Distributed Computing manuscript No.
(will be inserted by the editor)

Probabilistic Quorums for Dynamic Systems

Ittai Abraham, Dahlia Malkhi

The Hebrew University of Jerusalem e-mail: {ittaia,dahlia}@cs.huji.ac.il

The date of receipt and acceptance will be inserted by the editor

1 Introduction

A classic approach for maintaining information in distributed settings is to post data items to, and retrieve information from, subsets of the system called quorums that have the property that any two quorums have a non empty intersection. Due to the intersection property, each quorum of processes collectively guarantees access to all previously posted data items. Quorum systems are attractive for two reasons. First, they offer high availability, as the system may continue serving requests in face of failures, so long as some quorum of accessible processes exists. Second, they promote load balancing, as each individual process suffers only the load of requests done on quorums that include it. For a comprehensive theory of quorum techniques, see [26,32]. The theory of quorum systems tells us, among other things, that there is an inherent tradeoff between the reduced load on individual system members, inflicted by quorum accesses, and the high fault tolerance of the system.

Probabilistic quorum systems (PQS), introduced in [29], use randomization to circumvent this tradeoff and provide optimal load and availability simultaneously, while relaxing the strict intersection property to a probabilistic one. Informally, in a probabilistic quorum system, quorum members are selected at random according to some distribution, such that two selected quorums intersect with high probability. Compared with strict quorum systems, PQSs are natural for dynamic and non-structured environments: Finding members can be done in parallel and efficiently, and replacing failed members is trivial.

This paper introduces enhanced PQS techniques that cope with scalable, highly decentralized and highly dynamic settings. It addresses two challenges. First, it assumes that no process in the system has a global view of the system participants. More concretely, we allow each process to maintain connections with, and even knowledge of, only a constant number of other members. This restriction stems both from our vision of having processes run on ubiquitous, low-memory devices, and participate in Internet resource sharing applications; and from the desire to keep the amount of state that needs to be updated at reconfiguration very low. To this end, it addresses the issue of selecting quorum members without such global view. Second, it provides an evolution scheme over time for quorum-replicated data, when the system of which this quorum is part of grows/shrinks.

Non-uniform PQS. The first problem we address is that in order to uniformly select quorum members at random, a process would need to know all other processes. In order to avoid global information, we introduce a *non-uniform PQS* as follows. Let us have any probability distribution $p : S \rightarrow [0..1]$ over individual members of the system S, where $|S| = n$. We define the *flat access strategy* $f(p, m)$ as the quorum selection distribution obtained by randomly choosing m members (with repetitions) where each member is independently chosen according to the distribution p. We generalize the uniform distribution PQSs by showing that for any parameter ρ, a quorum system with the access strategy $f(p, \rho\sqrt{n})$, for any probability distribution p, guarantees quorums intersection with probability $1 - e^{-\rho^2/2}$.

It is left to show how to realize a process selection distribution p in dynamic settings and how to preserve it in evolving quorums. Our approach is to form an on-the-fly overlay graph among the processes. Then from any process, selecting $\rho\sqrt{n}$ other processes is done by performing random walks on the graph.

More specifically, our design employs a dynamic routing graph based on the dynamic approximation of the de Bruijn network introduced in [1] [1]. Using techniques introduced in [1], we show how to insert nodes into the overlay graph and how to remove them. We also prove that random walks of reasonable ($\log(n)$) length give us the independent node selection distribution p that we need.

The dynamic graph also allows to estimate the size of the network n with a constant error factor. Using this estimation, the flat access strategy $f(p, \rho\sqrt{n})$ is approximated by performing roughly $\rho\sqrt{n}$ random walks of $\log(n)$ steps each. We obtain at any instant in time

[1] The dynamic De Bruijn construction appeared independently also in [30,11,20].

an $e^{-t^2/2}$-intersecting PQS. Accessing quorums is done in $\log(n)$ parallel time.

Quorum evolution. The second issue is how to evolve quorums as the system grows. We devise an evolution strategy that grows the quorums along with the system's growth automatically and distributively. We prove that our evolution technique keeps quorums sufficiently large, as well as maintains the individual member selection distribution p. The cost of our maintenance algorithm is w.h.p. [2] a constant number of random walks per system reconfiguration. Each single walk incurs a logarithmic number of messages and a state change in one process.

Summary of contribution. To summarize, the result of our construction is a scalable information sharing mechanism based on dynamic PQSs. For a fixed $\varepsilon > 0$, the construction maintains $1-\varepsilon$ intersection probability in any dynamic setting, without central coordination or global knowledge. We extend the treatment of PQSs to cope with scalability and high dynamism in the following ways. First, we allow each participant only partial knowledge of the full system, and avoid maintaining any global information of the system size and its constituents. To this end, we develop a theory of PQSs whose individual member selection probability is non-uniform. We demonstrate a realization of such a non-uniform PQS that is fully adapted for the dynamic and scalable settings we aim for. The second extension of PQSs we address is to evolve quorums as the system grows/shrinks in order for them to remain viable. We provide both a formal definition of quorum evolution and the algorithms to realize it.

The resulting scheme achieves the following performance measures with high probability: The cost of a member addition (join) is a logarithmic number of messages, and a state-change in a constant number of members. Because the $O(\sqrt{n})$ processes are chosen independently at random using $O(\log n)$ length random walks, quorum selection may be done in $O(\log n)$ parallel time.

1.1 Related Work

Our work touches both on quorum systems techniques for managing distributed data and on scalable data sharing systems. We describe related work in both of these areas.

We begin with a brief mentioning of landmark works on strict quorum systems. Quorum systems, originally introduce by Gifford [13] and Thomas [38], are tools for increasing the availability and efficiency of replicated services. A *quorum system* for a universe of servers is a collection of subsets of servers, each pair of which intersect. Intuitively, each quorum can operate on behalf of the system, thus increasing its availability and performance, while the intersection property guarantees that operations done on distinct quorums preserve consistency. The connection between voting and quorum systems was explored in [12], demonstrating that the latter includes more flexibility. Maekawa suggests in [25] an efficient quorum construction, the first quorum system with quorums of size $\sqrt{n}$, where n is the size of the system, and in which every pair of quorums intersect in exactly one element. Naor and Wool introduce in [32] formal performance measures for quorum systems, including load and availability. They provide a lower bound of $1/\sqrt{n}$ on the load of any quorum system, and demonstrate an inherent availability-load tradeoff. Malkhi and Reiter initiate in [27] the study of Byzantine quorum systems, intended for environments prone not only to data inavailability resulting from benign process or communication failures, but also to arbitrary data corruption.

We now remind the reader of PQSs and motivate their use. The PQSs of Malkhi et al. [29] are an attractive approach for sharing information in a large network. Using a PQS, each participant can disseminate new updates to shared data by contacting a subset (a probabilistic quorum) of $\rho\sqrt{n}$ processes chosen uniformly at random, where ρ is a reliability parameter. Likewise, participants query data from such quorums. Intuitively, analysis similar to the famous "birthday paradox" (e.g., see [10]) shows that each pair of update/query quorums intersect with probability $1-e^{-\rho^2/2}$. The result is that with arbitrarily good probability a query obtains up to date information, and with a small calculated risk it might obtain stale data.

The benefit of the PQS approach is that publicizing information and global querying are done each with only a $O(1/\sqrt{n})$ fraction of the participants. At the same time, PQSs maintain availability in face of as many as $O(n)$ faults. In deterministic approaches these two features are provably impossible to achieve simultaneously (see [32]). Indeed, PQSs have been employed in diverse and numerous settings. To name just a few deployments, PQSs were used for designing probabilistic distributed emulations of various shared objects [22,23]; they were used for constructing persistent shared objects in the Fleet system [28] and the Aquarius system [7]. Probabilistic quorum techniques were also applied for maintaining tracking data in mobile ad-hoc networks [15,14,4]. In the latter applications, quorum member selection policy is highly influenced by the dynamics and geometry of the ad-hoc network, hence not necessarily uniform. The simulations presented in these works indicate good performance, i.e., high probability of quorum intersection.

We now get to survey works on dynamic quorums. A limitation of all the quorum systems mentioned above is that they require their clients to know a priori the quorum construction and its members. In scalable and dynamic settings, this may pose an infeasible cost. Recently, a number of dynamic quorum selection techniques were developed for the purpose of tracking mobile hosts in wireless ad hoc networks [36,37]. These works vari-

[2] In this paper w.h.p. means that the probability of this event is at least $1-1/n$ where n is the number of nodes in the system.

ate on the following basic principle. Location servers are situated in a plane, such that each host can communicate with its close-by neighbors. A quorum emanating from any participant is constructed by forming on the fly a north-south path and an east-west path (or more general trajectories), thus guaranteeing high likelihood of intersection among quorums. Hosts arrive and depart dynamically, replacing or being replaced in the quorums they belong to accordingly. This guarantees intersection of past quorums with new ones. Naor and Wieder adopt the same approach in [31] using virtual plane coordinates for dynamic quorum maintenance in general settings. The approach taken in [16,9] for dynamic quorum maintenance is to assign virtual coordinates for quorum members, and dynamically assign actual servers to take their role according to network settings. Compared with our approach, all of the above works deal with strict quorum systems. We view probabilistic quorum systems as particularly suitable for scalable dynamic settings, in that they naturally allow quorum members to be accessed in parallel, and in that a failed access attempt is easily replaced by another member access.

Previously, the construction of dynamic probabilistic quorums was addressed in AntWalk [33]. In order to provide uniform member selection with partial knowledge, AntWalk necessitates periodic, global "re-mixing" of the links of old members with those of the new processes that arrived. We consider the price of this approach too heavy for Internet wide applications.

We now more generally look at scalable data sharing facilities. The advent of the Internet and ubiquity of computing resources led to rising interest in information sharing services. We do not attempt to provide a comprehensive survey of information sharing projects here, but focus on distributed directory services, that had influenced our work. A distributed directory associates information with a name (a key), and scatters directory entries around the network. In large scale distributed directories the main challenge is to route queries to the location (or locations) that contain the value corresponding to a particular name. They achieve this by constructing an overlay routing infrastructure. Examples of directories include DNS, the most widely deployed distributed directory to date, as well as a number of Distributed Hash Tables (DHTs) [35,34,39]. Unlike our approach, directory services do not support global querying of names, and cannot support fuzzy forms of searching (this was stated as an open problem in [18]). Additionally, the data itself is often held in such systems in a single location, whereas part of our goal is to provide high availability data. Thus, the main idea we borrow from the DHT paradigm is the dynamic routing overlay, which we employ in order to find quorum members.

Content sharing facilities currently deployed on the Internet do facilitate more flexible searching. For example, Gnutella (`http://en.wikipedia.org/wiki/Gnutella`) supports global querying through a probabilistic depth-bounded multicast. This approach is effective, and resembles ours, yet it is ad hoc. Our work provides formal background and analysis by which rigorous semantics can be assured.

Another problem area closely related to ours is information dissemination and content delivery in scalable networks. Several recent systems employ peer-to-peer overlay routing infrastructure to build scalable group multicast services, e.g., Bayeux [40], SplitStream [6], and SCRIBE [24]. Other protocols employ randomization using epidemic style gossiping for scalable information dissemination, including Bimodal multicast [5] and [21]. All of these achieve our goal of having partial system view at each participant. However, their goal is to quickly spread write-once information, and they do not handle searching or data longevity.

Finally, it should be noted that this paper is based on the preliminary conference version in [3].

2 Problem Definition

We consider a (potentially infinite) universe W of possible processes. The system consists of a dynamic subset of processes taken from W that evolves over time as processes join and leave the system. We use a logical discrete time-scale $T = \{0, 1, \dots\}$. At each time-step $i \in T$ there exists a set $U(i)$ of processes from W that are considered members of the system at that time. Each time-step i consists of a single event $e(i)$, which is one of the following: Either a process joins the system, or a process leaves the system. For each time step $t > 0$, the partial history of events uniquely determines the universe $U = U(t)$ consisting of all the processes that joined the system minus those that have left. It should be clear that this abstraction is done for reasoning purposes only; in practice, many arrivals and departures occur simultaneously, but their modeling unnecessarily complicates the treatment.

Focusing on a fixed time step $t > 0$ for now, we first recall the relevant definitions from [29]. A *set system* $\mathcal{Q}$ over a universe U is a set of subsets of U. A *(strict) quorum system* $\mathcal{Q}$ over a universe U is a set system over U such that for every $Q, Q' \in \mathcal{Q}$, $Q \cap Q' \neq \emptyset$. Each $Q \in \mathcal{Q}$ is called a *quorum*. An *access strategy* ac for a set system $\mathcal{Q}$ specifies a probability distribution on the elements of $\mathcal{Q}$. That is, $ac : \mathcal{Q} \to [0, 1]$ satisfies $\sum_{Q \in \mathcal{Q}} ac(Q) = 1$. From here on, we use the notation $Q \sim ac$ to denote that Q is a random variable whose probability function is ac. We are now ready to state the definition of probabilistic quorum systems:

Definition 1 (ϵ-intersecting quorum system[29]). *Let $\mathcal{Q}$ be a set system, let ac be an access strategy for $\mathcal{Q}$, and let $0 < \epsilon < 1$ be given. The tuple $\langle \mathcal{Q}, ac \rangle$ is an* ϵ-intersecting quorum system *if $\Pr[Q \cap Q' \neq \emptyset] \geq 1 - \epsilon$, where $Q, Q' \sim ac$, the probability of choosing Q and Q' is taken with respect to the strategy ac.*

We now proceed to define time-evolving quorums. A time-evolving quorum system has a fixed creation time s and a series $\langle \mathcal{Q}(i), ac_i \rangle$ of quorum systems for every time step after its creation $s \leq i \in T$. In order to formally

define the way the quorum system evolves, we first define an evolution strategy as follows:

Definition 2 (Evolution strategy). *For every $t \in T$, let $\mathcal{Q}(t)$ be a set system over the system $U(t)$. An* evolution strategy *ev_t specifies a probability distribution on the elements of $\mathcal{Q}(t)$ for each given element of $\mathcal{Q}(t-1)$. Formally, $ev_t : \mathcal{Q}(t-1) \times \mathcal{Q}(t) \to [0,1]$ satisfies*

$$\forall Q' \in \mathcal{Q}(t-1) : \sum_{Q \in \mathcal{Q}(t)} ev_t(Q', Q) = 1 \ .$$

Thus, $ev_t(Q', Q)$ for $Q' \in \mathcal{Q}(t-1)$ and $Q \in \mathcal{Q}(t)$ indicates the probability that Q' evolves into Q.

The access strategies over $U(1), U(2), \ldots$ together with an evolution strategy determine the probability that a certain subset occurs as the evolution of any previously created quorum. The following definition captures this distribution:

Definition 3 (Evolving probability distribution). *Fix a creation time $s \in T$. For every time step $s \leq i \in T$, let $\langle \mathcal{Q}(i), ac_i \rangle$ be a probabilistic quorum system and ev_i be an evolution strategy. The* evolving probability distribution *$p_t^s : \mathcal{Q}(t) \to [0,1]$ for quorums created at time s that evolved up to time t, for $t \geq s$, is defined recursively as follows:*

$$\forall Q \in \mathcal{Q}(t) :$$

$$p_t^s(Q) = \begin{cases} ac_s(Q) & t = s, \\ \sum_{Q' \in \mathcal{Q}(t-1)} p_{(t-1)}^s(Q') ev_t(Q', Q) & t > s. \end{cases}$$

Our goal is to devise a mechanism for maintaining ε-intersecting probabilistic quorums in each $U(i)$ for all $s \leq i$, and to evolve quorums that maintain information (such as updates to data) so that their evolution remains ε-intersecting with quorums in later time steps. Any two quorums created at times s and t will evolve in a manner such that at any later time r, their intersection probability remains $1 - \varepsilon$. This is captured in the following definition:

Definition 4 (Dynamic ϵ-intersecting probabilistic quorum system). *For every time step $i > 0$, let $\langle \mathcal{Q}(i), ac_i \rangle$ be a probabilistic quorum system and ev_i be an evolution strategy. Let $0 < \epsilon < 1$ be given. Then $\langle \mathcal{Q}(i), ac_i, ev_i \rangle$ is a* dynamic ϵ-intersecting quorum system *if for all $r \geq s \geq t > 0, Q, Q' \in \mathcal{Q}(r)$:*

$$\Pr[Q \cap Q' \neq \emptyset] \ \geq \ 1 - \epsilon$$

where the probability is taken over the choices of Q and Q', distributed respectively according to $Q \sim p_r^s$ and $Q' \sim p_r^t$.

In words, if Q is a quorum chosen at time s from $U(s)$ using strategy ac_s and evolved using ev_i until time r ($r - s$ evolution steps), and Q' is a quorum chosen at time t from $U(t)$ using strategy ac_t and evolved using ev_i until time r ($r - t$ evolution steps) then the probability that Q and Q' do not intersect is less than ϵ.

2.1 Performance goals

Driven by our goal to maintain quorums in very large and dynamic environments, such as Internet-wide peer-to-peer applications, we identify the following four performance goals. (Formal definitions are given in Section 6.) First, we strive to keep the join/leave message complexity low (logarithmic), and the number of state-changes per reconfiguration a small constant. Second, we wish for an efficient accessing procedure that finds quorums and reads or writes from/to them. Additionally, we consider two traditional measures that were defined to assess the quality of probabilistic quorum systems [32,29]: The *load* inflicted on processes is the fraction of total updates/queries they must receive. The degree of *resilience* is the amount of failures tolerable by the service. Our goals with respect to the latter two measures are to preserve the good performance of PQSs in static settings. Specifically, we wish for the load to be $O(1/\sqrt{n})$ and the resilience to be $O(n)$.

3 Non-uniform Probabilistic Quorum Systems

In this section, we extend the treatment of probabilistic quorum systems of [29] to constructions that employ non-uniform member selection.

Let S be a system containing n members (e.g., $S = U(t)$ for some $t > 0$). Let $p(s)$ be any distribution over the members $s \in S$. We first define a flat non-uniform selection strategy that chooses members according to p until a certain count is reached.

Definition 5 (Flat access strategy). *The* flat access strategy *$f(p,m) : 2^S \to [0,1]$ is defined as follows: for $Q \in 2^S$, $f(p,m)(Q)$ equals the probability of obtaining the set Q by repeatedly choosing m times (with repetitions) from the universe S using the distribution p.*

The flat strategy $f(p,m)$ strictly generalizes the known access strategy for PQSs in which members are chosen repeatedly m times using a uniform distribution. In the Lemma below, we obtain a generalized probabilistic quorum system with non-uniform member selection.

Lemma 1. *The construction $\langle 2^S, f(p, \rho\sqrt{n}) \rangle$ is an $(e^{-\rho^2/2})$-intersecting quorum system.*

Proof. Consider two sets $Q, Q' \sim f(p, \rho\sqrt{n})$. For every $s \in S$ denote an indicator variable x_s that equals 1 if $s \in Q \cap Q'$, and equals 0 otherwise. Thus, $E[\sum_{s \in S} x_s] = t^2 n p^2(s)$. By the Cauchy-Schwartz inequality, we have $\sum_{s \in S} p^2(s) \frac{1}{n} \geq \left(\sum_{s \in S} p(s) \frac{1}{n}\right)^2$. Combining the above: $E(\sum_{s \in S} x_s) = t^2 n \sum_{s \in S} p^2(s) \geq t^2$.

We now wish to apply Chernoff bounds to bound the deviation from the mean. Since the x_s's are dependent, we cannot apply the bounds directly. Rather, we define i.i.d. random variables $y_s \sim x_s$. Clearly, $E[\sum_s y_s] = E[\sum_s x_s]$. Due to a result by Hoeffding [19], we have: $Pr[Q \cap Q' = \emptyset] = Pr[\sum_{s \in S} x_s = 0] \leq Pr[\sum_{s \in S} y_s = 0] < e^{-t^2/2}$. □

Interestingly, the flat access strategy is overly conservative in the following sense. Generally, a quorum selection strategy with non-uniform member selection distribution need not necessarily have a fixed quorum size. Intuitively, this is because "heavier" members (that are chosen with a higher probability) are more likely to occur in the intersection among pairs of quorums. An example might clarify this point: Suppose that some member $s \in S$ has $p(s) = 1/2$. Clearly, if s belongs to a quorum, then the probability of intersecting with any other quorum is at least a half, even if quorums have only one element each. In the general case, the total number of selected members could therefore depend on their combined weight. We encountered a difficulty in obtaining such a "weighted" access strategy, namely that the likelihood that a member is included in a quorum depends on the ordering of sampling. We are currently still investigating whether there is a way to implement a non-uniform variable-size quorum access strategy along these lines.

Finally, note that implementing $f(p, \rho\sqrt{n})$ requires global knowledge of n, which is difficult in a dynamic setting. The remaining of this paper is devoted to approximating f, i.e., we show how to (roughly) maintain a non-uniform flat quorum access strategy and how to evolve quorums, over a dynamic system.

4 Non-uniform Probabilistic Quorums in Dynamic Systems

4.1 The dynamic graph

A key component in the construction is designing a dynamic overlay routing graph among the processes. The graph allows processes to search for other processes during quorum selection while maintaining low memory overhead. We assume that the underlying communication network is a complete graph but the overlay routing graph may be partial. Denote by $G(t) = \langle V(t), E(t)\rangle$ a directed graph representing the system's routing overlay network at time t as follows. $V(t)$ is the set of processes in $U(t)$ at time point $t > 0$. A directed edge $(u, v) \in E(t)$ indicates that u knows the network identifier of v and hence u can communicate directly with v. Henceforth, we refer to system participants as processes or as nodes interchangeably.

One option is to maintain $G(t)$ as a complete graph over all the system participants. However this would cause high (linear) join/leave complexity. Driven by the need to maintain the goals stated above in Section 2.1, we wish to maintain a dynamic graph $G(t)$ with the following properties: (1) Small constant degree (so as to maintain constant join/leave complexity). (2) Logarithmic routing complexity, so time to select a quorum logarithmic. (3) Rapid mixing time [3], so that we can maintain a fixed individual selection distribution independent of the origin using a small number of steps from each node.

We choose to employ for $G(t)$ a routing graph that approximates a de Bruijn routing graph. In the de Bruijn [8] of order k there are 2^k nodes each with a unique identifier in $\{0,1\}^k$. Every node has two outgoing links: Node $\langle a_1, \ldots, a_k\rangle$ has an edge to the two nodes $\langle a_2, \ldots, a_k, 0/1\rangle$ (shift, then set the last bit). We employ a dynamic approximation of the De Bruijn graph that was introduced in [1]. This dynamic graph has w.h.p. a constant-degree, logarithmic routing complexity, and logarithmic mixing time.

The dynamic graph is constructed dynamically as follows. Each node has a binary identifier. We maintain two properties. First, the identifiers of the nodes always form a *complete prefix code*.

Definition 6 (Complete prefix code). *We say that a set of nodes has a* Complete prefix code *property if no identifier is a prefix of another identifier and for every infinite binary string $S \in \{0,1\}^*$ there exists an identifier that is a prefix of S.*

Second, the graph linking (edge set) is a *dynamic de Bruijn* linking, defined as follows:

Definition 7 (Dynamic de Bruijn linking). *We say that a graph has a* dynamic de Bruijn *linking if each node whose id is $\langle a_1, \ldots, a_k\rangle$ has an edge to each node whose id is $X = \langle a_2, \ldots, a_k\rangle$ or whose id is a prefix of X, or whose id has X as a prefix.*

For example consider a system with 5 nodes whose identifiers are $11, 10, 01, 001, 000$. They clearly form a complete prefix code. For Dynamic de Bruijn linking, node 11 has links to 10 and itself, node 10 has links to nodes $01, 001, 000$, node 01 has links to $11, 10$, node 001 has links to node 01 , node 000 has links to 001 and itself.

We assume that initially, G_1 has two members that bootstrap the system, whose id's are 0 and 1. Joining and leaving of members is done as follows:

JOIN: When a node u joins the system, it chooses some member node v and "splits" it. The way a node chooses a member will be described in the load balancing section. Specifically, let $v.id = \langle a_1, \ldots, a_k\rangle$ be the identifier v has before the split. Then u uniformly chooses $i \in \{0,1\}$, obtains identifier $u.id = \langle a_1, \ldots, a_k, i\rangle$ and v changes its identifier to $v.id = \langle a_1, \ldots, a_k, (1-i)\rangle$. The links to and from v and u are updated so as to maintain the dynamic de Bruijn linking, as follows. If previously v had a link $\langle a_2, \ldots, a_k\rangle$ or a prefix, then both u and v link to it; a link of the form $\langle a_2, \ldots, a_k, i\rangle$ or for which this is a prefix, is dropped from v's links and is copied to u, and links of the form $\langle a_2, \ldots, a_k, (1-i)\rangle$ or for which this is a prefix remain v's links, and are not copied over to u.

[3] Formally, every connected unweighted non-bipartite graph G induces an irreducible and aperiodic Markov chain M. The *stationary distribution* is the unique distribution π such that $\pi = M\pi$. The *mixing time* of M is the minimum integer m such that for any initial π_0, $\pi_0 M^m$ is the stationary distribution.

LEAVE: When a node u leaves the system, it finds a pair of 'twin' nodes $\langle a_1,\ldots,a_k,0\rangle$, $\langle a_1,\ldots,a_k,1\rangle$. The way a 'twin' is found will be described in the load balancing section. If u is not already one of them, it uniformly chooses $i\in\{0,1\}$, and swaps with $\langle a_1,\ldots,a_k,i\rangle$. Swapping exchanges both identities and links: The node whose identifier was $\langle a_1,\ldots,a_k,i\rangle$ now has u's identifier (and maintains all of u's links) and u's assumes the identifier $\langle a_1,\ldots,a_k,i\rangle$. More importantly, now u is one of the twins.

Now node u, with id $\langle a_1,\ldots,a_k,i\rangle$, leaves the system. Its twin $u'=\langle a_1,\ldots,a_k,(1-i)\rangle$ changes its identifier to $\langle a_1,\ldots,a_k\rangle$. The links to and from $\langle a_1,\ldots,a_k\rangle$ are updated so as to maintain the dynamic de Bruijn linking: The incoming and outgoing links of both u and u' are merged and kept by $\langle a_1,\ldots,a_k\rangle$.

Load balancing. Recall that by the dynamic construction above, we are hoping to maintain a constant-degree, logarithmic diameter graph. The key component required for this to occur w.h.p. is a *load balancing* strategy. A load balancing strategy determines which node to split upon arrival, and which node(s) to merge upon departure. The goal is to keep all nodes at approximately the same id length. More precisely, we introduce the following technical definitions:

Definition 8 (Level). *Given a node v with id $\langle a_1,\ldots,a_k\rangle$ with a length k identifier, we define its* level *as $\ell(v)=k$.*

Definition 9 (Global gap). *The* global gap *of a graph $G(t)$ is defined as* $\max_{v,u\in V(t)}|\ell(v)-\ell(u)|$.

Techniques for maintaining a constant-bound w.h.p. on the global gap in dynamic graphs such as $G(t)$ are presented in [1] with logarithmic per join/leave cost. Briefly, the randomized load balancing there is to select for splitting the lowest-level node among $\log(n)$ randomly drawn nodes; and to the contrary for merging, choose the highest-level pair among $\log(n)$. In [30] techniques are presented for maintaining a global gap of 2 with linear cost per join/leave.

From here on, we assume that w.h.p. a constant bound C on the global gap is maintained.

Graph properties. If the global gap is small, then a node can estimate the size of the network by examining its own level. This is stated in the following lemma:

Lemma 2. *Let $G(t)$ be a dynamic de Bruijn graph with global gap C. Then for all $u\in V(t)$: $2^{\ell(u)-C}\leq|V(t)|\leq 2^{\ell(u)+C}$.*

Proof. Since the global gap is C and the node has a length $\ell(u)$ identifier, than the maximum length of an identifier is $\ell(u)+C$. Since all ids are unique, there can be at most $2^{\ell(u)+C}$ nodes. Similarly, identifiers form a prefix code and have at least $\ell(u)-C$ digits. Hence there are at least $2^{\ell(u)-C}$ nodes. □

In addition, for a global gap C the dynamic de Bruijn graph has out-degree at most $O(2^C)$, and diameter $\log(n)+C$ [1].

4.2 Quorum selection

We now describe the quorum selection algorithm. A quorum is selected by performing multiple random walks. The elements of the quorum are the end recipients of these random walks. The number of random walks performed is a function of the bound on the global gap C, and the required probabilistic guarantees. For a ε-intersecting quorum system we fix a parameter $\rho=\sqrt{\ln\frac{1}{\varepsilon}}$.

For a node u to establish a read or a write quorum, it initiates

$$\left\lceil \rho\sqrt{2^{\ell(u)+2C}}\right\rceil$$

random walk messages. For every random walk that node u initiates, it creates a message M with a hop-count $\ell(u)$, an id $u.id$, and appends any payload A to the message, i.e., $M=\langle\ell(u),u.id,A\rangle$. Each node (including u) that receives a message $\langle j,id,A\rangle$ with a non zero hop-count $j>0$, forwards a message $M'=\langle j-1,id,A\rangle$, randomly to one of its outgoing edges. If $(u,v)\in E$ then the probability that u forwards the message to v is:

$$\Pr[u\text{ forwards to }v]=\frac{1}{2^{\max\{\ell(v)-\ell(u)+1,1\}}} \qquad (1)$$

Lemma 3. *The above function (Equation 1) is a well defined probability function.*

Proof. The proof is by induction of the splits and merges of the dynamic graph. Denote $f(u,v)=\frac{1}{2^{\max\{\ell(v)-\ell(u)+1,1\}}}$. For the initial graph with two nodes $0,1$ and edge set $0\to 0, 0\to 1, 1\to 0, 1\to 1$ it is clear that f is well defined.

Assume the function is well defined for $G(t)$ and let $G(t+1)$ be formed by a split operation that splits node $w=\langle a_1,\ldots,a_k\rangle$ into nodes $u=\langle a_1,\ldots,a_k,0\rangle$ and $v=\langle a_1,\ldots,a_k,1\rangle$. Due to the Dynamic de Bruijn linking the following will be true. Consider any node x that had a link to w. If $\ell(w)\geq\ell(x)$ then x will have a link to both u and v. Otherwise, if $\ell(w)<\ell(u)$ then x will have a link to either u or v but not both. From the definition of f is is clear that $f(x,w)=f(x,u)+f(x,v)$ and hence $\sum_{\{y|(x\to y)\in G(t)\}}f(x,y)=\sum_{\{y|(x\to y)\in G(t+1)\}}f(x,y)=1$.

Similar analysis holds for merge operations. □

We call the node that receives a message with hop-count 0 the destination of the message.

The number of random walks is chosen to be more than $\rho\sqrt{n}$. Informally, if the random walks are rapidly mixing then an ϵ-intersecting quorum system is formed. Intuitively, the reason a random walk originating from a level k node needs to take k steps is that this ensures the resulting distribution is independent of the originating node. The reason the random walk is non-uniform is that the graph itself is imbalanced. A uniform random walk may create a higher load and a longer mixing time. Our non-uniform random walk ensures that the stationary distribution formed is relatively balanced up to a constant factor that is a function of C. We formally perform the analysis of the quorum selection in the next subsection.

4.3 Analysis of quorum selection

Let $G(t)$ be a dynamic graph on n nodes. Recall the probability distribution of message forwarding as defined in Section 4.2, Equation 1. We represent this distribution using a weighted adjacency $(n \times n)$-matrix $M(t)$ as follows:

$$m_{v,u} = \Pr[u \text{ forwards to } v] = \begin{cases} \frac{1}{2^{\max\{\ell(v)-\ell(u)+1,1\}}} & (u,v) \in E(t), \\ 0 & \text{otherwise.} \end{cases}$$

We first explicitly state the stationary distribution on the dynamic graph, and then prove that the weighed random walk algorithm makes a perfect sampling of this distribution.

We begin with a technical lemma. Intuitively, it states each node in $G(t)$ is pointed to by edges whose total weight is proportional to its own level. The precise sense in which this holds is stated by the lemma.

Lemma 4. *Let $v \in G(t)$ be a node whose id is $v.id = \langle a_1, a_2, \ldots, a_k \rangle$. Denote by $N_0(v)$ the nodes in $G(t)$ whose id's match $\langle 0, a_2, \ldots, a_k \rangle$, or are a prefix of it, or have a postfix added to it. (Similarly, denote by $N_1(v)$ the nodes that match $\langle 1, a_2, \ldots, a_k \rangle$, its prefix or postfix.) Then for any $i \in \{0,1\}$*

$$\sum_{u \in N_i(v)} m_{v,u} \frac{1}{2^{\ell(u)}} = \frac{1}{2^{\ell(v)+1}} \ .$$

Proof. By our graph construction, there are two cases to consider. The first one is $|N_0(v)| = 1$. In this case, denote $N_0(v) = \{w\}$, and it follows that $\ell(w) \leq \ell(v)$. We therefore have:

$$m_{v,w} \frac{1}{2^{\ell(w)}} = \frac{1}{2^{\ell(v)-\ell(w)+1}} \frac{1}{2^{\ell(w)}} = \frac{1}{2^{\ell(v)+1}} \ .$$

The second case is $|N_0(v)| > 1$. Then $\forall w \in N_0(v) : \ell(w) > \ell(v)$, and the nodes $w \in N_0(v)$ have the form $w.id = \langle 0, a_2, \ldots, a_k \rangle$, or are a prefix of it or have a postfix appended to it. By a trivial induction on split and merge operations, we have $\sum_{w \in N_0(v)} \frac{1}{2^{\ell(w)}} = \frac{1}{2^{\ell(v)}}$. Thus:

$$\sum_{w \in N_0(v)} m_{v,w} \frac{1}{2^{\ell(w)}} = \sum_{w \in N_0(v)} \frac{1}{2} \frac{1}{2^{\ell(w)}} = \frac{1}{2^{\ell(v)+1}} \ .$$

By symmetry, the analogous statement on $N_1(v)$ also holds. □

We now analyze the stationary distribution of $M(t)$.

Theorem 1. *The stationary distribution of $M(t)$ is the vector x, such that $\forall v \in V(t)$, $x_v = \frac{1}{2^{\ell(v)}}$*

Proof. Showing $\sum_{v \in V(t)} x_v = 1$ is trivially done by induction on the series of node additions and removals.

For $v \in V(t)$, we show that $\sum_{u \in V(t)} m_{v,u} x_u = x_v$. Suppose $v = \langle a_1, a_2, \ldots, a_k \rangle$. In our graph, nodes that have directed links to v are those in $N_0(v)$ and in $N_1(v)$. The sets $N_0(v)$ and $N_1(v)$ are disjoint by definition. Thus, using Lemma 4, we have:

$$\sum_{u \in V(t)} m_{v,u} x_u = \sum_{w \in N_0(v) \text{ or } w \in N_1(v)} m_{v,w} x_w = \frac{1}{2^{\ell(v)+1}} + \frac{1}{2^{\ell(v)+1}} = \frac{1}{2^{\ell(v)}} = x_v \ .$$

□

For every $t > 0$, denote $x(t)$ as the stationary distribution on $M(t)$, that is $x(t) = M(t)x(t)$. We now show that the random walk algorithm described in Section 4.2 chooses nodes according to $x(t)$. We begin by proving that the choices made in the random walk are independent and uniform.

Lemma 5. *All the bits of a random walk message with hop count 0 are independently uniformly distributed.*

Proof. Let v be a starting node whose level is k. We show by induction that after i hops, the walk reaches a node whose bits beyond the first $(k-i)$ bits are selected independently uniformly at random. Since the hop count starts with k, then when $i = k$ the hop count is 0 and the lemma follows.

Denote v's id by $v.id = \langle a_1, \ldots, a_k \rangle$. The first hop must go to a node whose id matches $\langle a_2, \ldots, a_k, r_1 \rangle$, or a prefix thereof, or with a postfix appended. and where r_1 is chosen to be 0/1 with uniform probability. In case the destination node has a postfix appended, the postfix bits are chosen uniformly at random by our construction, since every split operation divides the weight of an edge pointing to the split node into half. Thus, we have the induction basis.

For the induction step, suppose that after $i-1$ hops, the walk reaches a node $\langle a_i, \ldots, a_k, \sigma \rangle$, or a prefix thereof, where σ is a sequence of randomly chosen bits. Then at hop i we move to a node whose id matches $\langle a_{i+1}, \ldots, a_k, \sigma, r_i \rangle$, or a prefix thereof, or with a random postfix added to it, and where r_i is chosen to be 0 or 1 with uniform probability. By the same argument as above, in case the destination node has a postfix appended, the postfix bits are chosen uniformly at random.

Thus when the hop count reaches zero, the bits of the target node which equals the bits of the random walk message are all random, independent and uniformly selected. □

We also note the following simple combinatorial claim:

Proposition 1. *For $k \leq j$, given a fixed sequence of bits $A = \langle a_1, \ldots, a_k \rangle$, and a sequence $B = \langle b_1, \ldots, b_j \rangle$ of bits each independently chosen with uniform probability then*

$$Pr[A \text{ is a prefix of } B] = \frac{1}{2^k}$$

The number of hops until a random walk has distribution $x(t)$ is exactly the level of the node initiating the walk. This is the reason why a level k node performs k-hop random walks.

Theorem 2. *The mixing time of a random walk on $M(t)$ starting from a node of level k is k.*

Proof. A random walk message starting at a level k node will walk k steps until its hop count reaches 0. By Lemma 5 all its bits are independently uniformly chosen. Thus for $v \in V(t)$ the probability that the random walk reaches v by Fact 1 is $2^{-\ell(v)}$. □

Theorem 1 and Theorem 2 above together imply that our graph maintenance algorithm together with our random walk quorum selection strategy implement a non-uniform selection strategy over the members of $V(t)$, where the probability of choosing $v \in V(t)$ is $1/2^{\ell(v)}$. As an immediate consequence we have our main theorem as follows:

Theorem 3. *For a system S on a dynamic graph with global gap C and parameter $\rho = \sqrt{\ln \frac{1}{\varepsilon}}$, the quorum selection strategy as described above forms a ε-intersecting probabilistic quorum system.*

Proof. The theorem follows from the fact that each quorum access includes at least $\rho\sqrt{2^{\ell(u)+2C}} \geq \rho\sqrt{n}$ independent selections, each one done according to the distribution $x(t)$. □

5 Quorum Evolution

In this section we describe the evolution algorithm for maintaining *dynamic ε-intersecting quorum systems*. For such a construction, quorums need to evolve along with the growth of the system in order to maintain their intersection properties. This property must be maintained in spite of any execution sequence of join and leave events that may be given by an adversary.

One trivial solution would be to choose new quorums instead of the old ones each time the network's size multiplies. Such a solution has a major drawback, as it requires a global overhaul operation that may affect all the system at once. Even if we consider amortized costs, this process requires changing the state of $\sqrt{|V|}$ nodes for some join events. In contrast, our evolution scheme w.h.p. resorts only to local increments for each join or leave event, each causing only a constant number of nodes to change their state.

The intuition for our algorithm comes from the following simple case. Suppose the network is totally balanced, i.e., all nodes have the same level m, hence there are 2^m nodes. In this state a quorum using $\rho 2^{m/2}$ random walks is chosen. Hence the size of the quorum is $\rho\sqrt{n_1}$ where $n_1 = 2^m$. Further assume that after a series of join events, the network's size multiplies by 4 and all nodes have level $m+2$. Our evolution algorithm works as follows in this simple scenario. Each time a node splits, each data entry stored on the split node randomly chooses which sibling to move to. In addition, if the node that splits has an even level then each of its data entries (it may have several data entries for several different quorums) also creates one more duplicate data entry and randomly assigns it, using a random walk, to a new node. Thus the number of data entries doubles from $2^{m/2}$ to $2^{(m+2)/2}$ and each data entry is randomly distributed on the network. Hence the new quorum has $\rho\sqrt{n_2}$ random members where $n_2 = 2^{m+2}$.

Our evolution algorithm simulates this behavior on approximately balanced networks. Thus, its success relies on the fact that the global gap of the dynamic graph w.h.p. is at most C. In order to avoid fractions, we set the bound C on the global gap to be an even number.

5.1 Informal description of the evolution algorithm

Recall that a join (respectively, leave) event translates to a split (respectively, merge) operation on the dynamic graph. We now explain how the random walk algorithm is enhanced, and what actions are taken when a split or a merge operation occurs.

We divide the levels of the graph into phases of size C, all the levels $(i-1)C+1, \ldots, iC$ belong to phase iC. When a node in phase iC wants to establish a quorum, it sends $\rho 2^{(i+1)C/2}$ random walk messages. Each such message also contains the *phase* of the sender which is iC.

When two child nodes are merged into one parent node, all the data entries stored in the two children are copied to the parent node. If the parent node is later split, we want each data entry to go to the sibling it originally came from. Otherwise, the distribution of the data entry's location will be dependent on the execution sequence. Thus, each data entry also stores all the random choices it has made as a sequence of random bits called *dest*. When an entry is first created, *dest* is set to the id of the node that the data entry is in.

When a node of level i is split into two nodes of level $i+1$, there are two possibilities: Either $|dest| \geq i+1$ and the data entry moves according to the (i+1)th bit of *dest*. Otherwise, the data entry randomly chooses which one of the two sibling to move to, and it records this decision by adding the appropriate bit to *dest*.

The number of data entries is increased only when a data entry is split on a node whose level is a multiple of C. If a data entry with phase iC is involved in a split operation on a node with level iC then $2^{C/2} - 1$ new data entries with phase $(i+1)C$ are created. These data entries are randomly distributed using the random walk algorithm.

Additionally, whenever a random walk message from a node in phase jC arrives at a node u with phase $(j+1)C$, we simulate as if the message first arrived at an ancestor node of level jC that is a prefix of u, and later this ancestor node had undergone some split operations. Thus, if a phase $(j+1)C$ node receives a message with hop count 0 initiated by a node in phase jC then, in addition to storing the data entry, the node also creates $2^{2/C} - 1$ new data entries with phase $(j+1)C$. This simulation technique is recursively expanded to cases where a node in phase $(j+\ell)C$ receives a message initiated by a node in phase jC.

5.2 Evolution algorithm

Enhanced random walk: Denote $phase(i) = C\lceil i/C \rceil$. When a node u initiates a random walk it creates a message M with a hop-count $\ell(u)$, phase $phase(\ell(u))$, id $u.id$, and payload A to it, i.e., $M = \langle \ell(u), phase(\ell(u)), u.id, A \rangle$. Each node that receives a message $\langle j, ph, id, A \rangle$ with a non zero hop-count $j > 0$, forwards a message $M' = \langle j-1, ph, id, A \rangle$, randomly to one of its outgoing edges v with probability $\Pr[u \text{ forwards to } v] = \frac{1}{2^{\max\{\ell(v)-\ell(u)+1,1\}}}$.

Nodes store information as a *data entry* of the form $(dest, ph, id, A)$, where $dest$ is a sequence of bits that describes the location of the entry, ph is the phase, id is the identity of the quorum initiator, and A is the payload.

When node w receives a message $M = \langle 0, ph, id, A \rangle$ it stores the data entry $(w.id, ph, id, A)$. If $phase(\ell(w)) > ph$ then for every i such that $\lceil ph/C \rceil < i \leq \lceil \ell(w)/C \rceil$, w sends $2^{C/2} - 1$ messages of the form $\langle \ell(w), iC, id, A \rangle$.

Create: A node u creates a quorum by initiating $\rho 2^{(phase(\ell(u))+C)/2}$ enhanced random walk messages.

Split: Suppose node u wants to enter the system, and $v = \langle a_1, \ldots, a_k \rangle$ is the node to be split into nodes $\langle a_1, \ldots, a_k, 0 \rangle$ and $\langle a_1, \ldots, a_k, 1 \rangle$. For every data entry (d, ph, id, A) held in v do the following. If $|d| \geq k+1$ then store (d, ph, id, A) at node $\langle a_1, \ldots, a_k, dest_{k+1} \rangle$ where $dest_i$ is the ith bit of $dest$. Otherwise, if $|d| < k+1$ then with uniform probability choose $i \in \{0, 1\}$ and send to node $\langle a_1, \ldots, a_k, i \rangle$ the message $\langle 0, ph, id, A \rangle$. Node $\langle a_1, \ldots, a_k, i \rangle$ will handle this message using the enhanced random walk algorithm (in particular, if the split has crossed a phase boundary, it will generate $2^{C/2} - 1$ new data replicas).

Merge: Suppose node u wants to leave the system, and the twin nodes $\langle a_1, \ldots, a_k, 0 \rangle, \langle a_1, \ldots, a_k, 1 \rangle$ are the nodes that merge into node $v = \langle a_1, \ldots, a_k \rangle$. If u and one of the twins swap their ids then they also swap the data entries that they hold. After the swap, the merged node $v = \langle a_1, \ldots, a_k \rangle$ copies all the data entries that the nodes with ids $\langle a_1, \ldots, a_k, 0 \rangle, \langle a_1, \ldots, a_k, 1 \rangle$ held.

5.3 Analysis of quorum evolution

Given a network $G(t)$ on n nodes, we seek to show that the evolved quorum's distribution is at least as good as the flat access scheme $f(x(t), \rho\sqrt{n})$ where $x(t)$ is the stationary distribution $G(t)$ as proven in Theorem 1. So we must show a set of data entries that are independently distributed, whose size is at least $\rho\sqrt{n}$ where n is the size of the current system. Note that the existence of some of the data entries is dependent on the execution history. Therefore, it is not true that all data entries are independently distributed. However, we use a more delicate argument in which we analyze the size of a subset of the data entries whose existence is independent of the execution sequence.

The main result we pursue is that a non-uniform PQS is maintained despite any system reconfiguration, and is given in the Theorem below. The following two lemmas are crucial for proving it.

Lemma 6. *For any time t, data entry D, the distribution of D's location on $V(t)$ is $x(t)$.*

Proof. For every data item $D = (dest, ph, id, A)$ we prove the following by induction: $dest$ is a sequence of bits that are independently and uniformly distributed and D is stored in the node whose id is a prefix of $dest$. Since the identifiers form a complete prefix code and $dest$ will always be at least as long as any identifier with the same prefix, this claim is well defined.

When a data entry is created it is stored at a node v chosen by the random walk algorithm and $dest$ is set to v. By Lemma 5 all the bits of v are independently uniformly distributed. Thus the induction base holds.

Now assume at time t that entry D is stored in node $v = \langle a_1, \ldots, a_k \rangle$. Suppose the next event $e(t+1)$ is a leave that causes a merge operation on v and its twin. This will cause D to be stored in $\langle a_1, \ldots, a_{k-1} \rangle$ which still remains a prefix of $dest$.

Suppose $e(t+1)$ is a join that causes a split operation on v. If $|dest| \geq k+1$ then D moves to $\langle a_1, \ldots, a_k, dest_{k+1} \rangle$ and the claim holds. If $dest$ has only k bits then the evolution algorithm independently with uniform probability chooses which sibling to move to and thus the new destination maintains the induction hypothesis.

Therefore, the $dest$ sequence is i.i.d. and the data entry resides in a node which is a prefix of $dest$. Thus by Fact 1 and Theorem 1, the location of D at time t has a distribution $x(t)$ □

Definition 10. *Denote $L(t)$ as the lowest phase on $G(t)$, $L(t) = \min_{v \in V(t)} phase(\ell(v))$.*

Lemma 7. *Let $t > 0$, and let the dynamic graph $G(t)$ have global gap C. Consider any quorum initiated by a node u at phase i with payload A. If $L(t) \geq i$ then the number of data entries of the form (d, ph, u, A) such that $ph \leq L(t)$ is exactly $\rho 2^{((L(t)+C)/2}$.*

Proof. The proof is by induction. When a quorum is established by a node of phase i, it creates $\rho 2^{(i+C)/2}$ data entries with phase i. Since the gap is at most C, i is either $L(t)$ or $L(t) + C$. If $i = L(t)$ then the induction base holds. If $i = L(t)+C$ then the base holds vacuously, since $L(t) < i$.

If $e(t+1)$ is a join that causes $L(t+1) = L(t)+C$ then all of the data entries with phase at most $L(t)$ must have been spilt on a node of level $L(t)$ (or split on a lower level node that simulates this split). If $t+1$ is the first time that $L(t+1) \geq phase(id)$, and the induction hypothesis was vacuously true before $t+1$, then the phase of the quorum establisher was $L(t)+C$, thus $\rho 2^{((L(t)+2C)/2}$ data entries of phase $l(t)+C$ exist in the network as required.

Otherwise, by the induction hypothesis there are $\rho 2^{(L(t)+C)/2}$ data entries of phase at most $L(t)$. Each one creates

$2^{C/2}-1$ more data entries of phase $L(t)+C$. Together, there are $\rho 2^{((L(t)+2C)/2}$ entries of phase at most $L(t)+C$. □

Theorem 4. *On dynamic networks with global gap C, and parameter $\rho = \sqrt{\frac{1}{\varepsilon}}$, the evolution algorithm maintains a dynamic ε-intersecting quorum system.*

Proof. By Lemma 6 the locations of all data entries of all quorums are distributed by $x(t)$, the stationary distribution of $M(t)$. Consider a quorum initiated at a phase i node. If $L(t) < i$ then the initial $\rho 2^{(i+C)/2} \geq \rho\sqrt{|V(t)|}$ data entries suffice. If $L(t) \geq i$ then by Lemma 7 every quorum has $\rho 2^{(L(t)+C)/2}$ entries whose existence is independent of the execution history. Since the network has global gap of C, then $\rho 2^{(L(t)+C)/2} \geq \rho\sqrt{|V(t)|}$. Thus at any time t, the evolving probability distribution p_t^r of the above subset of data entries of any quorum, for any establishment time r, is a flat access strategy $f(x(t), m)$ in which $m \geq \rho\sqrt{|V(t)|}$. By Lemma 1 this access scheme forms an ε-intersecting quorum system as required. □

Our construction implements, for any history of events, access strategies and an evolution strategy that maintains the evolving probability distribution p_t^r as a flat access strategy on $V(t)$ using the distribution $x(t)$ with more than $\sqrt{|V(t)|}$ independent choices. Thus, at any time t, all quorums (both newly established and evolved) are ε-intersecting.

6 Performance Analysis

In this section, we state more formally the four performance measures alluded to in Section 2.1, and analyze them with respect to our dynamic quorum system. Generally, we note that our protocols hinge on the network balancing algorithms we employ, e.g., from [1], and on their ability to maintain the bound C on the global level gap. We note that the network construction of [1] incurs a constant number of state-changes per join/leave and a logarithmic number of messages. It maintains the global gap bound C w.h.p. Below, the analysis stipulates that the global gap C is maintained, and calculates additional costs incurred by our algorithm.

Join/leave complexity. The *complexity of handling join/leave* events is measured in terms of the count of messages and the number of processes that must incur a state-change. In our dynamic quorums, when a new process joins the system, it may cause a split of a node. In that case, we allocate a constant number of new data entries, that incur a constant number of random walks. Thus, the message cost is $O(\log(n))$ and the number of processes incurring a change in their state is constant. Leave events generate one message and a state change to one process.

Quorum access complexity. Our second measure is the *complexity of accessing a quorum*, measured both in messages and in (parallel) time. When selecting a quorum in our system, we initiate $O(\sqrt{n})$ random walks in parallel. The parallel time is $O(\log(n))$, and the total number of messages is $O(\sqrt{n}\log(n))$.

Load The load of a quorum system, defined by Naor and Wool [32], captures the probability of accessing the busiest server. Load is a measure of efficiency. All other things being equal, systems with lower load can process more requests than those with higher load.

Definition 11 (Load). *Let w be a strategy for a set system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ over a universe U. For a server $u \in U$, the* load *induced by w on u is $l_w(u) = \sum_{Q_i \ni u} w(Q_i)$. The* load *induced by a strategy w on $\mathcal{Q}$ is $L_w(\mathcal{Q}) = \max_{u \in U}\{l_w(u)\}$.*

Let $\langle \mathcal{Q}, w\rangle$ be an ε-intersecting quorum system. Then the load *of $\langle \mathcal{Q}, w\rangle$ is $L(\langle \mathcal{Q}, w\rangle) = L_w(\mathcal{Q})$.*

It is known that for any quorum system $\mathcal{Q}$ over n servers, $L(\mathcal{Q}) \geq \max\{\frac{1}{c(\mathcal{Q})}, \frac{c(\mathcal{Q})}{n}\}$ where $c(\mathcal{Q})$ is the size of the smallest quorum in $\mathcal{Q}$ [32]. In particular, this implies that for any quorum system $\mathcal{Q}$, $L(\mathcal{Q}) \geq 1/\sqrt{n}$.

In our dynamic system, the load on a process v during quorum selection at time t is $O(1/\sqrt{n})$. In order to see this, observe that Theorem 1 and Theorem 2 above together imply that our graph maintenance algorithm together with our random walk quorum selection strategy implement a non-uniform selection strategy over the members of $V(t)$, where the probability of choosing $v \in V(t)$ is $1/2^{\ell(v)}$. By Lemma 2, at most $\rho 2^{(\log(n)+c+2c)/2}$ processes are accessed in every quorum access. The probability that some node v is selected is therefore bounded as follows:

$$1-\left(1-\frac{1}{2^{\ell(v)}}\right)^{\rho 2^{(3/2)C}\sqrt{n}} \leq$$

$$1-\left(1-\frac{1}{2^{\log(n)-C}}\right)^{\rho 2^{2C}\sqrt{n}} \leq \frac{\rho 2^{2C}\sqrt{n}}{2^{\log(n)-C}} = \frac{\rho 2^{3C}}{\sqrt{n}} .$$

As the system grows, the load above continues to hold. If the system dramatically diminishes, the relative fraction of data entries could grow, causing high load on processes. Naturally, a practical system must deploy garbage collection mechanisms in order to preserve resources. The discussion of garbage collection is left outside the scope of this paper.

Fault tolerance The fault tolerance and failure probability measures for PQSs defined in [29] measure the resilience of the system to failures. There is some technical complexity in defining resilience of PQSs, which can intuitively be explained as follows. Suppose that we have a certain PQS over an n-member universe. Suppose that we add to this PQS "quorums" of all individual

members, each "quorum" containing one member. These quorums are non-intesecting, but we do not care: We select these quorums with zero probability! This strange construction is fine in the context of PQS. However, for resilience measurement, we must take care **not** to count the individual-quorums as being available, because this would artificially inflate the apparent resilience.

The formal definitions below take care of this by relating resilience to the access probability of quorums. The key notion with which fault tolereance is defined is that of *high quality* quorum, which are selected with non-marginal probabilities.

Definition 12 (δ-high quality quorums). *Let $\langle \mathcal{Q}, w\rangle$ be an ε-intersecting quorum system, and let $0 \leq \delta \leq 1$ be given. The set of* δ-high quality quorums *of $\langle \mathcal{Q}, w\rangle$ is*

$$\mathcal{R} = \{Q \in \mathcal{Q} : \Pr[Q \cap Q' \neq \emptyset] \geq 1 - \delta\},$$

where $Q' \in \mathcal{Q}$ is chosen according to w.

Definition 13 (High quality quorums). *Let $\langle \mathcal{Q}, w\rangle$ be an ε-intersecting quorum system. Then the* high quality quorums *of $\langle \mathcal{Q}, w\rangle$ are the $\sqrt{\varepsilon}$-high quality quorums of $\langle \mathcal{Q}, w\rangle$.*

Definition 14 (Fault tolerance). *Let $\langle \mathcal{Q}, w\rangle$ be an ε-intersecting quorum system. Let $\mathcal{R}$ be the set of high quality quorums of $\langle \mathcal{Q}, w\rangle$, and let $\mathcal{S} = \{S : S \cap Q \neq \emptyset$ for all $Q \in \mathcal{R}\}$. Then the* fault tolerance $A(\langle \mathcal{Q}, w\rangle)$ *is* $\min_{S\in\mathcal{S}} |S|$.

Definition 15 (Failure probability). *Let $\langle \mathcal{Q}, w\rangle$ be an ε-intersecting quorum system, and let $\mathcal{R}$ be the set of high quality quorums of $\langle \mathcal{Q}, w\rangle$. The* failure probability $F_p(\langle \mathcal{Q}, w\rangle)$ *is the probability that every $Q \in \mathcal{R}$ contains at least one crashed server, under the assumption that each server in U crashes independently with probability p.*

For the availability analysis, note that all quorums of size $\rho 2^C\sqrt{n}$ are high quality. Because only $\rho 2^C\sqrt{n}$ processes need be available in order for some (high quality) quorum to be available, the fault tolerance is $n - \rho 2^C\sqrt{n} + 1 = \Omega(n)$.

The failure probability is given by $F_p = e^{-\Omega(n)}$. Let p denote the independent failure probability of processes. In order that the system fail, at least $n - \rho 2^C\sqrt{n} + 1$ processes must fail. By a standard analysis of threshold-resilience using a Chernoff bound (see [32]), the failure probability can be bounded by the following:

$$F_p = \Pr(\#\text{fail} > n - \rho 2^C\sqrt{n}) \leq e^{-2n\left(\frac{\rho 2^C}{\sqrt{n}}+\delta\right)^2} = e^{-\Omega(n)},$$

for all $p \leq 1 - 2\frac{\rho 2^C}{\sqrt{n}} - \delta$. This probability decreases rapidly to zero for values of p approaching 1 (according to the selection of δ), which is the best we can expect. Furthermore, this failure probability is optimal [32].

7 Discussion

In this paper we assumed the read-write ratio to be roughly equal. It is possible to extend the techniques of this paper to differentiate between read-quorums and write-quorums, and achieve better performance. Given any read-write ratio, instead of having all operations select $cn^{1/2}$ nodes, read operations select cn^{α} nodes, and write operations select $cn^{1-\alpha}$ nodes for some predetermined $0 < \alpha < 1$.

We presented a system with a constant $1-\varepsilon$ intersection probability for a fixed constant ε. In the AntWalk system [33], $O(\sqrt{n \log n})$ processes are randomly chosen thus leading to intersection with probability $1 - n^c$. Our quorum selection and evolution algorithm can be modified along similar lines to achieve a $1 - n^c$ dynamic intersecting quorum system.

Our analysis is sketched in a model in which changes are sequential. While we believe our construction to be efficient in much stronger settings, where a large number of changes may occur simultaneously, it is currently an open problem to provide a rigorous analysis.

The fault tolerance analysis concerns the robustness of the data which the system stores against $O(n)$ failures. While the data will not be lost due to such catastrophic failure, clearly our constant degree network, which is used to access the data, may disconnect. Network partitioning can be reduced by robustifying the network through link replication. But unless each node has $O(n)$ links, $O(n)$ failures will disconnect any network. Once the network is partitioned, the problem of rediscovering the network's nodes is addressed in [17,2]. When the network is reconnected, the dynamic de-Bruijn can be reconstructed. After recovering from this catastrophic failure, the system will maintain consistency, since the information itself was not lost.

References

1. I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, D. Malkhi, and E. Pavlov. A generic scheme for building overlay networks in adversarial scenarios. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2003)*, 2003.
2. I. Abraham and D. Dolev. Asynchronous resource discovery. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, 2003.
3. I Abraham and D. Malkhi. Probabilistic quorums for dynamic systems. In *Proceedings of the 17th International Symposium on DIStributed Computing (DISC 2003)*, October 2003.
4. Sangeeta Bhattacharya. Randomized location service in mobile ad hoc networks. In *MSWIM '03: Proceedings of the 6th ACM international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 66–73. ACM Press, 2003.
5. K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

6. M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *SOSP'03*, October 2003.
7. G. Chockler, D. Malkhi, B. Merimovich, and D. Rabinowitz. Aquarius: A data-centric approach to corba fault-tolerance. In *The workshop on Reliable and Secure Middleware, in Proceedings of the 2003 International Conference on Distributed Objects and Applications (DOA)*, 2003.
8. N. G. de Bruijn. A combinatorial problem. *Konink. Nederl. Akad. Wetersh. Verh. Afd. Natuurk. Eerste Reelss*, (A49):758–764, 1946.
9. S. Dolev, S. Gilbert, N. Lynch, A. Shvartsman, and J. Welch. Geoquorums: Implementing atomic memory in mobile ad hoc networks. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC 2003)*.
10. W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley & Sons, 1967. 3rd edition.
11. P. Fraigniaud and P. Gauron. The content-addressable network D2B. Technical Report 1349, LRI, Univ. Paris-Sud, France, 2003.
12. H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, 1985.
13. D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles*, pages 150–162, 1979.
14. J. L. Welch H. Lee and N. H. Vaidya. Location tracking with quorums in mobile ad hoc networks. *Ad Hoc Networks*, 1(4):371–381, 2003.
15. Z. J. Haas and B. Liang. Ad hoc mobility management with randomized database groups. In *Proceedings of the IEEE Internation Conference on Communications*, 1999.
16. Z. J. Haas and B. Liang. Ad hoc mobility management with uniform quorum systems. *IEEE/ACM Transactions on Networking*, 7(2):228–240, 1999.
17. M. Harchol-Balter, T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, pages 229–237, 1999.
18. M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica. Complex queries in DHT-based peer-to-peer networks. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.
19. W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
20. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal hash table. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, 2003.
21. A.-M. Kermarrec, L. Massouli, and A.J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3), March 2003.
22. H. Lee and J. L. Welch. Applications of probabilistic quorums to iterative algorithms. In *Proceedings of 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pages 21–28, 2001.
23. H. Lee and J. L. Welch. Randomized shared queues. In *Brief announcement in Twentieth ACM Symposium on Principles of Distributed Computing (PODC 2001)*, 2001.
24. A-M. Kermarrec M. Castro, P. Druschel and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.
25. M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, 1985.
26. D. Malkhi. Quorum systems, 1999. Invited chapter in the Encyclopedia of Distributed Computing, Joseph Urban and Partha Dasgupta, editors, Kluwer Academic Publishers, available from `http://research.microsoft.com/~dalia/pubs/quorums.ps`.
27. D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.
28. D. Malkhi and M. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):187–202, 2000.
29. D. Malkhi, M. Reiter, A. Wool, and R. Wright. Probabilistic quorum systems. *Information and Computation*, 170(2):184–206, 2001.
30. M. Naor and U. Wieder. Novel architectures for p2p applications: the continuous-discrete approach. In *The Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '03)*, 2003.
31. M. Naor and U. Wieder. Scalable and dynamic quorum systems. In *proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC 2003)*, 2003.
32. M. Naor and A. Wool. The load, capacity and availability of quorum systems. *SIAM Journal of Computing*, 27(2):423–447, April 1998.
33. D. Ratajczak. Decentralized dynamic networks. Technical Report M. Eng. Thesis Proposal, MIT, May 2000.
34. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
35. 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 the SIGCOMM 2001*, 2001.
36. I. Stojmenovic. A scalable quorum based location update scheme for routing in ad hoc wireless networks. Technical Report TR-99-09, SITE, University of Ottawa, 1999.
37. J. Tchakarov and N. Vaidya. Efficient content location in wireless ad hoc networks. In *IEEE International Conference on Mobile Data Management (MDM)*, 2004.
38. R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, 1979.
39. B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.
40. S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, 2001.