Flexible Update Management in Peer-to-Peer Database Systems

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

Abstract

Promising the combination of dynamic configuration, scalability and redundancy, peer-to-peer (P2P) networks have garnered tremendous interest lately. Before long, this interest extended to P2P sharing of disparate databases. While many have explored the data placement and coordination difficulties involved in such systems, far fewer have examined the unique challenges involved in updating P2P-distributed data items. 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 this flexible update scheme, especially in the areas of data freshness, data availability, access latency and redundancy. Our simulation results demonstrate that if the data replication or quorum levels are sufficiently high it is possible to achieve a near-zero probability of stale data access. However, since more nodes are involved in quorum-building, the cost of high data confidence is slower access times. Keeping with the flexibility theme, our philosophy is to leave control in the hands of individual peers, letting them choose parameters and tradeoff performance 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 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 for a completely decentralized network, relying on the cooperation of individual hosts to negotiate and maintain connections, store and transfer data, or perform distributed computations. Of late, P2P architectures have generated a great deal of interest and activity (with few signs of abating). Any justifiably skeptical individual would have to ask: why all the enthusiasm?

This work supported in part by NSF grants IIS-0208758, CCR-0329609.

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 (different hardware, OS and languages) and unreliable; they just have to 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. If things work 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 almost for free. Yet another important benefit of is scalability [22]. 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 as its resource demands and network size grow. Can improved redundancy and scalability *really* be achieved with minimal setup and configuration costs? Almost sounds too good to be true, and clearly we are ignoring some major challenges that have to be addressed first.

1.1 Peer-to-Peer Databases

While a number of applications have been suggested for peer-to-peer networks [4], P2P databases stand out as particularly intriguing. Some of the key concerns of traditional and distributed databases: data availability, redundancy and concurrency, happen to fit pretty nicely with many of the benefits of peer-to-peer systems just discussed. Many organizations have a sizable collection of disparate database systems that actually become more valuable when meaningfully linked together. For instance, a retail company might have one database for customer purchase history, a separate database to track in-store inventory and a third detailing the company's important suppliers. Since customer purchasing patterns influence desired inventory levels, which in turn affect orders to

product suppliers, it makes sense to link the three databases. Many organizations will do just that, but in a very rigid, per-database way. Peer-to-peer database systems offer the promise of dynamic, on-the-fly configuration; flexibility with regard to adding and removing data nodes. Resolving the combination of data sharing and P2P network challenges in a simple, flexible way may indeed seem daunting.

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. [12]) 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 choices and tradeoffs include: the level of shared (global) knowledge and autonomy peers have to make decisions, how dynamic the network is in terms of peer membership, the level of data replication and granularity, freshness and consistency requirements, etc. In contrast, the data coordination problem involves managing data dependencies and semantic mappings between peers [11]. Here the challenge is reconciling numerous individual data schemas to associate related data stored under different names and formats.

1.2 Peer-to-Peer Data Updates

In this paper, we focus on updates in peer-to-peer database systems. Although updating data with multiple, distributed copies can be difficult, it has been studied somewhat in traditional distributed database research [20]. However, the extreme flexibility of P2P systems often causes traditional approaches to break down. Most P2P database systems have no central location where metadata can be stored or updates synchronized. Further, in highly dynamic P2P networks, nodes can join and leave at anytime with autonomous peers wanting to change their local data at will (even when disconnected). 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 the outcome, for example, in terms of how many nodes will reflect the updated value and how long it will take. Another key concern is what kind of update consistency can be achieved in a peer-to-peer environment.

In this paper, we do not deal directly with the aforementioned data placement or configuration problems; instead we work from the assumption that basic yet reasonable solutions to these problems can be employed. 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 give an overview of traditional quorum consensus (Section 3), then move on to the specifics of our update scheme (Section 4). We discuss the flexible nature of our approach, including trade-offs between a variety of factors such as consistency, availability, query latency and convenience. After presenting performance simulations that explore some of these trade-offs (Section 5) we briefly discuss some important issues 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 its data placement and coordination challenges. The first step of any P2P query (regular or database) is to locate relevant data in the network [2]. 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 [17]. 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 of more reliable nodes that can manage a greater number of connections [18]. By building data summaries along these hierarchies, it is possible to support efficient semantic-level searching as well [23]. Other recent research makes use of distributed hash tables to quickly locate data and distribute it evenly throughout the network [24].

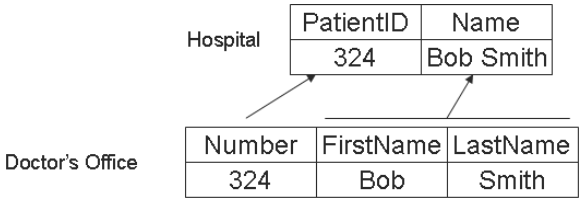

Figure 1: Sample data mapping between two schemas.

Beyond merely locating data, peer-to-peer database systems seek to enable dynamically established communication links between several heterogeneous databases (see Figure 1 for an example data mapping). Bernstein et al. [5] propose the Local Relational Model (LRM) as a framework for managing the differences between various local databases such that their contents 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, created and used without the benefit of global schema. Kementsietsidis et al. [16] have an algorithmic implementation, showing how a dynamic mapping between two distinct relations might be created.

The Piazza system [8] takes a similar tack, but makes it hierarchical, with an arbitrary collection of interconnected peer schema mappings. In order to process a query, it must be *reformulated* according to the mappings so that it is defined entirely in terms of the underlying data. The information retrieval-inspired PeerDB system [19] creates a kind of data keyword thesaurus to store name mappings. Standard search techniques can be used to locate data relevant to a given keyword query.

In addressing the data placement problem, Gribble, et al. [12] propose hierarchical *spheres of cooperation*. Nodes within a sphere share information, arriving at data placement decisions cooperatively. Spheres can try for optimization by consolidating queries or adjusting internal data placement. Zhang, et al. [26] pursue a hierarchical approach to P2P data placement, by mapping hierarchical structured path names to an underlying flat storage structure among peers. The authors encounter a tradeoff between latency and resource distribution fairness/ efficiency 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 P2P network as static, failing to address how one might write to them. Managing data updates across several heterogeneous databases with different structures, schemas, and access policies is by no means straightforward. Yet this ability to change values and have updates managed in a predictable way is crucially important to realizing the full power of a peer-to-peer database system. Rodrig and LaMarca [21] 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, choosing the most important areas of performance (consistency, response time, etc.). We evaluate the implications of these tradeoffs in Section 5.

3. Traditional Quorum Consensus

Quorum consensus [10] 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. To access a data item 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 item. Each properly functioning node that received a request would respond, indicating its number of votes and data version. Once the original requestor collects enough votes to meet the threshold it can read the value (from a copy with the most recent version number) or update the 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 few writes, it might make sense to set the read quorum (RW) low and the write quorum (WQ) high, thereby making the frequent reads fast. Writes would be slowed 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. Thus, it is impossible to read a stale data value. An additional benefit of this approach is that node failures do not affect 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 becomes impossible to get enough votes to continue with a read or write. 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 to peer-to-peer networks. What changes most are the implications of using this technique and the meanings of the quorum levels chosen. Traditional weighted voting (as described in [10]) assumes that a centralized location exists for storing the list of data item replicas, version numbers and 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. 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, tantamount to no). The requesting node would 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.

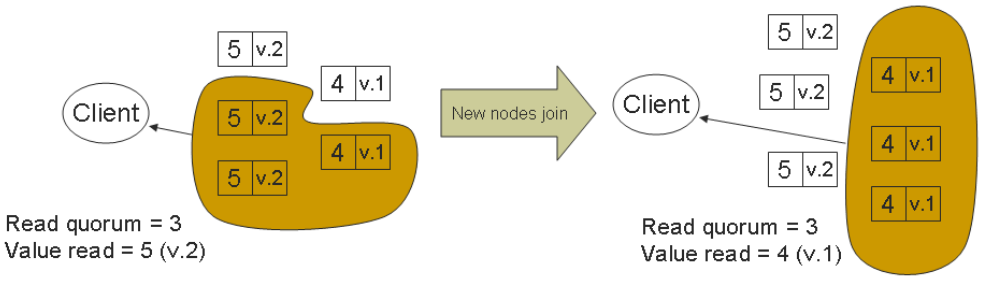

Figure 2: Lack of global replica knowledge can lead to accessing stale data values.

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; not just the number of copies currently accessible in the network, the total number of copies that could *ever* be present.
- Version numbers must be globally unique (not so difficult with suitable clock synchronization).
- Individual peers cannot read or update data items when disconnected. Reading or writing without a quorum jeopardizes consistency.

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

1. Requestor tries to establish a write quorum (according to its locally set threshold).
2. Peers that vote yes, lock their replica 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 copy.

This is effectively a version of the two-phase commit protocol, ensuring that a quorum of copies agrees to commit before proceeding. If nodes fail before committing, but a write quorum still exists among the functioning nodes, the update can proceed. On the other hand, if the write quorum is lost the transaction must abort. Updated values should be sent to all copies of a data item, even peers abstaining or casting no votes. A thorough exploration of update propagation strategies (like lazy propagation [6]), though important, is left as future work.

4.2 Relaxing Consistency

There are many situations where global knowledge of the number of replicas in existence simply isn't available to peers. The need for global knowledge at all is somewhat at odds with a true P2P system, which should be totally decentralized. Without global knowledge, however, our guaranteed consistency becomes impossible. Consider what can happen if a client chooses quorum thresholds based on its current knowledge of replicas in the network. If peers are truly free to join and leave the network at any time, the number of in-network data copies available will always be in flux. In Figure 2, when a node with an old data version joins yet the quorum level stays the same, a quorum is no longer guaranteed to contain at least one up-to-date replica.

A possible solution involves forcing every node that joins the network to update its copies to the most recent versions. Additionally, whenever a node joins or leaves the network, all other peers would have to be notified in order to adjust their quorum levels accordingly. Still, problems 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. Different updates could then happen simultaneously in both halves of the network. If the partitions were to later reunite, we would confront two different and equally valid versions of reality.

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 a dynamic, decentralized P2P environment. So instead we relax the strict sequential consistency and allow for the possibility of reading old data values. Figure 2 shows how reading old values is possible, 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. Assuming only 3 of the six replicas vote yes there's a 5% chance of an old

value being read. Typically, you'd expect a higher response rate; four or more responses would make reading old data impossible.

What makes the flexible quorum consensus approach even more powerful is that individual peers can control the risk of consistency violations 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 to tradeoff between data freshness and several other areas:

- *Availability* – Higher quorum thresholds imply a greater chance of access failure due to lack of votes. That's the price you pay for better data freshness.
- *Access Time* – Smaller quorums will tend to improve data access (response) times since fewer peers need be consulted before operations proceed.
- *Convenience* – Though inherently risky, disconnected peers can opt to read and write local data copies. Without knowledge of other updates in network it's easy to read stale values and upon reconnection the peer must reconcile disconnected writes.

This approach basically swaps firm guarantees for probabilistic ones, achieving a great deal of flexibility in the process. Such a strategy seems to fit better with the nature of P2P networks that already lack strong guarantees (in locating data or delivering messages).

4.3 Choosing Write Quorum Levels

Traditional quorum consensus requires that at least half of the data copies be available for an update to succeed. In P2P networks, since each node's local estimates of available data replicas are likely to be slightly shy of the actual total, it probably makes sense to require write quorum levels that are somewhat higher than half of the number of votes (perhaps WQ > 3/5 of the estimated total). Since even in the presence of large scale node failure, a high percentage of nodes are still reachable (Section 5.2), the probability of updates in non-overlapping quorums is relatively low. And of course peers can adjust write quorum restrictions based on the tolerable risk levels for their application.

Even if WQ is set sufficiently high, care must still be taken in resolving write conflicts. If simultaneous write requests are initiated for a data item, the competition for votes may leave both requests without a quorum. Further, if both writers continue to resubmit upon detecting this failure, the repeated locking of data items could block subsequent accesses. The resolution lies with the nodes in the overlap that receive both requests. These nodes should reject both requests, notifying requesters of the conflict. Additionally, some arbitrary, yet consistent tie-breaking decision must be made by the peers in the overlap so that one request can proceed first.

4.4 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 [7] for example, organize sites into a 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). Although such structuring seems well suited for static networks with globally available information, it doesn't map well to dynamic P2P networks with changing sets of data replicas and peer links. Similar dynamic reconfiguration concerns also apply to tree quorums [1] which impose a logical tree structure over the set of data replicas. Tree quorums also have a significant reliance on the root and upper level nodes on the tree; in the extreme [1], all write operations 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 [13]. Although sites are organized hierarchically into an n-ary tree, hierarchical quorum consensus doesn't have the same root node dependence as tree quorums. Such a hierarchy might map nicely onto a P2P network with superpeer nodes (see Section 6), but sadly, such efforts must be relegated to future work. Jimenez et al. [14] did a comparison of various quorum approaches and found that read-one copy/write all available was often 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 approach could certainly enact a read-one/write-all-available scheme by selecting appropriate read and write quorum levels. Read-one/write all is particularly attractive with known data replica locations, because a read simply involves contacting the nearest copy. With P2P data, this isn't the case. Depending on the protocol, typically several messages must be propagated to nearby peers to find copies. As a result, the overhead of gathering a read consensus may not be so severe compared to what is already necessary to read a single copy. In terms of maintaining the P2P goals of dynamic reconfiguration, heterogeneity, scalability and redundancy, it seems that simple, adaptable strategies like flexible quorum consensus are more appropriate.

5. Evaluation

To get a better understanding of the effectiveness of our flexible quorum consensus approach along with various tradeoffs it engenders, we constructed a basic

implementation and simulated its operation using NeuroGrid [15]. NeuroGrid is capable of simulating a variety of different underlying peer-to-peer protocols (including Gnutella, FreeNet and NeuroGrid) at a fairly high level of abstraction. As a result, it can simulate a fairly large number of nodes relatively quickly. For the purposes of evaluation, we assumed that all peers were working cooperatively together, none were actively trying to undermine or subvert the network or it's data. We opted to use the Gnutella P2P protocol [17]. Though 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.

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 already-connected peer. After successfully contacting this peer, the node has joined the network and sends out *ping* requests (through its active connection) to locate additional peers on the network. The joining node will cache a list of peers that responded to the ping requests, initiating and maintaining 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 neighbors it maintains direct 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 to the query originator. The QueryHit should travel backward along the same path the query request followed.
- Decrement the message's TTL field. Once this field reaches 0, the message is not propagated further. In this way the Gnutella protocol prevents queries from running indefinitely.
- If the TTL field is not 0, forward the query message to all of the node's active connections.

For our experiments, we used following basic parameters:

- 500 node network
- 3 active connections per node
- Initial TTL of 7 hops per message

These active connection and message TTL settings are fairly typical, according to the protocol documentation. Although, the network is on the small side for Gnutella file sharing networks, it seems reasonable for P2P database sharing which is likely to involve fewer peers.

Consistent with real P2P networks we use 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 with the number of replicas set to some fraction of the number of peers. For the following experiments, 500 simulation trial runs were performed for each configuration (data point) and averaged; 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 nodes of the P2P network would still be reachable, once a certain percentage of nodes had failed (Figure 3).

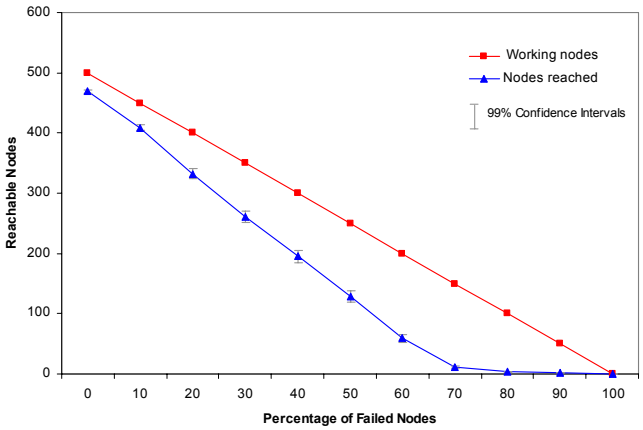

Figure 3: Network reachability under node failure.

These results show the impact of a fairly large fraction of nodes failing simultaneously, a pretty catastrophic situation. If much of the network became unreachable or partitioned due to a small number of node failures, it would become nearly impossible to obtain a quorum, 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 nodes maintain a small number of connections, without restrictions on the active set. Many peers will probably end up with connections to the same set of nodes, reducing the chance that a message will reach the entire network before its TTL runs out. Increasing the number of active connections or the message start TTL would improve reachability, but at the expense of more network flooding.

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 only becomes possible after a severe 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 the risk of inconsistent writes. Each node's set of active connections could also be kept large enough to ensure excellent network coverage and minimize fault suceptibility [8]. In short, network partitioning is quite unlikely, preventable through parameter tuning and usually detectable; not quite the show-stopper it once seemed.

5.3 Message Transfers for Quorum Building

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 quorum levels and degrees of data replication (Figure 4). Though the set of replication levels tested is not enormous, general trends and tradeoffs between different levels are still readily apparent and could easily be extrapolated.

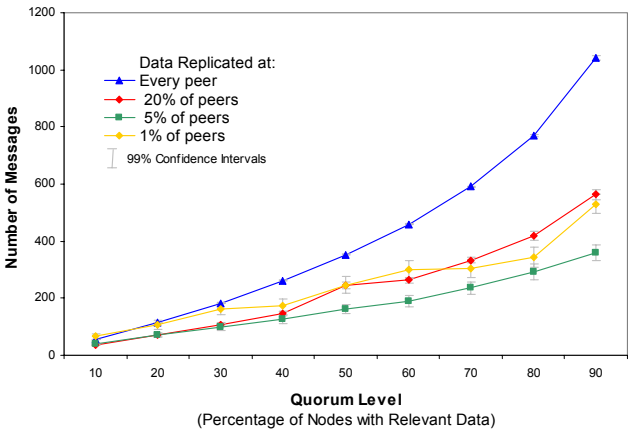

Figure 4: Message transfers involved in quorum building.

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 influences query response times (next section). From Figure 4 we see that as data is replicated at more nodes, more nodes need to be involved for consensus and more messages need to be exchanged: the price of 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 with the 5% replication-level striking a good balance between redundancy and quorum-building effort. If greater redundancy is needed, more messages will be sent, but trying to further lower the quorum-building effort by reducing redundancy is 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 (Figure 5) as opposed to the number of message transfers.

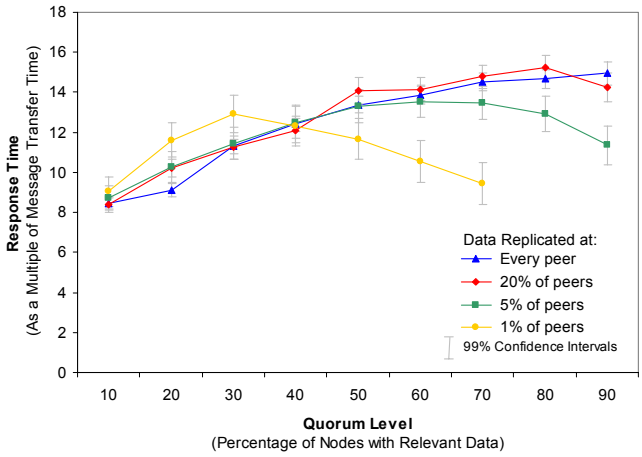

Figure 5: Query response time for different quorum and replication levels.

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 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 used, 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:

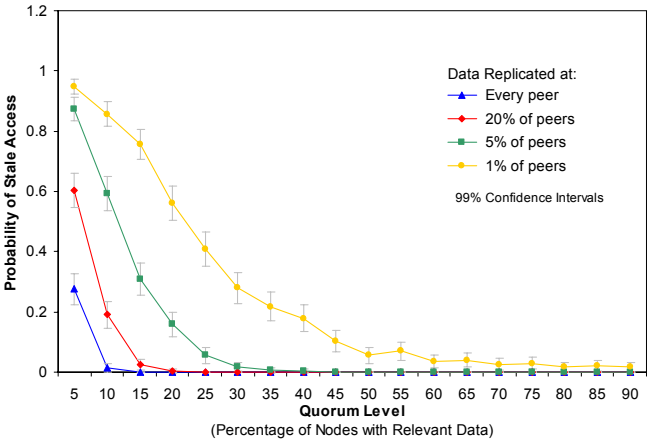

Figure 6: Probability of stale data access for different quorum and data replication levels.

the trade-off between latency and data freshness. While many of the replication levels have good response times when quorum levels are low, as quorum levels (and data confidence) increase, some redundancy-related differences start to emerge. The response times of lower replication levels tend to be better, merely because fewer nodes (in absolute terms) are needed to achieve a quorum. At the high data freshness end of the spectrum, you're forced to settle for worse response times if you need more redundancy. Note, with data replication levels of 1% or below, the number of nodes with relevant data was so small that quorum levels above 70 percent became unattainable. 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 choose quorum levels for a given peer it is important understand how such decisions affect the probability of reading an old data value. This experiment explored this data confidence, quorum threshold trade-off 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 number of nodes needed for a quorum. If the quorum level were set at 20%, only 20% of the relevant nodes are assumed to hold up-to-date values. Though relatively unlikely in a real-world network, this is useful for illustrating the general relationship between quorum thresholds and data confidence (Figure 6). We would expect the probabilities of stale data access in a real network with decent update propagation to be even lower than those shown.

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. So, 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*

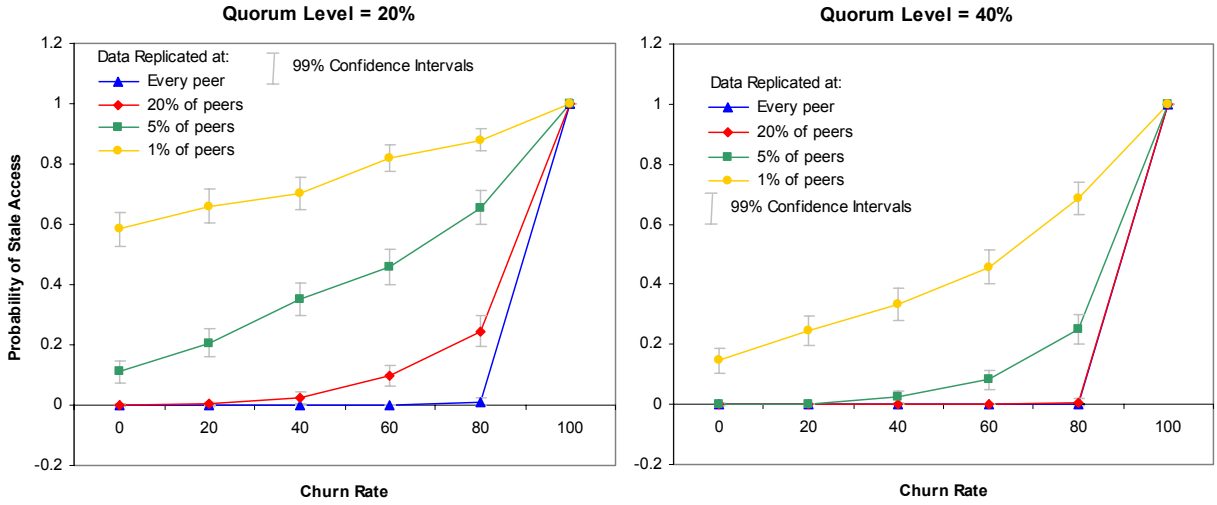

Figure 7: Probability of stale data access (for different quorum and replication levels) as affected by churn rate.

rate. The churn rate represents the percentage of the network experiencing membership changes between queries. 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 size of simulated network and the data replication rate stay constant between queries while the versions of data items 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.

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 bring old values. Even at low quorum levels (like 20%), if the data is replicated at 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, since a 30% node turnover between queries not likely in a real network.

5.7 Summary of Tradeoffs

Degree of Data Replication	Quorum Level: Low	Quorum Level: 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

Our flexible quorum scheme is all about trade-offs. The table summarizes of how quorum and data replication levels can be adjusted to meet the performance goals in each quadrant. How might these parameters and confidence levels be used in the context of various applications? In Section 1.1, we suggested a P2P database application involving shared inventory data among retailers and suppliers. Such data might reside on a dedicated peer-to-peer network, with fast connections between a small set of peers. The quorum and replication levels could be set fairly high (upper right box) ensuring excellent data confidence. The small network size would tend to keep the network traffic low, despite the extra quorum assembly effort.

While manipulating data replication levels makes sense for a privately managed network, a totally open system often renders this impossible. Although, it might still be possible to manipulate the degree of replication somewhat, (e.g. creating several more peers and adding them to the network), with a large set of replicas, peers would have to rely on adjusting the quorum levels instead.

One such open application might be a distributed data crunching project along the lines of SETI@home [2]. The confidence in individual accesses might be less important, instead a reliance on multiple computations of same data set is used for confidence. If peers wrote and shared partial computation results, there might be relatively frequent writes in addition to reads. Due to the potentially large size of the network, degree of replication might difficult to influence, but low network traffic would probably be desirable. To achieve this, peers could choose low quorum levels when the replication level is high (upper left quadrant), and slightly higher quorum levels when the replication level dropped (between the bottom two quadrants).

Much scientific research (like genomic sequencing) also benefits from data sharing. Such networks would be tend to be semi-private, with more parameter control than SETI@home, but perhaps less than the supply chain network. The size of the network might similarly fall in the middle-ground. Fortunately, with the flexible P2P update system we've been discussing, it would still be possible to tune parameters to meet users' needs.

6. Conclusions and Future Work

Though challenging, managing updates in peer-to-peer database systems is an important practical concern. The weighted quorum consensus scheme, adapted for P2P networks has great potential here. If global replica knowledge is known, P2P quorum consensus can provide localized consistency guarantees. More commonly, without global knowledge, 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 flexibility, a truly adaptable approach.

Still, there are a number of facets of P2P data updates that remain unexplored, and would provide ample

opportunities for future pursuit. One already mentioned is update propagation. While a variety of strategies could be employed, each would doubtless impact update performance in the areas of network traffic, query response time and stale access probability. One 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, important data with higher availability requirements could be located on superpeers. Such nodes would be expected to have a larger allocation of votes to reflect their superior reliability. What affect such strategies would have on performance remains to be seen. On a related note, it might be worth exploring how underlying network protocols affect performance. Several promising P2P protocols have come to light in recent years; some might prove better than others for P2P data updates.

In this paper we looked at how adjusting the quorum and data replication levels can affect response times and data confidence. Ideally, however, users would first decide on their data confidence requirements, then ask the network or individual peers to choose quorum and replication levels to minimize response times. Since users often only control a small set of nodes, global optimization of this sort could be tricky. Parameters need not be static either, but could conceivably be adjusted in real-time to meet changing needs. Although such intelligent peer behavior is not without its challenges, even without it, flexible quorum consensus seems well-suited to the dynamic, heterogeneous world of P2P networks.

7. References

[1] D. Agrawal and A.E. Abbadi. *The tree quorum protocol: an efficient approach for managing replicated data*. 16th Int'l Conference on Very Large Databases, 1990, 243-254.

[2] D. P. Anderson, et al. *SETI@home: An Experiment in Public-Resource Computing*. Communications of the ACM, 45 (11), November 2002, pp. 56-61.

[3] H. Balakrishnan, et al. *Looking up data in P2P systems*. Communications of the ACM. 46 (2), 2003, pp. 43-48.

[4] 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.

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

[6] Y. Breitbart, et al. *Update Propagation Protocols for Replicated Databases.* ACM SIGMOD Int'l Conference on Management of Data, 28 (2), June 1999, pp. 97-108.

[7] S.Y. Cheung, et al. *The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data*. 6th Int'l Conference on Data Engineering, Feb. 1990, 438-445.

[8] B. F. Cooper and H. Garcia-Molina. SIL: Modeling and Measuring Scalable Peer-to-Peer Search Networks. Stanford Technical Report, TR 2003-53, 4 August 2003.

[9] 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.

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

[11] F. Giunchiglia and I. Zaihrayeu. *Making Peer Databases Interact - A Vision for an Architecture Supporting Data Coordination*. Conference on Information Agents, Madrid, September 2002.

[12] S. Gribble, et al. *What Can Peer-to-Peer Do for Databases, and Vice Versa?* 4th Int'l Workshop on the Web and Databases (2001).

[13] A. Kumar. *Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data*, IEEE Transactions on Computers, Sept. 1991, pp. 996-1004.

[14] R. Jimenez-Peris, et al. *Are Quorums an Alternative for Data Replication?* ACM Transactions on Database Systems (TODS), 28 (3), September 2003, pp. 257-294.

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

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

[17] T. Klingberg, et al. *RFC-Gnutella 0.6.* http://rfc-gnutella.sourceforge.net/developer/testing/ index.html.

[18] J. Lu and J. Callan. *Content-Based Retrieval in Hybrid Peer-to-Peer Networks*. 12th Int'l Conference on Info. and Knowledge Management, November 2003, pp. 199-206.

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

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

[21] M. Rodrig and A. LaMarca. *Decentralized Weighted Voting for P2P Data Management*. MobiDE 2003. San Diego, CA.

[22] R. Schollmeier and G. Schollmeier. *Why Peer-to-Peer (P2P) Does Scale: An Analysis of P2P Traffic Patterns.*2nd Int'l Conference on P2P Computing, Sep. 2002, 112-119.

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

[24] 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.

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

[26] Z. Zhang, et al. *Scalable, Structured Data Placement over P2P Storage Utilities*. HP Labs Technical Report, 2002, www.hpl.hp.com/techreports/2002/HPL-2002-40.pdf.