Efficient Hierarchical Quorums in Unstructured Peer-to-Peer Networks

Kevin Henry, Colleen Swanson, Qi Xie, and Khuzaima Daudjee

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
{k2henry,c2swanso,q7xie,kdaudjee}@uwaterloo.ca

Abstract. Managing updates in a peer-to-peer (P2P) network can be a challenging task, especially in the unstructured setting. If one peer reads or updates a data item, then it is desirable to read the most recent version or to have the update visible to all other peers. In practice, this should be accomplished by coordinating and writing to only a small number of peers. We propose two approaches, inspired by hierarchical quorums, to solve this problem in unstructured P2P networks. Our first proposal provides uniform load balancing, while the second sacrifices full load balancing for larger average quorum intersection, and hence greater tolerance to network churn. We demonstrate that applying a random logical tree structure to peers on a per-data item basis allows us to achieve near optimal quorum size, thus minimizing the number of peers that must be coordinated to perform a read or write operation. Unlike previous approaches, our random hierarchical quorums are always guaranteed to overlap at at least one peer when all peers are reachable and, as demonstrated through performance studies, prove to be more resilient to changing network conditions to maximize quorum intersection than previous approaches with a similar quorum size. Furthermore, our two quorum approaches are interchangeable within the same network, providing adaptivity by allowing one to be swapped for the other as network conditions change.

1 Introduction

Peer-to-peer (P2P) networks have become increasingly attractive over the last few years; systems such as Gnutella [1] and Kazaa/FastTrack [2] are used extensively for file-sharing over the Internet. The underlying idea of having a completely decentralized, dynamic configuration of heterogeneous peers promises several benefits over more traditional approaches, which are limited by the need for costly, high-maintenance central servers that often act as a bottleneck to the scalability of the system. P2P systems, on the other hand, can distribute workloads among all peers according to resource capabilities, thereby allowing the network size to grow. Moreover, with many nodes in the network, P2P systems can achieve high availability and redundancy at a lower cost.

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 183–200, 2009.
© Springer-Verlag Berlin Heidelberg 2009

An open question is how to efficiently use P2P networks in the distributed database setting. Several issues must be considered, including appropriate data placement and replication strategies, how best to achieve coordination among peers, and in particular, how to handle updates. It is this last issue that is the main focus of this paper; we seek to ensure consistency of updates in a completely decentralized, unstructured P2P setting. We devise a hierarchical quorum system for managing updates that i) requires no coordination among the nodes beyond the usual query messages and responses, ii) requires writes at only a small percentage of data copies (as low as 10% when replication levels are high), iii) guarantees that all requests will see an up-to-date version of the data in the absence of node failure, and iv) achieves a high probability of accessing up-to-date data even when the network suffers from a high degree of instability caused either by *network churn* (peers entering and leaving the network at a normal rate) or *peer failure* (peers leaving the network without new peers joining to maintain network connectivity).

Our work has been inspired by that of Del Vecchio and Son [3], who consider the problem of applying a traditional quorum consensus method to an unstructured P2P distributed database system. Our methods differ significantly from theirs, however, as they only give probabilistic estimates as to the likelihood of seeing up-to-date copies in a fully-functioning network. In addition, we achieve small quorums without sacrificing consistency, whereas Del Vecchio's scheme relies on a trade-off between quorum size and the probability of accessing stale data. Other than [3], little research has been done on quorum-based P2P update management, and none of the other schemes address the unstructured P2P setting. Instead, most relevant work relies heavily on structured P2P systems, such as those with distributed hash tables (DHTs). It is an interesting question whether or not there is a practical method of handling updates without the use of an overlay network such as a DHT, and it this problem that we explore in this paper.

We motivate our approach with a brief discussion of quorums and present Del Vecchio's flexible quorum system in Sect. 2, before giving a detailed description of our quorum selection algorithms in Sect. 3. We then analyze the performance of our system over a simulated Gnutella network in Sect. 4, and give a detailed comparison of our results with Del Vecchio's in Sect. 5. After presenting other relevant work in Sect. 7, we give concluding remarks and comment on future research directions in Sect. 8.

2 Quorums

Quorums are a well-known method of coordinating read and write operations on multiple copies of a data item. Each copy of the data item is assigned a certain number of votes, and to perform a read or write operation, a corresponding read or write quorum must be formed by assembling a given number of these votes, specified as the read/write quorum level (r and w, respectively). The underlying principle of a *correct* quorum system is that any two quorums intersect; in this

way, an up-to-date version of the data copy will always be present in any quorum. We remark that the smaller the quorum size (i.e. the fewer required votes), the more efficient the quorum system; we always seek to minimize the amount of communication necessary to perform a read or write operation.

The traditional, or majority, quorum consensus method relies on the rules $r + w > n$ and $2w > n$, where n is the number of copies of the given data item. These rules guarantee the desired quorum intersection property, thereby ensuring data consistency. An obvious question is how we can effectively relax these consistency guarantees—for example, if we can tolerate reading stale data from time to time, perhaps we can make the read quorum more efficient by reducing the number of votes, while still keeping the probability of reading an up-to-date copy at an acceptable level for the given application.

Del Vecchio and Son [3] investigate this issue in the P2P setting using a probabilistic, flexible version of the traditional quorum consensus method. That is, they investigate the probability of accessing stale data when write quorum levels do not require a majority of the data copies, and experiment with different quorum levels in a simulated Gnutella network. Their results are promising, in that, if the data is replicated at a relatively large percentage of the peers (20% or greater), the probability of accessing stale data is low even with quorum levels as low as 20% and the presence of network churn. We refer to the quorum system of Del Vecchio and Son as the *flexible quorum* approach, since their basic idea is to allow peers to vary quorum levels; a more detailed look at the probabilistic guarantees their scheme provides is given in Sect. 5.

Given the unpredictable nature of P2P networks, some sacrifice of consistency is necessary: we will always be faced with the possibility of peer failure and network churn, so we cannot guarantee that even one up-to-date copy of a file is present at a given time. Whereas Del Vecchio chooses to sacrifice consistency in order to lower quorum size, we focus on achieving small quorums while relaxing consistency *only* in the presence of network churn or peer failure.

In this paper, we propose two *hierarchical quorum consensus* algorithms for managing updates in unstructured P2P systems. We propose a scheme that builds on a hierarchical scheme [4], which imposes a logical multi-level tree structure on the location of data copies and recursively applies the traditional quorum consensus method to each level. That is, data copies are viewed as leaves of an n-level tree, and for each level i $(i = 1, 2, \ldots, n)$, a read quorum r_i and a write quorum w_i of nodes are assembled recursively that satisfy the following two properties: $r_i + w_i > \ell_i$ and $2w_i > \ell_i$, where ℓ_i is the number of nodes at level i of the tree. In particular, by recursive we mean that at each level i, we are assembling a quorum from the children of the nodes in the quorum at level $i - 1$. A relatively straightforward proof by induction shows the correctness of this quorum method; we shall not reproduce this result here. It is further shown in [4] that the optimal quorum size can be achieved using ternary, or degree three, trees, giving quorum sizes of $n^{0.63}$ copies as opposed to $\lceil \frac{n+1}{2} \rceil$ copies as in traditional quorum consensus. We describe our hierarchical P2P quorum system in the next section.

3 Hierarchical P2P Quorums

The benefits of the hierarchical quorum approach come at the cost of requiring a fixed tree structure. For this reason hierarchical quorums cannot be directly implemented in an unstructured P2P network. In this section two variants of the hierarchical quorum approach are developed that allow peers to randomly, but deterministically, select their position in a tree without any communication or coordination between themselves. Our approach is probabilistic in the sense that the tree structure is random; however, it has the property of failing gracefully toward the majority quorum in the worst case. Unlike the flexible quorum approach described in the previous section, our quorums are guaranteed to overlap when all peers are reachable.

Our first and simplest approach, the *random hierarchical quorum*, assigns peers to locations in the tree randomly and recursively builds quorum by selecting a random majority at each level. The second approach, the *hybrid hierarchical quorum*, utilizes a combination of random and fixed traversals of the tree in a manner that increases the expected intersection size, while still selecting most members of the quorum at random.

3.1 Random Hierarchical Quorums

Let M be an upper bound on the number of peers in the network; we assume this value is a public parameter. Let T be a complete ternary tree with depth d defined such that there are 3^d leaf nodes, where d is the smallest integer such that $3^d > M$, i.e., $d = \lceil \log_3 M \rceil$. The levels of the tree are labeled $0, \ldots, d$, with the root located at level 0 and the leaves at level d; the leaves of the tree are indexed $0, \ldots, 3^d - 1$. Peers will be located at level d of the tree. Since the network is unstructured, peers determine their own location in the tree by generating a random integer modulo 3^d for each data item. We recommend using a hash function, such as the SHA family [5], to compute location as a function of a peer's network address and the primary key of a given data item. In this manner, each tree is uniquely determined by the primary key of the data item, and each peer's location within the tree is determined by its network address, thereby allowing any peer to compute the location of any other peer without any communication. Given that each peer selects its position in the tree at random, it is expected that several peers will select overlapping locations, and also that several locations in the tree will be empty. Thus, instead of treating each leaf as a single peer, each leaf corresponds to a (possibly empty) set of peers. No peers are located at non-leaf positions in the tree, but one may conceptually visualize each non-root position in the tree as the set of all leaves/peers rooted at that position.

In our protocol, each data item is associated with a tree T as described above. Thus, each peer has a collection of tree locations: one for each data item the peer possesses. We comment that M may be chosen optimistically, as there is no penalty to over-estimating network size. Underestimating network size does not affect correctness, but will affect quorum size if the estimate is too low. In

particular, choosing M, and hence the depth of the tree, to be significantly larger than the size of the network has only the drawback of increasing the maximum size of the integer associated with each peer's location in the tree. On the other hand, choosing a very small tree depth increases the size of quorums.

Figure 1a shows a sample tree assignment on 9 peers labeled $A, \ldots, H$. The rest of Fig. 1 shows how the quorum generation algorithm, given by **RandQuorum**, traverses the tree to build quorum; in each part of the figure, the nodes selected by **RandQuorum** are bold. A summary of the algorithm, together with the running example from Fig. 1, is as follows: the peer wishing to establish quorum sends a search query into the network to locate peers containing relevant data and their corresponding tree positions. (Alternatively, the locations could be computed by the peer receiving the responses, if location is computed using a hash of a peer's network address.)

Once all responses have been received, a quorum is generated by starting at the root of T and recursively building quorum at each level. There are two possible cases: 1) If there are two or more non-empty children (i.e. the number of the peers rooted at the child is non-zero), then two non-empty children are selected at random and the process is repeated at each selected child (lines 1-4). We see this in Fig. 1b; here, the leftmost and rightmost children on the first level are chosen. Similarly, we see that the leftmost node of the first level has three non-empty children, so we choose two of them at random, namely the middle and rightmost (shown in Fig. 1c). 2) If there are not at least two non-empty children, the union of all peers rooted at the current position in the tree is selected and a majority is returned (lines 6-9). We see this case in the rightmost node of the first level of Fig. 1b. Here, as shown in Fig. 1c, the set of all peers rooted at this point are collected into a single set and a random majority is returned.

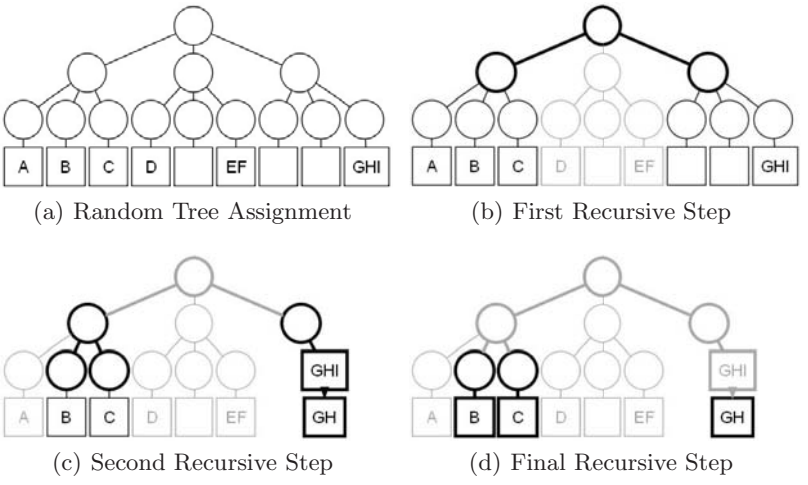
The algorithm terminates when the bottom of the tree has been reached, and the set of all peers returned by the algorithm forms the quorum. We see this in Fig. 1d: since neither selected subtree (from Fig. 1c) has at least 2 non-empty children, a random majority of the peers rooted at this point are returned. The union of all peers returned by the algorithm $\{B, C, G, H\}$ forms the quorum.

Algorithm 3.1: **RandQuorum**

Input: A non-empty ternary tree T

Output: A random hierarchical quorum on T

1. **if** T has two or more non-empty children **then**
2. Select two distinct non-empty children T_1 and T_2 at random
3. $S_1 \leftarrow$ **RandQuorum**(T_1) {Recursively build quorum on T_1}
4. $S_2 \leftarrow$ **RandQuorum**(T_2) {Recursively build quorum on T_2}
5. **return** $S_1 \cup S_2$ {Combine quorums generated from T_1 and T_2}
6. **else**
7. $S \leftarrow$ **LEAVES**(T)
8. **return** **MAJORITY**(S)
9. **end if**



(a) Random Tree Assignment (b) First Recursive Step

(c) Second Recursive Step (d) Final Recursive Step

Fig. 1. A sample run of **RandQuorum** on the set of peers $\{A, \ldots, I\}$. The bold path demonstrates how the tree is traversed at each recursive step, with the bolded boxes representing those peers returned by the algorithm. The union of the sets $\{B\}$, $\{C\}$, and $\{G, H\}$ forms the quorum.

RandQuorum makes use of two sub-routines, namely, **LEAVES**(T) and **MAJORITY**(S). The former returns the set of leaf nodes of T, while the latter returns a random majority quorum from the elements of S.

Recall that each peer selects its position in the tree independently from all others. Thus, while it is unlikely that this process results in a completely balanced tree, it is also unlikely that the result is a significantly unbalanced tree. At level k of the tree there are 3^k nodes, each the root of a subtree with 3^{d-k} leaves. If n peers randomly join the tree, then the probability that a subtree at level k is empty is $(1 - 3^{-k})^n$. For example, if the network contains 1000 peers, then a tree of depth 7 will be chosen. This tree has capacity for 2187 peers, and thus will always contain empty nodes at level 7. The probability that level 6 contains empty subtrees is approximately 25%, whereas the probability that level 5 contains empty subtrees is approximately 1%. Because the probability of empty subtrees becomes exponentially unlikely as one moves closer to the root, it is reasonable to assume that such randomly constructed trees will always be fully populated at a reasonable depth relative to the number of peers in the network. For this reason, our quorums are comparable in size to optimal hierarchical quorums, despite the fact that our trees are generated randomly.

Figure 2a shows the average size of quorums generated by our approach in comparison with the theoretical best performance of hierarchical quorums. For reference, a line representing the size of the quorum obtained from the majority quorum consensus algorithm is also included. These results were generated by taking the average size of a quorum returned by **RandQuorum** on 1000 different randomly constructed trees for each network size.

In the absolute worst case, each peer may choose the exact same location in the tree. In this case, **RandQuorum** is unable to move lower in the tree and simply returns a majority of the entire set of peers. Thus, the worst case performance is equivalent to the majority quorum consensus approach. Clearly, the event that each peer selects the same location in the tree is exceptionally rare, as discussed earlier in this section.

As peers enter and leave the network, the structure of the tree changes. Despite this, because each peer generates its position in the tree independently, the distribution of peers across the leaves of the tree will remain uniform over time. Thus, incoming peers do not need to assume a vacancy left by a peer that has just exited. This approach also has the benefit of being robust against failures, such as network outages or partitions, in the sense that there is no correlation between a peer's physical location in the network and logical location in the tree. After a large scale failure, the remaining peers will still be uniformly distributed among the leaves of the tree, and near-optimal quorum sizes will still be expected. We investigate the problem of strengthening the random hierarchical quorum against peer failure in the next section.

3.2 An Improvement: Hybrid Hierarchical Quorums

The random hierarchical quorum approach developed in the previous section guarantees that any two quorums generated from the same tree overlap at at least one point. In a fixed network, this would be sufficient to guarantee correctness; however, in a P2P network this is not sufficient. Between the generation of any two quorums, multiple peers may enter or leave the system. Furthermore, it cannot be guaranteed that every peer in the network can reach every other peer. Thus, if the average intersection size of two quorums is small, then even a relatively small number of peers becoming unreachable could cause two quorums to not overlap with unacceptably high probability. For this reason, it is desirable to maximize the intersection of any two quorums.

One method of maximizing quorum intersection is to always choose the same set of peers to be in a given quorum. This could be accomplished in the random tree approach by always selecting the same majority of nodes at each level of the tree. This approach always generates the same quorum for a given data item, with the same average quorum size as a random traversal, but it also means that only a fixed, small percentage of peers will ever be chosen to receive reads or writes for a given data item. Thus, a high resilience to peers becoming unreachable comes at the cost of focusing the load on small set of peers, rather than distributing the workload randomly among the available replicas. As peers are distributed randomly at the leaves of the tree, however, fixing the traversal to always select the left and middle children, for example, will still result in a different random quorum for each data item, as each data item is associated with a unique tree.

To compromise between maximizing quorum intersection and balancing load, we combine the load balancing of a random traversal with the large intersection of a fixed traversal in the following manner: at the root, we traverse the leftmost

subtree using a fixed majority at each level, and traverse one of the remaining two subtrees selecting a random majority at each level. This is summarized in the algorithms **HybridQuorum** and **FixedQuorum**.

FixedQuorum is identical to **RandQuorum** except that the algorithm proceeds recursively on the left and middle subtrees when both are non-empty, rather than on two random non-empty subtrees.

Algorithm 3.2: **HybridQuorum**

Input: A non-empty ternary tree T

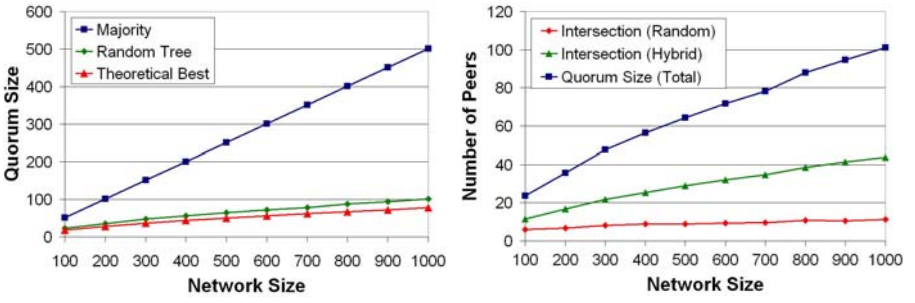
Output: A hybrid hierarchical quorum on T

1. **if** left subtree and at least one remaining subtree are non-empty **then**
2. $T_1 \leftarrow$ **LEFT**(T)
3. $T_2 \leftarrow$ a random non-empty non-left subtree
4. $S_1 \leftarrow$ **FixedQuorum**(T_1) {Build a fixed quorum from T_1}
5. $S_2 \leftarrow$ **RandQuorum**(T_2) {Build a random quorum from T_2}
6. **return** $S_1 \cup S_2$ {Combine quorums generated from T_1 and T_2}
7. **else**
8. $S \leftarrow$ **LEAVES**(T)
9. **return** **MAJORITY**(S)
10. **end if**

Figure 2b demonstrates the effectiveness of this approach. One can see that the intersection size is increased significantly compared to the randomly generated quorum, while still leaving plenty of room for load to be distributed among different peers in the tree. These results were generated by running either **RandQuorum** or **HybridQuorum** twice on the same tree and counting the number of common peers in each. This process was averaged over 1000 different trees.

One benefit of using the tree structure to select the fixed set of peers is that the intersection property is guaranteed between any two hierarchical based quorums, even if one is built using a random traversal, and the other a partially-fixed traversal. In a setting where peers are relatively stable, the random hierarchical quorum may be used, with the benefit of randomly distributing load among replicas of a data item. If network conditions change and a large number of failures or exiting peers are detected, peers can seamlessly switch to the hybrid hierarchical quorum, and all quorums generated from this point will be expected to intersect at a larger number of peers. Quorums generated prior to the switch will still be valid, although not as resilient to failure or changing network conditions.

Although the hybrid hierarchical quorum shifts extra load onto some fixed set of peers, it is important to recall that trees are built on a per-data item basis. Whether or not a given peer is a member of the fixed set for one data item does not have any bearing on whether or not it is a member of the fixed set for a different data item. The set of fixed peers for a given data item is chosen uniformly at random, and thus, although the load balance is skewed for any single data item, the load balance across all data items remains random.



(a) Average Quorum Size **(b)** Average Quorum Intersection Size

Fig. 2. (a) Comparison of average random tree quorum size to the majority quorum and optimal hierarchical quorums. Widest 99% confidence interval: ±0.60. **(b)** Comparison of average quorum intersection size for a random traversal and hybrid traversal of the tree. Generated by randomly assigning peers to 1000 different trees and averaging the intersection size of two quorums from each. Widest 99% confidence interval: ±0.64.

4 Performance Evaluation

In order to evaluate the performance of our random and hybrid hierarchical quorum approaches, we performed several experiments using the Gnutella protocol [1] simulated under NeuroGrid [6]. We note that the Gnutella protocol is a rather basic P2P system that relies on flooding the network with messages in order to locate documents; as such, the number of times a message is forwarded is limited by a time-to-live (TTL) parameter. The Gnutella protocol is well documented, widely used, and is completely unstructured in its basic form. Unlike some unstructured P2P protocols, such as FastTrack, Gnutella benefits from being an open specification. These properties make Gnutella an ideal candidate for testing the effectiveness of our random hierarchical quorum approaches. An additional benefit of performing our experiments using Gnutella is the ability to directly compare our results to Del Vecchio and Son [3].

4.1 Experimental Environment

Although NeuroGrid is a stand-alone P2P protocol, the NeuroGrid simulator provides functionality for simulating several P2P protocols, including Gnutella. A Gnutella network is modeled as a set of peers, each possessing one or more documents, and each document is associated with one or more keywords. Random networks are generated with respect to a connectivity parameter, specifying the number of connections a peer establishes with others when it joins the network. At network creation time, each peer chooses k distinct peers (with a default value of $k = 2$) and establishes a connection with each. Thus, each node has, on average, approximately $2k$ connections to other peers in the system. When implementing our algorithms, we noticed that the simulator disregards the connectivity specified in the configuration file and instead uses a hard-coded value

of 2 connections per node. For our simulations, we manually changed the connectivity of the network to $k = 3$. Our reasons for this choice are justified in Sect. 4.2

The main functionality provided by NeuroGrid is the ability to specify a distribution of documents across peers, perform a keyword search for a given document, henceforth referred to as a *query*, and generate result statistics, such as the number of peers with a match and number of messages sent between peers. By default, queries originate from a random peer and are for a single keyword. We set each node to contain a single document, with each document having a single keyword associated with it, and vary the number of keywords to model the replication rate of data items within the network. For example, 1 keyword implies this keyword is associated with every document (100% replication), whereas 5 keywords means each peer is assigned a given keyword with probability $\frac{1}{5}$, thus achieving 20% replication for each keyword. Because keywords are assigned randomly, we use a modified keyword distribution algorithm to guarantee exact replication rates.

For completeness, we also describe NeuroGrid's behavior with respect to the TTL of a message. The default TTL in NeuroGrid is 7, and the Gnutella specifications [7] state that the TTL is the "number of times the message will be forwarded by Gnutella servants before it is removed from the network." Each peer decrements the TTL by 1; if the TTL is non-zero and the message has not been seen before, the peer forwards the message to its neighbors. In NeuroGrid, the originator of the message decrements the TTL before forwarding it. Thus, a message with an initial TTL of 7 will only travel a maximum of 6 hops in NeuroGrid; we leave this functionality unchanged, as we find in Sect. 4.2 that 6 hops are sufficient for the network sizes we consider.

Unless otherwise specified, the results in this section are generated for a network size of 1000 peers, with replication rates of 1%, 5%, 20%, and 100%, generated as described earlier. When averaging our results, we sample 10 values for a given network, and then create a new network. This process is repeated 100 times for a total of 1000 random samples. The connectivity (number of connections each node establishes with others) is set to 3 and the default TTL is set to 7. For each data point a 99% confidence interval has been computed and a description of the largest interval accompanies each plot.

4.2 Network Reachability

In order to verify that our choice of network parameters (connectivity and TTL) is appropriate for a network of this size, we investigated the reachability of peers in the network as successively larger numbers of peers fail or leave the network. Reachability is determined by distributing a single keyword to all peers, terminating all connections for a fixed percentage of peers, and then counting the number of matches for this keyword when queried by a random non-failed peer. Our test included trials for connectivities of 2 and 3, respectively.

Our experiments showed that an initial connectivity of 3, or approximately 6 total connections per peer, is sufficient to retain access to virtually all peers in

the network so long as the failure rate is less than 30%. As the number of failed peers increases, the network becomes partitioned and the average reachability declines quickly. If the connectivity is 2, then full reachability is unlikely, and the number of reachable peers drops to less than 50% once 20% of peers have failed. It should be noted that in our simulation, peers did not attempt to establish new connections when a neighbor failed. Applying such a technique could maintain consistent connectivity in the presence of failure, so long as a large number of peers do not fail simultaneously, thereby partitioning the network before connectivity can be restored. By choosing to model peers that do not seek out new connections when neighbors fail, we are modeling the worst case performance under failure.

4.3 Cost of Communicating with Quorum

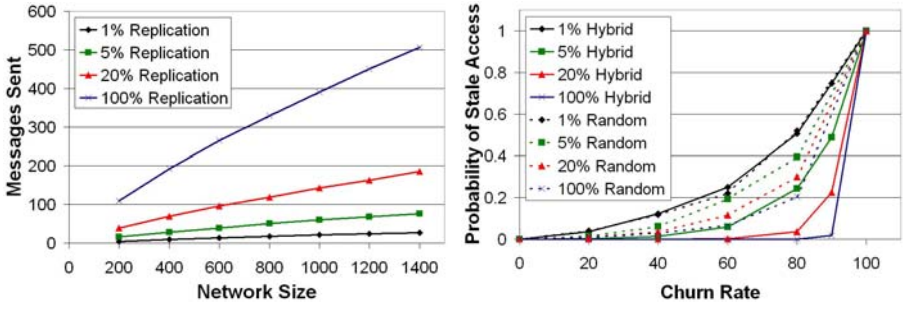
We can now determine the average cost for communicating with a quorum once it has been established. The communication cost for establishing the quorum takes exactly one query, the cost of which depends on the number of peers and the number of connections between them. The average cost of a query in the networks we consider is roughly constant; we therefore omit the messages involved in establishing quorum from our results.

Figure 3a was generated by recording the TTL of the query message for each peer possessing a document with the target keyword that received the query. From this, we calculate the number of hops necessary to reach each peer in the quorum. In Gnutella, communication with each peer occurs along the path from which it was located, so this gives the total cost of sending individual messages to each peer in the quorum as a function of hops. Optimizations may be possible if a single message may be used to communicate with all peers along a given path. In our simulation we observed that the number of messages required to communicate with the quorum is approximately 4 times the size of the quorum itself.

4.4 Stale Access under Churn

As mentioned in Sect. 3.2, a key goal is to compromise between maximizing quorum intersection and balancing load among replicas. This is motivated by the fact that in a P2P environment, peers are expected to continually leave and join the system. As in [3], we model this with network churn. The churn rate of a network can be defined as the number of peers that are expected to leave and join the system between queries for a given data item. We assume that the network size is constant, i.e., peers enter and leave at the same rate, and also that the replication rate of a data item stays constant. This is modeled by selecting a fixed percentage of peers at random and setting their data to be stale; that is, we assume that all peers rejoining the system have stale data.

Intuitively, the average intersection size of two quorums provides a good indicator of how resistant to churn a given quorum system is. Thus, we expect a

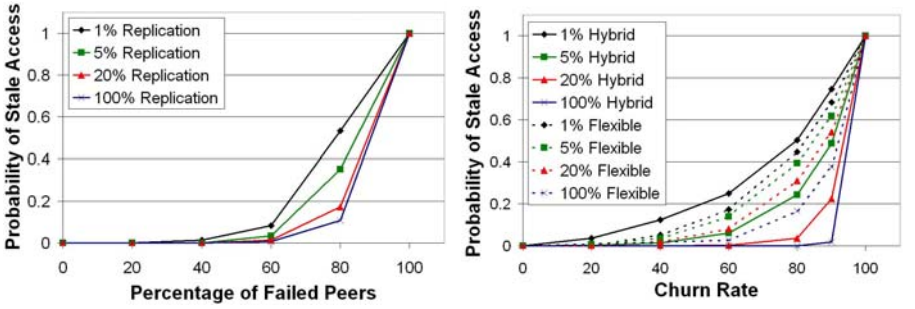


(a) Messages required to build quorum **(b)** Stale data access under node churn

Fig. 3. (a) The average cost of communicating with all peers in a quorum. Confidence intervals grow with the size of the quorum, ranging from ±0.2 to ±4.5 messages. **(b)** A comparison of the probability of stale data access under node churn for hybrid hierarchical quorums and random hierarchical quorums. Widest 99% confidence interval: ±0.04.

hybrid hierarchical quorum to perform much better on average than a random hierarchical quorum. Figure 3b demonstrates the performance of the hybrid hierarchical quorum for several levels of data replication. At 100% replication, we can tolerate 90% node churn without any significant probability of a quorum containing only stale data. This result may seem counterintuitive, but our simulation shows that hybrid hierarchical quorums intersect at over 40 peers on average, and thus, the probability that some of these peers remain after 90% churn is quite high. Indeed, the probability that a random set of 40 peers and a random set of 100 peers (i.e., the non-churned peers) do not intersect is approximately 1.4%. Stale access for lower replication rates is far more probable, as expected, although most replication rates can still handle a reasonable amount of churn.

As we would expect, random hierarchical quorums provide much less resilience against network churn. Figure 3b also shows a direct comparison of hybrid hierarchical quorums and random hierarchical quorums. Even at 40% churn, stale access begins to become a problem at 100% replication. Recall that the hybrid hierarchical increases the expected quorum intersection, but at the cost of requiring a small subset of the peers to always be part of the quorum. Figure 3b suggests that if the level of churn is low, then random hierarchical quorums are sufficient and in this case, load can be distributed uniformly among replicas. Conversely, if the level of churn is high, then the problem of stale data access can be mitigated by biasing the load slightly towards some peers through the use of hybrid hierarchical quorums. Our two quorum approaches have the property that any hybrid quorum and random quorum on the same tree will intersect. Thus, peers can default to the fully load-balanced random hybrid quorums, but switch to hybrid quorums if a large amount of network churn is detected.



(a) Stale Data Access Under Failure **(b)** Stale Data Access Under Churn

Fig. 4. (a) The probability of accessing stale data after a fixed percentage of peers have left the network. Widest 99% confidence interval: ±0.04. **(b)** A comparison of the probability of stale data access under node churn for hybrid hierarchical quorums and flexible quorums. Widest 99% confidence interval: ±0.04.

4.5 Stale Access under Failure

The tests for stale access under network churn are based on the assumption that peers enter and leave the network at the same rate, thus keeping the average connectivity of the network constant. In reality, network failures may cause a large number of peers to simultaneously become unreachable. The main difference between the results in this section and the previous section is that as the failure rate increases, the average connectivity of the network decreases. Once a significant portion of peers have failed, the network may become partitioned and full reachability is no longer expected.

Figure 4a shows the probability of stale access as an increasing number of peers fail for the hybrid hierarchical quorum. As expected, the probability of stale access becomes noticeable for lower failure rates than the corresponding churn rate. At 40% failure, the reachability of the network quickly begins to drop and the probability of stale access rises accordingly. In practice, we expect that large scale failure is detectable by individual peers, and that read/write operations are suspended if less than 50% of the expected peers are reachable. Del Vecchio and Son assume that the replication rate for a given data item is public knowledge. If this is the case, then quorum generation can be aborted if the number of responses to a query is significantly lower than expected.

5 Comparison with Flexible Quorums

To our knowledge, the only other attempt to apply quorums to unstructured P2P networks is the flexible quorum approach of Del Vecchio and Son [3]. As previously discussed, we have chosen to evaluate our approach in many of the same situations so that we can directly compare the performance of the two approaches, although some considerations must be made. In the flexible quorum

approach, the size of the quorum is a free parameter of the system and may be selected to provide the desired level of stale access. Because each flexible quorum consists of a fixed percentage of all replicas, this requires that the replication rate and total number of peers are known by each peer (or at least, that a reasonable estimate is known). In our hierarchical-based quorums, the size of the quorum is not a free parameter, but instead is a function of the replication rate; our approach constructs a quorum out of the entire set of peers that respond to a query. As seen in Fig. 4a, for 1000 peers with 100% replication, the probability of our biased tree quorums not intersecting at 50% failure is near 0. A flexible quorum with a quorum level of 60% would never succeed in establishing quorum at this failure rate, as fewer than 60% of the peers remain in the network. Additionally, the assumption that the replication rate is known allows peers to detect large failures, such as network partition, based on the number of responses to a query. In the presence of a large number of failures, read/write operations could be suspended until connectivity is restored.

A benefit of our hierarchical-based quorums over flexible quorums is that we can *guarantee* that two quorums always overlap if all peers are reachable. In order to guarantee intersection, a flexible quorum must contain at least 50% of the replicas. For 1000 peers with 100% replication, our tree quorums require approximately 10% of peers. It should be noted that, with high probability, flexible quorums do provide a low probability of stale access for quorum sizes smaller than 50%, but there is no guarantee. This behavior is shown in Fig. 4b, which directly compares our hybrid hierarchical quorum to flexible quorums in the presence of network churn.

Since quorum size is not a free parameter in our quorums, in order to compare our approach to flexible quorums, we first computed the expected quorum size of a hybrid hierarchical quorum for the given replication rate. To generate the probability of stale access for flexible quorums, we used quorums of the same average size as our tree quorums. For 1000 peers, the quorum sizes selected were 9.6%, 17.5%, 30%, and 60% for replication rates of 100%, 20%, 5%, and 1%, respectively. For all replication rates above 1%, biased tree based quorums provide a much lower probability of stale access. At the 1% replication level, there are too few peers to see any benefit from using a tree-based approach over the simple majority quorum. Comparing Fig. 3b and Fig. 4b, we see that random tree quorums are similar in performance to flexible quorums in general.

Another comparison point is the time necessary to build quorum, i.e., the query-response time, as a function of quorum size. Del Vecchio and Son observe that smaller quorums can be assembled faster than larger quorums, as it takes less time to hear a response from 10% of replicas than 90% of replicas. In comparison, our tree quorums require the full response set from the query before proceeding. In practice, taking the first set of responses to a query creates a bias towards peers located closer to the query originator. If two peers on opposite sides of a large network each assemble their closest 100 neighbors, then the probability of these two quorums overlapping in a large network is much less than if two nearby peers assemble quorums. To eliminate this bias towards

nearer neighbors, it is necessary to wait for the entire set of responses from the query before choosing a random quorum. We note that it is unclear whether Del Vecchio's experiments take this local bias into account; in particular, the experimental section does not specify whether random peers are used for each query, or if the same peer is used for repeated queries. In generating the above flexible quorum results, we used a new peer chosen at random for each query.

We remark that in a real-word scenario peers are limited to a local view of the network and so must use estimates of current network size and document replication rates, which can be obtained via gossip protocols [8], in order to decide on an appropriate time to build quorum. In addition, given that both our hierarchical-based quorums and Del Vecchio's flexible quorums are built by first querying the network, followed by contacting a certain percentage of peers, we expect the number of messages required by each to be approximately equal, particularly if local bias is eliminated. For these reasons, we feel that comparing the two approaches based on quorum size is reasonable.

6 Discussion

Our hierarchical systems are quite flexible in that quorums generated from the random and hybrid algorithms are interchangeable; peers can switch dynamically from one system to the other. Both the random and hybrid hierarchical quorums use the same randomly constructed logical tree structure, which carries the benefit of distributing peers in a ternary tree such that their logical location in the tree is unrelated to their physical location in the network. This means that correlation between peers failing in the physical network does not translate into localized failure within the tree. As the network structure changes, the expected distribution of peers within the logical tree remains constant.

The key difference between the random and hybrid hierarchical quorums is in the traversal of the tree. The random approach chooses a random majority, resulting in a completely random quorum, which has the effect of distributing reads/writes equally across all replicas of a data item. The hybrid approach traverses one subtree with a fixed majority and another with a random majority at each level, and thus shifts some of the load to a fixed set of peers for that data item. The effect of this shift is an increase in the expected size of the quorum intersection, and hence a higher tolerance to network churn and failure. Despite the non-uniform load balance, the fact that a different random tree is used for each data item allows the hybrid hierarchical quorum to still retain load balancing across all data items in the network.

We emphasize that there is a tradeoff between increasing the expected intersection size of two quorums and load balancing when we use the hybrid hierarchical approach. In our current method, although we fix the traversal over part of the tree, thereby increasing the expected intersection size of two quorums, we still leave a large proportion of the peers in sections of the tree that are traversed randomly. We remark that by changing the degree to which the traversal is fixed, various average quorum intersection sizes can be achieved. Since fixing a subset

of the peers has the affect of reducing the load balancing provided by our random tree assignment, it may be worth investigating the optimal intersection size for common sets of network parameters. For example, if network churn could be estimated ahead of time, a traversal strategy could be chosen that minimizes the fixed portion of peers, while still providing adequate stale access probabilities.

A drawback of our approach is that the entire network must be queried each time a quorum is constructed. However, the cost of sending messages is negligible compared to the cost of propagating and applying updates and our quorum approach minimizes the size of quorums, thus minimizing the number of peers an update must be sent to and thereby reducing the total cost of propagating and applying updates. A peer could choose to cache the results of a query, or more specifically the tree associated with a given data item. In most practical settings, however, it is unlikely that a peer could make use of a cached result before the instability of the network renders it invalid.

An interesting direction for future work is to consider the presence of Byzantine peers in the network. Such peers can misrepresent the freshness of their data or selectively choose not to forward messages in the network. Our hybrid hierarchical quorum could be used to guard against a small number of Byzantine peers, as the probability of stale access is low even if a large number of peers have failed or possess stale data. Because hybrid quorums are expected to intersect at a large number of peers, we could use a naive k-of-n voting protocol to combat Byzantine peers. In such an approach, a quorum is only accepted as valid if at least k of the n members agree on a specific version of the data.

7 Related Work

Much work has been done regarding the use of quorums in distributed database systems; a comprehensive survey and comparison of many quorum systems, including majority quorum consensus and weighted voting quorums [9], hierarchical quorum consensus [4], grid schemes [10,11,12], and tree quorum protocols [13] may be found in [14]. Wool [15] discusses general issues related to the usefulness of quorums for managing data replication and Naor and Wool [11], in addition to presenting the grid-based Paths quorum system, give a summary of the general quorum tradeoff between load balance and availability. We note that our choice of hierarchical quorum systems relies on its use of a logical structure in which responsibility is distributed symmetrically; as all copies of a data item are located at the leaves of the tree, there is no real hierarchy among the data copies themselves. This is in contrast to tree quorum protocols, for example, in which each node of the tree is assumed to have a copy of the data item and nodes higher in the tree are more important. Moreover, while grid protocols, in which nodes are organized into various grid patterns, appear promising at first glance, these protocols tend to suffer from poor availability in the presence of write operations.

Papers dealing with variations of the hierarchical quorum approach to the distributed database setting include [16], which analyzes multilevel, or hierarchical schemes, concentrating on the problem of determining the most suitable variant

for a given application, and [17], which examines a layered multilevel quorum scheme that recursively assembles quorums by alternating between a read-one write-all (ROWA) and majority quorum consensus strategy.

Apart from the flexible quorum approach, research has focused on the application of quorum systems to *structured* P2P networks. Baldoni et al. [18] investigate variants of the hierarchical quorum scheme in Chord [19], a DHT-based P2P system. Their approach relies heavily on the infrastructure of the DHT itself, and so is not applicable to the unstructured setting we consider. In particular, we remark that, in order to make their generalization of the hierarchical quorum system more compatible with Chord's infrastructure, they sacrifice optimal quorum sizes by using degree four trees rather than degree three. Additionally, DHTs like Chord map a keyspace onto a set of peers, whereas in our setting a peer possesses a set of data upon entering the network and is only required to participate in reads and updates for those data items it already possesses. Other examples of DHT-based quorum systems include [20,21,22,23]; as these methods are both unrelated to the hierarchical quorum system and heavily dependent on the infrastructure of the P2P network, we do not discuss them here.

8 Concluding Remarks

We have presented two new approaches to establishing read/write quorums in an unstructured P2P network: the random hierarchical quorum and the hybrid hierarchical quorum. These quorum systems are interchangeable (i.e., a random quorum will always intersect a hybrid quorum), which allows for a seamless transition from one to the other. We have investigated the performance of our random and hybrid hierarchical quorums and demonstrated that our hybrid hierarchical quorum is highly resilient against network churn, allowing one to retrieve fresh data even at churn rates of 90% or when up to half of the network has failed. Our observations show superior performance over previous quorum approaches in unstructured P2P networks. Furthermore, under realistic network sizes, we achieve a lower probability of stale data access for quorums of similar size.

References

1. Kirk, P.: RFC-Gnutella 0.6, `http://rfc-gnutella.sourceforge.net/index.html`
2. Liang, J., Kumar, R., Ross, K.W.: The fasttrack overlay: a measurement study. Comput. Netw. 50(6), 842–858 (2006)
3. Del Vecchio, D., Son, S.H.: Flexible update management in peer-to-peer database systems. In: IDEAS 2005: Proceedings of the 9th International Database Engineering & Application Symposium, Washington, DC, USA, pp. 435–444. IEEE Computer Society Press, Los Alamitos (2005)
4. Kumar, A.: Hierarchical quorum consensus: A new algorithm for managing replicated data. IEEE Trans. Comput. 40(9), 996–1004 (1991)
5. National Institute of Standards and Technology. FIPS 180-2, secure hash standard, federal information processing standard (FIPS), publication 180-2. Technical report, Department of Commerce (August 2002)

6. Joseph, S.: Neurogrid simulation setup, http://www.neurogrid.net/php/simulation.php
7. Kirk, P.: Gnutella protocol development: Standard message architecture, http://rfc-gnutella.sourceforge.net/developer/testing/message-Architecture.html
8. Kostoulas, D., Psaltoulis, D., Gupta, I., Birman, K.P., Demers, A.J.: Active and passive techniques for group size estimation in large-scale and dynamic distributed systems. J. Syst. Softw. 80(10), 1639–1658 (2007)
9. Gifford, D.K.: Weighted voting for replicated data. In: SOSP 1979: Proceedings of the seventh ACM symposium on Operating systems principles, pp. 150–162. ACM, New York (1979)
10. Cheung, S.Y., Ammar, M.H., Ahamad, M.: The grid protocol: A high performance scheme for maintaining replicated data. IEEE Trans. on Knowl. and Data Eng. 4(6), 582–592 (1992)
11. Naor, M., Wool, A.: The load, capacity, and availability of quorum systems. SIAM J. Comput. 27(2), 423–447 (1998)
12. Kumar, A., Rabinovich, M., Sinha, R.K.: A performance study of general grid structures for replicated data. In: Proceedings the 13th International Conference on Distributed Computing Systems, May 1993, pp. 178–185 (1993)
13. Agrawal, D., El Abbadi, A.: The tree quorum protocol: an efficient approach for managing replicated data. In: Proceedings of the Sixteenth International Conference on Very Large Databases, pp. 243–254. Morgan Kaufmann Publishers Inc., San Francisco (1990)
14. Jiménez-Peris, R., Patino-Martínez, M., Alonso, G., Kemme, B.: Are quorums an alternative for data replication? ACM Trans. Database Syst. 28(3), 257–294 (2003)
15. Wool, A.: Quorum systems in replicated databases: science or fiction. Bull. IEEE Technical Committee on Data Engineering 21, 3–11 (1998)
16. Freisleben, B., Koch, H.-H., Theel, O.: Designing multi-level quorum schemes for highly replicated data. In: Proc. of the 1991 Pacific Rim International Symposium on Fault Tolerant Systems, pp. 154–159. IEEE Computer Society Press, Los Alamitos (1990)
17. Freisleben, B., Koch, H.-H., Theel, O.: The electoral district strategy for replicated data in distributed systems. In: Proc. of the 5th Intern. Conference of Fault-Tolerant Computing Systems, pp. 100–111 (1991)
18. Baldoni, R., Jiménez-Peris, R., Patino-Martínez, M., Querzoni, L., Virgillito, A.: Dynamic quorums for DHT-based enterprise infrastructures. J. Parallel Distrib. Comput. 68(9), 1235–1249 (2008)
19. Brunskill, E.: Building peer-to-peer systems with chord, a distributed lookup service. In: HOTOS 2001: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, Washington, DC, USA, p. 81. IEEE Computer Society, Los Alamitos (2001)
20. Zhang, Z.: The power of DHT as a logical space. In: IEEE International Workshop on Future Trends of Distributed Computing Systems, pp. 325–331 (2004)
21. Lin, S., Lian, Q., Zang, Z.: A practical distributed mutual exclusion protocol in dynamic peer-to-peer systems. In: Voelker, G.M., Shenker, S. (eds.) IPTPS 2004. LNCS, vol. 3279, pp. 11–21. Springer, Heidelberg (2005)
22. Naor, M., Wieder, U.: Scalable and dynamic quorum systems. Distrib. Comput. 17(4), 311–322 (2005)
23. Silaghi, B., Keleher, P., Bhattacharjee, B.: Multi-dimensional quorum sets for read-few write-many replica control protocols. In: Fourth International Workshop on Global and Peer-to-Peer Computing (2004)