Flexible Update Management in Peer-to-Peer Database Systems

David Del Vecchio and Sang H. Son
Department of Computer Science
University of Virginia
{dad3e, son}@cs.virginia.edu

Abstract

Promising the combination of flexible, dynamic configuration, high scalability and built-in redundancy, peer-to-peer (P2P) networks have garnered tremendous interest in recent years. Before long, the potential for sharing disparate databases in a P2P fashion became apparent. Many have explored the data placement and coordination difficulties involved in such systems in an effort to make P2P database sharing a reality. Far fewer, however, have examined the unique challenges involved in updating data items distributed among numerous P2P nodes. In addressing that problem, we present an adapted version of the traditional quorum-consensus approach that is better suited to the P2P domain. Further, we explore the many tradeoffs introduced by such a flexible update scheme, especially in the areas of data freshness, data availability, access latency and redundancy. Our simulation results, for instance, demonstrate that it is possible to achieve a probability of stale data access that is near zero if the data replication or quorum levels are sufficiently high. The cost of this high data confidence is a slower data access time, since more nodes are involved in quorum-building. In keeping with the theme of flexibility, our philosophy is to leave control in the hands of users and individual peers, allowing them to choose parameters and make performance tradeoffs to meet their unique needs.

1. Introduction

The notable French pilot, poet and author, Antoine de Saint-Exupery once wrote that: “Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away.” In that same spirit, peer-to-peer (P2P) networks are less notable for what they add than for what they lack, in this case, a central server. P2P systems aim to achieve a completely decentralized network by relying on individual hosts to communicate with each other to negotiate and maintain connections, store and transfer data, or perform distributed computations.

Peer-to-peer architectures have generated a great deal of interest of late, accompanied by an increased level of activity which shows few signs of abating. Any justifiably skeptical individual would have to ask: why all the enthusiasm?

Many of the benefits of P2P networks are fairly straightforward and in fact are direct consequences of eliminating the central server. First, observe that removing the server means no expensive setup or maintenance is needed for this typically crucial component. Nodes in a peer-to-peer system can often be heterogeneous (with different hardware platforms, operating systems and languages) and unreliable, provided that they all speak the same protocol. A big part of P2P systems is dynamic configuration—nodes can join and leave the network at any time and both network and nodes must be able to seamlessly reconfigure themselves on the fly. Assuming this all works as intended, peer-to-peer systems can be remarkably low-maintenance. Further, with a large number of participating nodes, P2P networks can achieve a high level of availability and redundancy with little extra effort. Another important benefit of peer-to-peer architectures is scalability [21]. In traditional systems, the server's capacity often becomes a bottleneck. By spreading computation, data storage and message transfers to all the nodes in the network, a P2P system can scale effectively, even as its resource demands and network size grow. Can improved redundancy and scalability actually be achieved with setup and configuration costs that are lower than usual? This almost sounds too good to be true and clearly we are ignoring some major challenges that have to be addressed first.

1.1 P2P Databases

While a number of applications have been suggested for peer-to-peer networks [3], one particularly intriguing area is P2P databases. Consider some of the key concerns in traditional and distributed databases: data availability, redundancy and concurrency. These happen to fit pretty nicely with many of the benefits of peer-to-peer systems we just discussed. Many organizations have a large number of largely disparate database systems that actually become more valuable when meaningfully linked together. For instance, a retail company might have one database to record a history of customer purchases and transactions and a separate database to track in-store inventory and a third containing the company's important suppliers. Since customer purchasing patterns influence the desired inventory levels, and desired inventory levels affect product quantities requested from suppliers, it makes sense to link these three databases. In

fact, many organizations will do just that, but in a very rigid way on per-database basis. Peer-to-peer database systems offer the promise of dynamic, on-the-fly configuration in addition to flexibility with regard to adding and removing data nodes. Unsurprisingly, achieving this goal of simple, flexible data and resource sharing is not without its challenges (as alluded to earlier).

Perhaps chief among the obstacles to effective P2P databases are the problems of data placement and data coordination. The goal of P2P data placement (as defined by Gribble, et al. [10]) is to distribute data and computation among peers so that queries can be executed with the smallest possible cost in terms of response time, bandwidth usage or some other criteria. Data placement design choices and tradeoffs include: the level of shared (global) knowledge and autonomy peers have to make decisions, how dynamic the network is with regard to peer membership, the level of data replication and data granularity, and freshness and consistency requirements. The data coordination problem involves managing the data dependencies and data semantics among database peers [9]. Here the challenge is reconciling numerous individual data schemas to associate semantically related data stored under different names and formats.

1.2 P2P Data Updates

The work described in this paper focuses on updates in peer-to-peer database systems. Updating data with multiple copies in several locations can be difficult, but has been studied somewhat in traditional distributed database research [19]. However, the extreme flexibility of P2P systems often causes more traditional approaches to break down. Most P2P database systems have no central location where metadata can be stored or updates can be synchronized. Further, P2P networks are highly dynamic, nodes can join and leave at anytime and highly autonomous peers expect to be able to change their local data at will (whether connected to the network or not). This paper asks the question: can data updates be handled predictably in such an environment? By predictable, we mean that the peer making the update request can be reasonably certain of what the outcome of the update will be (i.e. how many nodes will reflect the updated value and how long it will take). A key concern is what kind of update consistency can be achieved in a peer-to-peer environment.

In this paper, we don't deal directly with the data placement or configuration problems discussed above. Instead we assume that reasonable, if primitive, solutions to these problems can be employed and work from there. We adapt a weighted-voting, quorum consensus update

scheme to the peer-to-peer architecture and explore the conditions under which consistency can be guaranteed both absolutely and probabilistically. After reviewing some of the related work in the P2P database field (Section 2), we will give an overview of traditional quorum consensus (Section 3), then move on to the specifics of our update scheme (Section 4). We will discuss the flexible nature of our weighted-voting strategy that allows users to trade-off a variety of factors including consistency, availability, query latency and convenience. After presenting the results some performance simulations that explore some of these tradeoffs (Section 5) we'll touch on some challenges for future consideration (Section 6).

2. Related Work

While still a relatively nascent field, P2P database research has experienced rapid growth and is making strides toward addressing the data placement and coordination challenges mentioned. The first step of any P2P query is to locate relevant data in the network [2]. Note that this is true for any P2P network, not just P2P databases. Perhaps the simplest lookup method is broadcast: each node forwards requests to its nearest neighbors until the relevant data is found or the query has reached some maximum time limit. A variation on this scheme is used by the Gnutella network [16]. Due to the large number of message transfers involved, pure broadcast schemes tend to scale poorly. A common modification is to add some hierarchy to the network in the form of super-peers, a set more reliable nodes that can manage a greater number of connections [17]. By building data summaries along these hierarchies, it is possible to support efficient semantic-level searching as well [22]. Other recent research makes use of distributed hash tables to quickly locate data and distribute it evenly throughout the network [23].

Beyond merely locating data, peer-to-peer database systems seek to make it possible to dynamically establish communication links between several heterogeneous database systems (see Figure 1 for an example data mapping). In particular, Bernstein et al. [4] propose the Local Relational Model (LRM) as a framework for managing the differences between various local databases such that the data they contained can be meaningfully linked in a peer-to-peer setting. LRM introduces *domain relations* as a way to map between data items in different databases and *coordination formulas* to specify semantic dependencies between the data. All of these mappings are inherently local and are created and used without the benefit of a global schema. Kementsietsidis et al. [15] have demonstrated an algorithmic implementation of how such a

dynamic mapping between two distinct relations might be established. The Piazza system [19, 5] takes a fairly similar approach and makes it hierarchical, supporting an arbitrary collection of interconnected peer schema mappings. In order to process a query, it is first *reformulated* according to the defined mappings so that it can be defined entirely in terms of the underlying data. The PeerDB system [18] gets its inspiration from information retrieval and creates a kind of data keyword thesaurus to store name mappings. Standard search techniques are then used to locate data that is relevant to a given keyword query.

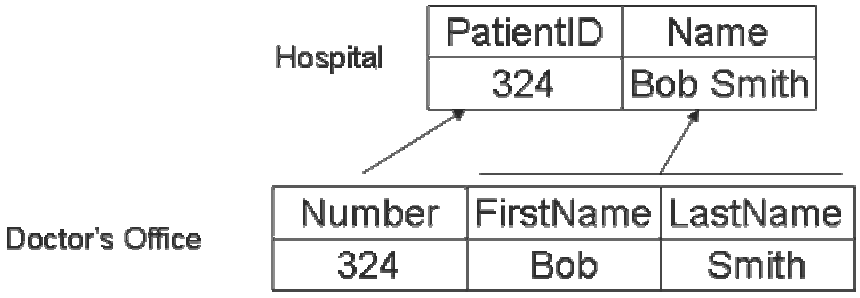

Figure 1: A sample data mapping between two different schemas.

In addressing the data placement problem, Gribble, et al. [10] propose hierarchical *spheres of cooperation*. Nodes within a sphere share information and arrive at data placement decisions cooperatively. Spheres can then try to consolidate queries or adjust data placement within a sphere as means of optimization. Zhang, et al. [26] also pursue a hierarchical approach to data placement, by mapping hierarchical structured path names to an underlying flat data storage structure among peers. The authors encounter a tradeoff between performance (in terms of latency) and fair and efficient resource distribution and try to strike a balance between the two.

Although much of this research is promising, it tends to consider the local databases in a peer-to-peer network as fairly static and fails to address the possibility that an individual might want to update the data stored in some of these local databases. How to manage data updates across several heterogeneous databases with different structures, schemas, and even access policies is by no means straightforward. Yet this ability to change values and have their updates managed in a predictable way is crucially important to realizing the full power of a peer-to-peer database system. Rodrig and LaMarca [20] aim to achieve sequential consistency for P2P data

updates and propose a weighted voting scheme. However, many of their assumptions seem at odds with typical P2P networks which are highly dynamic and in which the number of copies of a data item is rarely known with any certainty. Rather than providing strict sequential consistency which is often impossible to guarantee in typical P2P systems, our scheme provides probabilistic guarantees. Perfect knowledge of the number of replicas is no longer needed; instead a decent estimate is sufficient (and often easily obtainable from the underlying P2P protocol). Further in our scheme, users are allowed to make tradeoffs effectively by choosing the most important areas of performance (consistency, response time, etc.) and we evaluate some of the implications of these tradeoffs later in this paper.

3. Traditional Quorum Consensus

Quorum consensus [8] is a flexible variation of majority consensus [25], a technique for ensuring consistency among multiple data copies in a distributed environment. The basic idea is that each copy of a particular data item has two pieces of information associated with it: a certain number of votes and a version number. In order to access a data item (either for reading or writing) some minimum number of votes, a *quorum*, must first be obtained. The node requesting the data access would typically send its request to all nodes containing the data item. Each properly functioning node that received a request would then respond indicating how many votes it has and what version of the data that node has stored locally. Once the original requestor collects enough votes to meet the threshold it can then read the value (from a copy with the most recent version number) or update the data item.

The vote threshold, or quorum level chosen has important implications for the consistency and behavior of the system. Typically, a system will have separate quorum levels for reading (call it RQ) and writing (call it WQ). These thresholds indicate the number of votes required for reading and writing, respectively. The read quorum and write quorum do not have to be set at the same level and in fact it's often desirable for them to be different. For instance in a system with frequent reads, but relatively few writes, it might make sense to set the read quorum (RW) low and the write quorum (WQ) high. This would have the effect of making the frequent reads fast and while making writes slower since more votes would have to be collected.

In order to maintain data consistency, there are two key restrictions placed on the quorum values chosen: RQ + WQ must be greater than the total number of votes for that data item

available in the system and WQ must be greater than half the number of votes available. This guarantees that the read quorum and the write quorum overlap and every read operation must see at least one data copy that reflects the most recent write operation. As long as these conditions are maintained, it is impossible to read a stale data value. An additional benefit of this approach is that node failures do not affect the data consistency: as long as a quorum can be obtained from the set of working nodes, the transaction can safely proceed. However, if too many nodes fail, it will no longer be possible to obtain enough votes to continue with a read or write operation. Since consistency could no longer be guaranteed, the operation would have to be abandoned.

4. Peer-to-Peer Quorum Consensus

Remarkably, the traditional quorum consensus scheme can be adapted fairly easily for peer-to-peer networks. What changes most are the implications of using this technique and the meanings of the quorum levels chosen. The traditional weighted voting for distributed systems as described in [8] assumed a centralized location existed for storing the list of data item replicas, version numbers, vote distributions. Since P2P networks lack any centralization, our approach is to handle all of this locally. Rather than a master replica list, each individual node is responsible for finding the set of accessible copies of a data item. This would typically be done using the lookup strategy supported by the P2P network infrastructure. Among the lookup strategies already surveyed, broadcast is a commonly used (if less than ideal) approach. Once a node has established the copies of a data item it can access, it would distribute votes, assigning each copy a certain number. This vote allocation is stored locally and when a request for a data item is made, target nodes will respond with either a yes or no vote (or timeout, which is interpreted as no). The requesting node would then tally the votes according to its local vote allocation and decide whether the access can proceed based on the quorum levels the node has chosen to follow.

4.1 Enforcing Consistency

Observe that with this scheme, consistency has now become local, rather than global. Data accesses can appear sequentially consistent to a given node, provided that the following conditions are met:

- Each peer knows the total number of copies that exist for a data item. This cannot just be the number of copies currently accessible in the network; it must be the total number of copies that could *ever* be present.
- Version numbers must be globally unique. This is not that difficult to achieve, especially if some form of clock synchronization is used.
- Individual peers cannot read or update data items when disconnected. Clearly when nodes are not connected to the network, they cannot obtain a quorum and thus would jeopardize consistency.

With these requirements in mind, here are the basic steps a peer node would follow when trying to update a data value:

1. Requestor tries to establish a write quorum (according to its locally set threshold, as discussed above).
2. Peers that vote yes, lock their replicas and send back its (global) version number.
3. Requestor sends new data value and new version number to all copies in the quorum.
4. Peers update their value and unlock their copies.

This is effectively a version of the two-phase commit protocol that makes sure that a quorum of copies agrees to commit before proceeding. If some nodes fail before committing the new values, the update can still proceed as long as a write quorum still exists without those nodes. If with the loss of those nodes, a quorum no longer exists, the transaction must abort. Updated values should be propagated to all copies of the data item, even those casting no votes, or not voting at all. These updates could be propagated in a number of different ways; lazy propagation is one possibility [5]. Further exploration of update propagation alternatives, though important, is left as future work.

4.2 Relaxing Consistency

There are many situations where global knowledge of the total number of copies in existence simply isn't available to peers. That global knowledge is required at all is somewhat at odds with a true P2P system, which should be totally decentralized without shared global

information. Without this global knowledge, however, guaranteeing the same kind of consistency becomes impossible. Consider what can happen if a client chooses its quorum thresholds based on its current knowledge of the number of replicas in the network, a scenario depicted in Figure 2.

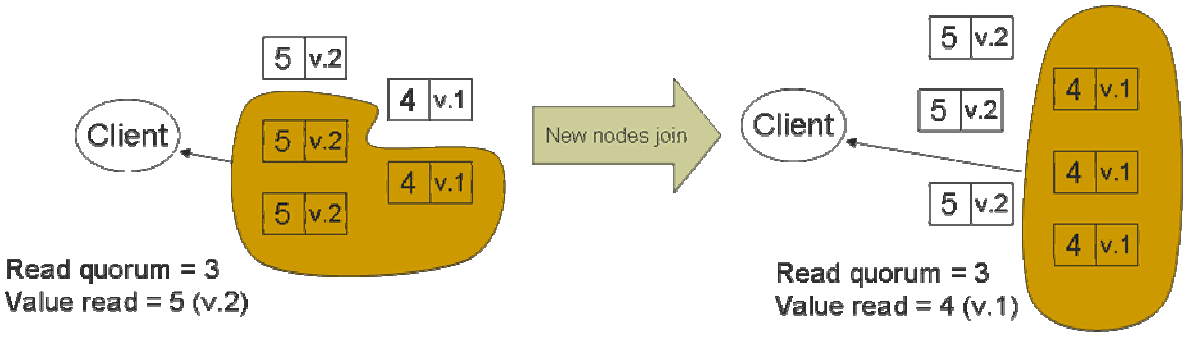

Figure 2: Lack of global replica knowledge can lead to accessing stale data values.

If peers are truly free to join and leave the network at any time, the number of copies of data items available in the network will always be changing. In Figure 2, when a node with an old version of the data item joins, yet the quorum level stays the same, obtaining a quorum no longer guarantees that at least one copy contains the most recent version.

A possible solution to this problem is to force every node that joins the network to update its data items to the most recent versions. Additionally, whenever a node joins or leaves the network, all the other peers would be notified so they could adjust their quorum levels accordingly. However, another problem can crop up if a substantial number of node failures cause a network partitioning. Since it would seem like the number of data replicas has dropped, nodes in each half of the partition would revise their quorum levels downward. Now, different updates could happen simultaneously in both halves of the network. Then if the partitions were to reunite, we would confront two different and equally valid versions of reality. Each would have its own most recent versions for data values and we would need a way to reconcile these.

Things are starting to look pretty bleak for the P2P quorum consensus approach. The problem is that the strict consistency we're trying to support is unreasonable in such a dynamic, decentralized peer-to-peer environment. So instead we relax the strict sequential consistency and allow for the possibility of reading old data values. The example in Figure 2 shows reading old values is indeed a possibility, but what is the probability that this actually happens? In the right half of Figure 2, there are 20 different 3 replica quorums possible, and only one will result in

reading an old value. So assuming only 3 of the six replicas vote yes there's a 5% chance of an old value being read. Typically, you'd expect more replicas than this to respond; if four or more responded, it would be impossible to read an old data value in this scenario.

What makes the flexible quorum consensus approach even more powerful is that individual peers can control what kind of risk of consistency violations they're willing to tolerate by adjusting their quorum levels. Quorum values are now better thought of as a level of confidence in the data being accessed: the higher the quorum levels, the greater the confidence. The key idea is that peers can choose the tradeoff between data freshness and a variety of other factors:

- *Availability* – The higher the quorum threshold, the greater the chance an attempted access will fail due to lack of votes. That's the price to be paid for better data freshness.
- *Access Time* – A smaller quorum will tend to improve data access (response) times since fewer peers have to be consulted before an operation can proceed.
- *Convenience* – If desired, disconnected peers can continue to read and write their local copies of data items. This is inherently risky; since the node has no knowledge of updates occurring in the P2P network it's easy to read a stale value. Further, the peer must reconcile disconnected writes with the rest of the network upon reconnection. Nevertheless, disconnected updates are made possible with the relaxed consistency provided.

This approach basically swaps firm guarantees for probabilistic guarantees while achieving a great deal more flexibility in the process. Such a strategy seems to fit better with the flexible nature of peer-to-peer networks that often lack any strong guarantees (with regard to locating data or message delivery) to begin with.

4.3 Choosing Write Quorum Levels

The traditional quorum consensus approach requires that at least half of the copies of a data item be available for an update to succeed. For our relaxed quorum consensus scheme this restriction is still important for ensuring that two different data updates do not occur simultaneously using two different write quorums. In fact, since each node's local estimates of

the number of copies of a data item available are likely to be slightly shy of the total number, it is probably a good idea to require write quorum levels that are somewhat higher than half of the number of votes (maybe WQ > 3/5 of the estimated total). Since even in the presence of a large number of node failures, a high percentage of nodes are still reachable (as we'll see in Section 5.3), the probability of such update conflicts are relatively low. And of course peers (or more likely the P2P network administrators) can adjust this write quorum restriction based on the update conflict risk level that is tolerable for their application.

4.5 Alternative Quorum Types

The quorum consensus (or majority voting) scheme we've been considering is just one of many quorum approaches that have been proposed over time. Grid quorums [6] for example, organize sites into some kind of geometric pattern (a rectangle at their simplest), then restrict quorum formation based on some properties of the grid (e.g. all elements in one column, or one element from every column). Such a structuring doesn't really map well to dynamic P2P network connections in which the set of data copies (and peer links) often changes, but instead seems better suited for static networks with globally available information. Tree quorums [1] are similar to grid quorums but instead impose a logical tree structure over the set of data replicas. The concerns about dynamically reconfiguring this structure still apply, but tree quorums also have a significant reliance on the root and upper level nodes on the tree. In fact in some tree quorum approaches [1], all write operations have to go through the root. Placing a large load on a small set of nodes can significantly reduce the scalability benefits of peer-to-peer networks.

One quorum approach that does seem promising for P2P applications is hierarchical quorum consensus [11]. Although sites are organized hierarchically into an n-ary tree, hierarchical quorum consensus doesn't have the same root node dependence as tree quorums and so would likely be a better fit for P2P's scalable nature. Indeed, such a hierarchy might map nicely onto a P2P network with superpeer nodes (see next Section 6 for more on these), but sadly must be relegated to future work. Jimenez et al. [12] did a comparison of various quorum approaches and found that the simple read-one copy/write all available strategy was often the best in terms of scalability, availability and communication overhead. Although their assumptions (homogenous, fully replicated sites with failure detection abilities) don't necessarily mesh with P2P networks, it's worth pointing out that peers in our flexible quorum consensus

could certainly choose a read-one/write-all-available strategy by setting their read and write quorum levels appropriately. In terms of maintaining the P2P goals of dynamic reconfiguration, heterogeneity, scalability and redundancy and for P2P database applications, it seems that simple, adaptable strategies like flexible quorum consensus are likely the way to go.

5. Evaluation

To get a better understanding of the effectiveness of our flexible quorum consensus approach and the ramifications of the various tradeoffs it engenders, we constructed a basic implementation and simulated its operation. We made use of the NeuroGrid peer-to-peer simulator [14] for our experiments. NeuroGrid is capable of simulating a variety of different underlying peer-to-peer protocols (including Gnutella, FreeNet and the NeuroGrid protocol [13]) at a fairly high level of abstraction. As a result, it can simulate a fairly large number of nodes relatively quickly. We opted to use the Gnutella P2P protocol [16] for these experiments. Although perhaps not the most advanced or efficient, the Gnutella protocol is well-established and has been used extensively by millions of people over the last few years. Though it might be possible to optimize performance depending on the underlying protocol in use, our general findings should hold regardless. We give a brief overview of the Gnutella protocol below.

5.1 Gnutella Protocol

The Gnutella protocol defines for peers a few basic actions fundamental to the operation of the network. These include *joining* the network, *pinging* to locate other peers and *querying* to locate data stored on other peers.

In order to *join* a Gnutella network, a peer must know the address of at least one peer that is already connected. After successfully establishing a connection to this peer, the new node has joined the network and sends out *ping* requests (through its active connection) to try and locate additional peers on the network. The joining node will cache a list of peers that responded to the ping requests and will initiate and maintain connections to a small number of these peers. If, at some point in time, one of these active connections is broken, a node will proceed to connect to another node in its cache of peers.

As a way of locating peers with resources of interest, a Gnutella node will send out a *query* message to the direct neighbors it maintains active connections to. Every query message

has a time-to-live (TTL) or hop-count field; whenever a peer receives a query message it should take the following actions:

- Check if the query matches any of the node's local resources. If there is a match, the peer should send a *QueryHit* message back to the original sender of the query. The QueryHit should travel backward along the same path that the original query message took.
- Decrement the message's TTL field. If this field reaches 0, do not propagate the message further. This is how the Gnutella protocol limits query lifetimes to prevent them from running forever.
- If the TTL field is not 0, forward the query message to all of the node's active connections.

For the purposes of our experiments, we settled on the following basic parameters:

- 500 node network
- Each node maintains 3 active connections
- Messages have an initial TTL of 7 hops

The number of active connections and message TTL were chosen due to suggestions in the Gnutella protocol documentation that these were fairly typical settings of most Gnutella clients. The network size is a little on the small side for Gnutella file sharing networks, but seems reasonable for P2P database sharing which is likely to involve fewer peers. This is mainly owing to the fact that far fewer people tend to have interesting databases they are willing to share than interesting files.

Consistent with real P2P networks the simulator uses a randomly assembled network topology (one that might result if a bunch of peers joined and left the network at random). Queries (data accesses) originate at randomly selected nodes and diffuse outward from there. Data replicas are distributed randomly throughout the network, but the total number of copies will be some fraction of the number of peers in the network. For each of the following experiments, 500 simulation trial runs were performed for each configuration (data point) and averaged. Additionally, 99% confidence intervals (two-sided t-distribution, $\alpha = 0.01$) are displayed for each point.

5.2 Reachability Under Node Failure

Our first simulation sought to determine how many functioning nodes of the P2P network would still be reachable, given that a certain percentage of the nodes had failed (Figure 3).

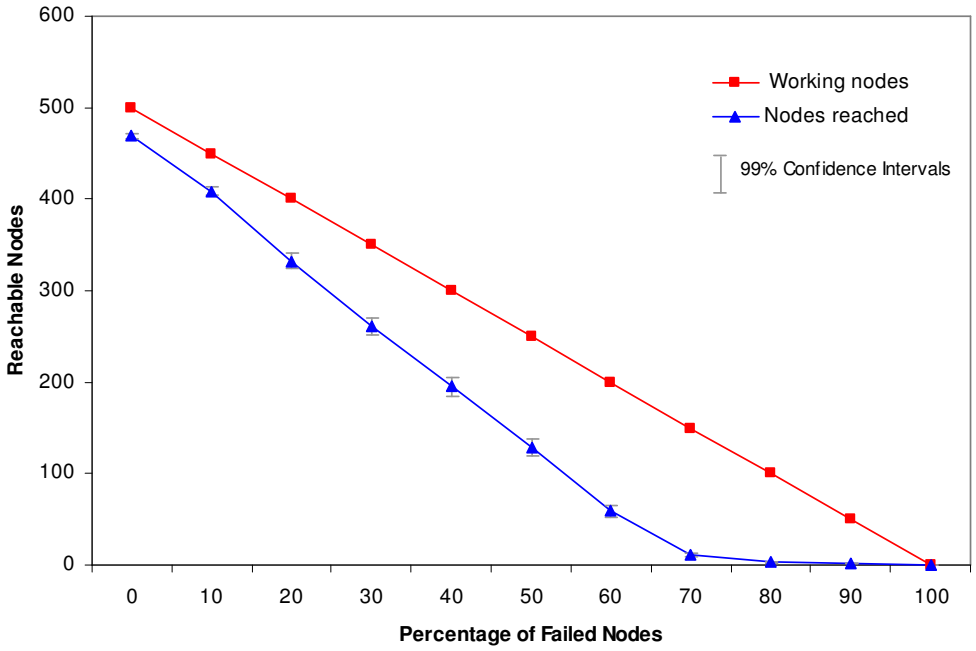

Figure 3: Network reachability under node failure

These results show the impact of a fairly large fraction of nodes failing simultaneously, really a pretty catastrophic situation. If much of the network happened to become unreachable or the network became partitioned as a result of a relatively small number of node failures, it would become nearly impossible to obtain a consensus, rendering the network fairly useless. We see from the results that even if the network is fully operational, the entire network is not reachable (although it's close, at about 95%). This is primarily because there is no restriction on which nodes peers maintain connections to. Many peers will probably end up with connections to the same set of nodes, reducing the chance that a query message will reach the entire network before its TTL runs out. In general, increasing the number of active connections or the message start TTL would improve reachability, but at the expense of more messages flooding the network.

As the percentage of failed nodes grows, most of the working nodes stay reachable; not until the failure rate reaches about 50% do things really start to deteriorate. Once 60% of the nodes have failed, less than half of the remaining nodes stay reachable, opening up the possibility of non-overlapping write quorums. If the still functioning peers were to revise their

quorum levels downward (adjusting for the failures), inconsistent parallel write operations could occur in two different subsets of the nodes. Although this may seem problematic, it really only becomes a possibility after a fairly extensive level of node failure and further, peers would have to become aware of the reduced node set before adjusting their quorum levels. Upon detecting the catastrophe, peers could choose to maintain current quorum levels or increase message TTL to reduce or eliminate the risk of inconsistent writes.

5.3 Message Transfers for Quorum Building

Much of the effort and many of the messages sent for both read and write transactions will involve first obtaining a quorum of copies. This experiment examines the network traffic (in total message transfers) required to build a consensus for several different consensus levels and degrees of data replication (Figure 4). Though the set of replication levels tested is not enormous, the general trends and tradeoffs between different levels are still readily apparent. Expected behavior for replication or quorum levels not tested could easily be extrapolated.

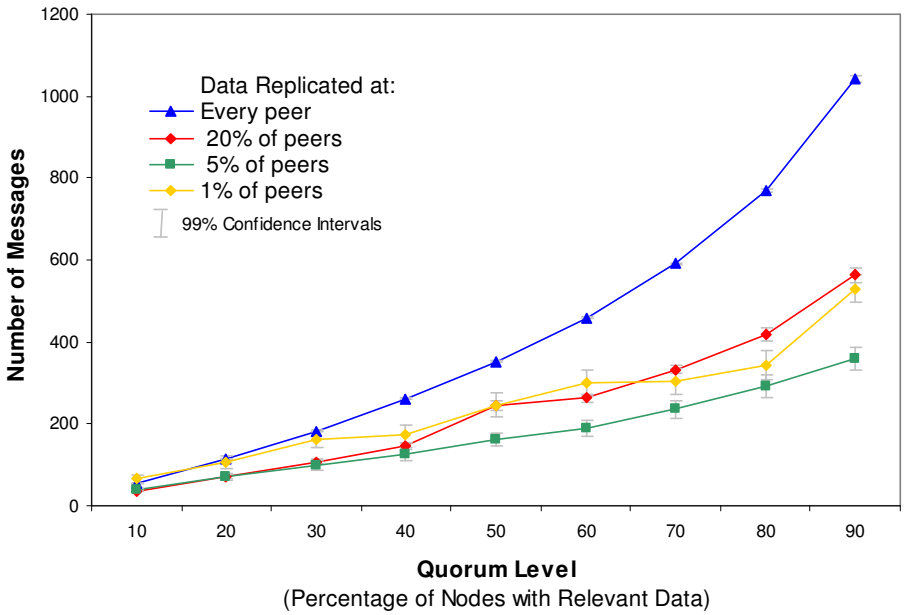

Figure 4: Message transfers involved in quorum building

The volume of network traffic involved in quorum building is important in determining the overall concurrency and volume of transactions the network can support. It also plays a role in influencing the query response time which we'll look at next. From Figure 4 we see that as data is replicated at more nodes, more nodes need to be involved to obtain a consensus and more

messages need to be exchanged. This is the price for greater redundancy. Interestingly enough, if the data is very sparsely distributed (at only 1% of peers), a large volume of messages must be sent just to coordinate these far flung nodes. At relatively low quorum levels, on the other hand, a high degree of replication becomes less of a penalty since several matching nodes can be found in the near vicinity without sending too many messages. The best all around performance seems to be achieved between the two extremes; in particular the 5% replication-level appears to strike a good balance between redundancy and quorum-building effort. If greater redundancy is needed, more messages will be needed, but trying to lower the quorum-building effort further by reducing redundancy actually appears self-defeating (as the 1% replication level shows).

5.4 Query Response Time

How long will it take to read or update a value distributed throughout the network? This too is directly related to the time it takes to build a quorum. But here we take a different view and measure elapsed response time as opposed to the number of message transfers (Figure 5).

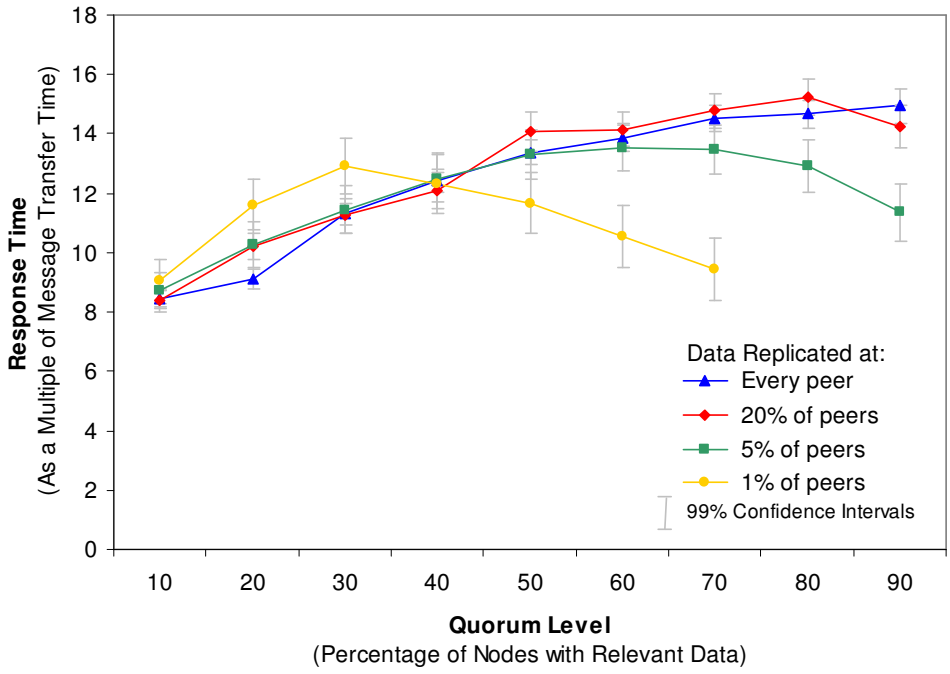

Figure 5: Query response time for different quorum and replication levels

In this experiment, time is not measured in seconds, but is instead measured relative to how long it takes to transfer one message over a single hop. This assumes that message transfer between different nodes is relatively consistent (or can be reasonably approximated by an average). A

benefit of this simplification is that the results are relatively independent of the performance of the underlying communication network.

Once again we see trade-offs between data-replication and response time. If a high degree of replication is desired, a low quorum level ends up being faster since a peer can quickly find enough nodes to meet the quorum threshold among its immediate neighbors. Unsurprisingly, as quorum levels increase, response times tend to as well: the trade-off between latency and data freshness. While many of the replication levels have good response times when the quorum levels are low as the quorum levels increase (as does data confidence), some differences due to redundancy start to emerge. The response times of lower replication levels tend to be better, merely because fewer nodes (in terms of absolute numbers) are needed to achieve a quorum. So, at the high data freshness end of the spectrum, you're forced to settle for worse response times if you need more redundancy. Note that when the data was only replicated at 1% of peers, so few nodes held the data that quorum levels above 70 percent were unattainable (explaining the missing data points). This indicates that it can be difficult, if not impossible to achieve both very high data confidence and very low redundancy.

5.5 Probability of Stale Data Access

Clearly, in order to decide what quorum levels are appropriate for a given peer it is important understand the consequences of that choice, particularly in terms of the probability of reading an old data value. This experiment explored this data confidence, quorum threshold tradeoff in more detail. For the simulations, we assumed a worst-case scenario in which the most recent copy of a data item was only replicated at the minimum possible number of nodes needed to obtain a quorum. So if the quorum level were set at 20%, only 20% of the relevant nodes are assumed to hold the most recent value. This is relatively unlikely in a real-world network, but it is useful for illustrating the general relationship between quorum thresholds and data confidence (Figure 6). We would expect the probabilities of reading stale values in a real network with a decent update propagation scheme to be even lower than those shown.

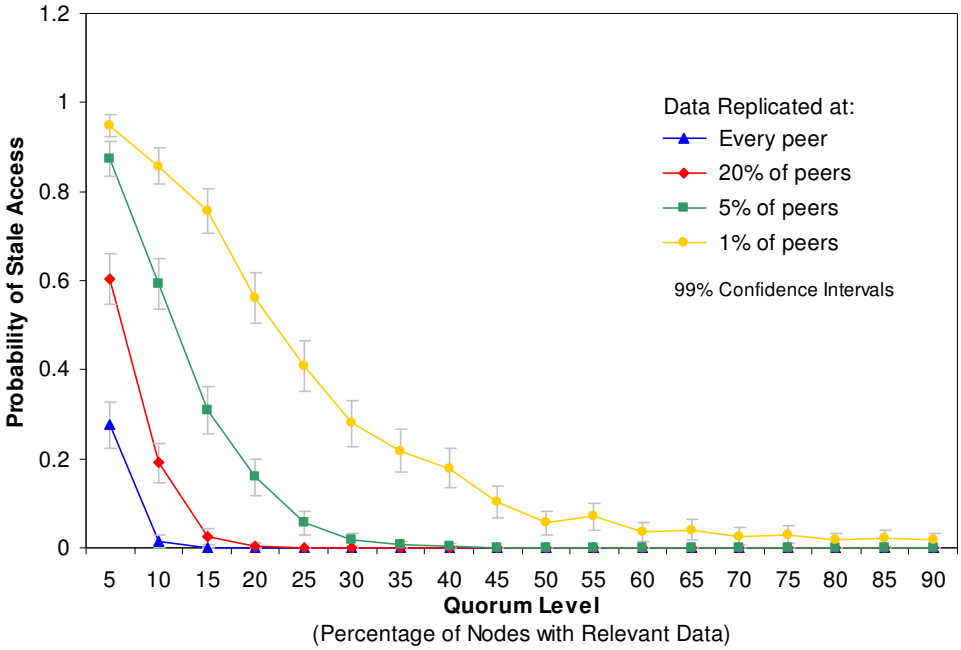

Figure 6: Probability of stale data access for different quorum and data replication levels

The results of this experiment are relatively straightforward. As the level of data replication goes up, the chances of reading an old value (even at the same quorum level) are reduced. This is primarily because a larger number of nodes are involved in the quorum, so it becomes far more likely that one of the nodes will possess an up-to-date copy. Stale access probabilities are fairly high below about 15% or so, but drop-off quite dramatically. Even with a relatively sparse 5% data redundancy, only a 40% quorum level is needed to achieve a stale access probability that's practically zero.

5.6 Stale Access under Node Churn

Since nodes can join and leave a P2P network at anytime, a network's configuration rarely (if ever) remains static. Next, we investigated how our flexible quorum consensus scheme would perform in a network with a fast changing set of peers (Figure 7). We modeled nodes joining and leaving the network in terms of a *churn rate*. The churn rate represents the percentage of the network experiencing changes in membership between queries. So for instance, a 20% churn rate would simulate 20% of the current peers (100 nodes in a 500 node network) leaving the network while another 100 nodes joined the network. In this way the simulated network always stays the same size and the data replication rate also stays constant between queries. However, the versions of different data items do change. Again, we assume a worst case

scenario, only a minimum quorum of nodes have the most recent copy. The set of nodes leaving is randomly chosen and so some leaving nodes will take their up-to-date copies out of the network with them. The set of joining nodes are all assumed to be bringing along stale copies into the network.

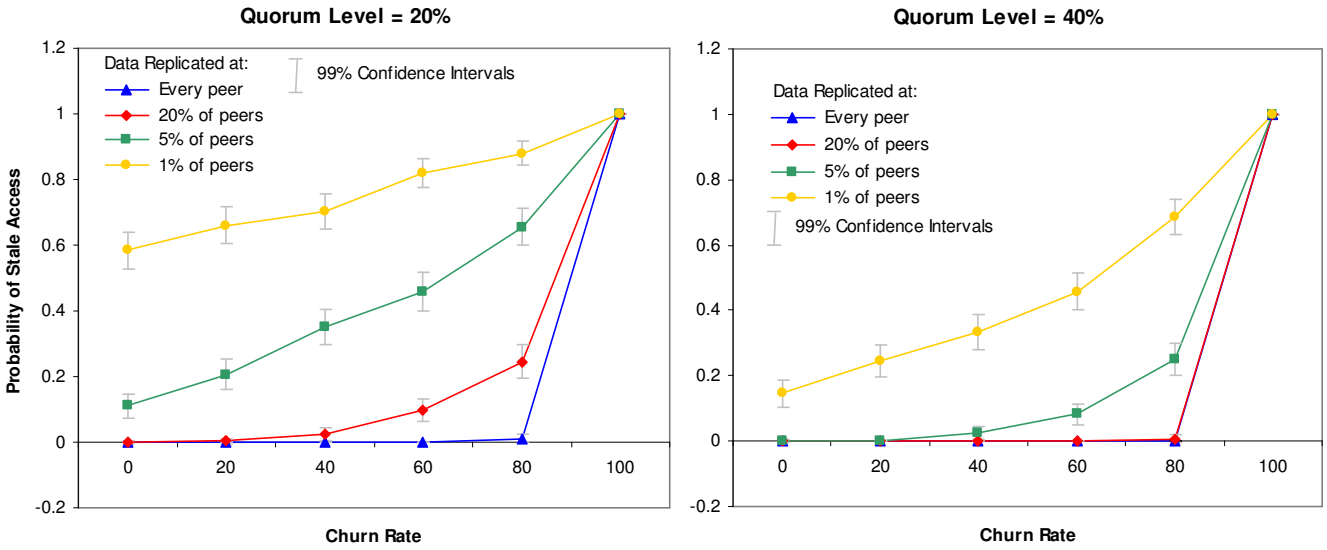

Figure 7: Probability of stale data access as it is affected by the network churn rate. Results are shown for two different quorum levels and varying degrees of data replication

With this set of adverse network conditions, one would expect the probability of accessing old data values to rise and indeed this is what happens. Although, we only show the results for two different quorum levels (20% and 40%) the general trends are still pretty clear. As network churn rates increase so does the stale access probability, due to the fact that newly joining nodes always have old values. Even at low quorum levels (like 20%), if the data is replicated a large percentage of nodes, the chance of accessing an old value is very low. As churn rates increase, high data replication schemes still perform well, inspiring confidence in the data being accessed. Another trend that we see is that increasing the quorum-level makes the network more resistant to node churn, even if the degree of replication stays the same. At a 40% quorum-level, the network is able to maintain high data confidence even as churn rates reach 30%, provided that the data is replicated on at least one out of every twenty peers. This is pretty remarkable, considering that 30% node turnover between queries is highly unlikely in a real network. Once again, it is worth pointing out that an effective update propagation scheme would improve performance even further, by sending along updated values to newly joining nodes and propagating updates to more than just a minimum quorum.

5.7 Summary of Tradeoffs

Our flexible quorum scheme is all about tradeoffs; below is summary of how quorum and data replication levels can be adjusted to meet other performance goals:

Degree of Data Replication		**Quorum Level**	
		Low	**High**
	High	▪ Least network traffic ▪ Fastest access time ▪ Medium data confidence	▪ Most network traffic ▪ Slowest access time ▪ Highest data confidence
	Low	▪ Low network traffic ▪ Short access time ▪ Lowest data confidence	▪ Moderate network traffic ▪ Short access time ▪ High data confidence

This table is structured according to two of the key parameters users can control: degree of replication and quorum level. The quadrants then indicate what performance effects should be expected as a result of the parameter levels chosen.

6. Conclusions and Future Work

Managing updates in peer-to-peer database systems is both a challenging and important task. If P2P database systems are to have any practical value, predictable updates are a nearly essential prerequisite. The weighted-voting for quorum consensus scheme seems to have significant potential, even when adapted for P2P databases, an area its original designers surely never envisioned. If global replica knowledge is known, the P2P quorum consensus approach can provide localized consistency guarantees. In the more common case when this information is not known, our quorum consensus strategy allows peers to tradeoff the desired level of data freshness with redundancy, availability and response time. Indeed the technique is rich with possible tradeoffs, resulting in a highly flexible and adaptable approach. For instance, those desiring high data redundancy could increase the level of data replication, but the level of network traffic and query response time would likely increase as a result. If latency were also

important, a low quorum level could be chosen, but then of course the chances of reading old values would increase.

There are still a number of facets of P2P data updates that remain unexplored, however, and would provide worthwhile opportunities for future pursuit. One such area already mentioned is update propagation. While there are a variety of strategies that could be employed to solve this problem, each would no doubt impact update performance in the areas of network traffic, query response time and stale access probability. Another promising extension of our P2P quorum consensus scheme involves the incorporation of *superpeers*. These typically more reliable nodes add hierarchy to the P2P network and provide several optimization opportunities. For example, data that is more important or has higher availability requirements could be located on superpeers. These special nodes would also be expected to have a larger allocation of quorum votes to reflect their superior reliability. What affect such strategies would have on performance, however, is not entirely clear. On a related note, it might be worth exploring how the underlying network protocol used affects performance. Several new P2P protocols have been developed in recent years and some might be better for P2P data updates than others.

In this paper we looked at how adjusting the quorum and data replication levels can affect response times and data confidence. Ideally, however, things would work the other way around. Users might specify the stale access probability they are willing to tolerate and ask the network or individual peers to optimally choose the quorum and data replication levels to minimize response times. In a typical situation, a user would only have control over a small set of nodes, not the entire network, in which case global optimization could be difficult. In addition to statically choosing these parameters and tradeoffs, many could conceivably be adjusted in real-time. Peers could dynamically adapt their quorum, vote distribution and data replication levels to improve performance and meet changing needs over time. Such intelligent peer behavior is not without its challenges, most of which are not currently well understood. Even without this advanced behavior, however, peer-to-peer quorum consensus still seems well-suited to the flexible, dynamic and heterogeneous world of peer-to-peer networks.

7. References

[1] D. Agrawal and A.E. Abbadi. *The tree quorum protocol: an efficient approach for managing replicated data*. Proceedings of the Sixteenth International Conference on Very Large Databases, September 1990, pp. 243-254.

[2] H. Balakrishnan, M.F. Kaashoek, D. Karger, R. Morris, I. Stoica. *Looking up data in P2P systems*. Communications of the ACM. Vol. 46, No. 2 (2003), pp. 43-48.

[3] D. Barkai. *Technologies for Sharing and Collaborating on the Net.* In Proceedings, First International Conference on Peer-to-Peer Computing, August 2001, pp. 13-28.

[4] P. Bernstein, et al. *Data Management for Peer-to-Peer Computing: A Vision.* WebDB Workshop on Databases and the Web, Madison, Winsconsin, USA, June 2002.

[5] Y. Breitbart, R. Komondoor, R. Rastogi, S. Seshadri, A. Silberschatz. *Update Propagation Protocols for Replicated Databases.* In Proceedings of the ACM SIGMOD International Conference on Management of Data, 28 (2), June 1999, pp. 97-108.

[6] S.Y. Cheung, M.H. Ammar and M. Ahamad. *The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data*. Proceedings of the Sixth International Conference on Data Engineering, 5 February 1990, pp. 438-445.

[7] A.Y. Halevy, et al. *The Piazza Peer Data Management System*. IEEE Transactions on Knowledge and Data Engineering, 16 (7), July 2004, pp. 787-798.

[8] D.K Gifford. *Weighted Voting for Replicated Data*. Proceedings of the Seventh Symposium on Operating System Principles, December 1979, pp. 150-162.

[9] F. Giunchiglia and I. Zaihrayeu. *Making Peer Databases Interact - A Vision for an Architecture Supporting Data Coordination*. In Proceedings of the Conference on Information Agents (CIA 2002), Madrid, September 2002.

[10] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. *What Can Peer-to-Peer Do for Databases, and Vice Versa?* In Fourth International Workshop on the Web and Databases (WebDB '2001).

[11] A. Kumar. *Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data*, IEEE Transactions on Computers, 40 (9), September 1991, pp. 996-1004.

[12] R. Jimenez-Peris, M. Patino-Martinez, G. Alonso and B. Kemme. *Are Quorums an Alternative for Data Replication?* ACM Transactions on Database Systems (TODS), 28 (3), September 2003, pp. 257-294.

[13] S. Joseph. *Adaptive Routing in Distributed Decentralized Systems: NeuroGrid, Gnutella and Freenet.* In Proceedings, Workshop on Infrastructure for Agents, MAS, and Scalable MS at Autonomous Agents, Montreal, 2001.

[14] S. Joseph. *Documentation for the NeuroGrid Peer-to-Peer Simulation Software.* Available at http://www.neurogrid.net/twiki/bin/view/Main/PeerToPeerSimulation. Last revised 9 April 2004.

[15] A. Kementsietsidis, M. Arenas, R. J. Miller. *Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues.* In Proceedings of the ACM SIGMOD International Conference on Management of Data, June 2003, p. 325-336.

[16] T. Klingberg, R. Manfredi, M. Bollaert, et al. *RFC-Gnutella 0.6.* Available at http://rfc-gnutella.sourceforge.net/developer/testing/index.html.

[17] J. Lu and J. Callan. *Content-Based Retrieval in Hybrid Peer-to-Peer Networks*. In Proceedings of the 12th International Conference on Information and Knowledge Management, November 2003, pp. 199-206.

[18] W.S. Ng, B.C. Ooi, K.L. Tan and A.Y. Zhou. *PeerDB: A P2P-based System for Distributed Data Sharing.* The 19th International Conference on Data Engineering (ICDE 2003).

[19] M. T. Ozsu and P. Valduriez. *Distributed Database Systems: Where Are We Now?* IEEE Computer, 24 (8), August 1991, pp. 68-78.

[20] M. Rodrig and A. LaMarca. *Decentralized Weighted Voting for P2P Data Management*. MobiDE'03. San Diego, CA, September 19, 2003.

[21] R. Schollmeier and G. Schollmeier. *Why Peer-to-Peer (P2P) Does Scale: An Analysis of P2P Traffic Patterns.* In Proceedings of the Second International Conference on Peer-to-Peer Computing, September 2002, pp. 112-119.

[22] H.T. Shen, Y. Shu and B. Yu. *Efficient Semantic-Based Content Search in P2P Network.* IEEE Transactions on Knowledge and Data Engineering, 16 (7), July 2004, pp. 813-826.

[23] I. Stoica, et al. *Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications.* In IEEE/ACM Transactions on Networking, 11 (1), February 2003.

[24] I. Tatarinov, et al. Mork. *The Piazza Peer Data Management Project*. SIGMOD Record, 32 (3), 2003.

[25] R.H. Thomas. *A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases.* ACM Transactions on Database Systems, 4 (2), June 1979, pp. 180-209.

[26] Z. Zhang, M. Mahalingam, Z. Xu, W. Tang. *Scalable, Structured Data Placement over P2P Storage Utilities*. Hewlett-Packard Labs Technical Report: HPL-2002-40 (2002). Available at www.hpl.hp.com/techreports/2002/HPL-2002-40.pdf.