

Minimizing Response Time for Quorum-System Protocols over Wide-Area Networks

Florian Oprea*

Michael K. Reiter†

Abstract

A quorum system is a collection of sets (quorums) of servers, where any two quorums intersect. Quorum-based protocols underly modern edge-computing architectures and throughput-scalable service implementations. In this paper we propose new algorithms for placing quorums in wide-area networks and tuning which quorums clients access, so as to optimize clients' average response time in quorum-based protocols. We examine scenarios in which the service is lightly loaded and hence network latency is the dominant delay, and in which client-induced load contributes significantly to the delay that clients observe. In each case, we evaluate our algorithms on topologies ranging from 50 to over 150 wide-area locations.

1. Introduction

A quorum system is a collection of sets (called *quorums*) such that any two intersect. Quorum systems are a standard tool to achieve efficient and fault-tolerant coordination in a distributed system. At a high level, the intersection property of quorums ensures that an update performed at one quorum of servers will be visible to any access subsequently performed at another quorum. At the same time, the fact that accesses need not be performed at all servers can lead to significant improvements in terms of system throughput and availability (e.g., [21]). For these reasons, they are a key ingredient in a range of practical fault-tolerant system implementations (e.g., [1, 8, 20]).

In this paper we consider the use of quorum-based protocols in a wide-area network. Our interest in the

wide-area setting arises from two previous lines of research: First, quorums have been employed as an ingredient of *edge computing* systems (e.g., [10]) that support the deployment of dynamic services across a potentially global collection of proxies. That is, these techniques strive to adapt the efficiencies of content distribution networks (CDNs) like Akamai to more dynamic services, and in doing so they employ accesses to quorums of proxies in order to coordinate activities. Second, there has recently been significant theoretical progress on algorithms for deploying quorums in physical topologies, so that certain network measures are optimized or approximately optimized (e.g., [9, 12, 14, 15, 19, 27]). These results have paved the way for empirical studies using them, which is what we seek to initiate here.

More specifically, in this paper we perform an evaluation of techniques for placing quorums in wide-area networks, and for adapting client¹ strategies in choosing which quorums to access. In doing so, we shed light on a number of issues relevant to deploying a service “on the edge” of the Internet in order to minimize service response times as measured by clients, such as (i) the number and location of proxies that should be involved in the service implementation, and (ii) the manner in which quorums should be accessed. A central tension that we explore is that between accessing “close” quorums to minimize network delays and dispersing service demand across (possibly more distant) quorums to minimize service processing delays. Similarly, as we will show, quorums over a small universe of servers is better to minimize network delays, but worse for dispersing service demand. Finding the right balances on these spectra is key to minimizing overall response times of a edge-deployed service.

We conduct our analyses through both experiments with a real protocol implementation [1] in an emulated wide-area network environment [28] and simulation of

This work was partially supported by U.S. National Science Foundation award CCF-0424422.

*Electrical & Computer Engineering Department, Carnegie Mellon University, Pittsburgh, PA, USA; foprea@cmu.edu

†Electrical & Computer Engineering and Computer Science Departments, Carnegie Mellon University, Pittsburgh, PA, USA; reiter@cmu.edu

¹While we refer to entities that access quorums as “clients”, they need not be user end systems. Rather, in an edge computing system, the clients could be other proxies, for example.

a generic quorum system protocol over models of several actual wide-area network topologies. The topologies are created from PlanetLab [5] measurements and from measurements between web servers [23]. Their sizes range from 50 to 161 wide-area locations, making this, to our knowledge, the widest range of topologies considered to date for quorum placement. That said, as the initial study at this scale, ours is limited in considering only “normal” conditions, i.e., that there are no failures of network nodes or links, and that delays between pairs of nodes are stable over long enough periods of time and known beforehand. We hope to relax these assumptions in future studies.

2. Related work

The earliest study of which we are aware of quorum behavior in wide-area networks is due to Amir and Wool [3]. Their analysis studied the availability of quorums deployed across three wide-area locations, with a focus on how the behavior of the wide-area network violated typical assumptions underlying theoretical availability analyses of quorum systems. Their focus on availability is complementary to ours on response time, and was conducted on much smaller topologies than what we consider.

Bakr and Keidar [4] conducted a wide-area study of a *communication round* among nodes, in which each node sends information to each other participating node. Their study also focused on delay, though otherwise their study and ours are complementary. Our study considers only direct client-to-quorum communication; their treatment of four different round protocols is more exhaustive. However, their study is confined to one node at each of ten wide-area sites—they do not consider altering the number or locations of nodes—and does not admit load dispersion (since all nodes participate in all exchanges). In contrast, our study shows the impact of scaling the number of servers in a wide-area quorum system; of the judicious placement of servers among the candidate sites; and of tuning client access strategies to disperse load.

Amir et al. [2] construct and evaluate a Byzantine fault-tolerant service with the goal of improving the performance of such a service over a wide-area network. In this context, they evaluate BFT [6] and their alternative protocol, which executes Paxos [17] over the wide area. These protocols can be viewed as employing Majority [11, 26] quorum systems in which a quorum constitutes greater than two-thirds or one-half of the wide-area sites, respectively; their evaluation overlaps ours in that we also evaluate these quorum systems (and others). As in the Bakr and Keidar evaluation,

however, Amir et al. constrain their evaluation to a fixed number of wide-area sites (in their case, five), and do not consider altering the number or locations of nodes. Consequently, our evaluation is complementary to that of Amir et al. in the same ways.

Oppenheimer et al. [22] examined the problem of mapping large-scale services to available node resources in a wide-area network. Through measurements of several services running on Planetlab [5], they extracted application resource demand patterns and CPU and network resource availability for Planetlab nodes over a 6-month period. From these measurements, they concluded that several applications would benefit from a service providing resource-informed placement of applications to nodes. Among other interesting results, their study reveals that inter-node latency is fairly stable and is a good predictor of available bandwidth, a conclusion that supports our focus on periods in which latencies between nodes are stable.

3. A motivating example

To motivate our study (and perhaps as one of its contributions), in this section we describe an evaluation of the Q/U protocol [1]. Q/U is a Byzantine fault-tolerant service protocol in which clients perform operations by accessing a quorum of servers. The goal of our evaluation is to shed light on the factors that influence Q/U client response time when executed over the wide area, leading to our efforts in subsequent sections to minimize the impacts of those factors.

We perform our evaluation on Modelnet [28], an emulated wide area environment, using a network topology developed from PlanetLab measurements. We deployed Modelnet on a rack of 76 Intel Pentium 4 2.80 GHz computers, each with 1 GB of memory and an Intel PRO/1000 NIC. We derived our topology from network delays measured between 50 PlanetLab sites around the world in the period July–November 2006 [24].

We varied two parameters in our experiments: the first was the number n of Q/U servers, where $n \geq 5t+1$ is required to tolerate t Byzantine failures. We ran Q/U with $t \in \{1, \dots, 5\}$, $n = 5t + 1$ and a quorum size of $4t+1$. The second parameter we varied was the number of clients issuing requests to the Q/U servers. We chose the location of clients and servers in the topology as follows: we placed each server at a distinct node, using a known algorithm (recounted in Section 4.1) that approximately minimizes the average network delay that each client experiences when accessing a quorum uniformly at random. For each such placement, we computed a set of 10 client locations for which the average

network delay to the server placement approximates the average network delay from all the nodes of the graph to the server placement well. On each of these client locations we ran c clients, with $c \in \{1, \dots, 10\}$.

Clients issued only requests that completed in a single round trip to a quorum. While not all Q/U operations are guaranteed to complete in a single round trip, they should in the common case for most services [1]. For each request, clients chose the quorum to access uniformly at random, thereby balancing client demand across servers. The application processing delay per client request at each server was 1 ms.

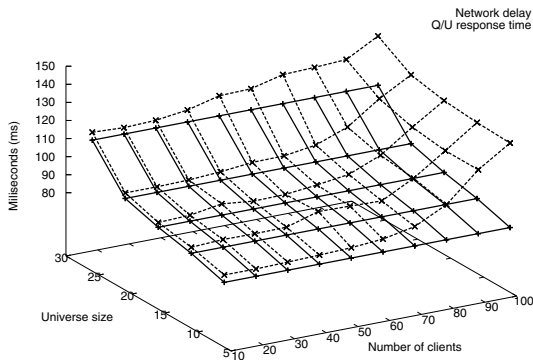


Figure 3.1. Average response time, network delay for Q/U on Planetlab topology

We compared two measures in our experiments: the average response time over all the clients and the average network delay over all the clients (both in milliseconds). Average response time was computed by running each experiment 5 times and then taking the mean. The variation observed was under 1 ms for up to 50 clients, and then increased with the client demand above that threshold.

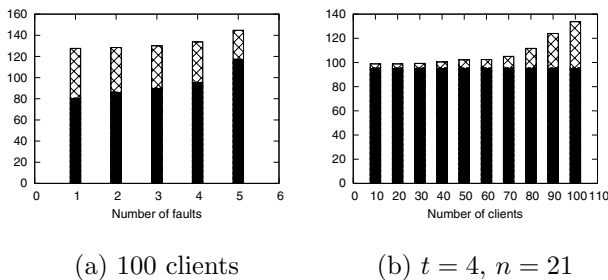


Figure 3.2. Avg network delay (black bars) & response time (total bars) for Q/U (ms)

Figure 3.1 shows how the two measures changed when we varied the universe size n and the number of clients issuing requests. As expected, increasing the client demand led to higher processing delay and hence higher average response time. Increasing the universe size had a similar effect on response time, but for a different reason: the average network delay increased since quorums tended to be spread apart more. This can be more easily seen in Figure 3.2a, where we keep the client demand constant and increase t and hence the universe size. However, increasing the universe size better distributed processing costs across more servers, and so decreased processing delay slightly.

In Figure 3.2b we can see how putting more demand on the system increased the average response time. For up to at most 50 clients, the major component of the average client response time was network delay. Increasing client demand beyond that point made processing delay play a more important role in average client response time (although network delay still represented a significant fraction of the overall response time). If request processing involved significant server resources, this effect would be even more pronounced.

To summarize, the Q/U experiments show that on a wide-area network, average response time depends on the average network delay and on the processing delays on servers. Increasing universe size tends to increase network delay but decrease per-server processing delay when demand is high. Thus to improve the overall performance of a protocol that uses quorum systems, we need algorithms that optimize either just the network delay (for systems where client demand is expected to be low) or a combination of network delay and processing delay (if client demand is expected to be high). The rest of this paper is devoted to balancing these tradeoffs by modifying how servers are placed in the network, and how clients access them.

4. Algorithms

To experiment with quorum placement in wide area topologies, we have implemented several known algorithms and developed others of potentially independent interest. Here we describe those algorithms. To do so, we introduce a number of concepts first.

Network We model the network as an undirected graph $G = (V, E)$, with each node having an associated capacity $\text{cap}(v) \in \mathbb{R}^+$; a node's capacity is a measure of its processing capability. There is a positive "length" $\text{length}(e)$ for each edge $e \in E$, which induces a distance function $d : V \times V \rightarrow \mathbb{R}^+$ obtained by setting $d(v, v')$ to be length of the shortest path between v and v' . We

assume that the set of clients that access the quorum system is V .

Quorum placement Given a quorum system $\mathcal{Q} = \{Q_1, \dots, Q_m\}$ over a universe U of logical elements, we define a *quorum placement* f as an arbitrary mapping $f : U \rightarrow V$. A placement specifies the node $f(u) \in V$ that hosts universe element u . Similarly, for $Q \in \mathcal{Q}$, we define $f(Q) = \{f(u) : u \in Q\}$. We call $f(U)$ the *support set* of the placement f (i.e., the nodes of the graph that actually host a universe element).

Load Given a quorum system \mathcal{Q} over a universe U , with $|U| = n$, and a client $v \in V$, an *access strategy* p_v of client v is a distribution on the quorums of \mathcal{Q} (i.e., $\sum_{Q \in \mathcal{Q}} p_v(Q) = 1$). Intuitively, in order to perform an operation, a client v samples a quorum according to the distribution p_v and then performs the operation at those servers. Consequently, an access strategy of client $v \in V$ induces a load on each element $u \in U$, given by $\text{load}_v(u) = \sum_{Q \in \mathcal{Q}: u \in Q} p_v(Q)$. For each client $v \in V$ and node $w \in V$, a placement f induces a load on w , namely $\text{load}_{v,f}(w) = \sum_{u \in U: f(u)=w} \text{load}_v(u)$ (i.e., equal to the sum of load of universe elements assigned to w by f). Finally, we define the load induced by f on node w as $\text{load}_f(w) = \text{avg}_{v \in V} \text{load}_{v,f}(w)$.

Response time We will attempt to deploy quorums in the wide-area to minimize response time, and to do so we model response time assuming that a client v issuing a request to a server node has to wait for an amount of time equal to the total roundtrip delay between itself and the server plus a time proportional to the number of requests the server has to process before servicing v 's request. In protocols that use quorum systems, a client completes a request only after receiving replies from a full quorum of servers. Thus we model the response time for a client request as:

$$\rho_f(v, Q) = \max_{w \in f(Q)} (d(v, w) + \alpha * \text{load}_f(w)) \quad (4.1)$$

Manipulating α allows us to adjust for absolute demand from clients and processing cost per request.

We model the *expected response time* (under p_v) for client v to access quorum \mathcal{Q} as:

$$\Delta_f(v) = \sum_{Q \in \mathcal{Q}} p_v(Q) \rho_f(v, Q). \quad (4.2)$$

The objective function we seek to minimize in our algorithms is the average response time over all clients: $\text{avg}_{v \in V} [\Delta_f(v)]$. If we set $\alpha = 0$, the response time for a client request becomes $\delta_f(v, Q) = \max_{w \in f(Q)} d(v, w)$. This transforms the objective function into average network delay, a measure studied previously [9, 14, 15, 19]. We stress that these definitions are merely tools for placing quorum systems; our interest is in seeing how well we can use these models to

efficiently deploy quorum systems in realistic wide-area topologies.

4.1. Previously introduced algorithms

In this section we briefly describe previously proposed algorithms for finding quorum placements that minimize network delay. In doing so, we make a distinction between one-to-one and many-to-one placements. Each of these two categories of placements potentially has advantages over the other: for instance, many-to-one placements can decrease network delay by putting more logical elements on a single physical node. One-to-one quorum placements, on the other hand, are important when we want to preserve the fault-tolerance of the original quorum system.

4.1.1 One-to-one quorum placements

For two known quorum constructions (Majority [11, 26] and Grid [7, 16]), Gupta et al. [14] propose optimal one-to-one placements assuming a single client $v_0 \in V$ issues requests using a uniform access strategy. They also show that to obtain a one-to-one placement without this assumption that is within a small constant factor of optimal, we can run the single-client placement algorithm using each node v as v_0 , compute the average network delay from all clients for each such placement, and pick the placement that has the smallest average delay.

Gupta et al. [14] show that, for Majorities, every one-to-one placement to a fixed set of nodes in V has the same average delay for a single client. Thus, in our evaluation we will pick an arbitrary one-to-one mapping f from the universe U , $|U| = n$, to the ball $B(v_0, n)$, i.e., the set of n nodes closest to v_0 (including v_0) such that each node v has capacity $\text{cap}(v) \geq \text{load}_f(u)$ for any $u \in U$. Recall that because p_{v_0} (and p_v for any $v \in V$) is the uniform access strategy, $\text{load}_f(u)$ is a constant independent of $u \in U$.

For the Grid quorum system the following algorithm is optimal for a single client $v_0 \in V$. For the sake of simplicity we describe the algorithm for finding the inverse of the optimal placement (i.e., that puts network nodes on the cells of a $n = k \times k$ grid). Let $d_1 \geq d_2 \geq \dots \geq d_n$ be the distances from the nodes in $B(v_0, n)$ to v_0 sorted in decreasing order. The algorithm places the largest ℓ^2 distances—or rather, the nodes with those distances—on the top-left $\ell \times \ell$ square of the grid. The next ℓ distances $d_{\ell^2+1}, d_{\ell^2+2}, \dots, d_{\ell^2+\ell}$ are placed on $(1, \ell+1), (2, \ell+1), \dots, (\ell, \ell+1)$, and the following ℓ distances $d_{\ell^2+\ell+1}, \dots, d_{\ell^2+2\ell+1}$ are placed on $(\ell+1, 1), \dots, (\ell+1, \ell+1)$. This completes the top

$(\ell + 1) \times (\ell + 1)$ square of the grid, and the construction proceeds inductively.

4.1.2 Many-to-one quorum placements

In this section we discuss two algorithms that result in many-to-one placements.

An almost-capacity-respecting placement The algorithm for finding many-to-one placements [14] has the same structure as the algorithm for one-to-one placements: it uses as building block an algorithm for a single client v_0 . To find the best placement we simply consider all possible nodes for v_0 . The difference from the case of one-to-one placements consists in the techniques used to place quorums for access by v_0 : the algorithm uses a linear programming (LP) formulation of the problem. At a high level, the algorithm is the following: We first solve the LP formulation to obtain a fractional placement. Then we use Lin and Vitter's filtering and rounding procedure [18] to obtain another fractional solution that does not (fractionally) assign a universe element to a node too far away from the single client v_0 . Finally, from the fractional solution we construct a *generalized assignment problem* (GAP) problem instance [25] and solve it to obtain the many-to-one quorum placement. An advantage of this algorithm is that it works for arbitrary quorum systems and an arbitrary access strategy p_{v_0} (where p_{v_0} is the same access strategy for all clients). However, the algorithm also allows for the node capacity to be exceeded by a small constant factor.

Singleton placement A special many-to-one quorum placement is the *singleton* quorum placement: this puts all the elements of U on a single network node (regardless of that node's capacity). The node on which we place all the universe elements is the node that minimizes the sum of the distances from all the clients to itself. When all nodes of the graph are clients, this node is also known as *the median* of the graph. Lin [19] showed that the singleton is a 2-approximation for the problem of designing a quorum system over a network to minimize average network delay.

4.2. New techniques

Optimizing access strategies Our first new technique is an algorithm that, given a placement, finds client access strategies that minimize network delay under a set of node capacity constraints. The algorithm allows one to improve network delay while preserving per-server load, something that will be useful when we want to minimize response time.

The algorithm is based on a LP with variables $p_{vi} \geq 0$, where p_{vi} specifies the probability of access

of quorum Q_i by client v . We assume a quorum placement $f : U \rightarrow V$ and a capacity $\text{cap}(v)$ for each $v \in V$ are given. A LP formulation of the problem is:

$$\text{minimize } \text{avg}_{v \in V} \sum_{i=1}^m p_{vi} \delta_f(v, Q_i) \quad (4.3)$$

$$\text{s.t. } \text{avg}_{v \in V} \text{load}_{v,f}(v_j) \leq \text{cap}(v_j) \quad \forall v_j \in V \quad (4.4)$$

$$\sum_{i=1}^m p_{vi} = 1 \quad \forall v \in V \quad (4.5)$$

$$p_{vi} \in [0, 1] \quad \forall v \in V, \forall Q_i \in \mathcal{Q} \quad (4.6)$$

Constraints (4.4) set capacity constraints for graph nodes. Constraints (4.5)–(4.6) ensure that the values $\{p_{vi}\}_{i \in \{1, \dots, m\}}$ constitute an access strategy, i.e., a distribution on quorums. Since $p_{vi} \geq 0$ are positive reals, we can always find a solution in time polynomial in $\max(m, n)$, if one exists. A solution might not exist if, e.g., the node capacities are set too low.

An iterative algorithm The first placement algorithm of Section 4.1.2 can be combined with the access-strategy-optimizing algorithm above in an iterative way. Let $\text{avg}(\{p_v\}_{v \in V})$ denote the access strategy p defined by $p(Q) = \text{avg}_{v \in V} p_v(Q)$. In addition, let p_v^0 be the uniform distribution for all $v \in V$, and let $\text{cap}^0(v)$ be the capacity of v input to the algorithm. Iteration $j \geq 1$ of the algorithm proceeds in two phases:

1. In the first phase of iteration j , the almost-capacity-respecting placement algorithm of Section 4.1.2 is executed with $\text{cap}(v) = \text{cap}^0(v)$ for each $v \in V$ and with access strategy $p = \text{avg}(\{p_v^{j-1}\}_{v \in V})$, to produce a placement f^j . Recall that it is possible that for some nodes v , $\text{load}_{f^j}(v) > \text{cap}^0(v)$, though the load can exceed the capacity only by a constant factor.
2. In the second phase of iteration j , the access-strategy-optimizing algorithm above is executed with $\text{cap}(v) = \text{load}_{f^j}(v)$ for each $v \in V$ to produce new access strategies $\{p_v^j\}_{v \in V}$.

After each iteration j , the expected response time (4.2) is computed based on the placement f^j and access strategies $\{p_v^j\}_{v \in V}$. If the expected response time did not decrease from that of iteration $j - 1$, then the algorithm halts and returns f^{j-1} and $\{p_v^{j-1}\}_{v \in V}$.

Note that this algorithm can only improve upon the almost-capacity respecting placement algorithm of Section 4.1.2, since the second phase can only decrease average network delay while leaving loads unchanged, and because the algorithm terminates if an iteration does not improve the expected response time.

5. Simulation methodology

We implemented the algorithms in Section 4 in C and GNU MathProg (a publicly available LP modeling language). To solve the LPs we use the freely available glpsol solver. The version of glpsol we use (4.8) can solve LPs with up to 100,000 constraints, which limits the systems for which we can evaluate our algorithms.

Network topologies The network topologies that we consider come from two sources: The first is a set of ping round trip times (RTTs) measured between 50 different sites in Planetlab between July and November 2006 [24]. We call this topology “Planetlab-50”. The second dataset is built from pairwise delays measured between 161 web servers using the king [13] latency estimation tool. The set of web servers was obtained from a publicly available list used in previous research [23]. We call this topology “daxlist-161”.

Quorum systems We evaluate our algorithms for four quorum systems: three types of Majorities commonly used in protocol implementations (the $(t + 1, 2t + 1)$, $(2t + 1, 3t + 1)$ and $(4t + 1, 5t + 1)$ Majorities) and the $k \times k$ Grid. In each experiment we vary the universe size by varying the t and k parameters from 1 to the highest value for which the universe size is less than the size of the graph.

Measures Our results in the following sections were obtained by computing one of two measures: average response time, $\text{avg}_{v \in V} [\Delta_f(v)]$, where $\Delta_f(v)$ is defined according to (4.2), or average network delay, which is computed identically but with $\alpha = 0$ in (4.1).

6. Low client demand

In this section we consider the case when client demand in the system is low. This can be modeled by setting $\alpha = 0$ in definition (4.1), as the response time in this case is well approximated by the network delay.

Lin [19] showed that the singleton placement yields network delay within a factor of two of *any* quorum system placed in the network. Thus, for a system with low client demand, i.e., in which network delay is the dominant component of response time, using a quorum system cannot yield much better response time than a single server. However, a quorum system might still yield advantages in fault-tolerance, and so our goal will be to determine the performance costs one pays while retaining the fault-tolerance of a given quorum system, i.e., by placing it using a one-to-one placement.

Since we are trying to minimize network delay, clients will always use the closest quorum for each access, i.e., $p_v(Q) = 1$ for v 's closest quorum Q , and

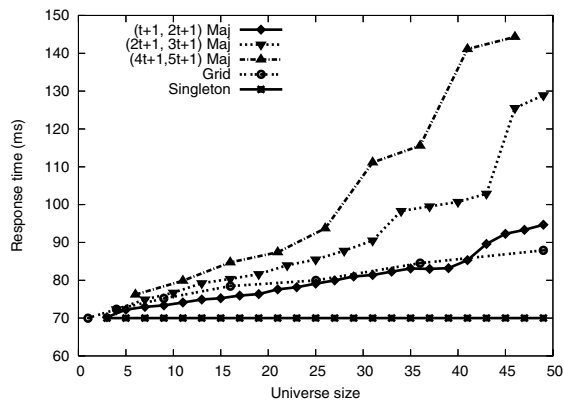


Figure 6.3. Response times on Planetlab-50;
 $\alpha = 0$; *closest* access strategy

$p_v(Q') = 0$ for all others; we call this the *closest* quorum access strategy.

The results of this analysis on the Planetlab-50 topology are shown in Figure 6.3. These results suggest that the average response time increases for each quorum placement as the universe size grows. In some cases the increase exhibits a critical point phenomenon: response time degrades gracefully up to a point and then degrades quickly. This can be best seen for the $(2t + 1, 3t + 1)$ and $(4t + 1, 5t + 1)$ Majorities.

A second observation is that for a fixed universe size, the response time is better for quorum systems with smaller quorums. In almost all the graphs the line corresponding to Grid is the best after the singleton, the $(t + 1, 2t + 1)$ Majority is the second, etc. More surprisingly, the response times for the quorum systems with small quorum sizes are not much worse than that of one server up to a fairly large universe size. The exact values depend on the topology; more generally, from other experiments we performed it seems that the values depend on the distribution of average distances from nodes of the graph to all clients.

In conclusion, under low client demand, using quorum systems with smaller quorum sizes gives better performance. For all quorum systems, the degradation in performance as compared to one server is fairly small up to a certain universe size that depends on the topology and the particular quorum system considered.

7. High client demand

In this section we evaluate algorithms for minimizing response time when there is high client demand in the system. We start by looking at one-to-one placements when clients use either the *closest* access strategy

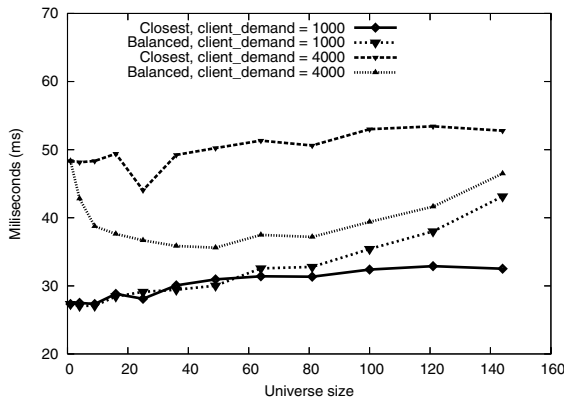


Figure 6.4. Response time for Grid under different client demands on daxlist-161

from Section 6 or a *balanced* strategy in which p_v is the uniform distribution for each client v .

To compute response time we set the α parameter as follows: $\alpha = op_srv_time * client_demand$. We use a value of .007 ms per request for the *op_srv_time* parameter (this is the time needed by a server to execute a Q/U write operation on a Intel 2.8GHz P4). We set *client_demand* to either 1000, 4000 or 16000 requests.

In Figure 6.4 we plot response times for the closest and balanced access strategies for the Grid quorum system when placed on the daxlist-161 topology and $client_demand \in \{1000, 4000\}$. The results show that for low client demand, closest seems to be the best access strategy in most cases (particularly for larger universe sizes where network delays are larger, as well), while balanced is the best access strategy for sufficiently high client demand.

Another interesting aspect illustrated by Figure 6.4 is the effect on response time obtained by varying the universe size (the line corresponding to balanced for a *client_demand* of 4000). For small universe sizes, the processing is spread on just a few nodes, which, in the case of high client demand, has a negative impact on the response time. At the same time, for large universe sizes, each node sees a much smaller load, but now network delay is sufficiently large to become the dominating factor in client response time.

To better illustrate the effect that load balancing has on response time, we also plot results for a higher value of client demand, $client_demand = 16000$ in Figure 6.5. We plot both response time and network delay on the same graph. The network delay component increases with the universe size, while the load component decreases for an access strategy that balances load on servers. Since the load induced by client demand in

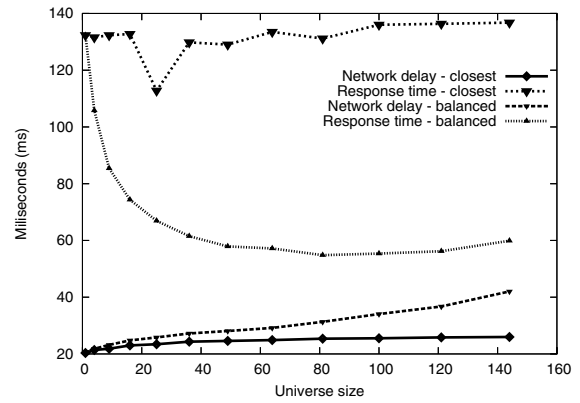


Figure 6.5. Grid with $client_demand = 16000$ on daxlist-161

this case is significantly larger than the network delay component, the response time for the balanced access strategy actually decreases with increased universe size. At the same time, the response time of the closest access strategy does not exhibit this behavior, since this provides no load balancing guarantees on the nodes.

In conclusion, while closest is the best access strategy for sufficiently low client demand (Section 6), balanced is the best for very large client demand. There is also a gray area of client demand values for which none of the two access strategies performs better than the other; this is clearly visible in Figure 6.4 where the lines of the two access strategies for $client_demand = 1000$ cross each other in multiple points. Below we present a technique for finding access strategies to minimize the response time for an arbitrary client demand.

Optimizing the access strategy To find the best access strategy for a given topology, quorum system, quorum placement and client demand, we will use LP (4.3)–(4.6) with different values for the capacity of nodes. While in practice the capacity of a machine is determined by its physical characteristics, here we use $cap(v)$ as a parameter to manipulate the clients' access strategies so as to minimize response time. To use this technique in the real world, we can simply determine an upper bound for $cap(v)$ of a machine v based on its physical characteristics and then run this tool with $cap(v)$ no higher than the obtained upper bound.

We evaluate this technique in the following way: we choose a set of 10 values c_i in the interval $[L_{opt}, 1]$ and set the node capacity of all nodes, $cap(v) = c_i$, for each $i \in \{1, \dots, 10\}$. L_{opt} here is the optimal load of the quorum system considered, for a fixed universe size. We solve LP (4.3)–(4.6) for each value c_i to obtain a set of access strategies (one for each client) and then

compute the response time corresponding to each such set of access strategies. Finally, we pick the value c_i that minimizes the response time. In our evaluation we choose the values c_i to be:

$$c_i = L_{opt} + i \cdot \lambda \quad (7.7)$$

for $i \in \{1, \dots, 10\}$, where $\lambda = (1 - L_{opt})/10$.

Figure 7.6 shows how the response time changes when we vary the node capacities for different universe sizes, assuming a client demand of $client_demand = 16000$. In general, setting a higher node capacity allows clients to access closer quorums, thus decreasing network delay but increasing the load component at some of the nodes at the same time. For high client demand, this can yield worse response times, since nodes with a high capacity will become the bottleneck; i.e., the costs of high load will outweigh the gains in network delay. In this case it makes sense to disperse load across as many nodes as possible, which can be enforced by setting low node capacities.

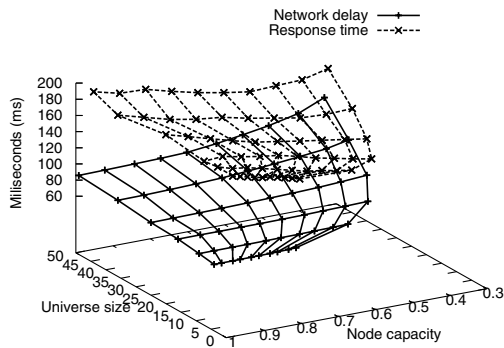


Figure 7.6. Grid when increasing node capacities on Planetlab-50

Non-uniform node capacities A variation of the previous technique can help improve the response time further. This approach is based on the following observation: for a given c_i , LP (4.3)–(4.6) will find access strategies that minimize network delay by selecting quorums that are close to clients, as much as the capacity of graph nodes permits. As a result some nodes will have their capacity saturated, and thus will handle the same volume of requests, irrespective of their average distance to the clients. For this set of nodes, the response time will thus depend on their average distance to the clients: for server nodes further away clients will have to wait more than for closer nodes.

This observation leads us to following natural heuristic: we can set nodes capacities inversely proportional

to their average network delay to the clients. This will hopefully spread load across servers in a way that minimizes response time.

We now present in more detail the algorithm for setting node capacities. Let $\{v_1, \dots, v_n\}$ be the support set of a given placement (we assume only one-to-one placements here), and let s_i be the average distance from all clients to v_i . Our goal is to set capacity $\text{cap}(v_i)$ to be inversely proportional to s_i and in a given range $[\beta, \gamma] \subseteq [0, 1]$. Let $le = \min_{i \in 1..n} \frac{1}{s_i}$ and $re = \max_{i \in 1..n} \frac{1}{s_i}$. We then assign

$$\text{cap}(v_i) = \frac{1/s_i - le}{re - le} (\gamma - \beta) + \beta$$

So, for example, if $v_i = \arg \min_{i \in 1..n} \frac{1}{s_i}$, then $\text{cap}(v_i) = \beta$, and if $v_i = \arg \max_{i \in 1..n} \frac{1}{s_i}$, then $\text{cap}(v_i) = \gamma$.

We evaluate this method for the Grid quorum system with universe size ranging from 4 to 49 on the Planetlab-50 topology. To compare with the results for uniform node capacities above, we use intervals $[\beta, \gamma] = [L_{opt}, c_i]$ for $i = 1..10$ (see (7.7)). We set $client_demand = 16000$ for this set of experiments.

Figure 7.7 shows response times for both uniform and non-uniform node capacities. For small capacities the two approaches give almost identical results. As capacities increase, the heuristic for non-uniform capacities gives better response time than the algorithm for uniform node capacities. The reason: for small values of c_i , the length of the $[\beta, \gamma]$ interval is close to 0, and as such, the nodes from the support set have almost the same capacity. As the $[\beta, \gamma]$ interval grows, the capacities are better spread out and better (inverse proportionally) match the distances s_i . This spreads load over nodes with larger average distances to the clients, which decreases response time.

To see the improvement given by this heuristic we also plot results for a fixed universe size ($n = 49$). Figure 7.8 shows that increasing node capacity increases the response time as well (due to the high load in the system) but at a slower pace for our heuristic than for the algorithm with fixed node capacities. As the size of client demand increases, we expect this effect to become more pronounced.

Evaluation of the iterative approach So far we have evaluated only algorithms yielding one-to-one placements. The last technique for improving response time that we evaluate in this paper is the iterative approach described in Section 4.2. Since this approach creates many-to-one placements, network delay will necessarily decrease: some of the quorums become much smaller, thereby allowing clients to reduce the distance they need to travel to contact a quorum.

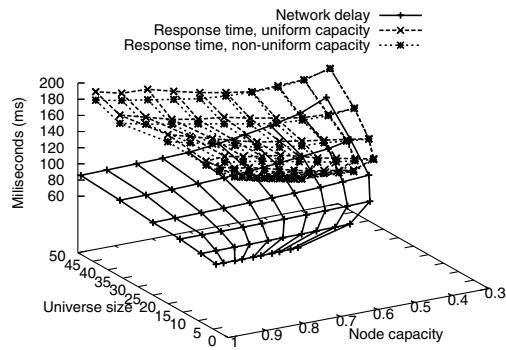


Figure 7.7. Grid on Planetlab-50 with uniform and non-uniform node capacities

In Figure 8.9 we show the performance gains in terms of network delay compared to a one-to-one placement for a 5×5 Grid. We run the iterative algorithm for different values of the node capacity to see whether the improvement in network delay depends on node capacity. For all node capacities the best improvement comes after the first phase, at the end of which many universe elements are placed on the same node. The second phase brings only small improvements. Most of the runs terminate after the first iteration.

We have also evaluated the response time for each intermediary point in this iterative process. The results show that using many-to-one placements can increase or decrease response time depending on the placement found in the first phase of the first iteration and on the client demand. For instance, if the placement found puts multiple quorum elements on many nodes of the graph, the response time increases with the client demand. For low client demand, response time is usually better than for the one-to-one placements. Finally, the response time is always improved between the first and the second phases of the first iteration, but only by small values (usually between 2 and 5 ms). Consequently, using many-to-one placements improves response time over other approaches mostly in the case of low client demand. However, the techniques discussed in Section 6 also excel in this case, while retaining the fault-tolerance of the original quorum system.

8. Conclusions

In this paper we have evaluated techniques for placing quorum systems on a wide-area network to mini-

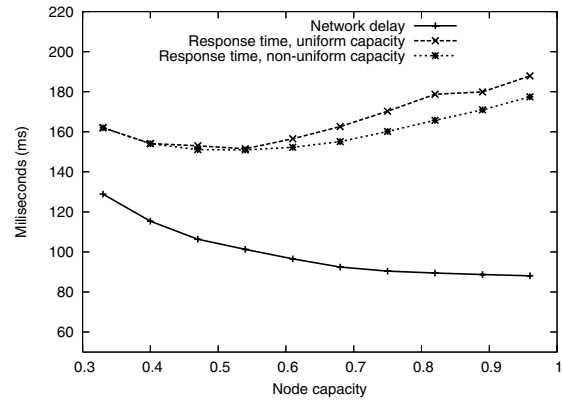


Figure 7.8. 7×7 Grid on Planetlab-50 with uniform and non-uniform node capacities

mize average client response time. The results of this evaluation reveal several interesting facts. First, for low client demand, using quorum systems up to a limited universe size in certain topologies does not substantially degrade performance compared to a single node solution. Thus, quorum systems are a viable alternative to the singleton solution in such cases, and offer better fault-tolerance. Second, as the client demand increases, it is important to balance load across servers to obtain good response time. When the network delay and client demand both play important roles in the response time, finding the right strategies by which clients access quorums is crucial to minimizing response time. Our methods for optimizing clients' access strategies, used with both uniform and non-uniform node capacities, are especially useful here.

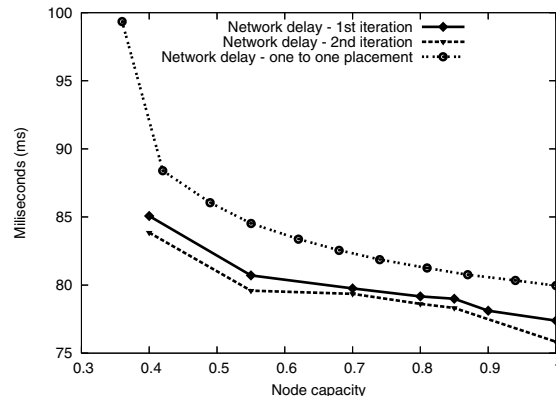


Figure 8.9. Network delay for iterative approach for 5×5 Grid on Planetlab-50

Finally, in our current framework, using many-to-one (instead of one-to-one) placements improves the response time only for low values of client demand. A variation of our model, in which a server hosting multiple universe elements would execute a request only once for all elements it hosts, can clearly improve the performance. We plan to analyze the benefits of such an approach in future work.

References

- [1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable Byzantine fault-tolerant services. In *Proc. 20th ACM Symposium on Operating Systems Principles*, 2005.
- [2] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Scaling Byzantine fault-tolerant replication to wide area networks. In *Proc. 2006 International Conference on Dependable Systems and Networks*, pages 105–114, June 2006.
- [3] Y. Amir and A. Wool. Evaluating quorum systems over the Internet. In *Proc. 26th International Symposium on Fault-Tolerant Computing*, June 1996.
- [4] O. Bakr and I. Keidar. Evaluating the running time of a communication round over the Internet. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, pages 243–252, July 2002.
- [5] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation*, Mar. 2004.
- [6] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4), Nov. 2002.
- [7] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *Knowledge and Data Engineering*, 4(6):582–592, 1992.
- [8] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementations*, Nov. 2006.
- [9] A. W. Fu. Delay-optimal quorum consensus for distributed systems. *IEEE Trans. Parallel and Dist. Sys.*, 8(1):59–69, 1997.
- [10] L. Gao, M. Dahlin, J. Zheng, L. Alvisi, and A. Iyengar. Dual-quorum replication for edge services. In *Proc. Middleware 2005*, pages 184–204, Dec. 2005.
- [11] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, pages 150–162, 1979.
- [12] D. Golovin, A. Gupta, B. Maggs, F. Oprea, and M. Reiter. Quorum placement in networks: Minimizing network congestion. In *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*, 2006.
- [13] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary internet end hosts. *SIGCOMM Comput. Commun. Rev.*, 32(3), 2002.
- [14] A. Gupta, B. Maggs, F. Oprea, and M. Reiter. Quorum placement in networks to minimize delays. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, 2005.
- [15] N. Kobayashi, T. Tsuchiya, and T. Kikuno. Minimizing the mean delay of quorum-based mutual exclusion schemes. *Journal of Systems and Software*, 58(1):1–9, 2001.
- [16] A. Kumar, M. Rabinovich, and R. K. Sinha. A performance study of general grid structures for replicated data. In *Proceedings 13th International Conference on Distributed Computing Systems*, pages 178–185, 1993.
- [17] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.
- [18] J.-H. Lin and J. S. Vitter. ϵ -approximations with minimum packing constraint violation (extended abstract). In *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 771–782, 1992.
- [19] X. Lin. Delay optimizations in quorum consensus. In *ISAAC '01: Proceedings of the 12th International Symposium on Algorithms and Computation*, pages 575–586. Springer-Verlag, 2001.
- [20] D. Malkhi, M. K. Reiter, D. Tulone, and E. Ziskind. Persistent objects in the Fleet system. In *Proceedings of the 2nd DARPA Information Survivability Conference and Exposition (DISCEX II)*, volume 2, pages 126–136, June 2001.
- [21] M. Naor and A. Wool. The load, capacity, and availability of quorum systems. *SIAM J. Comput.*, 27(2):423–447, 1998.
- [22] D. Oppenheimer, B. Chun, D. Patterson, A. C. Snoeren, and A. Vahdat. Service placement in shared wide-area platforms. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, 2005.
- [23] J. Padhye and S. Floyd. The tbit webpage. <http://www.icir.org/tbit/daxlist.txt>.
- [24] <http://ping.ececs.uc.edu/ping/>.
- [25] D. B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(3):461–474, 1993.
- [26] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems*, 4(2):180–209, 1979.
- [27] T. Tsuchiya, M. Yamaguchi, and T. Kikuno. Minimizing the maximum delay for reaching consensus in quorum-based mutual exclusion schemes. *IEEE Transactions on Parallel and Distributed Systems*, 10(4):337–345, 1999.
- [28] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, 2002.