Dynamic Quorums for DHT-based Enterprise Infrastructures

Roberto Baldoni[1] Ricardo Jiménez-Peris[2] Marta Patiño-Martínez[2]

Leonardo Querzoni[1*] Antonino Virgillito[1]

[1] Dipartimento di Informatica e Sistemistica "A. Ruberti"
Sapienza, Università di Roma
Via Ariosto 25, 00185 Roma, Italy
Tel. +39 06 77274014, Fax +39 06 77274002
{baldoni,querzoni,virgi}@dis.uniroma1.it

[2] Universidad Politécnica de Madrid (UPM)
Campus de Montegancedo s/n
Boadilla del Monte, Madrid 28660 Spain
Tel. +34 91 3367452, Fax +34 91 3367412
{rjimenez,mpatino}@fi.upm.es

*Corresponding author

Abstract

Peer-to-peer systems (P2P) have become a popular technique to design large-scale distributed applications in unmanaged inter-domain settings, such as file sharing or chat systems, thanks to their capabilities to self-organize and evenly split the load among peers. Recently, enterprises owning a large IT hardware and software infrastructure started looking at these P2P technologies as a mean both to reduce costs and to help their technical divisions to manage huge number of devices characterized by a high level of cooperation and a relatively low churn. Gaining a quick exclusive access to the system for maintenance or auditing purposes in these enterprise infrastructures is a fundamental operation to be implemented. Conversely, this kind of operation is usually not an issue in the previously mentioned inter-domain setting, where peers are inherently independent and cannot be managed.

In the context of classical distributed applications, quorum systems have been considered as a major building block for implementing many paradigms, from distributed mutual exclusion to data replication management. In this paper, we explore how to architect decentralized protocols implementing quorum systems in Distributed Hash Table based cooperative P2P networks. Our results show that quorum systems taken "as is" from the literature and directly applied to such networks are not scalable due to the high load imposed onto the underlying network. This paper introduces some design principles for both quorum systems and protocols using them that boost their scalability and performance. These design principles consist in a dynamic and decentralized selection of quorums and in the exposition and exploitation of internals of the DHT. As a third design principle it is also shown how to redesign quorum systems to enable efficient decentralization. We show that by combining these design principles in a cooperative environment with relatively low churn it is possible to minimize the imposed load in the system, in terms of sites contacted to obtain a quorum, and the latency of quorum acquisition

Keywords: quorum Systems, hierarchical majority, hierarchical grid, P2P Systems.

1 Introduction

Context and Motivations. Internet-scale applications like BitTorrent, KaZaA and Skype have led to a surge in research for large-scale inter-domain distributed systems. Solutions for such applications must be fault-tolerant, decentralized and adapt to dynamically changing settings. Peer-to-peer (P2P) networks have been shown to be the ideal substrate for such applications thanks to their self-organizing nature that is able to evenly share the network load generated by the application even in the presence of non-cooperative peers that can abruptly either leave the system or fail. These applications execute mainly simple operations like storing and locating objects (e.g. files, user contact information etc).

Recently it has been shown that P2P technologies can be successfully adopted also in large-scale IT enterprise infrastructures to reduce the complexity of their management (e.g., [21], [4], [1]) and thus the cost of ownership. The enterprise setting is dramatically different from the inter-domain one: even though the number of peers can be considered of the same order of magnitude[1], peers belonging to the substrate of an enterprise infrastructure work in a managed environment and, thus, they are much more stable that those in internet-based applications. We can therefore expect these peers to join and leave the application gracefully, except in the advent of failures. As a consequence, the enterprise setting is more stable than the inter-domain one, but operations that have to be executed on the P2P substrate by enterprise applications are definitely more complex than those implemented by Internet-based applications (e.g., prioritize and processing SLA alerts based on business impact vs. file location in file-sharing application). These complex operations may require a peer, for example, to take the control of the entire P2P system for auditing, monitoring or maintenance purposes. These latter functionalities are actually out of the philosophy, for example, of Internet-based file-sharing applications. As remarked in [21], such deep differences between enterprise and Internet-scale settings in terms of system model and operations lead to the need of designing P2P technologies (e.g., algorithms and mechanisms) optimized for a specific environment while keeping generic the very basic P2P functionalities such as the capability of peers to self-organize into a connected overlay network and

[1]Peers in a large scale enterprise infrastructure include any kind of devices and applications potentially reaching thus a number in the order of hundreds of thousands or millions [22].

collectively route data.

In this paper, we focus on an important abstraction for building decentralized protocols, namely quorum systems. Quorum systems [17] were proposed for regular distributed systems to attain a high degree of consistency, scalability and fault tolerance. They are still an important building block for the previously described innovative P2P enterprise applications that need to execute complex operations based on distributed consistency of replicated data or on distributed mutual exclusion. A quorum system over a set of sites (*system universe*) consists of a set of mutually intersecting subsets of sites (quorums). Quorums can be used for many different applications such as consistent data replication, distributed mutual exclusion, intrusion tolerance, distributed storage, trust management, multiplayer games, etc. A large variety of quorum systems have been proposed in the literature (see [9] for a survey). In traditional distributed systems, when a quorum is requested on a site, it autonomously chooses a single quorum among those in the system, and then proceeds trying to obtain permission (the locks) from the corresponding sites. All the requesters must precisely know the system universe. This makes this strategy hardly adaptable to the P2P (both enterprise and inter-domain) context, where the large size of the system makes unrealistic the assumption of precise global knowledge.

Another dimension that differs in the P2P context regards the performance of the quorum system. Quorum systems have been traditionally compared in terms of availability, load [17] and the number of messages needed 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 a traditional distributed system each peer can access any other peer with a single hop thanks to the assumption of precise global knowledge. However, as said above, in P2P systems this assumption does not hold anymore. Peers cannot maintain direct connectivity to all other peers for scalability reasons. As a result the cost of sending a message from one peer to another is, in general, higher than one hop. This means, that in a P2P system, acquisition cost will be strongly affected by the overhead induced by the routing performed by the underlying P2P network. Research in

P2P systems has come out with routing infrastructures (referred to as Distributed Hash Tables - DHTs) in which each peer is responsible for a set of keys defined over a virtual, continuous address space and holds the reference to a small subset of other peers (stored in the *finger table*). Message routing in DHTs exploits each peer's finger table, and the number of hops required to route a message between any two peers has an upper bound of $O(log(n))$ hops in a system with n peers.

Contribution. This paper presents a set of design principles for the conception, implementation and deployment of quorum systems in DHT-based enterprise infrastructures in a scalable and efficient fashion. A DHT can greatly help the realization of quorums in a large-scale setting: quorums can be defined over the key space, considering each key as a site, without any global knowledge on the number and identity of connected peers. However, simply reproducing the traditional quorum acquisition approach over the DHT does not lead to an efficient solution: if the quorum acquisition process is completely unaware of the key-to-peer mapping, routing can incur in a high overhead. For example, different sites belonging to the same quorum can correspond to keys mapped to the same physical peer in the DHT. Then, messages sent to all such keys will actually correspond to duplicated messages sent to the same peer.

We propose three design principles for the implementation of efficient quorum systems over P2P architectures[2], namely *delegation*, *integration* and *flexibility*, and discuss their impact on both acquisition cost and latency. The first design principle, *delegation*, lies in selecting quorums dynamically in a decentralized fashion as opposed to selecting the full quorum statically before initiating the acquisition. In other words, a quorum acquisition starts at a peer and is delegated recursively to subsets of other peers responsible for selecting and obtaining a grant from a subset of the whole quorum on behalf of the requesting peer. By forming quorums in this way, it becomes possible to have a significant degree of flexibility in selecting the most convenient sites in order to reduce routing overhead (and subsequently the acquisition cost). The second design principle, *integration*, consists in exposing internal information of DHT routing, more concretely, the finger table and the interval of keys for which a peer is responsible. This information enables to know

[2]It should be noted that, in general, one might not want to request a quorum over the full DHT, but only over a fraction of it. This is possible in hierarchical DHTs such as Crescendo [6], in which a fraction of the system is still a DHT by itself.

which sites can be reached directly and therefore to determine which are potentially most convenient. The third principle, *flexibility*, tells how quorum systems should be designed so they exhibit the necessary adaptiveness to exploit the first principle. That is, quorum systems should provide some flexibility to enable their dynamic acquisition in a balanced and efficient way from any peer, offering at each step enough alternatives to effectively realize delegation. Hence, by combining these three design principles it becomes possible to greatly reduce the number of visited peers and exchanged messages during a quorum acquisition.

The paper uses 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. The design principles are exercised in two quorum systems. The first one, named *farsighted*, is a novel quorum system that extends hierarchical quorum consensus [12] by offering higher flexibility in choosing quorums with the additional advantage of enabling smaller quorums. The second quorum system is the hierarchical grid quorum [12] incorporating two of the three design principles presented. 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 applying the three design principles with respect to the direct application of quorum systems to P2P networks. The study also points out an interesting tradeoff between the acquisition cost and latency. Moreover, quorum availability performance shows how the proposed solutions are suitable for a setting where the the number of faults is small and peers leave gracefully the system. When the number of failures increases, due to the fact for example that leaves are no longer graceful, we expect quorum availability delivered by our solutions to degrade severly. This is why our solutions well accommodates the needs of complex applications for P2P-based enterprise infrastructures.

Paper Organization. The paper is structured as follows: Section 2 presents the background on quorum systems and DHTs (Chord, in particular). In Section 3 we describe the farsighted quorum system. Section 4 introduces a general algorithm for acquiring a quorum over a DHT P2P system and several instantiations of it. Section 5 deals with problems related to peer failures. Experimental evaluation of all the proposed solutions is presented in Section 6. Section 7 presents the related work and Section 8 concludes the paper.

2 Background

In this section we introduce a few quorum systems defined for classical distributed systems and a brief summary of Chord that will be used as concrete DHT reference infrastructure[3].

2.1 Quorum Systems

A quorum system over a set of n sites, N, is defined as a set of subsets of sites, or quorums, with pair-wise non-empty intersection. More formally, a set system universe $S = \{S_1, S_2, ..S_n\}$ is a collection of subsets $S_i \subseteq N$. A *quorum system* defined over a set of sites N is a set system S that has the following *intersection property*: $\forall i, j \in \{1..n\}, S_i \cap S_j \neq \emptyset$. In a quorum system each subset S_i is called a *quorum*.

Quorum systems were originally proposed to cope with communication failures, ensuring that in case of partitioning of the system at most one partition will run, thereby preventing inconsistencies [8, 23]. Since then, many quorum systems have been proposed in the literature. In this paper we consider two well-known quorum systems, namely *hierarchical majority* and *hierarchical grid*.

1. Hierarchical majority (HMaj) [12] is a generalization of simple majority, where sites are organized in a hierarchy. This hierarchy is represented as a complete tree with sites at its leaves. A quorum is obtained locking recursively a majority of children nodes at each level starting from the root. The quorum size for HMaj is minimal when the tree degree is 3 [12], in such case the quorum size is $O(n^{0.63})$.

2. Hierarchical grid (HGrid) [13] is a variant of the grid quorum [3]. A hierarchical grid organizes sites into a multi-level hierarchy, such that they reside on the leaves of this hierarchy, while other levels are represented by logical nodes. Each node at level i of the hierarchy (beside leaves) is defined by a rectangular $m \times n$ grid of nodes at level $i + 1$.

   A quorum consists of the union of a *full row* and a *row cover* obtained recursively on the hierarchy. A *full row* at level i is defined as the set of $(i + 1)$-level nodes all pertaining to a single row of the grid,

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

while a *row cover* consists in a set of $(i+1)$-level nodes where each node pertains to a different row of the grid. Square-sized grids are optimal in terms of quorum size. The quorum size for a hierarchical square grid is $2 \cdot \sqrt{n} - 1$.

2.2 P2P Distributed Hash Tables and Chord

P2P Distributed Hash Tables (DHTs) are overlay networks deployed on top of existing networks (mainly TCP/IP infrastructures) to provide scalable self-organization and routing capabilities to applications. The common idea behind most such schemes is that messages, instead of being routed directly using physical peers' addresses ranging over a peer space N, are routed using logical *key* identifiers, defined over a key space K. In *ring-based* DHTs, like Chord [20], the key space is a unidimensional circular space, while in *range-based* DHTs like CAN [18] the key space is an n-dimensional space.

For the purposes of this paper, we consider a specific ring-based DHT routing protocol, namely Chord. The Chord protocol is based on a hash function that maps keys to actual peers. This function assigns each peer and key a m-bit identifier. These identifiers are ordered on an identifier ring, i.e. the *Chord ring*, modulo 2^m. Key k is mapped to the first peer (defined as the *successor peer* of key k) whose identifier is equal to or follows k in the ring.

The system automatically routes messages to the successor peer of the destination key, and maintains consistent key mappings in case of peers joining and leaving the system or peer failures. While peers may leave abruptly the system due to a failure, peers leaving gracefully the system can be requested to execute, before actually shutting down, a *hand-off* protocol to transfer data associated with keys mapped to them to their successor peer. This cooperative behavior is considered common to many existing P2P applications and is often leveraged to improve system performance [20]. In this paper we assume peers that do not fail follow this cooperative behavior.

For efficient key lookups, each peer maintains a table, called *finger table*, with m entries (i.e. the *fingers*). The i^{th} finger at peer n contains a pointer (e.g. the IP address and TCP port number) to the successor peer n' of the identifier $(n + 2^{i-1})$ modulo 2^m.

3 The Farsighted Quorum System

In this section we propose a hierarchical quorum system, namely the *farsighted quorum system* (FQ). FQ is similar to HMaj in the sense that sites are the leaves of a complete tree, but it is more general and flexible than HMaj in the choice of quorums. This allows to select quorums with smaller size in FQ than in HMaj.

Given a set of sites n, which are the leaves of a complete tree of degree d and odd height, a quorum in FQ is defined as the set of sites that results from applying $(height(tree) - 1)/2$ tuples, $(f_1, f_2, \cdots, f_d)$, with $0 \leq f_i \leq d$, to the tree. A tuple is *applied* to a generic odd level (outer level) and to the next level (inner level) of the tree . If $f_x \neq 0$, f_x is the number of children selected in the inner level for child x of the outer level (being children of a given level numbered from left (1) to right (d)). 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 FQ if for each pair of patterns $(f_1, f_2, \cdots, f_d)$, $(g_1, g_2, \cdots, g_d) \in tactic, \exists k \mid f_k + g_k > d$ (intersection property).

Figure 1 shows a system with 16 sites. In this example a possible quorum can be obtained applying one of the permutations of the $(4, 1, 1, 1)$ pattern, i.e. selecting 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 quorums defined by the pattern. In this case, a quorum consists of 7 sites while 9 sites would be selected with HMaj.

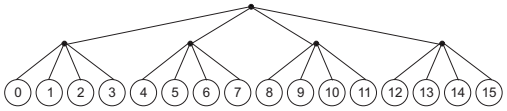


Figure 1: Hierarchical quorum consensus

FQ is more general than HMaj in the sense that HMaj is an instance of FQ. In particular, FQ 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 HMaj for a tree with a degree 4 is represented in FQ using the tactic containing all the permutations of the pattern (3, 3, 3, 0) (also called *pattern set*).

Various pattern sets can be used to build quorum based on FQ, but not all pattern sets are *compatible*, i.e. some combination of pattern sets can not guarantee the intersection property, thus leading to invalid tactics. For example the pattern set (4, 1, 1, 1) (all the permutations of this tuple) is compatible with the pattern set (3, 2, 2, 2) but not with the pattern set (3, 3, 3, 0). That is, the patterns (4, 1, 1, 1) and (0, 3, 3, 3) do not fulfill the intersection property. The tactic with the pattern sets (3, 2, 2, 2) and (3, 3, 2, 0) is also valid as it fulfills the intersection property.

There are patterns that are just an extension of other patterns, i.e. 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 than the one they extend, 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 an FQ. However, (4, 3, 2, 0), an extension of (3, 3, 2, 0), is compatible with the pattern set (4, 1, 1, 1).

The flexibility in choosing patterns in FQ allows selecting patterns exhibiting a lower quorum size than HMaj. The size of the quorum in FQ 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 FC considers pairs of levels, this is equivalent to defining a quorum over a tree of degree d^2 with half the height of the original tree. . A quorum consists of the leaves of a subtree of this tree in which at each level $\sum_i f_i$ children are taken, being f_i defined by the pattern. Hence, the quorum size is given by the number of leaves of a tree of degree $\sum_i f_i$ and height $log_{d^2} n$, that 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 $(p_1^1, p_2^1, \ldots, p_d^1), \ldots, (p_1^t, p_2^t, \ldots, p_d^t)$ 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 Implementation of Quorum Systems on DHTs

In this section we analyze in depth the issues related to the implementation of different quorum systems on top of a DHT network. Sites constituting a quorum system built upon a DHT network are actually the keys of the DHT itself. Therefore, differently from a quorum system for classical distributed systems, distinct sites can be mapped by the DHT to a same peer. We first introduce a general algorithm whose sole purpose is to represent a common implementation framework for all the types of quorum system we will consider. Then, we propose some general strategies that can be exploited for the implementation. Finally, we will show how the generic algorithm can be instantiated to implement the quorum systems introduced in previous sections, and how this can be done exploiting the cited strategies.

4.1 A General Algorithm

Function name: `GetQuorum`
Input: An interval *interval*
Output: *true* or *false*

List ← `ChooseStrategy`(*interval*)
for each $i \in List$
    if ¬ `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 ¬ `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 2: A general algorithm for the implementation of quorum systems on DHT-based P2P networks

The algorithm in Figure 2 is a generic canvas for the acquisition of quorums over DHT-based networks. It embeds several functions, namely *GetQuorum*, *Acquire*, and *ChooseStrategy*, as well as the handler for

messages exchanged between peers. 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 peer requesting a quorum (requester) starts the algorithm calling the function *GetQuorum* and passing it an interval representing the whole key space, i.e. the entire set of sites. *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 lock on the keys contained in 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 peer responsible for a key k receives a *GETQUORUM[interval]* message sent to key k, it tries to obtain a lock on *interval*; if *interval* is a subset of the key space controlled by that peer, denoted as *MyKeyspace*, then the peer locks *interval*, otherwise a distributed recursive call to *GetQuorum* must be done in order to lock on *interval*.

4.2 Implementation strategies

The proposed general algorithm enables a large variety of implementation strategies. As a trivial solution, the *ChooseStrategy* function can return subintervals composed by a single key. This corresponds to a *centralized* strategy, where the requester directly contacts all the keys forming the quorum. The centralized approach has an evident drawback: as each single peer is unaware of the mapping between keys and peers, it will send a different message to each single key even if most of such messages will be routed to the same peer. This inefficient behavior has a strong negative impact on the acquisition cost.

Through the general algorithm we can achieve a completely decentralized approach in the quorum formation. If the ChooseStrategy implementation returns a set of subintervals, each subinterval will lead to a recursive call to getQuorum, until the subinterval is a subset of the key space of a peer. In other words we

introduce a mechanism of *delegation* to delegate the acquisition steps to different peers. The advantage of this technique is that large intervals are progressively split into smaller subintervals, that are more likely to be completely contained in a peer's own interval and that can be acquired with a single message.

The selection of key intervals in *ChooseStrategy* can be done considering preferred intervals for delegation as those that will be delegated to peers contained in the finger table of the current peer. We refer to this new implementation strategy as a *integration* mechanism. The main goal of this mechanism is to select targets of the delegation such that, the routing mechanism of the DHT will be forced to make almost only single-hop routing steps, thus greatly reducing routing overhead. Implementations exploiting this mechanism will be referred to as *integrated* protocols, in contrast with protocols that are completely oblivious of the underlying DHT routing mechanism, namely *layered* protocols.

In the next subsections we instantiate the generic algorithm for two quorum systems, namely hierarchical majority and hierarchical grid, starting with centralized implementations and then adding decentralization as a first improvement and integration as the last step.

4.3 Layered Protocols

4.3.1 Hierarchical Grid

To implement HGrid on top of a ring-based DHT we need to find out a functional way to map the hierarchical bi-dimensional grid structure on top of the uni-dimensional keyspace. The keyspace is first partitioned into four intervals that are mapped to the four cells of a 2×2 grid, such that keys from 0 to $2^{m-2} - 1$ are mapped to cell $(1, 1)$ (row 1 and column 1) of the grid, keys from 2^{m-2} to $2 \cdot 2^{m-2} - 1$ are mapped to cell $(1, 2)$, keys from $2 \cdot 2^{m-2}$ to $3 \cdot 2^{m-2} - 1$ are mapped to cell $(2, 1)$, and keys from $3 \cdot 2^{m-2}$ to $4 \cdot 2^{m-2} - 1 = 2^m - 1$ are mapped to cell $(2, 2)$; this grid represents the first level of the hierarchy. Grids in lower levels of the hierarchy are obtained applying the same subdivision to the corresponding subintervals of the keyspace, until we reach a level where each single key is mapped to a different cell of the grid.

This mapping is shared among all peers in order to guarantee that the intersection property holds for any quorum chosen starting from any key. Figure 3 gives an example on how a three-level HGrid with 16

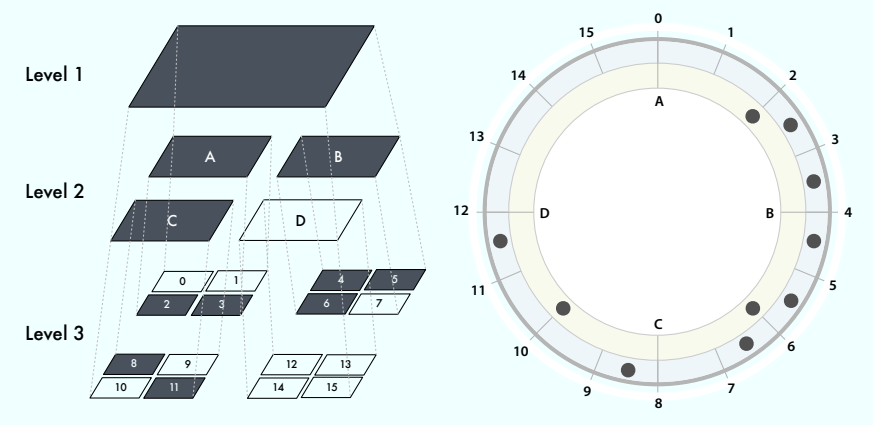


Figure 3: Example of a hierarchical grid quorum built over a 16-key Chord ring.

cells at the lower level can be mapped onto a 16-key Chord ring;

A centralized algorithm that builds HGrid quorums can be instantiated on the generic algorithm of Section 4.1 by implementing a *ChooseStrategy* function that acts locally by choosing recursively on the hierarchy a full row and a row cover, and then returns the corresponding keys. In the following we denote such an algorithm as *centralized HGrid*.

The same algorithm can be simply transformed in the corresponding decentralized version. The requesting peer starts two parallel actions to obtain a full row and a row cover on the root level grid. Each of these actions returns the key intervals corresponding to the selected cells. Choices at lower levels of the grid hierarchy are then delegated to those peers that are responsible for the first key of the chosen intervals. In the following we denote such an algorithm as *decentralized HGrid*.

Figure 3 shows an example of a hierarchical grid quorum obtained on a 16-cell grid structured as a three level hierarchy. Grey cells on the left side indicate those cells that have been selected by the algorithm, and dotted keys on the right side the corresponding locked keys in the Chord ring.

4.3.2 Hierarchical Majority

As mentioned in Section 2.1, the hierarchical majority quorum system (HMaj) 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 HMaj is obtained with a tree of degree 3, we will use $d = 4$ to simplify the matching between

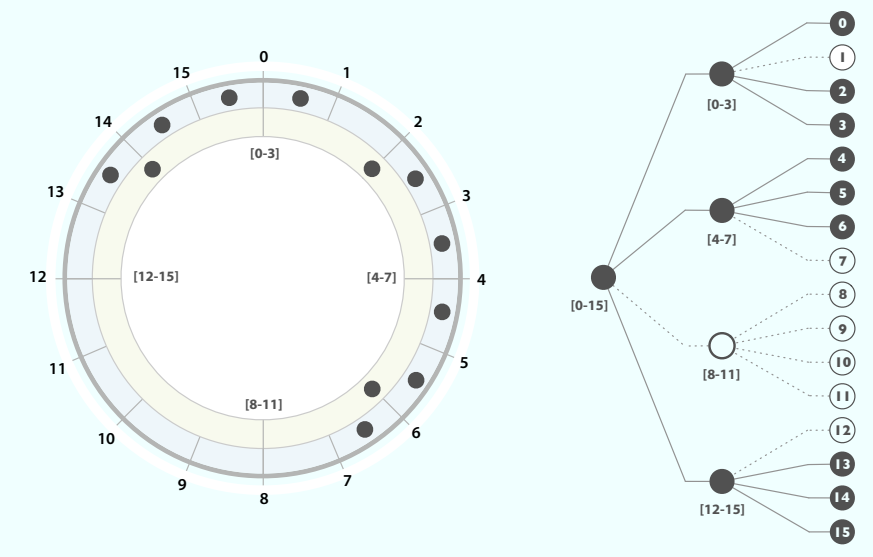


Figure 4: A decentralized hierarchical majority quorum built over 16-key Chord ring

key intervals defined by the tree and the key space size, $2^{2 \cdot x} = 2^m$.

In HMaj a quorum is formed recursively by selecting a majority of children at the first level and then selecting recursively a 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 4 for a 16-key Chord ring. In this figure black dots represent a HMaj quorum.

A centralized approach for the implementation of HMaj would consist in selecting one quorum at the requester peer 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 leaves 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 HMaj*.

Centralized HMaj can be also implemented by computing quorum keys in a decentralized manner. In the following we refer to such algorithm as *decentralized HMaj*. As in centralized HMaj, the requester in decentralized HMaj 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 peer responsible for that key will walk down one more step in the hierarchy. The decentralized recursion stops when a peer receives a message containing an interval which is completely contained in the portion of the key space the peer is responsible for. Therefore, the implementation of *ChooseStrategy* for decentralized HMaj 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 4). The first iteration of the algorithm occurs in *ChooseStrategy* at the peer 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 peers responsible for the first key of each subinterval.

4.4 Integrated Protocols

4.4.1 Hierarchical Grid

The Integrated version of the hierarchical grid quorum system (*integrated HGrid*) algorithm is an enhanced version of HGrid that exploits the knowledge contained at the DHT routing layer (i.e., finger table and the interval of keys the peer is responsible for). More specifically, *ChooseStrategy* of HGrid tries to select those cells in the grids that contain at least a finger. Only fingers pointing to peers that, given their position in the keyspace, could be able to obtain the desired keys are considered by the algorithm. When no finger is available to obtain all the required keys in a grid, then the algorithm proceeds with random choices.

4.4.2 Farsighted Quorum System

FQ algorithm employs (i) a tactic over a 4 degree hierarchy (as defined in Section 3) and (ii) the decentralized recursion used by decentralized HMaj (see Section 4.3).

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

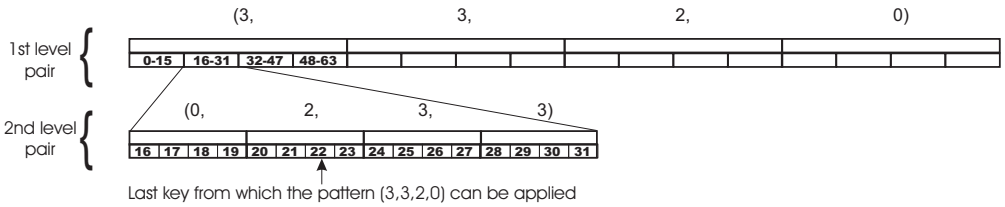


Figure 5: Last key from which the (3,3,2,0) pattern is applicable

favorable pattern among those applicable (i.e., the one that only involves peers holding keys part of the quorum). The set of applicable patterns depends on the interval managed by the peer responsible for the requesting key (the first key of the interval to be locked. See function *Acquire*). A pattern is applicable from a key owned by a peer, if the following two conditions are verified:

- A quorum can be obtained from that key only involving peers responsible for keys within the interval. For instance, the pattern (3, 3, 2, 0) selects keys from the first three subintervals. To apply that pattern in a 256-key space to interval [0..256[, the key must be located in the first subinterval ([0..64[) 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..128[and [128..174[) without resorting to peers managing keys located only in the last subinterval ([174..256[), which do not belong to the quorum. Moreover, the key can only be located in the first or second subintervals ([0..16[or [16..32[) of the subinterval [0..64[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 and the pattern (3, 3, 2, 0), if the value of the requester key is 31, from there it is not possible to obtain a quorum recursively because it is "too late" in that subinterval. That is, the pattern will be applied in the next recursive step and it is not possible to directly access the first, second and third subintervals of [16..31[from key

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

Figure 6: Numbering scheme for inner and outer levels

31 (see Fig.5), having to resort to peers only holding keys outside the original interval to reach them. 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 requester key is smaller than 22 (Fig 5). From there it is possible to apply in the next recursive steps the minimal pattern, (0, 2, 3, 3).

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. For this reason, intervals at a given level of the tree are numbered from left to right starting from 0. The same procedure is done at the next level (Figure 6) 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. (0, 2, 3, 3) is the minimal pattern in the previous tactic. So, the latest outer and inner intervals for the minimal pattern of a tactic, (o_{min}, i_{min}), are also needed. Assuming that the key space starts with key 0 on the left most leaf and finishes with key $2^{2m} - 1$ on the right most leaf, when a quorum is requested in an interval, since the tree in FQ is complete with a 4 degree, 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):

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

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 keys that can be skipped in the inner interval to apply the minimal pattern in all the next recursive steps. In a space of 256 keys and the [0..256[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 $[k, k + lastkey[$. 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* splits the given interval in subintervals recursively till a subinterval does not contain keys owned by more than one finger, that is, it contains keys of zero or one finger. In the latter case, the interval is delegated to that finger. In the former case, it will be delegated to the peer pointed by the closest finger before the interval. Those intervals owned by the current peer will be locked directly (if not locked). In this way intervals become smaller and smaller, and, therefore, the same happens with the jumps in the finger table, ending up with intervals fully owned by the receiving peer. We denote such algorithm as *farsighted HMaj*

5 Handling peer failures

The DHT-based overlay network is capable of handling peer failures, transferring the responsibility of keys pertaining to a failed peer to one of its neighbors (its successor in the ring, if we consider the specific case of Chord). This failure-handling mechanism should be "transparent" to applications running on top of the DHT thus enabling the developers to consider all the keys as always available (at least as long as a single peer is present in the overlay network). In our quorum-based system we can distinguish peers with three different roles:

Requesters – peers that try to obtain a quorum doing the first call to the *GetQuorum* function;
Delegators – peers that only handle delegation steps in the construction of a distributed quorum;
Leaves – peers responsible for the keys that actually form the quorum.

Failures of these three kinds of peers lead to different inconsistencies that must be avoided in order to guarantee that the quorum intersection property will always hold. A possible solution to this problem is the

employment of one of the many different replication techniques proposed by the literature [11]. However, in this section we propose a different approach to this problem that is based on the idea of restoring a peer state after its failure. Given the different roles played by different peers in a quorum request, we will discuss separately how to deal with their failures in the remainder of this section.

5.1 Failure of a leaf peer

Peers associate a state to each of the keys they are responsible for. In a ideal fail-free scenario, each key can be either *locked* (L), if the responsible peer had granted the lock on it for a quorum request, or *free* (F). Subsequently to a peer failure, its keys are automatically reassigned by the DHT to its successor, but they lose completely their state. In this case, a mechanism is needed to correctly restore this state after a failure. Reassigning to each key the correct state it had before the failure is fundamental to avoid inconsistencies in the system that can potentially lead to violations of the quorum intersection property. For this reason we introduce a third possible state, i.e. *unknown* (U) that is automatically assigned by a peer to all the keys inherited after the failure of its predecessor. Keys in this state cannot be granted to form quorums, as this guarantees that the quorum intersection property will always hold, despite failures.

(a) (b)

Figure 7: Key reassignment after the failure of peer P2, and subsequent state change.

Figure 7 shows a part of the Chord ring where three peers (P_1, P_2 and P_3) are located, along with the keys assigned to them. Each key is represented with its state. If P_2 fails, all the keys pertaining to it are automatically reassigned by the DHT to P_3 which puts them in the *unknown* state. Note that, after this failure, the keys now pertaining to P_3 can be divided in two distinct subsets: one containing only keys in the *unknown* state (i.e. those inherited after P_2's failure), and another containing keys in either *locked* or *free* states.

Each key can then suffer one of the following state transitions due to a peer failure (see Figure 8 for the

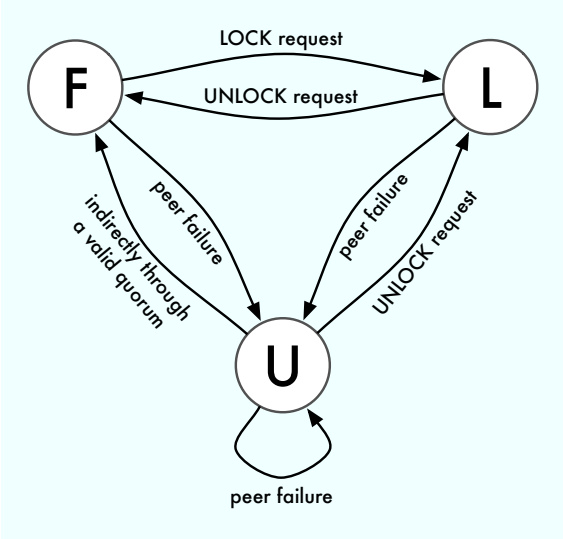


Figure 8: State transition diagram for keys in the DHT.

complete state transition diagram): (1) $U \rightarrow U$, (2) $L \rightarrow U$ or /3) $F \rightarrow U$.

Given that every transition produces the same state (i.e., U), the problem is how a peer can choose an inverse transition to correctly restore the original key state. State transition 1 happens when a peer that inherited the considered key from a failed neighbor peer, suffers itself a failure before being able to restore the original key state. Solving this case then actually boils down to solve either case 2 or 3.

A key undergoing through state transition 2 was previously locked and then was part of a quorum. This quorum could be either still forming or completely formed; in both cases we expect that eventually an UNLOCK message will be sent to this key from one of the peers involved in the quorum. The arrival of this message lets the peer currently in charge for that key assume that state transition 2 happened; then it can safely apply a transition $U \rightarrow L$ to the key before handling the UNLOCK message. We can then conclude that every key that underwent state transition 2 will eventually undergo the correct inverse state transition $U \rightarrow L$.

The case involving state transition 3 is the most difficult to solve. It requires a peer containing keys in the U state to receive a quorum request. If the quorum request is acknowledged, then it can safely apply a transition $U \rightarrow F$ to every key in the U state, as it can be sure that no other valid quorum can coexist at the same time (due to the quorum intersection property), and then every key can be safely moved to the

F state. Note that this actually is valid even for keys that underwent state transition 2, but whose original state was not yet restored. If the request is not acknowledged it means that non-U keys are not sufficient to achieve a quorum. If there are too many keys in the U state, this could lead to a deadlock. This situation can be prevented by letting keys return to the F state after a timeout expires. This is safe, provided the timeout is sufficiently long, because if the keys belong to a quorum they will receive an UNLOCK message before the timeout expires.

Even though keys forming the DHT can be considered as always available, the introduction of the U state actually renders part of the key-space not available for quorum requests. This leads to a behaviour that somewhat resembles the one in classical quorums, where peer failures actually affect the quorum availability.

5.2 Failure of a requester or a delegator peer

The requester peer and each delegator peer maintain state about the quorum hierarchy associated to the quorum, i.e. keys to which the quorum formation request was forwarded and, only for delegator peers, the peer from which the quorum formation request was received. A peer failure thus partially destroys information about the multicast tree used to request the quorum. If a quorum was granted and there are failures in the quorum hierarchy, some peers will never receive the UNLOCK message, and therefore a mechanism is required to detect this situation and perform the UNLOCK autonomously.

In order to recover this lost information we propose the usage of a simple technique based on KEEPALIVE messages forwarded periodically from the quorum requester down the quorum delegation hierarchy (if any) towards leaf peers. Each peer involved in the quorum waits for this message to be sure that the quorum structure at parent levels is still intact. If a message is not received from a peer, it can send toward its parent a QUORUMALIVE message after a predefined period of time has elapsed, to ask if the quorum is still valid. If a requester or delegator peer fails, then the peer that inherits its keys will eventually receive the QUORUMALIVE message from each of the peers at the lower level of the quorum hierarchy. For each of these messages it will reply with an UNLOCK message forcing all the locked keys to the free state. In case the failed peer was a delegator, the replacing peer will also receive the KEEPALIVE message from the

upper level, to which it will reply with an UNLOCK, forcing the removal of the quorum.

We would like to point out that the cost paid to maintain the quorum delegation structure through the periodic sending of KEEPALIVE messages is an addition to the costs paid by the DHT to consistently maintain its structure after a peer failure. However, we think that this cost can be considered significantly smaller that the one imposed by any standard state replication strategy. This claim is supported by the fact that (i) only peers involved in a quorum request must send these messages, (ii) the cost is incurred only during a quorum request and (iii) the only content of these messages is a simple quorum identifier, while a replication protocol would replicate the whole state associated to keys regardless of the stat content and the current activity of the system.

6 Performance Evaluation

To evaluate the impact on performance of the various techniques previously introduced we implemented different versions of the algorithms for hierarchical majority (HMaj) and hierarchical grid (HGrid) quorum systems.

We started evaluating performance of simple centralized implementations, then added decentralization techniques, and, finally, integration techniques and quorum flexibility (this latter only for hierarchical majority, the farsighted quorum). Note that the integration mechanism can work better with those quorum systems that permit more flexibility in the selection of the keys to be locked. For this reason we preferred a decentralized and integrated implementation of the farsighted quorum system introduced in Section 3, to a simpler, but less performant, decentralized and integrated implementation of the classical hierarchical majority. Therefore, the algorithms used in this simulation study were centralized HMaj, decentralized HMaj, farsighted HMaj[4], centralized HGrid, decentralized HGrid, and integrated HGrid All the algorithms were implemented on top of a Chord network simulator. The simulator routes messages among simulated peers while keeping track of performance indices. We ran tests simulating networks with various key space sizes

[4]Preliminary tests, not reported here, showed how the 4-1-1-1 pattern is the one that offers the best performance for the farsighted quorum algorithm. For this reason, for the test presented here, we always sticked with this pattern.

(of the form 2^{2x}) up to 2^{128}. During a simulation the network is first populated with p peers using a uniform distribution[5] over the whole key space. Then a quorum request is executed on a key chosen at random. This request generates messages that are routed across various peers. A monitor component keeps track of different performance metrics throughout the entire simulation run.

The implementation of quorum systems over DHTs introduces new performance metrics. Quorum systems applied to classical distributed systems, are usually evaluated on the basis of well-known metrics like *load* or *availability*. The implementation over the logical keyspace provided by DHTs requires to take into account the work done by the DHT routing mechanism to let peers communicate. The tests permitted us to observe and evaluate various aspects of the previously cited algorithms:

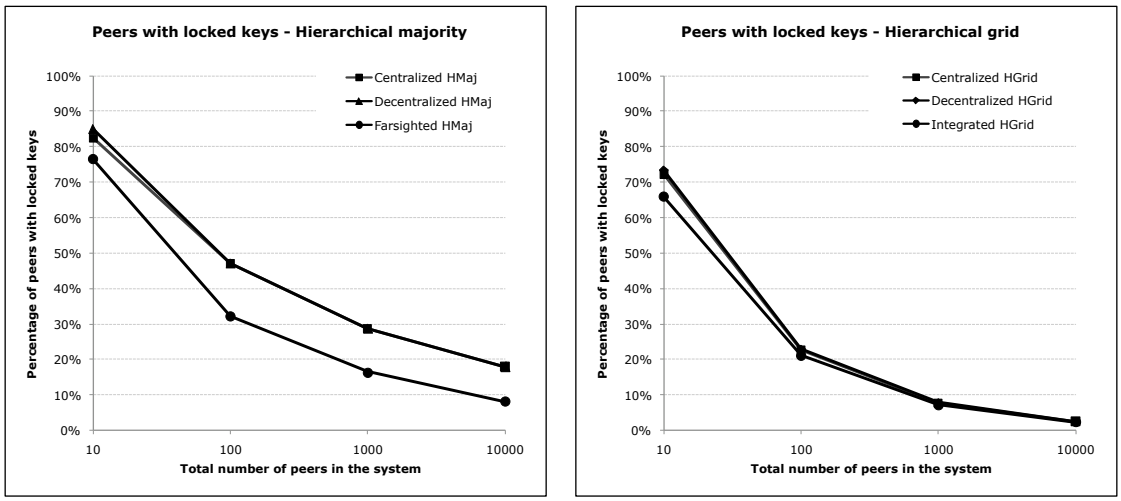
- **Percentage of peers containing locked keys**: reflects the real quorum size in terms of peers of the DHT.
- **Quorum acquisition cost**: the average number of messages exchanged at the DHT level to obtain all locks required by a quorum;
- **Depth of the generated multicast tree**: represents the length of the longest sequence of peers that must be contacted to obtain a grant on a key for a quorum;
- **Role of peers**: number of peers playing each role during a quorum acquisition, i.e. peers with locked keys, delegating peers or DHT routers;
- **Distribution of messages among peers**: gives a picture about the balance of the distribution of messages generated by a quorum request across peers of the DHT.
- **Availability under failure**: shows the average percentage of quorum requests that can be satisfied assuming that keys associated to one peer are in an unknown state (see Section 5)

In the following we report the results for each of these aspects. Each result is the average of 10000 independent runs. Confidence intervals are not represented as all measures showed a variance below 4%.

[5] A uniform distribution is used to simulate the correct behavior of the hash function used in Chord to map peers in the key space.

6.1 Percentage of peers containing locked keys

Here, we evaluate which percentage of peers contains, on average, keys locked by a quorum request. The tests were conducted varying the total number of peers in the system (log scale).



(a) (b)

Figure 9: Average percentage of peers with locked keys.

Figure 9(a) reports the results for hierarchical majority algorithms. Both the centralized and decentralized implementations (black and dark grey curves), as expected, show a similar behaviour, as the basic strategy used to choose keys forming the quorum is exactly the same in both cases. The farsighted implementation shows instead a lower number of peers with locked keys. This behavior is due to two reasons:

- thanks to its greater flexibility, the farsighted quorum algorithm, using the 4-1-1-1 pattern, is able to build smaller quorums w.r.t. the simple hierarchical majority; the average difference is about a 15%. For a system with 1000 peers, it would mean 150 peers less to get a quorum.
- the integration introduced in the implementation tends to choose peers that are responsible for larger sets of keys.

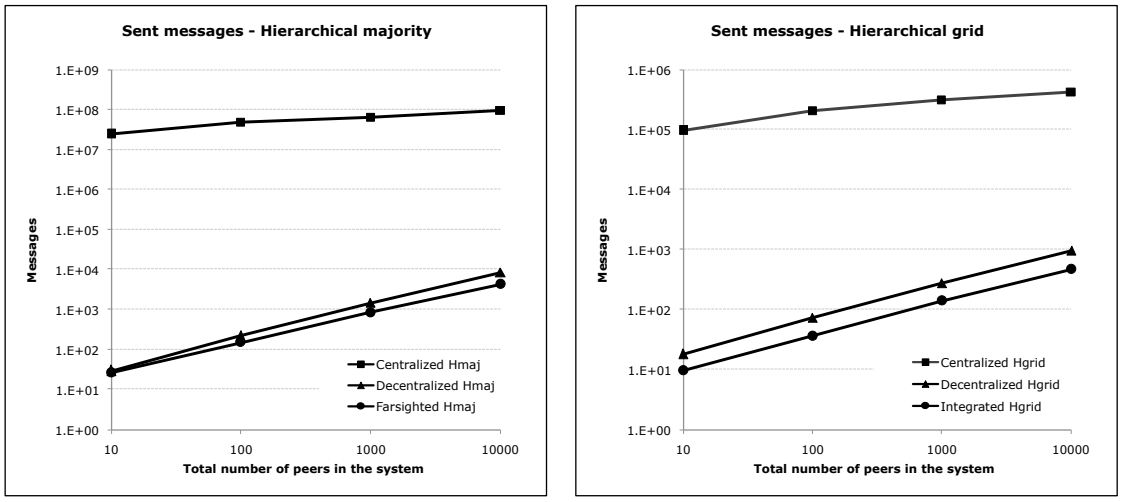
Figure 9(b) reports the results for hierarchical grid algorithms. In this case all three implementations show almost the same behaviour, with a slight advantage for the integrated version that is due to the tendency

of integrated algorithms to prefer peers with larger key sets. The tendency to decrease peer occupancy when more peers are in the system is a natural consequence of the quorum size, being a power of n with an exponent lower than 1. This means that a linear increase in the number of peers leads to a sublinear increase in the quorum size, resulting in smaller and smaller ratios between quorum and system sizes. Note that this metric only takes into account peers that contain at least one locked key, and thus cannot give a clear idea of the actual burden imposed by each algorithm on the whole Chord network.

6.2 Quorum acquisition cost

In classical distributed systems each peer can directly contact any other one. Therefore, the cost for the acquisition of a grant on a peer is usually one, in terms of application messages exchanged between peers. Given a quorum defined on a DHT, 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 and is one in the best case and $O(logN)$ in the worst case (being N the total number of peers in the system). 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. The *quorum acquisition cost* gives a comprehensive view about the load imposed by the quorum system on the DHT. From a practical point of view this metrics reports the number of messages generated by an algorithm from the initial quorum request until all key locks have been acquired, thus it also includes messages generated by the DHT routing level.

Figure 10(a) reports results for hierarchical majority. The advantage given by a simple decentralized implementation w.r.t. the centralized one is huge. This striking difference is due to the fact that with the centralized approach a single request is sent for every single key selected by the algorithm, and each of these requests can generate up to $log(N)$ messages at the DHT routing level (N is the number of peers in the system). With a decentralized implementation locking requests contain *intervals of keys* that are recursively refined. This mechanism saves a large portion of messages by generating a multicast tree that tries to minimize the number of messages. The same figure also reports results for the farsighted implementation that shows an improvement w.r.t. the decentralized one that tends to increase with system size. This can be



(a) (b)

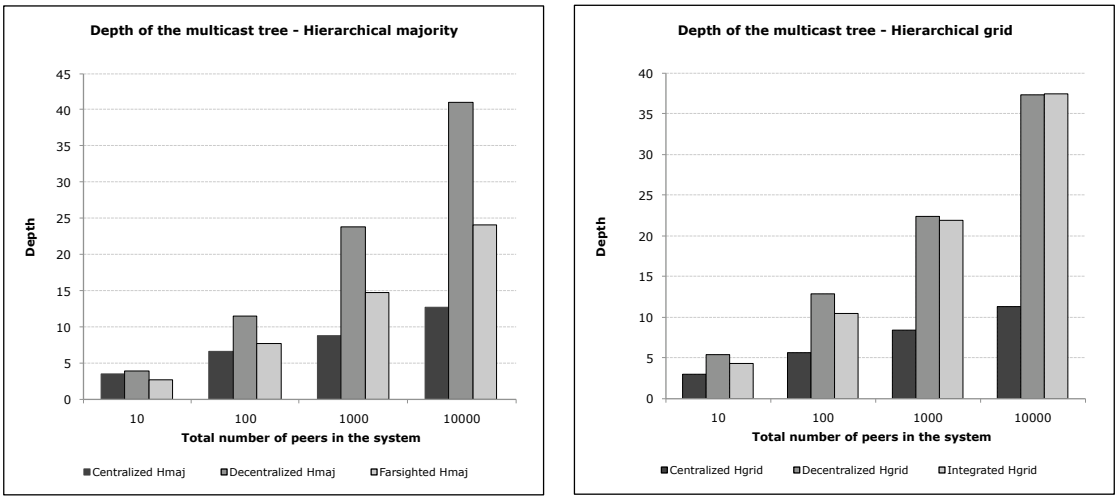
Figure 10: Average quorum acquisition cost.

explained by the fact that, with an increment of the system size, the smaller size of quorums obtained with the farsighted implementation gains more importance, thus reducing the overall number of locking requests generated.

Figure 10(b) reports the same results for hierarchical grid algorithms. Even in this case we find the same behaviour for centralized and decentralized implementations. This confirms that the better performance shown by decentralized implementations are general and due to the decentralized approach itself. In this case, the integrated implementation shows a constant advantage w.r.t. decentralized one that is completely imputable to its integration with the DHT.

6.3 Multicast tree depth

Here we show measures about the depth of the multicast tree that is implicitly generated by the algorithm as it sends locking request messages to all the target keys. In this case we also considered messages added by the DHT routing mechanism. The depth is defined as the longest path used by the quorum requester to lock a single key. Note that this measure is important as it greatly affects the overall latency experienced to obtain a quorum.



(a) (b)

Figure 11: Average depth of the multicast tree.

Figure 11(a) reports the results for hierarchical majority algorithms. In this case the centralized implementation behaves better than the two others, showing a smaller depth. This is obvious since the depth of the multicast tree for a centralized implementation will be $O(logN)$, whilst decentralized versions trade off depth for a lower load in the network. The depth of the centralized version provides a lower bound for the depth of the multicast tree to obtain a given quorum. It is worth noting here that the farsighted version clearly shows better performance than the decentralized one: this result is due to the integration techniques employed, that, as we will see in one of the next subsections, completely substitute the DHT routing mechanism. Basically, the integrated version decides how to create the quorum depending on finger tables and the distribution of keys to try to visit only peers that will become part of the quorum and, thus, reducing the load induced in the network. The decentralized version blindly chooses quorum intervals and then delegates routing to the DHT, what results invariably in a higher number of hops that its integrated counterpart.

The results for hierarchical grid algorithms are similar (Figure 11(b)). The main difference in this case is with the integrated implementation whose results are just a bit better than those of the decentralized one. The reason for these poorer results is that integration is able to express its complete potentials only when

coupled with a flexible quorum selection mechanism, and flexibility was included in farsighted HMaj but not in integrated HGrid.

6.4 Role of peers

In this section we explore which role is played, on average, by peers during a quorum acquisition. More specifically we counted, among the peers visited to acquire a quorum, how many of them acted only as routers at the DHT level (*DHT routers*), how many executed the quorum acquisition algorithm but ended without locked keys (*delegating peers*), and finally how many ended with one or more locked keys (*peers with locked keys*).

The big picture is given by Figure 12 that reports measurements separately for each algorithm. The curves are cumulative, and the sum of each single contribution represents the total percentage of peers involved in the quorum acquisition. There are three points in this picture that are worth noting:

- integrated versions reduce significantly the routing needed to acquire a quorum.
- for all the four implementations that use delegation, the ratio of peers that act as delegators or DHT routers w.r.t. those containing locked keys increases along with the size of the system;
- the measurements for the farsighted implementation of the hierarchical majority algorithm do not present even a single DHT router.

The last point is interesting as it points out that the integration mechanism used in this algorithm completely substitutes the DHT routing mechanism. This is done in order to realize a *more clever* routing in combination with the flexibility of farsighted HMaj that actually leverages information about the quorum that the algorithm is trying to obtain to reduce the overall number of messages sent. It should be highlighted that since quorum systems will target medium to large-sized systems, integration will be essential to reduce the number of routing peers and therefore reduce the load induced in the network.

Figures 13(a) and 13(b) show a more detailed comparison between hierarchical majority and hierarchical grid algorithms (respectively) for a 10k-peer Chord network. In this figure, it can be observed that even for

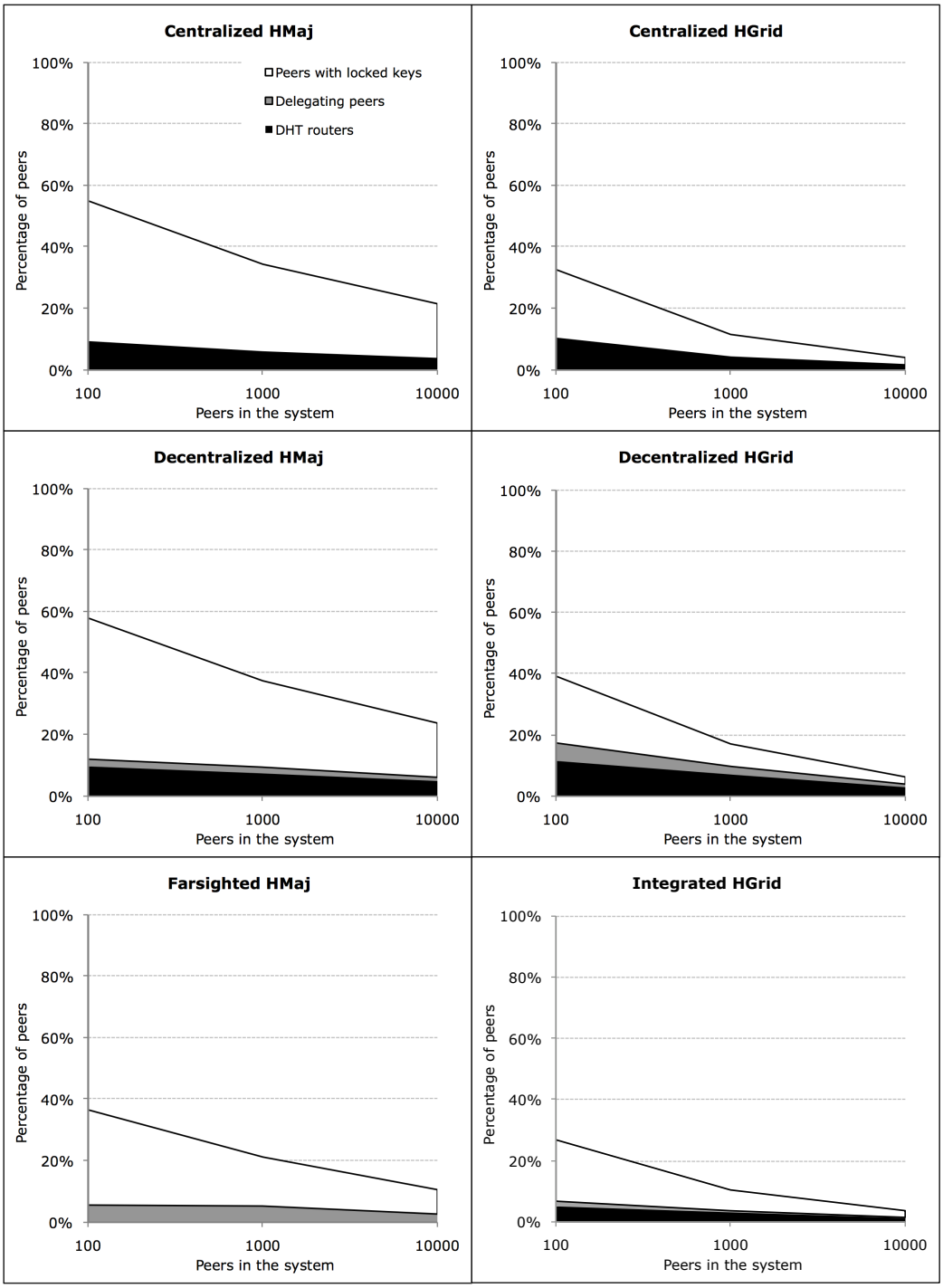
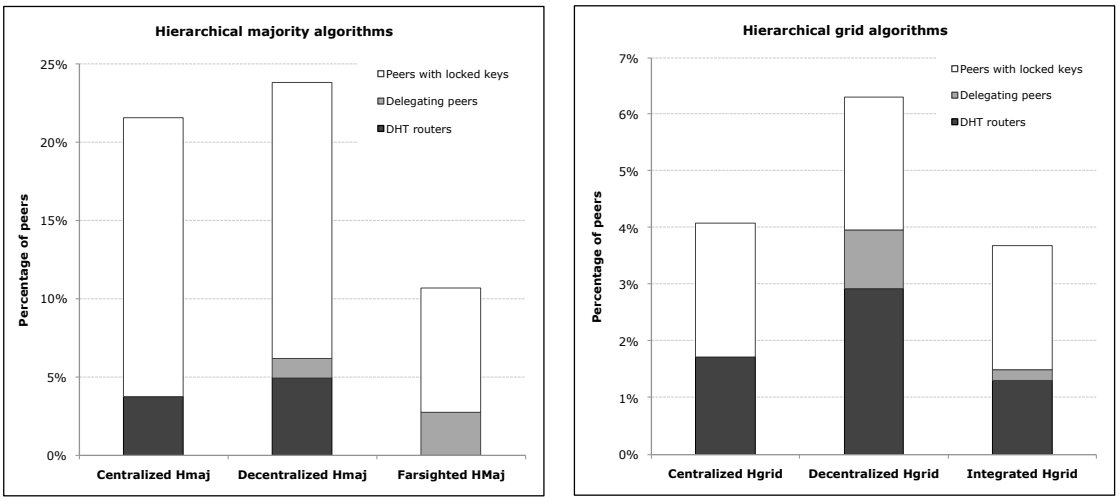


Figure 12: Role of peers during the acquisition of a quorum for different Chord network sizes.



(a) (b)

Figure 13: Role of peers during the acquisition of a quorum in a 10k peers Chord network.

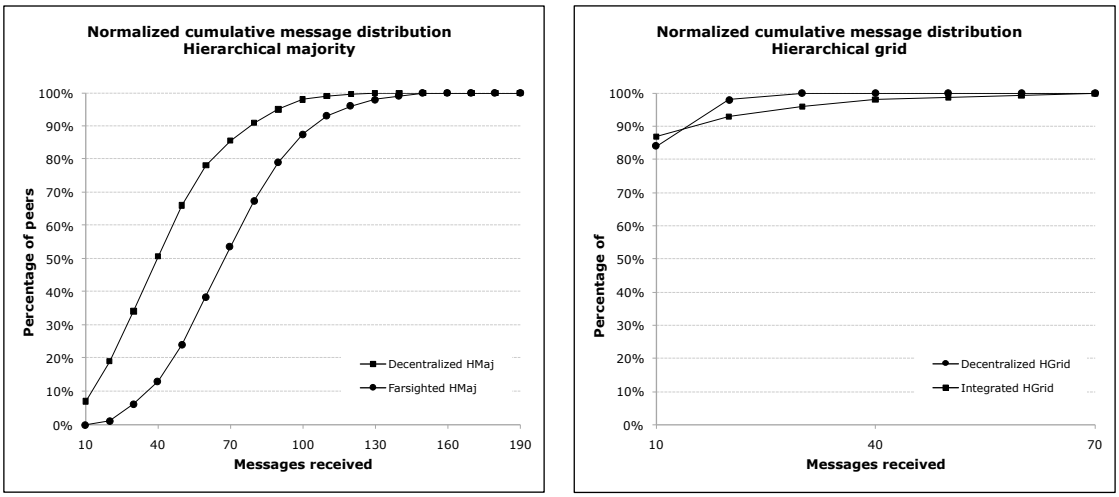
the 10k-peer case, in which the number of locked peers gets closer to the number of routing peers, integration has a substantial effect in reducing the number routing peers (either delegators or DHT routers). That is, decentralization should be combined with integration to be really effective.

6.5 Distribution evenness of messages among peers

In this subsection we explore how evenly the load for quorum acquisition is distributed among peers. To do so we plotted the distribution of messages among peers.

Figure 14 shows this distribution. The x axis represents the number of messages received, while the y axis shows the percentage of peers that received that specific number of messages. This percentage is w.r.t. the total number of peers that received at least one message. Note also that the distribution is cumulative.

Intuitively an implementation that perfectly balances the load among peers would present a step-like distribution, that would mean that every peer receives the same number of messages (i.e., if there are M messages and N peers, the graph would be at 0% till the M/N point in which it would go to 100%). A first observation is that all graphs show a very steep growth what means that the distribution is reasonably balanced in general. A second observation lies in that integrated versions distribute messages more evenly



(a) (b)

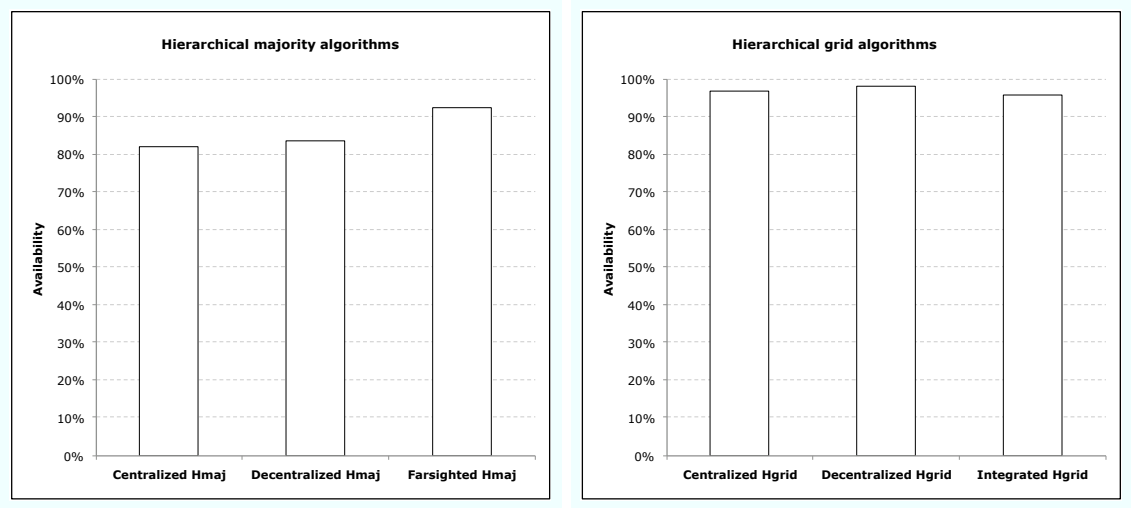
Figure 14: Distribution of messages among peers during a quorum acquisition.

as denoted by their steeper slope. HGrid versions converge faster to 100% due to their smaller quorum size.

6.6 Quorum availability under failure

Section 5 showed that the DHT-based overlay network is capable of handling peer failures, transferring the responsibility of keys pertaining to a failed peer to one of its neighbors. However, this transfer takes time and the new responsible peer for the transferred keys needs additional time to restore the original key state. During this time, in order to guarantee the quorum intersection property, transferred keys cannot be granted for quorum requests. As a consequence, during this transitory period of time, some quorums may not be available. This section evaluates the impact of a peer failure on quorum availability in this specific period of time.

To asses this impact we evaluated all six algorithms in a scenario with 10000 peers. In each test we chose one single failed peer at random and changed all its keys' state to *unknown*. We then started acquiring quorums at random and calculated the average quorum availability as the ratio between the number of successful quorum requests (i.e. requests of quorums that did not include keys in the *unknown* state) and the total number of quorum requests.



(a) (b)

Figure 15: Quorum availability after a peer failure.

Figure 15 shows the results for hierarchical majority (a) and hierarchical grid (b) algorithms. Each value reported in the graphs is the result of 5000 independent quorum requests executed on 5 different instances of the same scenario with a keyspace containing 2^{30} keys.

As the plots show the impact of a peer failure on global quorum availability is sometimes non negligible, especially for some of the hierarchical majority algorithms; this is a consequence of the fact that these algorithms tend to spread the keys chosen from a quorum more uniformly among the whole keyspace thus increasing the probability that one of these keys will be in the *unknown* state (note that keys in this state are usually clustered together in that part of the keyspace that was under the responsibility of the failed peer). This behaviour is confirmed by the values obtained for the farsighted hierarchical majority algorithm: thanks to its greater flexibility, and the adoption of the 4-1-1-1 pattern, this algorithm is able to build quorums that both (i) contain less keys and (ii) choose keys more clustered in the keyspace; these two aspects allow the farsighted hierarchical majority algorithm to reduce the probability of choosing one (or more) keys in the *unknown* state. Results for the hierarchical grid algorithms don't show the same degree of variability: all the three algorithm variants showed similar behaviours with respect to availability. Their overall better performance can be directly related to the fact the these algorithms are able to build quorums containing

less keys.

Note that we expect the impact of a peer failure on quorum availability to further decrease for settings with more participating peers: the larger is the number of peers, the smaller will be the number of keys assuming the *unknown* state as a consequence of a failure. This is a consequence of the ability of the hash function used by the DHT to uniformly distribute peers in the key space. Conversely, we expect quorum availability to severely decrease when multiple failures happen simultaneously; for this reason we believe the mechanisms introduced in Section 5 can hardly be adopted in a non cooperative setting (typical of common internet-based P2P applications like file sharing) with strong churn where a lot of peers can abruptly leave the system without handing off their keys to a neighbor peer.

7 Related Work

Quorum systems have been largely investigated in the distributed systems literature. Several surveys and comparison studies exist revising the various possibilities for building quorums as well as their performance trade-offs [17, 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. [24] 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 [2]) 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 [5]. Here, the finger table is used to forward messages to several peers 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, whether in our case the chosen keys have to form a quorum.

Besides [24], the same authors also propose in [14] algorithms and techniques for building a quorum system over a basic DHT, such as Chord. The idea is similar to that in [24]. Moreover, a random back-off technique is included in order to maintain constant throughput when concurrency of requests increases and subsequently there is more possibility 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 more sophisticated quorum systems that can be acquired more efficiently and with smaller quorum sizes.

In [16] a scalable dynamic quorum system supporting joins and departures of nodes is presented. This paper presents the dynamic counterpart of the Paths [17] quorum system, 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 [15]. In other words, rather that building the quorum over a separate DHT layer, 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 [18].

Another quorum system that exploits a geometrical CAN-like space is the d-space system presented in [19]. 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 [6, 7, 10]. The P2P architecture for multiplayer gaming presented in [10], relies on a single (but replicated) coordinator node for maintaining global shared data, that realizes in practice a monarchy quorum. Though monarchy quorum provides best latency and acquisition cost, it obviously presents poor availability, as confirmed by experimental results presented in [10].

8 Conclusions

P2P technology is becoming pervasive in new applicative contexts different from the classical internet-based file sharing, such as service management in data centers, distributed data storage and retrieval or trust-less distributed file systems for large enterprises. These new contexts have dramatically different characteristics and requirements (e.g., long vs. short peer lifetime, managed environments vs. unmanaged ones, cooperative vs. uncooperative peers) and they have to support much more complex operations (e.g., managing contractual SLA data vs. file location). As remarked in the introduction such deep differences between enterprise and Internet-scale settings lead to the need of designing P2P technologies (e.g., algorithms and mechanisms) optimized for a specific environment while keeping generic the very basic P2P functionalities such as the capability to of peers to self-organize into an overlay network and collectively route data.

Due to these reasons we studied how to architect quorum systems on top of a DHT-based P2P infrastructure which is able for example to provide a peer with low latency exclusive access to the entire P2P system when needed. This capability is vital for enterprise infrastructures (for example to implement auditing, monitoring and maintenance procedures) while, at the same time, it is not a main issue for inter-domain settings. Therefore we designed algorithms for building quorums in a P2P system characterized by graceful leaves and relatively small failure rates typical of a managed enterprise infrastructure.

Specifically, in this paper we focussed on design principles, namely integration, delegation and quorum flexibility, that should guide the construction of quorum systems and protocols in DHT-based enterprise infrastructures. These design principles have been exercised in two quorums systems, hierarchical majority and hierarchical grid. For each quorum system three versions have been implemented and evaluated: centralized, decentralized and integrated. For hierarchical majority the integrated version also included quorum flexibility by the use of the farsighted quorum system that provides higher flexibility than traditional hierarchical majority. The evaluation has proven that traditional centralized versions for acquiring quorums are not scalable in a P2P context due to the huge number of messages generated. What is more, the performance metrics were devised for networks in which the cost of communication between any pair of sites was the same, what is not true in a P2P network. We have introduced two novel performance metrics: quorum

acquisition cost and latency to better characterize the performance of quorum systems in P2P networks. The evaluation has also shown that decentralization by itself yielded to a high number of routing peers and latency. Integration effectively diminished the number of routing peers. Integration, mixed with quorum flexibility in the farsighted quorum system, was also able to reduce substantially the latency introduced by decentralization.

Acknowledgments

The authors want to thank the subject area editor, A. Schuster, for the insightful discussions during the review process that improved considerably the content of the paper.

References

[1] W. J. Bolosky, J. R. Douceur, D. Ely, M. Theimer, Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs, in: SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, ACM, New York, NY, USA, 2000, pp. 34–43.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Rowstron, Scribe: A large-scale and decentralized application-level multicast infrastructure, IEEE Journal on Selected Areas in Communications (JSAC) 20 (8) (2002) 1489–1499.

[3] S. Y. Cheung, M. H. Ammar, M. Ahamad, The grid protocol: A high performance scheme for maintaining replicated data, in: Proceedings of the 6th International Conference on Data Engineering (ICDE), IEEE Computer Society, 1990, pp. 438–445.

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, Dynamo: amazon's highly available key-value store, in: SOSP '07: Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles, ACM, New York, NY, USA, 2007, pp. 205–220.

[5] S. El-Ansary, L. O. Alima, P. Brand, S. Haridi, Efficient broadcast in structured p2p networks, in: M. F. Kaashoek, I. Stoica (eds.), Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS), vol. 2735 of Lecture Notes in Computer Science, Springer, 2003, pp. 304–314.

[6] P. Ganesan, P. K. Gummadi, H. Garcia-Molina, Canon in g major: Designing dhts with hierarchical structure, in: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS), IEEE Computer Society, 2004, pp. 263–272.

[7] P. Ganesan, Q. Sun, H. Garcia-Molina, Adlib: A self-tuning index for dynamic p2p systems, in: Proceedings of the 21st International Conference on Data Engineering (ICDE), IEEE Computer Society, 2005, pp. 256–257.

[8] D. K. Gifford, Weighted voting for replicated data, Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP) (1979) 150–162.

[9] R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, B. Kemme, Are quorums an alternative for data replication ?, ACM Transactions on Database Systems 28 (3) (2003) 257–294.

[10] B. Knutsson, H. Lu, W. Xu, B. Hopkins, Peer-to-peer support for massively multiplayer games, in: Proceedings of The 23rd Conference of the IEEE Communications Society (INFOCOM), 2004.

[11] S. Ktari, M. Zoubert, A. Hecker, H. Labiod, Performance evaluation of replication strategies in dhts under churn, in: MUM '07: Proceedings of the 6th international conference on Mobile and ubiquitous multimedia, ACM, New York, NY, USA, 2007, pp. 90–97.

[12] A. Kumar, Hierarchical quorum consensus: A new algorithm for managing replicated data, IEEE Transactions on Computers 40 (9) (1991) 996–1004.

[13] A. Kumar, S. Y. Cheung, A high availability $\sqrt{N}$ hierarchical grid algorithm for replicated data, Information Processing Letters 40 (6) (1991) 311–316.

[14] S. Lin, Q. Lian, M. Chen, Z. Zhang, A practical distributed mutual exclusion protocol in dynamic peer-to-peer systems, in: G. M. Voelker, S. Shenker (eds.), Proceedings of the 3nd International Workshop on Peer-to-Peer Systems (IPTPS), vol. 3279 of Lecture Notes in Computer Science, Springer, 2004, pp. 11–21.

[15] M. Naor, U. Wieder, Novel architectures for p2p applications: the continuous-discrete approach, in: Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), ACM, 2003, pp. 50–59.

[16] M. Naor, U. Wieder, Scalable and dynamic quorum systems, in: Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC), 2003, pp. 114–122.

[17] M. Naor, A. Wool, The load, capacity, and availability of quorum systems, SIAM Journal of Computing 27 (2) (1998) 423–447.

[18] S. Ratnasamy, M. Handley, R. M. Karp, S. Shenker, Application-level multicast using content-addressable networks, in: J. Crowcroft, M. Hofmann (eds.), Networked Group Communication, vol. 2233 of Lecture Notes in Computer Science, Springer, 2001, pp. 14–29.

[19] B. D. Silaghi, P. J. Keleher, B. Bhattacharjee, Multi-dimensional quorum sets for read-few write-many replica control protocols, in: Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid), IEEE Computer Society, 2004, pp. 355–362.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM), 2001, pp. 149–160.

[21] C. Tang, R. N. Chang, E. So, A distributed service management infrastructure for enterprise data centers based on peer-to-peer technology, in: Proceedings of the IEEE International Conference on Services Computing (SCC), IEEE Computer Society, 2006, pp. 52–59.

[22] C. Tang, M. Steinder, M. Spreitzer, G. Pacifici, A scalable application placement controller for enterprise data centers, in: C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, P. J. Shenoy (eds.), Proceedings of the 16th international conference on World Wide Web (WWW), ACM, 2007, pp. 331–340.

[23] R. H. Thomas, A majority consensus approach to concurrency control for multiple copy databases, ACM Transactions on Database Systems 4 (2) (1979) 180–209.

[24] Z. Zhang, The power of dht as a logical space, in: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS), IEEE Computer Society, 2004, pp. 325–331.