Evaluation of Peer-to-Peer Database Solutions

By: W Anthony Young
20161423
Date: July 30th, 2004
For: Prof. T. Ozsu
Subject: CS 654 - Spring 2004

1) Introduction

Database systems have been in use for decades as a means to store information. Large organizations store employee, customer and product information. Healthcare professionals store patient information. Researchers keep data on research projects. No matter where they are used, databases have the power to organize our information and help us access it efficiently.

Distributed databases have also been in use for many years. They allow for local autonomy, improved query performance, a high level of expandability, improved reliability, easy data sharing, and improved availability of access [14]. Distributed databases provide an organization the flexibility to tune storage and access protocols to suit their infrastructure. For example, data that is frequently used can be replicated to reduce response time. As well, data can be spread across many servers to increase redundancy.

Recently, peer-to-peer architecture has been applied to database systems. This architecture allows a system to act as both a client for performing queries and interacting with the user, and as a server to provide results to queries posed by other clients on the network [13].

1.1) Fundamental Differences

Peer-to-peer databases are fundamentally different from distributed and traditional databases in several ways. One of these fundamental differences is that nodes may join and leave a peer-to-peer network at any time. In distributed and traditional databases, nodes (peers) are added out of necessity (i.e. for redundancy or growth) and are known to

the cluster ahead of time [12]. However, a peer may join an information-sharing network at any time, and should only have its resources added to the resource pool when it comes online. Also, peer nodes are not necessarily known ahead of time (except in special cases of security restricted networks). This "arbitrary join" paradigm for peer systems poses challenges to finding data with queries, providing complete data through queries, and locating peers on a network. Take the following query as an example:

```
SELECT Min(Price)
FROM Products
WHERE ProductName = 'Apple PowerMac G5 Dual 2GHz'
```

With this query, the user is looking for the lowest price for a product. If peer node n_1 contains a cheaper product than peer node n_2, and n_1 is unavailable, the user will receive an incorrect result from their query. Also, how does the node performing the query know to which IP addresses the SQL statement should be sent in order to reach other peers?

A second fundamental difference between peer-to-peer databases and distributed and traditional databases is that the schema for a peer-to-peer database is not global. In traditional and distributed databases, the schema is standardized across each node. In a peer-to-peer database network, several schemas may be used to represent the same type of data at different nodes [12]. For example, imagine a peer network with two nodes each containing a database of contact information. Node n_3 may store contact information in a schema such as the following:

```
TABLE: Contacts {
        LName VARCHAR 25;
        FName VARCHAR 25;
        Phone INT;
        Email VARCHAR 50;
        Address VARCHAR 100;
}
```

Also, node n_4 may store contact information in a schema such as the following:

```
TABLE: contact_info {
        Lastname TEXT 15;
        Firstname TEXT 15;
        Phonenumber TEXT 10;
        Emailaddress TEXT 40;
        Homeaddress TEXT 50;
}
```

This "arbitrary schema" paradigm for peer systems poses problems for query completeness. For example, n_3 might be searching for contact information and receive a false negative when querying n_4 with a statement such as:

```
SELECT *
FROM Contacts
WHERE FName = 'John' AND LName = 'Doe'
```

In this case, n_4 might actually contain a record for John Doe. But, because records are stored in a different schema, the query will not return it.

A third fundamental difference is that the set of available data in a peer-to-peer network might not be complete. Distributed and traditional databases contain a complete set of data in each server cluster. However, a cluster of peers might not have the complete set of data required to accurately and completely answer a query [12]. This is because a node containing some information required by a query might be offline. Obviously, this query will not be able to return proper and complete results because of missing information. This "missing data" paradigm for peer systems poses problems for query completeness and correctness. The former, completeness, is obvious. If data is offline it will not be found and a complete result set will not be returned. The latter, correctness, is somewhat more difficult to understand. An incorrect result would be attributed to incomplete data providing a false result to a query. For example, take the SQL statement:

```
SELECT Count(DISTINCT Address)
FROM Customer
```

If a peer containing part of the Customer table is offline, the user will not receive the correct number of distinct addresses.

A fourth fundamental difference is that queries in peer-to-peer databases must be routed to many nodes. In distributed databases, a query can be routed to a relatively small set of nodes. In peer-to-peer databases, the query must be passed to many nodes in order to return an accurate result set [12]. This "query routing" paradigm for peer systems poses problems for locating nodes, routing queries, retrieving results, and doing all this in an efficient manner (nb: efficiency in this context refers to low consumption of network resources). For example, a peer system may contain upwards of 10 000 nodes. It is not feasible for the peer performing the query to store the IP address of each other node in the network, contact each node individually and submit its query, and wait for results from all 10 000 nodes before continuing. Such a system would require mass amounts of space at each node to store IP addresses, consume mass amounts of network bandwidth when submitting queries and retrieving result sets, and experience a long response time.

1.2) Applications

Peer database systems have many different applications. One application of peer systems is to special interest communities. The internet holds many special interest groups that share information, have meetings, etc. It would be advantageous for such groups to have individual communities that could be used to share files, host chats, etc. [9]. This type of system would distribute information across many nodes and require some sort of efficient search mechanism to find data. A peer network would be ideally

suited for this purpose: information would not need to be centrally located, and users would be able to find information spread across multiple nodes with relative ease.

Finding contact information is another application of peer systems. Frequently, people do not have a phone number or an email address for a person or business they are trying to contact. In such instances, a peer database could be queried to retrieve the required information. Each person could store their own vCard, and the vCards of those people they have already searched for. Locating contact information would be faster as users would not have to look through old emails or a hierarchy of web pages in order to find the information they need. Also, caching would speed up subsequent queries for the same data.

A third application of peer systems is development environment configuration management. When engineers work together to develop a piece of software, they must adhere to standards in order to ensure that everyone working on the project can follow the code and documentation. In programming environments, it can be particularly challenging to ensure that everyone uses the same conventions for variable naming, file structure, etc [5]. Such information could be stored in a searchable database. Due to the scale of some projects, it might also be necessary to distribute templates, project documents, etc. to many different people. Instead of sending this information via email, and in order to ensure that each person working on the project has the most up-to-date copy, a query could be performed on a peer database system to allow each employee to download the information they require.

Transfer of patient records is another application of peer database systems. Hospitals and doctor's offices must share information about patients: their medical

history, contact information, etc. [12]. This sharing usually requires faxing or mailing large amounts of information. This can sometimes take days to accomplish. If, for example, a patient has been taken to an emergency room because of complications resulting from surgery they had at a hospital 1 000 kilometers away, it is imperative that information take minutes rather than days to arrive. In these instances, it would be ideal to search hospital and doctor's office databases remotely to retrieve a patient's information.

Another application of peer systems is storing banking information. Customers often hold accounts at one single bank branch. Such a branch services the client on a regular basis, but records are often stored elsewhere on a central server. This results in the branch needing to connect to the central server every time a transaction needs to be performed on the client's account. In this situation, a peer system would speed up access to the client's records by distributing them across individual branches. Queries could then be performed over all branches in order to find account information for a client who goes to a different branch, an ATM, or another bank.

Another application of peer systems is distributed file storage. Distributing office documents across employee computers is usually done as a consequence of work style rather than as a conscious decision. However, this distribution paradigm is actually quite efficient for storage. Adding a simple searching mechanism would allow employees to access files stored on each other's computers without the need for a large file server, or for file requests to be made by email, phone, etc. [9]. Of course, security protocols would be a required addition to such a system.

Criminology is the final application of peer systems that will be discussed here. Law enforcement officials around the world keep DNA, fingerprints, criminal records, etc. in nationwide central databases for easy access; however, this creates a bottleneck at the server and the server is a single point of failure. As such, it would be advantageous to distribute this data across multiple nodes. For example, the agency that collected the evidence or arrested the individual could be responsible for holding records, updating them as new information is discovered, etc. Using a peer system, officials from around the world would be able to search this information, and cache it for frequent use and to speed up searching by other agencies.

1.3) Strengths

There are several strengths associated with peer systems. Some of the greatest are:

- No single point of failure: In a traditional or distributed database environment, if network links or server machines crash, the entire system can be unavailable. Distributed databases attempt to solve this problem by replicating or partitioning data. Although replicated server clusters may survive crashes, traditional and partitioned server crashes will result in a loss of data availability. Peer systems overcome this problem by keeping data highly distributed using local caching. In this case, if a peer crashes, queries can still be processed. Also, it is highly unlikely that enough network links would fail at one time to cause a significant number of peers to become inaccessible.
- Minimal administration: In a traditional database environment, a database administrator must keep a watchful eye on server performance and load, and

design queries to be run by users. The level of administration increases to include replication and partitioning concerns as well as interconnection performance in distributed database environments. Peer systems do not require high amounts of overhead. Users create their own databases, store their own information, perform their own queries, and connect their nodes with the click of a button. Minimal administration is required at the bootstrap nodes in order to keep registration and discovery of peers fast and efficient.

- Vast amounts of data: Peer databases provide users with access to a very large amount of data of many different types. Also, users can search for many different data types at once. Traditional and distributed database systems often store single types of information (i.e. information regarding customers and products). Users must then query many different servers in order to get the information that they require.
- Replicated data for fast retrieval: In peer systems, data is replicated over many nodes as queries are performed and results are cached locally. This allows access to locally cached copies of information if the original node is busy or unavailable. In traditional, and to a lesser extent distributed, database systems, queries will run slower as the load on the database server increases. Also, data will not be obtainable if the server is too busy or is unavailable.

1.4) Weaknesses

There are several weaknesses associated with peer systems. Some of greatest are:

- Discovery of peers: In traditional and distributed database systems, all cluster nodes are known ahead of time. This makes performing queries very easy, as applications know exactly what server(s) to contact. In peer systems, nodes may join and leave the cluster at any time. As such, nodes never have a complete picture of the peer network. It is thus necessary to "discover" other peers on the network before performing queries.
- Query routing: In peer systems, queries must be routed to all or large subsets of nodes in order to find peers that have the information a user is interested in. Implementing efficient routing algorithms that return complete results is difficult. In traditional and distributed database systems, queries are routed to a single or a small number of servers for processing. This is because all the required information is stored in a single or small number of places.
- Consumption of network resources: Because peer systems must route queries to a large number of nodes, they often consume a large amount of network resources. This is in sharp contrast to traditional and distributed database systems in which often only one request and one reply is transmitted.
- Mobility of users: In traditional and distributed database systems, servers are usually given static IP addresses. Applications and users then contact servers at those addresses to perform queries. In peer systems, nodes are usually assigned dynamic IP addresses as they log onto a network. Because of this, mapping tables must be maintained by bootstrap nodes or neighbours and read before queries can be routed.

2) Key Issues

There are several key issues that must be taken into account when designing a peer database system. This paper will present five of the key issues that need to be addressed and evaluate different peer systems based on them.

2.1) Scalability

Scalability refers to how well a system performs as the amount of load on it increases. A peer system must be able to handle an ever-increasing community of users [7]. If a system performs adequately with 1 000 users, it should also perform adequately with 100 000 users. As the size of the community increases, so does the consumption of network bandwidth, the number of messages passed between peers, and the amount of information that a bootstrap node or central server must keep track of. Ideally, a peer system will be able to scale in a less than linear fashion. Otherwise, an increase in size has the potential to choke the network a user community runs on top of, and overload the bootstrap nodes or central server attempting to hold the community together.

2.2) Availability

Nodes should be able to communicate with, and receive data from, each other. As well, data should be replicated at, and retrievable from, several sources [4]. We therefore define availability in the context of this paper to mean the ability to find and retrieve resources using a peer system. The importance of availability should be obvious. If a peer system does not have a way to find or retrieve resources, it does not serve its purpose. In

an ideal system, if user A has a resource and user B caches that resource for later use, user C should be able to obtain that resource from user A or B.

2.3) Performance

The network should return results with the smallest latency possible. As well, communication between nodes should be as efficient as possible [8]. Efficiency in the context of this paper refers to a low consumption of bandwidth and as few messages being passed between nodes as possible. Performance is critical to any peer system because users of such a system expect to have fast access to resources. Also, the usefulness of a system often decreases as performance degrades. As such, for a peer system to remain useful, it must have adequate performance.

2.4) Data Authenticity

Data authenticity is concerned with determining whether or not responses to a query are factual [4]. As such, a node should be able to tell the difference between a correct and an incorrect query hit. For example, if a user is searching for contact information for John Doe, how does the user know that the information returned is correct? Furthermore, if two different sets of information are returned, both purporting to be for John Doe, how does the user determine which set of information is correct? Ideally, a peer system would avoid returning incorrect results, or at least provide the user with a mechanism to aid in determining which information is factual.

2.5) Security

A peer system must ensure that authorized users are the only ones who make use of privileged data [3]. For example, in a system that shares patient records between doctor's offices and hospitals, only doctors, nurses, and their respective staff members should be able to access private information. As well, information should be encrypted during transport and while in storage to ensure that it cannot be stolen. In the context of this paper, we define security to mean that only authenticated users with proper privileges can access privileged system resources.

3) Proposed Peer-to-Peer Database Systems

Several different peer database systems have been proposed in recent years. However, very little critical evaluation of such systems has been performed. In this section, four recently proposed peer database systems are presented and evaluated according to the five key issues outlined in section two.

3.1) APPOINT

APPOINT is an acronym for "Approach for Peer-to-Peer Offloading the INTernet." [2] It is a system that seeks to provide access to large amounts of spatial data in a fast and efficient manner. Users may share or obtain entire spatial data sets over the internet using the SAND (Spatial And Non-spatial Data) browser. SAND is a java-based browser that allows a user to visualize different spatial data sets that are obtained from APPOINT. When users first contact the central APPOINT server, they are asked whether or not they will share their files, extra disk space, and bandwidth in a peer-to-peer

manner. If so, clients can be called upon by the central server to function as servers to other peers. If not, clients operate under the client-server paradigm.

APPOINT is a central server system and functions under a traditional client-server paradigm when system load is low. Users making requests to the system can download files directly from the server. The central server keeps track of what data sets a user has downloaded, as well as whether or not the user is online. When system load starts to increase, APPOINT defers download requests for data sets to peers in an attempt to optimize performance. The requesting user then downloads the file from a peer who has already cached it and the performance of the central APPOINT server remains acceptable (nb: all queries are routed through the central server regardless of whether they will be deferred to peers or not).

Uploading data under APPOINT functions in a manner similar to downloading. When a peer wishes to share a data set and system load is low, the data set will be uploaded to the central server and made available for download. If system load is high, data sets are uploaded to other peers and eventually propagated to the central server when the server load decreases.

3.1.1) Scalability

The central APPOINT server must keep state on all system users. This may not take a significant amount of space on a per-user basis, but when there are many peers using the system, the amount of data being tracked may overwhelm the server. In this case, the amount of information stored will grow proportional to the number of users, negatively affecting scalability.

The central APPOINT server hosts all data sets that may be downloaded. According to F. Brabec, H. Samet, and E. Tanin, authors of "Remote Access to Large Spatial Databases", spatial data sets usually take up a large amount of disk space. Also, as the number of users grows, the number of data sets to be stored could grow in an exponential manner, negatively affecting scalability.

3.1.2) Availability

The use of file caching by peers increases file distribution. As such, peers do not always have to download from the central server. This increases availability of files and positively affects availability. However, due to the use of a central server, APPOINT suffers greatly from the single point of failure problem. Should the central server be inaccessible for any reason, users will not be able to access any resources on the network. As such, overall availability of the APPOINT system is poor.

3.1.3) Performance

Since the central APPOINT server is used to fulfill queries or route them to nodes with cached copies of data, a very small number of messages must be transferred between client and server. In the case that the central server handles the query, the client will only receive one reply to their one request. In the case that a peer handles the query, the client should have to send a request and receive a reply from the server as well as from the peer. Therefore, the greatest number of messages required for a query should be two requests and two replies. For this reason, APPOINT experiences very good performance due

directly to the low number of messages passed. But, the amount of bandwidth consumed by transferring the actual query result is quite large. This negatively affects performance.

3.1.4) Data Authenticity

Data is uploaded to the central APPOINT server and downloaded by peers. Peers upload their information by propagating it to the server. Using this scheme, it is possible for malicious users to upload a falsely named data set, or a data set containing incorrect data. Further, APPOINT does not have any mechanism for validating the data that is uploaded or downloaded by peers. This makes authenticating data very difficult. As such, APPOINT has poor data authenticity characteristics.

3.1.5) Security

Since APPOINT does not require any security mechanism (i.e. it is a freeware system), any user may download any file. For the current free distribution of information that the system is being used for, this does not pose a problem. However, should APPOINT be employed on a network with sensitive or private data, it would require security measures to be built in. As such, in environments that require security, APPOINT would not be able to deliver.

3.1.6) Overall

Overall, APPOINT scales well and experiences decent performance. However, APPOINT requires some work to provide data authenticity and security schemes for its

users. As well, availability suffers greatly from the single point of failure problem. This problem is by far the single greatest failing of APPOINT.

3.2) DBGlobe

DBGlobe is a knowledge management system that forms dynamic information "communities" of nodes (also called "Primary Mobile Objects" or PMOs) that store information of a common type [1]. Information at a PMO is represented as a service. Thus, PMOs may publish several services of information to the bootstrap servers so that user agents may retrieve and use them. PMOs may also cache services that they make use of in order to publish them for others. Conceptually, each node in the DBGlobe network contains a data store (PMO) that stores and provides access to information, and an application that registers information with the bootstrap node and performs queries.

DBGlobe makes use of a number of "geographically" distributed Cell Administration Servers (CAS) (nb: it is unclear from the literature whether geographical area refers to physical location or network location such as IP address or domain). These bootstrap servers are responsible for publishing services to PMOs for retrieval and to other CAS's to aid in query routing. A CAS also registers and holds metadata for the PMOs that fall within its assigned geographical area. CAS's receive and route all queries from PMOs to appropriate PMOs and CAS's that should be able to provide meaningful results. Since the CAS's store metadata for each PMO, this query routing is easy.

As mentioned above, DBGlobe tracks information communities. This is done through the use of Community Administration Servers (CoAS's). A CoAS keeps track of each PMO that contains information related to the theme of its assigned community. For

example, a "history community" may be assigned to $CoAS_2$. As well as being tracked by its geographically defined CAS, a PMO that stores historical information would also be tracked by $CoAS_2$. When a PMO is looking for information about historical events, its query can then be routed to $CoAS_2$ to receive faster and more accurate results. Conceptually, a CoAS is identical to a CAS except that it tracks PMOs in information communities instead of geographic regions. Communities can be created statically by users, or dynamically by the CAS's analysis of commonalities in stored metadata.

DBGlobe makes use of filters to route queries to the appropriate neighbouring sites (i.e. PMOs, CAS's, and CoAS's) in a network. Each site maintains a "local" filter that summarizes all the services it provides, as well as a set of "merged" filters that summarize the services offered by its neighbours. Once a site receives a query (either from a PMO or from a neighbour), the query can be routed to nodes containing relevant data for further processing.

3.2.1) Scalability

Since PMO's are distributed amongst CAS's on the network, the amount of user load on a CAS can be expected, on average, to increase less than linear to the number of users of the system. However, in the worst case, all PMOs will be from the same geographic region, and will thus connect to the same CAS, causing the system to scale linearly. As such, DBGlobe experiences good scalability of user load on average.

Similarly to APPOINT, the CAS's must store metadata about other nodes in the network, and as such, the space required for metadata storage could grow at an

exponential rate. However, since metadata does not require large amounts of disk space, scalability of space is still decent.

3.2.2) Availability

In DBGlobe, query results are cached at the machine performing the original query. This allows future queries by other peers to retrieve these cached results, positively affecting availability. However, peers are required to connect to a geographically defined CAS. Should this CAS be unreachable for any reason, the system may be inaccessible to some peers. This negatively affects availability. But, DBGlobe experiences better availability than APPOINT as there are several CAS servers, each servicing part of the network. Therefore, DBGlobe does not suffer from the same single point of failure problem that APPOINT suffers from, and experiences fairly good availability.

3.2.3) Performance

Since the CAS's handle query routing to appropriate sites, a small number of messages are passed around the network in order to perform a query. However, since the number of messages passed depends on the distribution of data relevant to a particular query, it is possible that $O(2(n-1)^m)$ messages will be sent across the network for each query (where n is the number of PMOs and m is the number of CAS's and CoAS's). This is because each request could, in theory, be routed to each CAS in the system and in turn forwarded to each PMO. Then, replies would be sent back to the querying PMO. However, performance should be fairly good on average.

The amount of bandwidth consumed by DBGlobe should be quite small. When querying databases, unless objects are transferred as part of a request or reply, only a small amount of data should be sent, keeping bandwidth low.

In contrast, DBGlobe experiences many more messages being passed than APPOINT, but should experience much less bandwidth consumption due to the relatively small size of query result sets.

3.2.4) Data Authenticity

DBGlobe provides no means with which to authenticate information. So, for any returned query results, users will not know whether the results are factual. However, the use of CoAS's does mean that queries can be routed to PMOs that claim to have information related to a specific subject. But, no attempt is made to verify that the purporting PMO actually holds that type of information. With this scheme, DBGlobe does experience better data authenticity than APPOINT. But, as with APPOINT, there is still the possibility of malicious users providing false results.

3.2.5) Security

As with APPOINT, DBGlobe does not provide any authentication or encryption mechanisms to its users. As such, any user may make use of any piece of information in the system. This is very poor security, but is still within the bounds of the current use of the system. As with APPOINT, security measures would need to be added should DBGlobe be used in a problem domain with sensitive data.

3.2.6) Overall

DBGlobe experiences better scalability of load and disk space than APPOINT. This is mainly due to the distribution of user load amongst several CAS's as apposed to one central server. This distribution aids in the high data availability of the DBGlobe system. There is no single point of query, and thus, there is no single point of failure (i.e. queries can still be routed if a few of the CAS's are unavailable). Therefore, data is available with much greater frequency.

DBGlobe experiences fair performance in number of messages passed during queries. This is mainly due to the "flooding" method in which queries are performed. However, it can be expected that in the average case, DBGlobe will consume less bandwidth than APPOINT, making its performance good.

DBGlobe experiences poor data authenticity, but does gain ground over APPOINT due to the use of the CoAS's. This increases the likelihood that returned results are relevant to the query. However, security of data is minimal and, as with APPOINT, quite poor.

3.3) Edutella

Edutella makes use of a "super peer" organization strategy for connecting peers to each other [10]. Nodes wishing to join the network are either defined as peers or super peers. Super peers organize themselves into a hypercube topology with edge degrees defining a neighbouring super peer as degree 0, 1, etc. Peers then connect, using a clustering algorithm, to the super peer that will provide them with the best performance.

Thus, peers are organized in a star topology around their super peer, and route all queries to the super peer.

Super peers maintain two types of indices: “Super Peer / Peer” (SP/P) and “Super Peer / Super Peer” (SP/SP). The function of the two types of indices is the same: to hold metadata about peers connected to the network in order to route queries to only those neighbours who can provide results. However, the information stored in the two types of indices is what sets them apart. SP/P indices hold metadata for each peer that is connected to a specific super peer. Thus, each super peer has a different SP/P index. Incoming queries are then routed to the peers that have relevant data to contribute to the query. Data can then be returned to the requesting peer through the connection of super peers. SP/SP indices hold metadata for each peer cluster. So, each super peer would report to its neighbours what information its peers hold. Metadata would propagate around a hypercube similar in fashion to the way routing information propagates in the distance vector routing algorithm: all SP/SP indices would converge to a steady state over time. Each super peer would then know where to route a query in order to get it to a peer that has appropriate results, using other super peers as intermediate routers.

Peers in the Edutella network route queries to their super peer who in turn forwards them through the network according to the edge degree system discussed above. When a query request is routed, the edge degree the message was forwarded on is included in the message. Then, when a super peer receives a query request, it only routes it to those relevant super peer neighbours that have a degree higher than the degree from which the query originated.

In the Edutella network, peers may be clustered to super peers in three ways: using “ontology clustering”, “rule-based clustering”, or “query clustering”. With ontology clustering, peers are assigned to clusters according to the data that they are “interested” in. For example, all peers that are interested in holding or searching for DNA data would be clustered into one group. With rule-based clustering, peers are clustered according to a certain set of rules. For example, peers could be clustered according to domain, IP prefix, average response time, average throughput, etc. With query clustering, peers are grouped according to the frequency with which they have performed queries on certain types of data. For example, the number of times they have searched for a person’s name, a product price, or a fingerprint. This requires peers to store statistics about the type of information they search for

3.3.1) Scalability

Edutella makes use of the super peer paradigm, which allows peers and super peers to self-regulate their organization. Load is balanced using one of several clustering algorithms in order to make sure that no super peer has more peers connected to it than it can handle. Thus, scalability of peers is good. Further, scalability of peers is better than that of DBGlobe as a more robust algorithm than simple geographical clustering is used to distribute peers.

However, Edutella makes use of two centralized indices that must be stored at the super peer level. The SP/P indices could grow at an exponential rate as with metadata storage in DBGlobe and APPOINT. As well, the SP/SP indices could grow quite

exponentially as more and more data is added to each super peer, requiring it to be propagated around the hypercube.

3.3.2) Availability

Similar to DBGlobe and APPOINT, all peers and super peers have the ability to cache results. As such, data is available to the entire network even if the originating peer goes offline. Once results are cached, the SP/P and SP/SP indices are updated to reflect that a peer holds new information.

As with DBGlobe, super peers are decentralized. This means that there is no single point of failure in Edutella as there is in APPOINT. Further, since the clustering algorithm employed by Edutella is more robust than that employed by DBGlobe, if a super peer goes down, peers may be absorbed into a different cluster and will not have to wait for their super peer to come back online to use the network. Thus, availability in Edutella is quite good.

3.3.3) Performance

Edutella routes queries according to edge degrees. This means that a query can effectively be routed to all relevant nodes in the topology using at most N-1 messages (where N is the number of nodes in the network). This minimizes the request messages that must be sent. Also, all nodes can be reached after a maximum of $\log_2 N$ message forwarding steps. This means that super peers can aggregate the results to the query and return it as one large result to the requesting peer. Thus, at most another N-1 message is required to return a result to the querying peer. This is an improvement over DBGlobe's

query flooding strategy. It is still, however, a far cry from APPOINT's single request and single reply strategy.

Bandwidth consumption under Edutella can be expected to be relatively low in the average case for the same reasons as DBGlobe. The small amount of data to be returned by a query would mean that only a small amount of bandwidth would be required. Overall, performance is good.

3.3.4) Data Authenticity

As with DBGlobe and APPOINT, Edutella does not provide any means to authenticate that the data being returned to a peer is valid. In fact, unless the ontology clustering algorithm is used, Edutella's authenticity scheme is just as poor as the scheme employed in APPOINT. If the ontology clustering algorithm is used, Edutella can expect to see authenticity mechanisms only as strong as DBGlobe, and there is still much room for improvement.

3.3.5) Security

As with DBGlobe and APPOINT, Edutella does not provide any means to secure data on a network. Edutella's current implementation aims to support the free distribution of information, meaning that it does not require security services. However, employing Edutella on a network requiring security would mean it would not provide a suitable solution.

3.3.6) Overall

Edutella experiences relatively good scale of users, but does experience exponential growth in the size of its stored indices. Comparatively, Edutella scales better in users than both DBGlobe and APPOINT, but worse in space than DBGlobe. As well, Edutella slightly improves data availability over DBGlobe in that a more robust clustering algorithm means that peers can use the network even if their previously assigned super peer goes offline.

Edutella experiences a reduced number of messages transmitted over DBGlobe due to its edge degree message forwarding. This reduces the number of messages required to be sent to O(2[n-1]) as opposed to $O(2[n-1]^m)$ with DBGlobe. Bandwidth consumption is comparable to that of DBGlobe.

Edutella suffers from the same lack of security mechanisms that both APPOINT and DBGlobe suffer from. As well, without the use of ontology clustering, data authenticity is reduced to be as poor as with APPOINT. However, with ontology clustering, data authenticity is as good as DBGlobe.

3.4) PeerDB

PeerDB is a robust data management solution that supports data sharing through agent based communication with other peers [7, 8]. In a PeerDB network, each node has a DBMS, local and export dictionaries, and an agent. The DBMS is responsible for storing local data that the user has entered through the user interface. It also stores cached results of queries that have been performed by the agents. The local dictionary stores metadata and keyword descriptions for all locally stored data. The export dictionary stores metadata and keyword descriptions for all shared data, and is searchable by remote

agents. Agents perform both local queries on the local dictionary and remote queries on peers export dictionaries.

PeerDB is built on top of the BestPeer architecture [11]. Using BestPeer, peers are organized into a mesh based upon neighbour assignments from a Location Independent Global Names Lookup (LIGLO) server. A peer initially contacts the LIGLO server and is assigned a unique identifier and a set of neighbour peers to communicate directly with. LIGLO servers store the IP address of the peer, its assigned neighbours, and its unique ID. In this manner, the server can keep track of peer status. When a user logs in again, the LIGLO server knows its new IP address and can assign it new neighbours. As well, the LIGLO server can assign the peer as a neighbour to others. When queries are performed, peers are able to make remote nodes neighbours if they are found to be providing many useful query hits. This self-reconfiguration of neighbours allows peers to communicate faster with those nodes that are providing relevant results to its queries. This reduces response time for relevant results, and allows the forming of information “meshes” similar in nature to Edutella and DBGlobe communities.

When a user creates a database schema in PeerDB, they assign keywords to it to describe the data they are storing [7, 8]. This schema is then searchable by keyword in order to avoid the problems associated with the arbitrary schema paradigm discussed above. In order to perform queries on neighbour export dictionaries, peers make use of agents. Agents are dispatched to each neighbour of a peer when a query is issued. The agents will query the remote export dictionary by keyword and return any results to the user (nb: no data is actually returned at this point, only descriptions/keywords). Each remote query has a time-to-live counter associated with it. If the counter is not zero when

it reaches a neighbour, the counter is decremented and the agent is cloned and dispatched to all directly connected peers of the neighbour. Replies to queries are returned directly by the agent to the calling peer. Replies from the agents are then ranked for the user according to agent-determined relevance. The user then selects the data they wish to have returned and the agent fetches the full data set.

3.4.1) Scalability

There are several LIGLO servers distributed around a PeerDB network. Peers may contact any one of these servers in order to register themselves when they start up. The LIGLO servers must store minimal information about each peer (i.e. its status, IP address and unique ID), and do not have to store any related metadata. As such, this system scales well both in users and space. PeerDB scales similar in users and better in space than APPOINT, DBGlobe and Edutella (nb: LIGLO servers are analogous to super peers and CAS's).

One drawback of using LIGLOs to store information related to a peer is that data be must replicated across each server. Otherwise, peers would only be able to contact one LIGLO server each time they log in, and this would cause a single point of failure problem. However, it is a trivial operation to replicate user information across the servers. This replication process is not discussed in detail in the literature, but the author of this paper assumes that some messaging overhead and bandwidth consumption would be necessary to keep information current across all LIGLOs.

3.4.2) Availability

Since PeerDB distributes the LIGLO servers across the network, and peer registration information is replicated at each site, the system should be available to all peers even if many of the LIGLO servers become unreachable. This is because the peers are all capable of contacting any of the servers for registration and gathering neighbours. As well, cached data can be exported to other querying peers when the originating node is offline. As such, PeerDB experiences availability similar to DBGlobe and Edutella.

3.4.3) Performance

PeerDB makes use of a time-to-live (TTL) counter on all queries. This value allows remote nodes to further distribute the query request to each of its neighbours, and so on. This means that a maximum of $O(2(n^{TTL}))$ messages are passed through the network for each query (where n is the maximum number of neighbours at any peer in the network). This is because one request and one reply message is required for each remote peer, and in the worst case, each peer will have n neighbours. Then, when data is to be requested and returned for a query, an additional $O(2(n^{TTL}))$ messages is required. This is because a user may request all results from each queried neighbour. This number of messages is significantly larger than Edutella, and a query is not necessarily guaranteed to reach each peer in the system. Thus, PeerDB experiences fairly poor performance in number of messages passed.

PeerDB attempts to limit wasted bandwidth by returning only those relevant result sets that the user has chosen. This ensures that a minimal amount of bandwidth is used when transferring data. This may significantly decrease, on average, the amount of bandwidth consumed by PeerDB over that consumed by Edutella, DBGlobe and

APPOINT. However, the user may still request all data sets be transferred, and this would provide bandwidth consumption on par with Edutella and DBGlobe in the worst case.

3.4.4) Data Authenticity

PeerDB attempts to provide some data authenticity through the use of keyword searching, ranking, and user data selection. Instead of treating all returned results as valid, PeerDB presents a list of rankings to allow the user to determine which data is most likely to be relevant to their query. However, there is still no mechanism built in to ensure that the data a peer makes available is factual. But, through the use of this ranking and keyword searching system, much better authenticity mechanisms are in place than all of Edutella, DBGlobe and APPOINT.

3.4.5) Security

PeerDB provides some security for sensitive data by restricting what data may be searched by remote agents. This is done by restricting remote agent queries to the export dictionary. As with Edutella, DBGlobe and APPOINT, there is no mechanism to support searching of sensitive data by authorized users, or data encryption; however, by allowing only certain information to be exportable, PeerDB is one step closer to providing security functionality. As such, PeerDB still experiences relatively poor security support, but its mechanisms are slightly better than the others.

3.4.6) Overall

PeerDB's use of LIGLO servers and lack of indexing and metadata storage at the server level afford it better scalability than each of Edutella, DBGlobe and APPOINT. PeerDB experiences availability on par with the others through its use of result caching, distribution of the LIGLO servers, and redistribution of peers after a failure.

PeerDB experiences poorer performance in terms of number of messages passed than the others, but does improve on average case bandwidth consumption by allowing the user to select which data sets are returned.

PeerDB's data authenticity is superior to Edutella, DBGlobe and APPOINT due to its use of keyword searching and ranking. This helps the user determine which results are most relevant, and speeds up the process of finding useful data. As well, PeerDB makes some strides towards offering a secure information sharing environment with the use of the export dictionary. However, this dictionary does not actually allow authentication of users and encryption of data: it merely supports user-specified global access rights to local information.

4) Concluding Remarks

For the most part, systems that make use of true peer-to-peer technology experience good scalability, availability and performance. The only exception is APPOINT, which suffers greatly from the single point of failure problem whether it is acting under the client-server paradigm or the peer-to-peer paradigm.

Query routing and clustering algorithms have also come a long way through the study of file sharing networks. For these areas, Edutella provides the optimal solution in terms of number of messages sent, and PeerDB provides the optimal solution in terms of

bandwidth consumption. As well, neighbour organization through a number of clustering algorithms is very useful for adapting Edutella to different problem domains.

It appears that much work still needs to be done in the area of data authenticity and data security. None of the four solutions evaluated properly supports either of these two very important functions. However, PeerDB does come closest.

In general, the systems discussed each present a viable solution for their intended problem domain. However, each system has drawbacks that limit its deployment in a setting outside this domain.

References

1. S. Abiteboul, D. Pfoser, E. Pitoura, G. Samaras, and M. Vazirgiannis. DBGlobe: A Service-Oriented P2P System for Global Computing. In ACM SIGMOD Record 32(3), September 2003.
2. F. Brabec, H. Samet, and E. Tanin. Remote Access to Large Spatial Databases. In Proceedings of the Tenth ACM International Symposium on Advances in Geographic Information Systems.
3. M. Ciglaric, and T. Vidmar. Management of peer-to-peer systems. In Proceedings of the Parallel and Distributed Processing Symposium, April 2003.
4. N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In 9th International Conference on Database Theory, January 2003.
5. D. Heimbigner, A. van der Hoek, and A. L. Wolf. A generic, peer-to-peer repository for distributed configuration management. Proceedings of the 18th International Conference on Software Engineering, March 1996.
6. W. Hoschek. A Unified Peer-to-Peer Database Protocol. Proceedings of the International IEEE/ACM Workshop on Grid Computing, November 2002.
7. M. Jovanov. Scalability Issues in Large Peer-to-Peer Networks. Accessed June 19th, 2004 from http://www.ececs.uc.edu/~mjovanov/Research/paper.html.
8. J. Liu, Q. Zhang, X. Zhang, and W. Zhu. Measure: a group-based network performance measurement service for peer-to-peer applications. In Proceedings of the Global Telecommunications Conference, 2002.

9. T. Nakata, H. Sunaga, and M. Takemoto. In Proceedings of the Third International Conference on Peer-to-Peer Computing 2003, September 2003.
10. W. Nejdl, W. Siberski, and M. Sintek. Design Issues and Challenges for RDF- and Schema-Based Peer-to-Peer Systems. In ACM SIGMOD Record 32(3), September 2003.
11. W. S. Ng, B. C. Ooi, and K. L. Tan. BestPeer: A Self-Configurable Peer-to-Peer System. The 18th International Conference on Data Engineering, 2002.
12. W. S. Ng, B. C. Ooi, K. L. Tan, and A.Y. Zhou. PeerDB: A P2P-based system for Distributed Data Sharing. In Proceedings of the International Conference on Data Engineering, 2003.
13. B. C. Ooi, Y. Shu, and K. L. Tan. Relational data sharing in peer-based data management systems. In ACM SIGMOD Record 32(3), September 2003.
14. Özsu, M.T. and Valduriez, P. 1991. Principles of Distributed Database Systems. Prentice Hall, Englewood Cliffs, NJ.