Dynamic Quorums for DHT-based P2P Networks[*]

Roberto Baldoni[1], Ricardo Jiménez-Peris[2], Marta Patiño-Martínez[2], Leonardo Querzoni[1] and Antonino Virgillito[1]

[1] Dipartimento di Informatica e Sistemistica - Università di Roma “La Sapienza”, Roma, Italy
{baldoni,querzoni,virgi}@dis.uniroma1.it

[2] Universidad Politécnica de Madrid (UPM), Madrid, Spain
{rjimenez,mpatino}@fi.upm.es

Abstract

Peer-to-peer systems (P2P) have become a popular technique to architect decentralized systems. However, despite its popularity most P2P systems consist in simple applications such as file sharing or chat systems. The main reason is that more complex applications require levels of consistency that nowadays are not offered by P2P systems. In this paper, we explore how to provide consistency based on distributed mutual exclusion via quorum systems in DHT-based P2P networks. Our results show that quorum systems applied directly to such networks are not scalable due to the high traffic imposed onto the underlying network. The paper introduces some design principles for both quorum systems and protocols using them that help to boost their performance. These design principles consist in dynamic and decentralized selection of quorums and in the exposition and exploitation of internals of the DHT such as the finger table. We show that by combining both design principles it is possible to minimize the number of visited sites and the latency needed to obtain a quorum.

1 Introduction

Despite the growing popularity of peer-to-peer (P2P) technology, its applicability remains constrained to simple systems such as file sharing and chat systems. Complex applications have not been built on top of P2P mainly due to the lack of consistency guarantees in such systems. Several research contributions aiming at increasing the consistency guarantees in P2P applications have appeared in recent years [14, 17, 5, 12, 1, 10], confirming the need for research on these topics. An example of more sophisticated infrastructure for P2P systems is Adlib [8] that provides support for complex index structures. In Adlib the configuration of the index structure, its hierarchical organization, should be stored (and maintained) in the underlying P2P network with very high resilience. Additionally, its consistency should be guaranteed in the presence of multiple writers. Typically this information is kept in a fraction of the network by resorting to hierarchical P2P systems, such as Crescendo [7], a hierarchical Chord ring.

In this paper, we focus on one of the main mechanisms used to attain consistency: mutual exclusion. P2P systems are inherently decentralized what means that approaches to mutual exclusion should be decentralized as well. In this paper we have chosen quorum systems for implementing distributed mutual exclusion. A quorum system over a set of sites consists of a set of mutually intersecting subsets of sites (quorums). Mutual exclusion based on quorum systems consists in requesting permission to a random quorum of sites. If all the sites in the quorum grant permission, no other quorum could have granted permission. Mutual exclusion can be used to solve conflicts in concurrent access to shared data. These accesses may occur once in a while in all common distributed applications deployed over a P2P system (such as distributed storage, trust management, multiplayer games).

A large variety of quorum systems have been proposed in the literature. Quorum systems have been traditionally compared in terms of availability, load [15] and the number of messages required to acquire a quorum (referred to in the following as *acquisition cost*). While load and availability depend on combinatorial properties of the quorum structure, acquisition cost is related to the number of messages required to contact all the sites in a quorum. That is, the message complexity or acquisition cost depends on how a quorum is obtained. In traditional distributed systems each site can access any other site directly (one message), while in a P2P system each site only knows a few neighbor

[*] This work has been partially funded by an Integrated Action Italy-Spain HI2003-0036 and the Spanish Research Council (MEC) TIN2004-07474-C02-01

sites to enable scalability in large scale settings. This means that the cost of accessing different sites is not fixed. A few sites are accessible within one hop, but the rest of the sites require routing through other sites and therefore, additional hops. Research in P2P systems has come out with routing infrastructures (Distributed Hash Tables - DHTs) in which the number of hops has an upper bound of $O(log(n))$ hops. In a P2P system, acquisition cost is strongly affected by the overhead induced by the routing in the P2P network.

The paper presents a general canvas in the form of a generic algorithm to reason and compare the different protocols for obtaining quorums in terms of acquisition cost and latency. We propose two design principles for quorum systems over P2P architectures, namely *delegation* and *integration*, and discuss their impact on both the acquisition cost and the latency. The first design principle lies in selecting quorums dynamically in a decentralized fashion. In other words, a set of delegated sites is responsible for selecting and obtaining a grant from a subset of the whole quorum on behalf of the requesting site. By forming quorums in this way, it becomes possible to have some flexibility in selecting the most convenient sites in order to reduce the acquisition cost. The second design principle consists in exposing internal information of DHT routing, more concretely, the finger table and the interval of keys for which a site is responsible. This information enables to know which sites can be reached directly and therefore to determine which are potentially most convenient. Hence, by combining the two design principles it becomes possible to minimize the number of visited sites and exchanged messages. We exercise these two design principles in two quorum systems. The first one, namely *farsighted*, is a novel system that extends the traditional hierarchical quorum consensus [11] offering higher flexibility in choosing quorums and allowing to obtain smaller quorums. The second quorum system is the classical grid quorum [4]. We developed a decentralized version of the grid quorum with a strong accent on delegation that also minimizes the number of visited sites. Finally, we provide an extensive simulation study of the deployment of all the aforementioned quorum systems over a ring-based DHT, namely Chord. Simulations show the performance gain obtainable by using the integrated approach with respect to the layered one for farsighted, grid and other quorum systems. Moreover, the study points out an interesting tradeoff between the acquisition cost and latency.

It might be argued that using a quorum at system level is not a good idea for large P2P networks. Optimal quorum systems have a size of $\sqrt{n}$ sites that is much higher than the targeted $log(n)$ complexities in P2P systems. In this paper we target hierarchical DHTs such as Crescendo [7], where a quorum request will be performed on a fraction of the whole P2P network. This means that in practical terms for a large network of a few millions of sites, one Crescendo subring can contain a few thousands of sites (depending on the actual hierarchy used).

The paper is structured as follows: Section 2 presents some background on quorum systems and DHT infrastructures (Chord, in particular). The Farsighted quorum system is described in Section 3. In Section 4 we introduce a general algorithm for acquiring a quorum over a DHT P2P infrastructure and in Section 5 we propose several instantiations of the general algorithm. Experimental evaluation of all the proposed solutions is presented in Section 6. Section 7 presents related work and Section 8 concludes the paper.

2 Background

In this section we introduce quorum systems defined for classical distributed systems and a brief summary of Chord that will be used as concrete DHT infrastructure[1].

Quorum Systems. Given a set of sites N, a set system universe $S = \{S_1, S_2, ..S_n\}$ is a collection of subsets $S_i \subseteq N$ over N. A *quorum system* defined over N is a set system S that has the following *intersection property*: $\forall i, j \in \{1..n\}, S_i \cap S_j \neq \emptyset$. Many quorum systems have been proposed in the literature. In this paper we consider two quorum systems, namely *rectangular grid* and *hierarchical quorum consensus*. A rectangular grid quorum system organizes N sites in a grid of r rows and c columns ($N = r \cdot c$) [4]. A quorum consists of an entire row and one element from each of the remaining rows (quorum size = $c + r - 1$). When a grid is a square, it becomes optimal in terms of quorum size. The quorum size for a square grid is $2 \cdot \sqrt{n} - 1$. Hierarchical quorum consensus (HQC) [11] consists in organizing the sites into a hierarchy. This hierarchy is represented as a complete tree where sites appear at the leaves of the tree. A quorum consists of the leaves obtained choosing a majority of the root children ($\lfloor d/2 \rfloor + 1$) and applying the same procedure recursively for each selected child. The quorum size for HQC is minimal when the tree degree is three [11] yielding a quorum size of $O(n^{0.63})$.

P2P Distributed Hash Tables and Chord. Peer-to-peer Distributed Hash Tables (DHTs) are overlay networks deployed on top of existing networks (mainly TCP/IP infrastructures) to provide self-organization and routing capabilities to applications. The common idea behind most such schemes is that, instead of being routed directly using physical sites' addresses ranging over a site space N, messages are routed using logical *key* identifiers, defined over a key space K. In *ring-based* DHTs, like Chord [18], the key

[1] Even though the experiments have been conducted on the top of Chord, the underlying DHT can actually be any ring-based structured P2P system.

space is a unidimensional circular space, while in *range-based* DHTs like CAN [16] the key space is a n-dimensional space. For the purposes of this paper, we consider a specific ring-based DHT routing protocol, Chord. The Chord protocol is based on a hash function that maps keys to the actual sites (a key is *covered* by some site). This function assigns each site and key an m-bit identifier. These identifiers are ordered on an identifier ring modulo 2^m. The identifier ring is called the *Chord ring*. Key k is assigned to the first site (called the *successor site* of key k) whose identifier is equal to or follows k in the ring. The system automatically routes messages to the site which is responsible for the destination key and repairs key assignment in case of sites joining and leaving the system. For efficient key lookup, each site maintains a table called *finger table*. A finger table contains up to m entries in an N-site network. The i^{th} entry in the table at site n is the successor site s of the identifier $(n + 2^{i-1})$ modulo 2^m. Site s is called the i^{th} *finger* of site n. Each entry also contains the IP address and the port number of the relevant node.

3 The Farsighted Quorum System

In this section we propose a hierarchical quorum system, *farsighted quorum consensus system* (FQC). FQC is similar to HQC in the sense that sites are the leaves of a complete tree although, FQC is more general and flexible than HQC in the choice of quorums. This allows the selection of quorums with lower size in FQC than in HQC. The difference between HQC and FQC is the quorum formation. As in HQC, keys are the leaves of a complete tree with a degree d. In HQC a majority of the root children ($\lfloor d/2 \rfloor + 1$) are selected and for each selected child the same procedure is applied recursively. In FQC, a given number of children of the root are chosen. For each of them, some of their children are selected again. The same procedure is applied recursively for the latter children. The number of children selected at each level depends from the number of children at the level above, as we detail below. For instance, if there are 16 sites organized in tree of degree 4, we can select 4 children in the first level and then 4 children from the first child, and 1 child from each of the other children. As an example, $\{0, 1, 2, 3, 4, 8, 12\}$, $\{0, 1, 2, 3, 5, 9, 12\}$, $\{0, 1, 2, 3, 6, 10, 15\}$ are valid quorums. In this case, a quorum consists of 7 sites, while 9 sites are needed for HQC. It is easy to see in Figure 2 that it is a quorum system (i.e. it fulfills the intersection property).

Given a set of n sites, which are the leaves of a complete tree of degree d and odd height, a quorum in FQC is defined as the set of sites that result from applying $(height(tree) - 1)/2$ tuples, $(f_1, f_2, \cdots, f_d), 0 \leq f_i \leq d$ to the tree. A tuple is applied to an odd level (outer level) and to the next level (inner level). If $f_x \neq 0$, f_x is the number of children selected in the inner level for child x of the outer level. If $f_x = 0$ no children of x are selected. We call each tuple a *pattern* and a set of patterns a *tactic*. A tactic over a complete tree of degree d with an odd number of levels is an FQC if for each pair of patterns $(f_1, f_2, \cdots, f_d)$, $(g_1, g_2, \cdots, g_d) \in tactic, \exists k \mid f_k + g_k > d$. That is, they fulfill the intersection property of quorum systems.

FQC is more general than HQC in the sense that HQC is an instance of FQC. In particular, FQC looks at two consecutive levels of the hierarchy at a time. Given two consecutive levels, it decides how many children of the inner level will be taken for each children of the outer level. The quorum chosen in HQC for a tree with degree 4 is represented in FQC using the tactic containing all the permutations of the pattern (3, 3, 3, 0) (also called *pattern set*). The pattern (3, 3, 3, 0) means that the first three children of a level are selected. For each of these children again 3 of its children are selected. The tactic for HQC satisfies the FQC quorum formation condition. Another example of valid pattern for a tree of degree 4 is the pattern set (3, 2, 2, 2). The tactic consisting of that pattern set satisfies the quorum condition. (3, 2, 2, 2) and (2, 2, 2, 3) intersect in the first and the last components ($3 + 2 > 4$). The pattern sets (3, 3, 3, 0) and (3, 2, 2, 2) also fulfill the FQC condition. The pattern set (4, 1, 1, 1) is compatible with the pattern set (3, 2, 2, 2) but not with the pattern set (3, 3, 3, 0). The tactic with the pattern sets (3, 2, 2, 2) and (3, 3, 2, 0) also fulfills the intersection property. There are patterns that are just an extension of other patterns. That is, they have a bigger value in some components. For instance, (3, 3, 2, 1) is just an extension of (3, 3, 2, 0). These extension patterns require more children, and therefore, they increase the quorum size. Although, they may help to increase the flexibility of a tactic. For instance, the pattern set (4,1,1,1) does not intersect with the pattern set (3, 3, 2, 0). Therefore, they are not a FCQ. However, (4, 3, 2, 0), an extension of (3, 3, 2, 0), is compatible with the pattern set (4, 1, 1, 1). The combination of the two pattern sets yields a highly flexible tactic in which for the first outer interval it is possible to pick either 0, 1, 2, 3 or 4 inner intervals.

The flexibility in choosing patterns in FCQ allows to select patterns exhibiting a lower quorum size than HQC. The size of the quorum in FCQ depends on the tactic. For a tactic composed of a single pattern set $(f_1, \ldots, f_d)$ the quorum formation is defined over a tree of degree d and height $(log_d n) + 1$. Since FQC considers pairs of levels, this is equivalent to define a quorum over a tree of degree d^2 with half the height of the original tree, $log_{d^2} n = (log_d n) + 1/2$. A quorum consists of the leaves of a subtree of this tree in which at each level $\sum_i f_i$ (where f_i is defined by the pattern) children are taken. Hence, the quorum size is the number of leaves of a tree of degree $\sum_i f_i$ and height $log_{d^2} n$, that it is: $(\sum_i f_i)^{log_{d^2} n}$. For arbitrary tactics, the quorum size depends on the pattern applied at each level. Generalizing the previous formula, the quorum size for a sequence $(p_1^1, p_2^1, \dots, p_d^1), \dots, (p_1^t, p_2^t, \dots, p_d^t)$ of patterns is computed as the product of the sum of each of them, that is, $\prod_{i=1}^{t} \sum_{j=1}^{d} p_j^i$.

4 Quorum Systems over P2P Networks

The basic idea about quorum systems applied to a P2P network is to allow a process to gain exclusive control of the key space. This can be obtained adapting classical quorum algorithms where a quorum is actually a set of keys over the whole key space. Exclusive control of the key space can be requested by any key and, upon this request, the site responsible for that key starts the protocol for the quorum acquisition.

The Notion of *Quorum Acquisition Cost*. In classical quorum systems the cost, in terms of application messages exchanged between sites, for the acquisition of a grant on a single site is usually one, as each site can contact any other site directly. Given a quorum defined on a P2P network, the cost for the acquisition of a grant on a key k, in terms of message overhead, depends both on the position of k and on the position of the quorum requester in the key space. This overhead is due to the routing mechanisms of the DHT, i.e. key k is reachable by the quorum requester in $O(logn)$ hops. This overhead must be taken into account to obtain a realistic view of the effective load generated by a specific quorum system on the underlying network. For this reason, we define the *quorum acquisition cost* of a quorum system over a P2P network as the average number of messages exchanged at the DHT level to obtain all grants required by a quorum.

A General Algorithm. In this section we introduce a general algorithm for the acquisition of quorums over DHT-based P2P networks. The algorithm (Fig. 1) is a generic canvas that embeds several functions, namely *GetQuorum*, *Acquire*, and *ChooseStrategy*, as well as the handler for messages exchanged between sites. The canvas also embeds other functions whose meaning is intuitive. Key to this canvas is the *ChooseStrategy* function which implements the logic related to a specific quorum system (i.e., grid, hierarchical etc.). In other words, only the *ChooseStrategy* must be changed in order to implement a different quorum system. The site requesting a quorum starts the algorithm calling the function *GetQuorum* and passing it an interval representing the whole key space. *GetQuorum* calls the *ChooseStrategy* function. *ChooseStrategy* splits the given interval in various subintervals and returns only those that will form the quorum. Then, *GetQuorum* tries to obtain a grant on the subintervals returned by *ChooseStrategy*. This is done via multiple calls (one for each subinterval) to the *Acquire* function that simply sends a *GETQUORUM* message to the first key of the subinterval passed as parameter and waits for the corresponding response. When the site responsible for a key k receives a *GETQUORUM[interval]* message sent to key k, it tries to obtain a grant on *interval*; if *interval* is a subset of the key space controlled by that site, denoted as *MyKeyspace*, then the site locks *interval*, otherwise a distributed recursive call to *GetQuorum* must be done in order to lock on *interval*.

```
Function name: GetQuorum
Input: An interval interval - Output: true or false
List $\leftarrow$ ChooseStrategy(interval)
for each $i \in$ List
    if $\neg$ Acquire(i)
        return false
return true

Function name: Acquire
Input: An interval interval - Output: true or false
send GETQUORUM[interval] to FirstKeyOf(interval)
wait for message from FirstKeyOf(interval)
if message = ACK[interval]
    return true
else
    return false

Function name: handler for GETQUORUM messages
Input: An interval interval and a key k from which the message
was received
if interval $\subseteq$ MyKeyspace
    if $\neg$ IsLocked(interval)
        Lock(interval)
        send ACK[interval] to k
    else
        send NACK[interval] to k
else if GetQuorum(interval)
    send ACK[interval] to k
else
    send NACK[interval] to k
```

Figure 1. A general algorithm for the implementation of quorum systems on DHT-based P2P networks

The Notion of *Latency*. Keys forming a quorum can be computed either locally at a single site (i.e., in a *centralized* way) or in a *decentralized* manner. This influences *ChooseStrategy*'s implementation. In the first case, the site requesting the quorum actually computes locally the whole key set forming the quorum. In the latter case, the quorum is computed in a decentralized fashion, in which several sites contribute to compute this set, i.e., the site requesting a quorum actually “delegates” other sites for this purpose (this delegation happens recursively). The *latency* of a quorum actually represents the longest sequence of sites that must be contacted to obtain a grant on a key for that quorum. The longer is this sequence, the higher number of messages is sent to obtain a quorum. On one hand a centralized computation of the quorum keys presents generally a high degree of parallelism (i.e. low latency, $O(logN)$); on the other hand, it leads to high quorum acquisition costs because for each key a lookup operation is issued. A decentralized computation of the quorum keys (as described in the next sections) tries to reduce the quorum acquisition cost introducing a delegation mechanism. The delegation mechanism enables to amortize the cost of a message across many keys. That is, a message from the requester to one of its fingers is not only useful to lock a single key k, but it is also useful to obtain all the keys belonging to the quorum with a higher value than k. However, delegation might increase the latency. The potential tradeoffs between the acquisition cost and latency will be analyzed in Section 6.

5 Layered vs. Integrated approach

There are two possible implementations of the previously introduced general algorithm: *layered* and *integrated*. In the integrated approach the generic algorithm exploits the information contained in the finger table of a site. In the layered approach, the generic algorithm is built on the top of the standard interface provided by the underlying P2P system. As a consequence, in this approach the algorithm does not have access to any internal information of the DHT. In this section we instantiate the generic algorithm for three quorum systems, namely grid, HQC and farsighted, using both layered and integrated approaches.

5.1 Layered Protocols

Grid Quorums. To implement a grid quorum system on top of a P2P system we proceed in a straightforward way: keys are organized, starting from key 0 to key $2^m - 1$, in a rectangle-shaped grid with r rows and c columns, such that $r \cdot c = 2^m$. The grid structure is shared among all sites so, the quorum intersection property is guaranteed for any chosen quorum starting from any key. When a site requests a quorum, it selects the smallest key, k, that site is responsible for. Assuming k is on the i-th row of the grid, that site tries to obtain a grant from all the keys belonging to row i, and a random single key for any row other than the i-th. A centralized algorithm that builds grid quorums can be instantiated on the generic algorithm of Section 4 by implementing a *ChooseStrategy* function that simply returns $c + r - 1$ intervals. The intervals correspond to an entire row of the grid and one key for any other row. More specifically, c intervals contain one key each corresponding to the range $[(i-1) \cdot c, \ldots, (i \cdot c) - 1]$ (actually the i-th row of the grid). The remaining $r - 1$ intervals contain also one key each, randomly chosen from the remaining rows. In the following we denote such an algorithm as *Grid Quorum Consensus* (GQC).

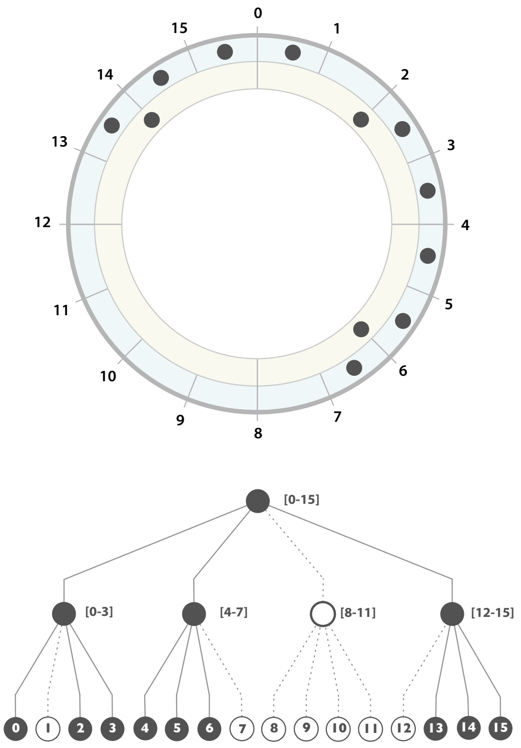


Figure 2. A decentralized hierarchical quorum over 16-key space

Hierarchical Quorums. As mentioned in Section 2, the hierarchical quorum consensus (HQC) consists in a hierarchical organization of the sites into a tree of degree d, in which sites (keys) are the leaves. Although the optimal quorum size for HQC is obtained with a tree of degree 3, we will use $d = 4$ to simplify the matching between key intervals defined by the tree and the key space size, $2^{2x} = 2^m$. In HQC a quorum is formed recursively by selecting a majority of children at the first level and then selecting recursively majority in each of the selected children until the leaves are reached. Let us point out that a child at a certain level of the hierarchy is directly mapped to a subinterval of the key space. Therefore, when a majority of children are chosen, it is actually being decided which underlying intervals of the key space are being selected as shown in Figure 2 for a 16-key Chord ring. In this figure black dots represent a HQC quorum. A centralized approach for the implementation of HQC would consist in selecting one quorum at the requester site and obtaining a grant for each key in the selected quorum. The *ChooseStrategy* function that implements this approach is straightforward: it simply recursively walks down the hierarchy choosing each time three children out of four until the leaf level is reached. Each key selected at this final stage is returned by the function as a different interval. We denote such algorithm as *Centralized Hierarchical Quorum Consensus* (C-HQC). HQC can be also implemented by computing quorum keys in a decentralized manner. In the following we refer to such algorithm as *Decentralized Hierarchical Quorum Consensus* (D-HQC). As in C-HQC, the requester in D-HQC selects three children (key subintervals) out of four, but only for a single level of the hierarchy. Then, a message with each selected interval is sent to the first key of that interval. The site responsible for that key will walk down one more step in the hierarchy. The decentralized recursion stops when a site receives a message containing an interval which is actually a subset of the portion of the key space the site is responsible for. Therefore, the implementation of *ChooseStrategy* for D-HQC boils down to the selection from the given interval of three randomly chosen subintervals out of four. For example, let us assume that key 0 is the requester key (Figure 2). The first iteration of the algorithm occurs in *ChooseStrategy* at the site responsible for key 0 and works on interval $[0-15]$. *ChooseStrategy* subdivides the interval into four subintervals and chooses three of them: $[0-3]$, $[4-7]$ and $[12-15]$. The next iteration occurs in parallel on the sites responsible for the first key of each subinterval.

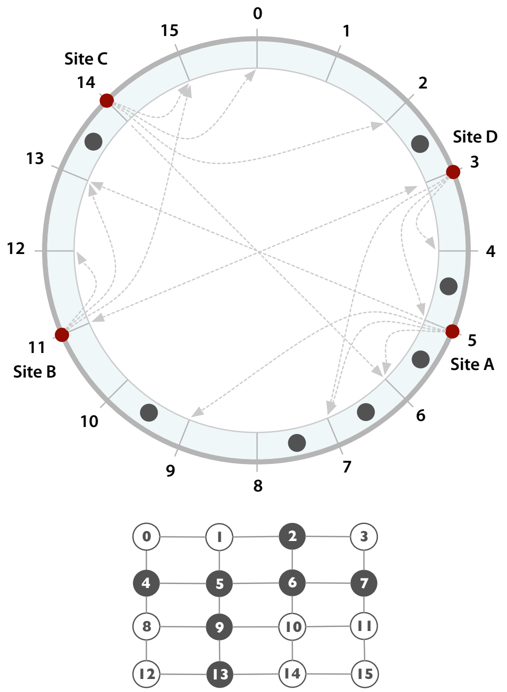


Figure 3. A 4×4 grid quorum over 16-key space

5.2 Integrated Protocols

Smart Grid Algorithm. The Smart Grid Quorum Consensus (S-GQC) algorithm is an enhanced version of GQC that exploits the knowledge contained at the DHT routing layer (i.e., finger table and the interval of keys the site is responsible for). More specifically, *ChooseStrategy* of S-GQC implements the following two points for selecting, in a decentralized way, the row and the remaining keys which form the quorum:

- Selection of the best row to be locked based on the keys the requesting site is responsible for. If the site is responsible for the keys of an entire row (or more than one), that row is selected, otherwise the next row is selected with respect to the one containing the first key the site is responsible for.

- The requesting site chooses a key in the row following the one fully locked, possibly among those contained in the site's finger table. Moreover, the site also delegates to the site responsible for the chosen key the locking of a key in the next row.

This step is repeated until a key is locked in each row.

As an example, let us consider Figure 3 where a square grid is represented over a 16-key Chord ring. This figure also points out the finger tables at each site representing fingers as arrows. Let site A (with key 5) be the requesting site. According to *ChooseStrategy* running at site A, the row corresponding to interval $[4-7]$ is selected. Then, looking at the finger table site A selects a key in the row following $[4,7]$. Let 9 be this key and site B be the responsible for this key. Finally, A starts in parallel to ask for both a grant for key 9 and grants for keys in row $[4-7]$ to all the sites responsible for those keys. The latter action is sequential in the sense that grants are acquired one after the other. The choice for getting a grant from a key in row $[12-15]$ is left to site B (since it is responsible for key 9) and so on. Therefore, the latency will be given by the longest sequence of contacted sites generated by the two parallel actions. Specifically in the example of Figure 3 the latency is 3 because site A can obtain grants for keys 9, 13 and 2 forwarding a message through sites B,C and D.

***Farsighted* Algorithm** FQC algorithm employs (i) a tactic over a 4 degree hierarchy (as defined in Section 3) and (ii) the decentralized recursion used by D-HQC (see Section 5.1).

ChooseStrategy is in charge of dividing an interval into subintervals and therefore, encapsulates the pattern selection for a given tactic. A pattern in FQC determines how many intervals are selected at the current (outer) level and

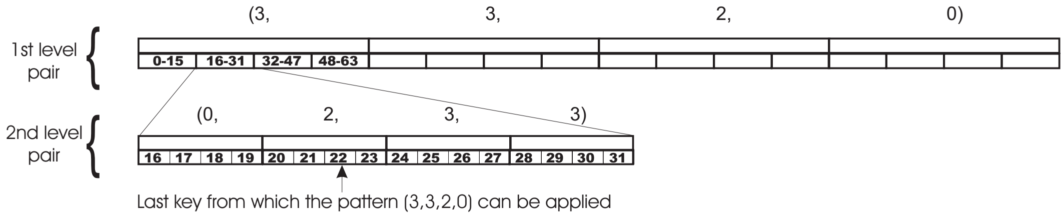


Figure 4. Last key from which the (3,3,2,0) pattern is applicable

for each of these subintervals how many intervals at the next (inner) level are chosen again. In doing the task of pattern selection, *ChooseStrategy* faces a big issue: selecting the most favorable pattern among those applicable. The set of applicable patterns depends on the interval managed by the site responsible for the requesting key. A pattern is applicable from a key, if the following two conditions are verified:

- A quorum can be obtained from that key without resorting to sites outside the interval. That is, only involving sites responsible for keys within the interval. For instance, to apply the pattern (3, 3, 2, 0) in a 256-key space to interval [0..255], the key must be located in the first subinterval ([0..63]) at the first level of the hierarchy because the pattern selects keys from the first subinterval, and from there it is possible to reach the second and third subintervals ([64..127] and [128..173]) without resorting to lookups. Moreover, the key can only be located in the first or second subintervals ([0..15] or [16..31]) of the subinterval [0..63] in order to reach 3 subintervals of that interval from that key.

- The key should be located at a position in the key space such that, at the next recursive steps of the quorum formation, at least the minimal pattern in the tactic (i.e. the one with lower lexicographical order) can be applied. Considering the previous example with a 256-key space, if the value of the key is 31, from there it is not possible to obtain a quorum recursively because it is “too late” in that subinterval. If the patterns (3, 3, 2, 0) and (0, 2, 3, 3) belong to the tactic, the pattern (3, 3, 2, 0) can be applied if the first key of the site is smaller than 22 (Fig 4). This allows to use the minimal pattern ((0,2,3,3)) in the rest of the levels.

0				1				2				3			
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

Figure 5. Numbering scheme for inner and outer levels

In order to calculate when a pattern is applicable to a key and interval, intervals at a level of the tree are numbered from left to right starting from 0. The same procedure is done at the next level (Figure 5)

As a consequence, one of the main tasks of *ChooseStrategy* is to calculate the last key from which a pattern is applicable for a given interval. In order to gain this knowledge, it is necessary to know the latest outer and inner intervals, (o, i), at which that pattern is applicable. In the previous example with only patterns (3, 3, 2, 0) and (0, 2, 3, 3) in the tactic, for pattern (3, 3, 2, 0), $(o, i) = (0, 1)$, which means that the last key to which the pattern (3, 3, 2, 0) can be applied is in the first interval (0) and in the second subinterval (1) within that interval. The minimal pattern must also be considered in order to check the applicability of a pattern. The minimal pattern enables the latest start in the subsequent levels. So, the latest outer and inner intervals for the minimal pattern of a tactic, (o_{min}, i_{min}), are also needed. Assuming a complete tree with degree 4, and that the key space starts with key 0 on the left most leaf and finishes with key 2^{2m} on the right most leaf, when a quorum is requested in an interval, it is possible to know the level of the tree (l) that corresponds to that interval. Using that level it is possible to know the last key from which a pattern is applicable at level l (where j iterates downwards to 1):

$$lastkey = o \cdot 2^{2l} + i \cdot 2^{2l-1} + \sum_{j=\frac{l-2+1}{2}}^{1} o_{min} \cdot (2^2)^{2j} + i_{min} \cdot (2^2)^{2j-1}$$

The first term of the sum represents the number of keys (leaves) before the latest outer interval. The second one represents the number of keys before the latest inner interval. And the third one represents the number of leaves that can be skipped before the subsequent pairs of intervals, if the minimal pattern is applied. In a space of 256 keys and the [0..255] interval, the pattern (0, 2, 3, 3) would be applicable from key 0 to key 120.

If the starting key (k) of the interval under consideration is not zero, the interval in which the pattern is applicable is $[k, k + lastkey - 1]$. It should be noted that in the first level of the tree, there is circularity, in the sense that any of the subintervals could be numbered 0 as far as the next on the right is numbered 1 and so on. However, this does not hold for the rest of the levels.

Once the selection of the pattern has been done for the current level, *ChooseStrategy* applies a first local recursion for those subintervals for which a pattern is applicable; recursion for the other subintervals is delegated to other sites (as in D-HQC). Preference is given to those sites present in the local finger table.

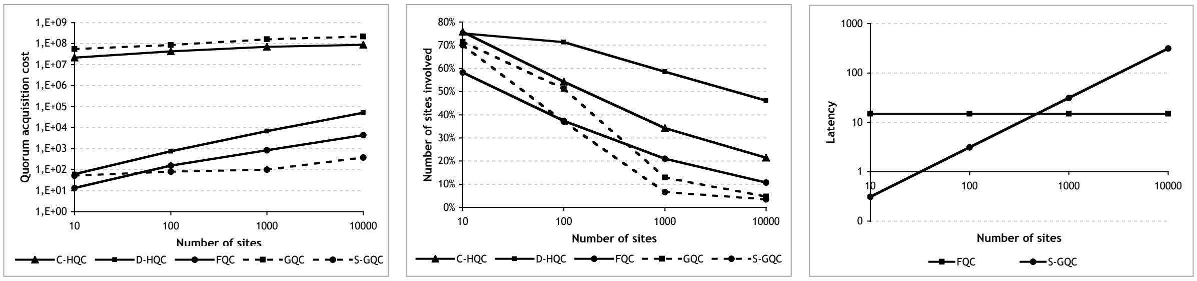
6 Layered vs. Integrated Approach: a Performance Evaluation

We have implemented the algorithms proposed in previous sections over a Chord network simulator we built. The simulator routes messages among simulated sites while keeping track of performance indices. We ran tests simulating networks with various key space sizes (of the form 2^{2x}) up to 2^{128}. We considered different (and large) key spaces to test different realistic Chord deployments. During a simulation the network is first populated with p sites using a uniform distribution over the whole key space. The uniform distribution actually simulates the correct behavior of the hash function used in Chord to map sites in the key space. Then, a quorum request is simulated on a key chosen at random. This request generates messages that are routed across some of the sites. A monitor component keeps track of the different performance metrics during the simulation to enable the comparison among the different algorithms. We measured the quorum acquisition cost (number of exchanged messages), the number of sites involved in the quorum formation (sites in the quorum plus sites only devoted to DHT routing) and the latency, varying the number of sites participating in the DHT. Plots in Figure 6 report the results. Further experiments and plots are reported in [2]. Each point in the plots represents the average value of the results obtained from the acquisition of 100 quorums. The variance was always within 3% so, it is not reported in the plots.

Evaluation of the quorum acquisition cost. We first determined experimentally the optimal grid configuration for the network used in the experiments (1000 sites and 2^{30} keys). We obtained that, contrarily to classical grid quorum systems, S-GQC reaches its optimal performance with a non-square grid ($2^5 \times 2^{25}$). Tests for Figure 6 have been computed using this optimal grid configuration. The dashed curves, reporting results for GQC and S-GQC, clearly show the impact of the usage of information internal to the P2P network (key intervals and finger tables). There is indeed a performance gap in the quorum acquisition cost between the layered and the integrated approach of approximately 3 orders of magnitude (Figure 6(a)). The number of involved sites, i.e. the percentage of sites with locked keys, for S-GQC also remains below that of GQC (Figure 6(b)). Figure 6(a) also shows the quorum acquisition cost for three different hierarchical quorum systems, namely C-HQC, D-HQC and FQC (using the pattern-set 4-1-1-1). All plots show an exponential growth ratio of the quorum acquisition cost with respect to the number of sites. As expected C-HQC exhibits the worst acquisition cost performance even though the number of sites involved in quorum formation is in between D-HQC and FQC as shown in Figure 6(b). The gap with D-HQC is due to the fact that D-HQC contacts sites that are used only for delegation purposes. FQC shows better performance with respect to D-HQC for both performance metrics thanks to its strategy for choosing the keys to be locked. Finally, the distance between D-HQC and FQC in Figure 6(a) gives a clear idea of the performance gap between layered and integrated approaches in hierarchy-based quorums. Since the main target of the proposed techniques is a hierarchical DHT such as Crescendo [7], the quorum system will be defined on a subring. A Crescendo subring (a Chord ring itself) will contain a few thousands of sites for a large network of a few millions of sites (it depends on the actual hierarchy used). This means that the segment of the curves that are of practical interest would be those in the interval 1000-10000.

A comparison between S-GQC and FQC. Figure 6 shows that S-GQC performs better than FQC with respect to the number of involved sites and the quorum acquisition cost. S-GQC is actually very well suited to P2P systems as it exploits the fact that routing information on DHT is more accurate in the proximity of a site. The great majority of the keys that must be locked by S-GQC in the optimal setting lies in a single row (i.e., they are contiguous). This means that a site on that row always tries to get a grant from a key covered by the following site in the network ring. This implies that in S-GQC each delegation step involves only sites to which at least a grant on one of their keys is requested. The set of keys constituting an FQC quorum does not have this contiguity. Therefore, it is possible that sites are involved in the quorum formation process only for delegation purposes (no grant on one of their keys is requested to them).

Latency Evaluation. It can be easily devised that layered algorithms have a latency which falls in a range of values. More specifically C-HQC and GQC have a latency ranging between 1 and 30. This is due to the fact that both algorithms, for each key that has to be contacted, spend from 1 up to $log(n)$ routing hops. The latency for D-HQC stems from the depth of the hierarchical tree used to build the quorum. Each step in the hierarchy can take up to $log(n)$ routing hops, therefore the latency for D-HQC can range between 30 and 30×30. Figure 6(c) shows a comparison between S-GQC and FQC (in their most favorable configurations) with respect to the latency. FQC shows a constant latency that is equal to half of the depth of the used



(a) Quorum acquisition cost versus number of sites

(b) Involved sites versus number of sites

(c) Latency for various algorithms

Figure 6. Measures for a network of 2^{30} keys

tree. Contrarily to layered algorithms, this number uniquely identifies the latency, thanks to the fact that the delegation mechanism is done between sites at one hop distance (typical behavior of the integrated approach). It can be also pointed out that S-GQC does not scale as FQC with respect to the latency. This is due to its low parallelism. S-GQC latency is heavily influenced by the average number of keys handled by each site as this has a great impact on the average number of sites forming a row. As an example, if we consider the points in the plots that correspond to 1000 sites the latency is around 30 for S-GQC, while FQS shows a latency of $log(2^{30})/2 = 15$. This means that, on the average, in that setting, FQC can obtain a quorum in half of the time of S-GQC. This difference in time will increase with the number of sites participating in the DHT.

7 Related Work

Quorum systems have been largely investigated in the distributed systems literature. Several surveys and comparison studies exists revising the various possibilities for building quorums as well as their performance trade-offs [15, 9]. In the following we concentrate on realizations of quorum systems based on P2P infrastructures. The idea of exploiting an overlay network as the underlying infrastructure for distributed systems is becoming more and more popular. [19] presents a series of examples of applications that can be realized in that fashion, including a quorum system. The idea is to build a simple majority quorum by requesting mutual exclusion access to a majority of the keys in the key space. Authors suggest the usage of a DHT-based multicast algorithm (such as [3]) for reaching all the required keys efficiently. However, building and maintaining such a multicast tree introduces further overhead which adds to those of DHT maintenance and quorum construction. The integrated approach we propose in this paper exploits a technique for dispatching the quorum requests to all the involved keys resembling the broadcast algorithm for a generic DHT infrastructure presented in [6]. Here, the finger table is used to forward messages to several nodes in parallel and maintaining a $\log(n)$ bound on the latency of broadcasted messages. However, differently from our paper, the aim of this algorithm is just to reach a certain number of keys, while in our case the chosen keys have to form a quorum. The same authors of [19] also propose in [12] algorithms and techniques for building a quorum system over a basic DHT, such as Chord. The idea is similar to that in [19]. Moreover, a random back-off technique is included in order to maintain constant throughput when concurrency of requests increases and subsequently there is a higher probability of access conflicts. With respect to our paper, the problem of efficiently gaining access to keys is not taken into account. Moreover, our suite of integrated protocols is based on quorum systems with smaller quorum sizes which require less keys than majority. In [14] a scalable dynamic quorum system supporting joins and departures of nodes is presented. This paper presents the dynamic counterpart of the Paths quorum system [15], Dynamic Paths, featuring low load, high availability and efficient quorum construction. In particular, Dynamic Paths manages dynamic behavior of nodes through DHT-like techniques borrowed from [13]. In other words, rather that building the quorum over a separate DHT layers, like we do, Dynamic Paths embeds management of node joins and departures following an approach based on a geometrical decomposition of a 2-dimensional space similar to CAN [16]. Another quorum system that exploits a geometrical CAN-like space is the d-space system presented in [17]. d-space has the objective of improving the efficiency of read operations, assuming they are more frequent than writes. The quorum size is proved to be optimal for read operations. Communication costs due to the DHT deployment are not computed. Other approaches for gaining consistency guarantees in P2P applications have been presented in [7, 8, 10]. In particular, in [10] a P2P architecture for multiplayer gaming is presented, relying on a single (but replicated) coordinator node for maintaining global shared data, that in practice is a singleton quorum. Though a singleton quorum provides best latency and acquisition cost, it obviously presents poor availability and/or consistency, as confirmed by their own experimental results.

8 Conclusions

Even though P2P systems are characterized by independent behavior of the participants, complex applications may require rare, but unavoidable, concurrent access to shared data. Quorum systems are a useful tool to obtain mutual exclusion access on such data. In this paper we focused on design principles, namely integration and delegation, that should guide the construction of quorum systems and protocols in DHT-based P2P systems. We introduced two novel performance metrics, quorum acquisition cost and latency to measure the performance of quorum systems in P2P systems. These metrics have a greater impact in a P2P than previous metrics used for quorums in classical distributed systems (like the quorum size). Then, several quorum algorithms have been proposed which basically differ in the way keys are selected to form a quorum. Quorum algorithms which use a centralized selection mechanism (e.g., C-HQC and GQC) exhibit a very high quorum acquisition cost. Decentralized algorithms aim to reduce this cost exploiting the notion of delegation to define the set of keys forming the quorum (e.g., D-HQC, FQC and S-GQC). However, some decentralized quorum algorithms suffer from a higher latency. This fact points out an interesting tradeoff between quorum acquisition cost and latency in some algorithms. More specifically, the quorum algorithm that scales better with respect to the quorum acquisition cost, namely S-GQC, does not scale as well with respect to latency. All the other algorithms show an opposite behavior. In particular, FQC, a new quorum system introduced in this paper, exhibits a constant latency while maintaining good performance with respect to quorum acquisition cost. All quorum approaches involve a number of sites higher than $log(n)$ what can question their scalability. However, by resorting to hierarchical DHTs a quorum can be obtained over a fraction of the P2P network. Currently we are working in applying principles of delegation and integration to hierarchical grid to combine its low quorum size with the flexibility and low latency of FQC.

References

[1] E. Anceaume, R. Friedman, M. Gradinariu, and M. Roy. An Architecture for Dynamic Scalable Self-Managed Persitant Objects. In *Proceedings of DOA 2004*, 2004.

[2] R. Baldoni, R. Jiménez-Peris, M. Patino-Martínez, L. Querzoni, and A. Virgillito. Dynamic quorums for dht-based p2p networks. Technical report, Uniersità di Roma "La Sapienza" - Universidad Politécnica de Madrid (UPM), 2005- http://www.dis.uniroma1.it/˜midlab/articoli/BJPQV05techrep.pdf.

[3] M. Castro, P. Druschel, A. Kermarrec, and A. Rowston. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8), 2002.

[4] S. Y. Cheung, M. Ahamad, and M. H. Ammar. The grid protocol: a high performance scheme for maintaining replicated data. In *Proceedings of the 6th International Conference on Data Engineering*, 1990.

[5] A. Datta, M. Hauswirth, and K. Aberer. Updates in Highly Unreliable, Replicated Peer-to-Peer Systems. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, 2003.

[6] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured peer-to-peer networks. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[7] P. Ganesan, P. K. Gummadi, and H. Garcia-Molina. Canon in g major: Designing dhts with hierarchical structure. In *ICDCS*, pages 263–272, 2004.

[8] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A self-tuning index for dynamic p2p systems. In *ICDE*, 2005.

[9] R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and B. Kemme. Are quorums an alternative for data replication. *ACM Transactions on Database Systems*, 28(3), 2003.

[10] B. Knutsson, H. Lu, W. Xu, and B. Hopkins. Peer-to-peer support for massively multiplayer games. In *Proceedings of INFOCOM 2004*, 2004.

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

[12] S. Lin, Q. Lian, M. Chen, and Z. Zhang. A practical distributed mutual exclusion protocol in dynamic peer-to-peer systems. In *Proceedings of the 3nd International Workshop on Peer-to-Peer Systems*, 2004.

[13] M. Naor and U. Wieder. Novel Architectures for p2p Applications: the Continuous-Discrete Approach. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[14] M. Naor and U. Wieder. Scalable and dynamic quorum systems. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, 2003.

[15] M. Naor and A. Wool. The Load, Capacity, and Availability of Quorum Systems. *SIAM Journal of Computing*, 27(2), 1998.

[16] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *LNCS*, 2233:14–34, 2001.

[17] B. Silaghi, P. Keleher, and B. Bhattacharjee. Multi-Dimensional Quorum Sets for Read-Few Write-Many Replica Control Protocols. In *Proceedings of the 4th International Workshop on Global and Peer-to-Peer Computing*, 2004.

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[19] Z. Zhang. The power of dht as a logical space. In *Proceedings of the 10th International Workshop on Future Trends in Distributed Computing Systems*, 2004.