that for BP, the completion time fo the first search is much higher the other searches for the same query. This is because for the first search, BP also need to route through the entire intermediate peers before reaching nodes with the answers. For subsequent searches, BP's reconfiguration feature ensures that it can directly connect to these nodes with answers. As such, for subsequent searches, the response time is significantly reduced. We also observe that BP outperforms Gnutella in all runs. This is because, in this experiment, we do not return the data files as output as Gnutella will not return results directly - it simply sends the list of files that matches the query. Therefore, while BP and Gnutella return results out-of-network, this feature is not used in the experiment. In addition, Gnutella requires messages to be sent to the peers along the path of the query traversal, i.e., the list of files have to be transmitted through the query traversal path! On the other hand, under BP, nodes with matching files will send the information direclty back to the initiating node.

From Figure 8(b), we see the effect of the number of peers over 4 queries each time. As the number of directly connected peers increases, BP remains superior. While Gnutella's performance also improves with more peers, traversing the same path each time and returning answers along the query path lead to its poorer performance.



(a) Number of peers = 8.

(b) Effect of number of peers.

Figure 8: BestPeer vs Gnutella.

5 Related Works

The Concordia platform [13, 3] developed by Mitsubishi Electric provides support for Java-based mobile agents. Agent's mobility is achieved via Java's serialization and class loading mechanisms. Each agent object is associated with a separate Itinerary object, which specifies the agent's migration path (using DNS hostnames) and the methods to be executed at each host. In [9], the Aglets environment allowed the creation of a group of agents that can work cooperatively to solve a complex task. In [10], Ajanta, a Java-based system for support agent mobility was developed which also make use of Java's serialization for state capture. Agent-code is loaded on demand, from an agent-specified server.

The above agent-based technology provides 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, there are already many P2P systems [4, 5, 14, 8, 15, 12]. However, these systems cannot be easily extended to meet users changing needs.

More recently, the database community has begun to exploit P2P technologies for database applications [7, 16]. In [7], data placement issues were addressed. In [16], 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.

6 Conclusion

In this paper, we have presented a P2P system called BestPeer that can be used to support a wide range of applications. BestPeer has several nice features. First, because it integrates agent and P2P technologies, it provides easy extensibility to existing systems. Second, it provides a mechanism to reconfigure a nodes' peers based on some optimization criterion. Third, it supports distributed LIGLO servers to maintain some crucial information of BestPeer participants. Our extensive experimental studies show that BestPeer is a promising system for distributed processing. We plan to extend this work in several directions. First, our current implementation provides no optimization schemes - basically, a node will always send its agent to the destination node to process the data there. 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 plan to further study how placement of data and replication can be exploited to improve performance. Finally, the issue of finding a consistent data set has to be addressed.

Acknowledgements

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

References

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

[2] S. Bressan, C.L. Goh, B.C. Ooi, and K.L. Tan. Supporting extensible buffer replacement strategies in database systems. In *SIGMOD 1999*, 1999.

[3] 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.

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

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

[6] 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.

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

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

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

[10] 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.

[11] D. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets.* Addison-Wesley, 1998.

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

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

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

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

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