1-1-1990

Finding Optimal Quorum Assigments for Distributed Databases

Donald B. Johnson
Dartmouth College

Larry Raab
Dartmouth College

FINDING OPTIMAL QUORUM ASSIGNMENTS FOR DISTRIBUTED DATABASES

Donald B. Johnson and Larry Raab

Technical Report PCS-TR90-158

Finding Optimal Quorum Assignments for Distributed Databases

Donald B. Johnson* Larry Raab†
Dartmouth College‡

Abstract

Replication has been studied as a method of increasing the availability of a data item in a distributed database subject to component failures and consequent partitioning. The potential for partitioning requires that a protocol be employed which guarantees that any access to a data item is aware of the most recent update to that data item. By minimizing the number of access requests denied due to this constraint, we maximize availability. In the event that all access requests are reads, placing one copy of the data item at each site clearly leads to maximum availability. The other extreme, all access requests are write requests or are treated as such, has been studied extensively in the literature. In this paper we investigate the performance of systems with both read and write requests. We describe a distributed, on-line algorithm for determining the optimal parameters, or optimal *quorum assignments*, for a commonly studied protocol, the quorum consensus protocol[9]. We also show how to incorporate these optimization techniques into a dynamic quorum reassignment protocol. In addition, we demonstrate via simulation both the value of this algorithm and the effect of various read-write ratios on availability. This simulation, on 101 sites and up to 5050 links (fully-connected), demonstrates that the techniques described here can greatly increase data availability, and that the best quorum assignments are frequently realized at the extreme values of the quorum parameters.

Keywords: availability, discrete event simulation, distributed database, quorum assignment, reliability, replication.

1 Introduction

Replication has been proposed as a means of improving the availability of data in a distributed database. Although numerous copies of a data item maybe present in a computer network with replication, we wish to have them operate as if there existed only one copy of each data item. We, therefore, require that any operation, read or write, on a data item access the value most recently written to that item, independent of which physical copies have the most

*e-mail address:djohnson@dartmouth.edu
†e-mail address:raab@dartmouth.edu
‡Department of Mathematics and Computer Science. Hanover, N.H. 03755

current value. In the event that all access requests are read requests, the best approach is clearly to put one copy of the data item at every site. The situation in which all requests are write requests, or, equivalently, when no distinction is made between reads and writes, has been studied extensively in the literature[3, 5, 8, 13, 14, 17]. In [15], we show that the benefits of replication under these circumstances is limited. In this paper, we examine the situation in which there occur both read and write accesses and they are not necessarily treated in the same manner.

Our examination can be viewed in two ways. We can think of ourselves as investigating how availability changes in response to changes in the read-write ratio, or how, given a particular read-write ratio, reads and writes are treated in order to maximize availability. We answer the former question in section 5 and the latter in section 4. We also examine, in section 5.5, the extent to which previous research exclusively on write requests applies to databases with a significant number of read requests.

These are difficult questions as evidenced by the dearth of research on the subject. The dependence of availability on many factors, including the reliability of the network components, the topology of the network, the distribution of the access requests, and the read-write ratio, complicates the optimization of a single parameter, like the quorum assignment. An analytical model in which each of these factors is parameterized would be exceedingly complex. Our approach, for any given network and distribution of access requests, is to express availability in terms of a set of functions $F = \{f_1, f_2, ..., f_n\}$, where f_i expresses the probability density over the "size" of the network component containing site i. In some cases, these functions can be evaluated off-line. In other cases, it is advantageous to calculate these functions in a distributed, on-line manner which incorporates temporal characteristics of the component reliabilities, the network topology, and the access request distribution.

The protocol most frequently studied is the quorum consensus protocol[9]. Unlike its predecessor, the majority consensus protocol[20], the quorum consensus protocol distinguishes between read and write transactions. Thus read throughput, or the percentage of read requests granted, can be increased by relaxing the criteria for allowing a read to proceed. Unfortunately, any relaxation of the read constraints must be matched by a corresponding tightening of the write constraints. We define more precisely these constraints, or *quorums*, and the relationship between them in section 2.1.

In [1], Ahamad and Ammar investigate optimal quorum assignments under the assumption that if two sites are operational then they can communicate. Although this eliminates the possibility of the partitioning which necessitates consistency control, it does provide some interesting analytic results which we show also hold in networks with fallible links. They prove, for instance, that the minimum or maximum of the availability function (see step 3 of Figure 1) occurs at the extreme quorum values. Figures 2-7 exhibit this same property. Ahamad and Ammar also show, as we discuss in section 5.5, that requiring a majority of votes for both read and write accesses is optimal for a wide range of network parameters. Due to the exponential computations involved in purely analytic studies, they are unable to investigate networks with more than nine copies. For networks as large as those studied in the present paper, we have shown that greater availability can be achieved with significantly

more copies[14]. In [7], the quorum assignment problem is addressed in conjunction with determining the optimal vote assignment. Again assuming a non-partitionable network, the authors find the optimal quorum assignment by exhaustively searching an exponential set of candidate coteries.[1] Numerical results are given for networks with up to seven sites.

Although the possibility of incorporating dynamic quorum assignments in consistency control protocols is suggested in [13], the mechanisms for doing so are neither fully-defined nor applied. Herlihy[11] carefully defines and rigorously proves a protocol based upon a hierarchy of quorum assignments. An access which is denied in the current quorum may be able to graduate to a higher quorum in which the access may be allowed. The mechanism for selecting and ordering quorums from the exponential set of possible quorums is not discussed, nor is the performance of this scheme evaluated.[2]

In the present paper, we describe how the quorum consensus protocol can efficiently calculate the optimal quorum assignment from information gained during normal transaction processing. We present both a method for incorporating quorum assignments and for adjusting these assignments to optimize for changing system characteristics, including component failures and recoveries, and shifting access request patterns. This can provide a significant performance advantage over static quorum assignments which must be calculated off-line based upon an abstraction of the real network.

We begin by describing the quorum consensus protocol, a dynamic version of this protocol, and the measure of data availability which we employ to evaluate performance. In section 4 we give the algorithm in terms of the distribution of component "sizes". We then describe the various methods for calculating or approximating this distribution. In section 5 we apply our algorithm to seven different topologies and five read-write ratios. This motivates an enhancement, described and applied in section 5.4, which allows the specification of a minimum write throughput. We conclude with a discussion of the influence of read requests on the performance of replication, and the influence of these techniques on the potential benefits of replication.

2 Protocols

This section briefly describes each of the protocols studied in this paper. We consider two classes of protocols, *static* and *dynamic*. *Static* protocols, including the the quorum consensus protocol, use fixed criteria for determining the *distinguished* component, the set of communicating sites in which access may be granted. *Dynamic* protocols, on the other hand, change these criteria based upon past events in the system. The quorum reassignment protocol of section 2.2 is a dynamic protocol.

[1] Coteries provide a single mechanism, more general than voting, for specifying both vote assignments and quorum assignments[8]. The method given in [7] for enumerating all candidate coteries provides a significant reduction over previous enumeration schemes in the number of candidate coteries.

[2] Our constraints, which we define in section 2.1, differ slightly from those of [11]. We require that each read reports the value of the most recent write, thereby ensuring *one-copy serializability*[6]; whereas, Herlihy guarantees *multiversion serializability*, which allows reads on old values of the data item.

2.1 Quorum Consensus Protocol

In the quorum consensus protocol[9], each copy of a replicated object is assigned a number called its *vote assignment*. Every read access must acquire a minimum number of the total votes called the *read quorum*, q_r; every write access must acquire a minimum number of the total votes called the *write quorum*, q_w. In order to ensure consistency in a network with a total of T votes, the read and write quorums must satisfy the following conditions:

1. $q_r + q_w > T$, and
2. $q_w > \frac{T}{2}$.

The first condition guarantees that each read is aware of the most recent write. The second condition not only ensures that each write is aware of the most recent write, but also prohibits simultaneous writes.

The performance of the quorum consensus protocol is therefore dependent upon the values chosen for q_r and q_w. Condition 2 implies that $\frac{T}{2} < q_w \leq T$. In addition, conditions 1 and 2 taken together allow the assumption that $0 < q_r \leq \frac{T}{2}$, since requiring $q_r > \frac{T}{2}$ would be unnecessarily restrictive. We consider q_r to be our primary variable, and we assign $q_w = T - q_r + 1$, by condition 1.

The quorum consensus algorithm works as follows: when an access request is submitted to a site, that site collects the votes from every site in its current component. If the request is for read, then the read is granted if the number of votes collected is at least q_r. Likewise, if the request is for write, then the write granted if the number of votes collected is at least q_w. If, in either case, the required number of votes is not collected, the access is denied.

Instances of the quorum consensus protocol are known by other names. When $q_r = \lfloor \frac{T}{2} \rfloor$ and $q_w = \lfloor \frac{T}{2} \rfloor + 1$, the quorum consensus is equivalent to the majority consensus protocol[20]. The case $q_r = 1$ and $q_w = T$ is also known as the read-one, write-all protocol. Finally, when the vote assignments and quorum assignments are such that accesses may be issued only from a component containing a particular site (called the *primary site*), the quorum consensus protocol is reduced to the primary copy protocol[2].

2.2 Dynamic Quorum Reassignment Protocol

Since we can calculate on-line the optimal quorum assignment using the techniques of section 4, we may wish to dynamically change the quorum assignment to adjust for temporal characteristics of the network. Clearly, we can wait to make this change until the entire network is operational, that is, until every site can communicate with every other site, but this wait could be lengthy and is unnecessary. Using the version number techniques employed by protocols which dynamically reassign votes[4, 5, 12, 13, 17] we can allow quorum assignments to change without waiting until the entire network is operational. We call this method the quorum reassignment protocol, or QR.

We associate with each copy of the data item both a quorum assignment and a number called the *version* number. The version number is initially one and is incremented with

each change in the quorum assignment. Quorum assignments can only be changed in sets of communicating sites, or *components*, which possess at least a write quorum of votes (using the old quorum assignment). When an access request is submitted to a site x, the quorum assignment in effect is the assignment associated with the site having the highest version number in the component containing x.

For this method to be valid, it must guarantee that no access be granted in a component that is unaware of the new quorum assignments. This constraint is ensured by allowing the assignments to change only in a component, C_1, containing a write quorum, q_w, of the votes, and by the requiring, as in section 2.1, that only one such component exists at a time. The quorum consensus protocol also requires that $q_r + q_w > T$. Thus C_1 is the only component with q_r or more votes. Since $q_w > q_r$, no other component, C_2, may access the data item until some site, x, from C_1 joins C_2. When x joins C_2, C_2 becomes aware of the new quorum assignment, at which time every site in C_2 updates their quorum assignment and version vector. Therefore, no access can take place in a component using an old quorum assignment.

3 Measures of Data Availability

There are two definitions of data availability in the literature[13]. The metric more commonly used, which we will call *Survivability*($SURV$), is the probability that at an arbitrary time there exists at least one site which may access the data object. This is equivalent to the probability that a distinguished component exists or to one minus the probability of a halting state. The second metric, which we will call *Accessibility*(ACC), is the probability that at an arbitrary time an arbitrary site may access the data object.

Informally, the difference between these two metrics is that $SURV$ measures the fraction of time that there exists a site in the network which can access a data object, while ACC takes into account the number of sites which are able to perform that access. For example, policies which increase the number of sites which may access the data will increase ACC but may leave $SURV$ relatively unchanged. On the other hand, $SURV$ favors dynamic protocols since they tend to produce smaller distinguished components than do the static protocols. In any case, the reliability of a single site is a lower bound for $SURV$, since $SURV$ is always realizable by a single copy, and an upper bound for ACC, since at least the site at which the request originates must be up.

We have chosen to present our results using the ACC metric since we are interested in the probability that an arbitrary site will be able to access the data item and therefore would not expect availability greater than the reliability of that site. It is our view that ACC reports more nearly the availability as experienced by a user of the system, who typically cannot readily move from site to site or have knowledge a priori of which sites are functioning.[3]

[3] Our method could be adapted to find optimal quorum assignments using the $SURV$ metric by substituting, in step 1 of the algorithm in Figure 1, the distribution of the number of votes in the *largest* component rather than the distribution of the number of votes in the component to which the transaction is submitted.

1 Assume that the following are known:
α = the fraction of the accesses which are read requests,
r_i = the fraction of the read accesses submitted to site s_i,
w_i = the fraction of the write accesses submitted to site s_i, and
f_i = the probability density function for each site s_i.

2 Let $r(v) = \sum_{i=1}^{n} r_i * f_i(v)$, and
$w(v) = \sum_{i=1}^{n} w_i * f_i(v)$.

3 Let
$$A(\alpha, q_r) = \alpha \sum_{k=q_r}^{T} r(k) + (1-\alpha) \sum_{k=T-q_r+1}^{T} w(k)$$

4 Find q_r for which $A(\alpha, q_r)$ is maximized, and assign $q_w = T - q_r + 1$.

Figure 1: **Optimal Quorum Assignment Algorithm**

4 Finding Optimal Quorum Assignments

The algorithm in Figure 1 describes how to determine the optimal quorum assignments given the read rate, α; the access distributions, r_i and w_i; and the component size distribution, $f_i(v)$. In section 4.1 we define each of these parameters and then describe the steps necessary to find the optimal quorum assignments. In the presentation of the algorithm, $S = \{s_1, s_2, s_3, \ldots, s_n\}$ denotes the set of all sites s_i in the network, and T denotes the total number of votes in the system.

4.1 Description of the Algorithm

The optimal quorum assignment algorithm assumes that four parameters are known. The first three of these, α, r_i, and w_i, are likely to be explicit in the model or can be directly measured by the system. The fourth parameter, $f_i(v)$, is the probability that site s_i is in a component containing exactly v votes. The derivation of this probability density function is discussed extensively in section 4.2.

In Step 2 we form two new probability density functions from the function $f_i(v)$. The first function, $r(v)$, is the probability that an arbitrary read request will be submitted to a site within a component containing v votes. Likewise, $w(v)$ is the probability that an arbitrary write request will be submitted to a site within a component containing v votes. Notice that if the access requests are uniformly distributioned, that is $r_i = w_i = \frac{1}{n}$ for all sites i, then $r(x) = w(x)$.

In Step 3 we use the cumulative distribution functions $R(q_r) = \sum_{q_r}^{T} r(k)$ and $W(q_r) = \sum_{T-q_r+1}^{T} w(k)$ to calculate availability given α and q_r. Thus $R(q_r)$ is the probability that an arbitrary read request will be granted, and $W(q_w)$ is the probability that an arbitrary write request will be granted.

The final step requires that, given α from step 1, we find the q_r which maximizes $A(\alpha, q_r)$. Since q_r can only assume integer values between 1 and $\lfloor \frac{T}{2} \rfloor$, one could naively, yet in polyno-

mial time in the number of sites, conduct an exhaustive search for the optimal q_r. However, a number of characteristics of the function $A(\alpha, q_r)$ can be used to significantly reduce the computation time. As will be shown in section 5.3, $A(\alpha, q_r)$ is frequently maximized when $q_r = 1$ or $q_r = \lfloor \frac{T}{2} \rfloor$. This fact can be used in a numeric technique such as the so-called golden section search in one dimension (described, for example, in [18]). Other techniques can be applied to the continuous approximations of A, where $f_i(v) = \frac{dF_i(v)}{dv}$ such that $F_i(v_1) - F_i(v_2)$ is the probability that the component containing site s_i has between v_1 and v_2 votes. The Brent's Method (again, see [18]) makes use of the derivative of A, which we know from $r(v)$ and $w(v)$.

4.2 Finding Component Sizes

Step 1 of the algorithm in Figure 1 assumes that we know the probability density function $f_i(v)$. The function $f_i(v)$, the probability that site s_i is in a component containing v votes, can easily be found for some symmetric networks including ring, fully-connected, and single bus networks.

A ring of n sites with a copy at each site and one vote per site has density function

$$f_i(v) = \begin{cases} vp^v r^{v-1}(1-r) + p^v r^v & \text{if } v = n = T \\ vp^v r^{v-1}\Big((1-p) + p(1-r)^2\Big) & \text{if } v = T-1 \\ vp^v r^{v-1}(1-pr)^2 & \text{if } 0 < v < T-1 \\ (1-p) & \text{if } v = 0 \end{cases}$$

where p and r are the reliability of the sites and links, respectively.

The density function for a fully-connected network of n nodes is more complicated than that of a ring network. The expression for $f_i(v)$ below involves another function, $Rel(m, r)$, which is the probability that all m sites of a fully-connected network can communicate assuming that the sites never fail and the links have reliability r. Gilbert in [10] presented the following recursive formula for $Rel(m, r)$,

$$Rel(m, r) = 1 - \sum_{i=1}^{m-1} \binom{m-1}{i-1} (1-r)^{\,i(m-i)}\, Rel(i, r)$$

Using Rel, we can express the probability density function as

$$f_i(v) = \binom{n-1}{v-1} p^v \Big((1-p) + p(1-r)^v\Big)^{n-v} Rel(v, r)$$

The density function of the single bus network depends upon the design of the network. If the architecture is such that no site can function when the bus is inoperative, then $f_i(v) = \binom{n-1}{v-1} r p^v (1-p)^{n-v}$, where r is the reliability of the bus. If, on the other hand, the bus failure does not necessitate site failure, then

$$f_i(v) = \begin{cases} p & \text{if v=1} \\ \binom{n-1}{v-1} r p^v (1-p)^{n-v} & \text{otherwise.} \end{cases}$$

Unfortunately, it is very unlikely that f_i can be calculated efficiently in general graphs. In [14], we prove that calculating the expected size of the component containing a site i is $\#P$-complete. Since this expectation is equivalent to the mean of the distribution f_i when the votes are uniform, and since the mean of this distribution can be calculated in polynomial time in the number of sites, finding f_i must be $\#P$-complete.

Although $\#P$-complete in general, it is not difficult to approximate f_i based upon past performance of the database system. This method may even be preferable to exact calculation since it requires very little computation time, is able to recognize changes in the system not anticipated by the model, and can accommodate dynamic protocols (see section 4.3). Periodically, each site s_i queries every site with which it can communicate, recording the total number of votes possessed by all the sites in its component. If past history is indicative of future performance, then the values acquired in this manner approach $f_i(v)$.[4] Rather than performing broadcasts solely to acquire this vote total, site i can record the totals received while performing other functions required by the consistency control algorithm such as acquiring permission for data access.

4.3 Dynamic Quorum Reassignments

Not only does the algorithm of Figure 1 describe an efficient method for finding optimal quorum assignments for static systems, it also provides a method for dynamically adjusting quorum assignments. This allows the consistency control algorithm to adjust for temporary or unanticipated changes in system state, such as a shifting pattern of data access or periodic component failure. The potential benefits of dynamic quorum reassignment are significant. A static quorum assignment must optimize over the entire stream of access requests and component failures and recoveries, whereas, by dynamically adjusting the quorum assignments, we can adjust for temporal characteristics of the access request stream and network configurations. Moreover, a static quorum assignment generated off-line using a mathematical model reflects the assumptions, such as full link reliability or component failure independence, made in order to generate the assignments. If any of these assumptions are inaccurate, the quorum assignments will be suboptimal. Superior availability is therefore achieved by basing quorum assignment upon local, real-world performance characteristics, rather than abstract, frequently unrealizable models.

The reassignment of quorum values is quite simple. Periodically each site i determines f_i and uses the algorithm of Figure 1 to determine optimal quorum assignment. If site i cannot communicate with some site j, then site i may approximate f_j, use an old value for f_j, or wait until communication with site j is restored. When a site finds that the current quorum assignment differs significantly from the optimal quorum assignment, the site attempts to install the new assignment using the QR protocol described in section 2.2.

[4] Since non-operational sites cannot record access requests, density functions approximated in this manner yield availability A', the probability that an access request submitted to an arbitrary *operational* site will succeed. The distinction between A' and A, the probability that an access request submitted to an arbitrary site will succeed, is of little consequence since $pA' = A$, where p is the reliability of a site. Therefore q_r maximizes $A(\alpha, q_r)$ if and only if q_r maximizes $A'(\alpha, q_r)$.

5 Examples and Enhancements

In this section we demonstrate the use of the algorithm as presented in section 4.1 by finding the optimal quorum assignments for five read-write ratios on each of seven topologies. Figures 2-7 show the effect of the quorum assignments on availability. For every site i, we approximate the density function f_i using the on-line technique described in section 4.2.

Although we find that optimal quorum assignments significantly improve performance, we also discover that in some cases achieving the optimal availability eliminates nearly all writes. We present and demonstrate an enhancement to our algorithm that finds optimal quorum assignments given a minimum write throughput constraint. These new assignments are optimal with respect to the new constraint; that is, they yield the highest possible availability while ensuring that some minimum fraction of all write requests is granted. Before demonstrating these new algorithms, we discuss our system model and simulator.

5.1 System Model

The system we consider is composed of sites and bi-directional links. Links fail by failing to transmit messages; partial failures such as links operating in only one direction or garbling messages are not considered. Our assumption is that any message transmitted is correct in its entirety and that the sequential order of transmitted messages is preserved. Processors are fail-stop [19]; although they may fail to send or receive a message, byzantine failures[16] are not considered. Since message passing is the only inter-node communication mechanism, processor and link failures can partition the network into components. Finally, all node and link failures are eventually repaired, although once repaired they are again subject to failure. All events, data accesses and site and link failures and recoveries, are modeled to occur instantaneously. Therefore no site or link can either fail or recover while an access request is processing.

The results cited below are for a single data object with one copy at every site. We employ a uniform vote assignment of one vote per copy, since the data access distribution and component reliabilities are all uniform and the topologies are roughly symmetric. Although we investigate other scenarios in [14], our purpose here is not to be exhaustive but to illustrate and enhance the algorithm for determining optimal quorum assignments.

We examine environments comprised of 101 sites configured into various topologies beginning with a ring, and adding links until all the sites are fully connected. We chose to consider ring-based networks since a ring is completely connected with the minimum number of links necessary to guarantee at least two disjoint paths between every pair of sites. In this paper we denote by Topology i a ring with $i = 0, 1, 2, 4, 16, 256, 4949$ additional links, or chords. The exact placement of the chords can be found in [14].

5.2 Simulator

The events, site and link failures and recoveries and access requests are generated by a steady-state discrete event simulator. A detailed explanation of the necessity of simulation and of

the parameters used by our simulator is contained in [14]. We list these parameters and their values below:

- The submission of data access requests by each site is modeled as a Poisson process with mean $\mu_t = 1$.
- The ratio, ρ, of the mean time-to-next-access to the mean time-to-next-failure is $\frac{1}{128}$.
- Site and link failures and recoveries are modeled as Poisson processes. The mean time-to-next-failure of each component, μ_f, is the same for both sites and links. Likewise, the mean time to recovery, μ_r, is the same for both.
- Each component is 96% reliable. Therefore $\frac{\mu_f}{\mu_f+\mu_r} = .96$.
- In order to overcome the initial state, we do not monitor system state or performance until an initial 100,000 accesses have passed.
- The simulation is run for 1,000,000 accesses beyond the initial 100,000.
- We assume that both read and write requests are submitted uniformly at random to every site in the network. Therefore $r(v) = w(v)$ as noted in section 4. We regard a down site as a member of a component of size zero. This implies, as discussed in section 4, that the availability includes transactions submitted to down sites.

All simulations were run on a DEC Station 5000 and each batch required from one-half to two hours depending upon the topology. Each availability figure reported in this paper reflects the average availability over a number of *batches* each consisting of 1,000,000 accesses. A *batch* is a series of events, and the number of batches, which ranges from five to eighteen, is dictated by the desired confidence interval. The network is reset to the initial state before each batch is begun. Availabilities reported are with a 95% confidence interval with an interval half-size of at most $\pm 0.5\%$.

5.3 Optimal Quorums

Figures 2-7 show the various availability curves for seven different topologies, each with α, the percentage of access requests which are reads, equal to 0, .25, .50, and .75. The availability curves produced by topology 4949 (fully-connected) are not shown since they are nearly identical to those produced by topology 256.

The most striking observation from these figures is that all curves for a given topology converge at $q_r = \lfloor \frac{T}{2} \rfloor$. This occurs since q_r and q_w are nearly equal and therefore no distinction is made between read and write requests. We also see that for a given α the availabilities at $q_r = 1$ is independent of the topology, the read succeeds whenever the site to which the request was submitted is operational. Since the probability that a site is operational is 96%, the availability at $q_r = 1$ is $.96\alpha$.

The graphs also show that all the curves, with only the exception of topology 16 at $\alpha = .75$, have maximum value at an endpoint of the curve. If the maximum occurs when $q_r = \lfloor \frac{T}{2} \rfloor$,

then this is clearly the best quorum assignment. On the other hand, a maximum that occurs at $q_r = 1$ may be unsuitable since $q_w = T$ and writes will succeed only when every copy is accessible. Remedying this shortcoming is the subject of the next section. We return to the case $q_r = \lfloor \frac{T}{2} \rfloor$ in section 5.5.

5.4 Write Constraint

As shown in the previous section, the optimal quorum assignment may be unacceptable due to very low write availability. We now describe two techniques for introducing a write constraint into the algorithm for finding optimal quorum assignments. In this context, the optimal quorum assignment is that assignment which maximizes availability given the write constraint.

The first technique for increasing write throughput is simply to weight the writes in the calculation of "availability". In this case, the formula for availability becomes

$$A(\omega, \alpha, q) = \alpha \sum_{k=q}^{T} r(k) + \omega(1-\alpha) \sum_{k=T-q+1}^{T} w(k)$$

where ω is the weight given to writes. This changes the definition of availability to a *weighted* linear combination of the availabilities of read and write accesses. We do not demonstrate this method by showing any such curves in this paper for two reasons. Firstly, there are infinitely many choices for ω and no clear criteria for choosing a value. Secondly, the method which follows is preferable since it does not require any change in the definition of availability.

If the optimal quorum assignment is unacceptable due to low write availability, then let us consider only those assignments which yield write availability of at least A_w. Such assignments require read quorums q_r such that $A(0, q_r) \geq A_w$. We can now maximize $A(\alpha, q_r)$ given this new constraint on q_r.

We demonstrate this method in Figure 4. Notice that the bottom curve on each graph is $A(0, q_r)$, and therefore we can use this curve to find the range of q_r for which $A(0, q_r) \geq A_w$. Suppose that $\alpha = 75\%$. Then the optimal availability is 72% and is achieved when $q_r = 1$. But at this point $q_w = T$ and therefore a write request will succeed only when all copies are accessible. Since this is very unlikely in a system of 101 copies, we can require $A_w \geq 20\%$, from which we find that q_r must be greater than 27 votes. Since the availability at $\alpha = 75\%$ decreases monotonically as q_r increases, $q_r = 28$ is the quorum assignment that optimizes availability when we require $A_w \geq 20\%$. The availability at this point is 50%.

5.5 Effects of Read-Write Ratio

As mentioned in the introduction, previous work has concentrated on the effects of replication without distinguishing between read requests and write requests. This approach is inherent in the majority consensus protocol[20], the coterie-based protocols[8], the primary copy protocol[2], and the quorum consensus protocol with $q_r = q_w$ [9].

From Figures 2-7, we see that one-half of the curves have maximum at $q_r = \lfloor \frac{T}{2} \rfloor$. The situations in which this is true include low read rates and highly-connected topologies, supporting the conclusion of [1] as mentioned in section 1. In these cases, the results of previous research apply directly to the case where there are both read and write accesses. On the other hand, the remaining curves indicate that we cannot ignore the read-write ratios in determining availability. In fact, this ratio can have a profound effect on the optimal quorum assignment and on consequent availability. Frequently, in fact, the assignment $q_r = \lfloor \frac{T}{2} \rfloor$, $q_w = \lfloor \frac{T}{2} \rfloor + 1$ yields the lowest availability.

6 Conclusion

The results of this paper demonstrate both the critical influence of the quorum assignment on the availability of replicated data and the ability of our algorithm to determine the optimal quorum assignment. Although we have shown that a seemingly necessary calculation is $\#P$-complete, we have described and used a method for approximating this value on-line. In addition to being feasible, this on-line method has the advantage of changing the quorum parameters over time in response to changes in the network topology, component reliabilities, or access request distribution. This property allows our algorithm to be employed by a dynamic quorum reassignment protocol, thereby adjusting quorum assignments to exploit temporal characteristics of these parameters.

We have demonstrated via simulation the effectiveness of our algorithm and have shown that optimal quorums frequently occur either when both read requests and write requests require a majority of the votes or when read requests require only one vote and write requests require all votes. We have then described an enhancement for our algorithm that modifies quorum assignments of the latter type, which, while maximizing availability, incur an intolerable reduction in write throughput. Quorum assignments found in this way yield the maximum availability that can be achieved while guaranteeing some minimum write throughput.

References

[1] Mustaque Ahamad and Mostafa H. Ammar. Performance characterization of quorum consensus algorithms for replicated data. In *Proceedings of the 6th Symposium on Reliability in Distributed Software and Database Systems*, pages 161–168. IEEE, 1987.

[2] P. A. Alsberg and J. D. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the 2nd Annual Conference on Software Engineering*, pages 627–644, October 1976.

[3] Daniel Barbara and Hector Garcia-Molina. The reliability of voting mechanisms. *IEEE Transactions on Computers*, C–36(10):1197–1208, 1987.

[4] Daniel Barbara, Hector Garcia-Molina, and Annemarie Spauster. Protocols for dynamic vote reassignment. In *Proceedings of the 5th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 195–205. ACM, 1986.

[5] Daniel Barbara, Hector Garcia-Molina, and Annemarie Spauster. Increasing availability under mutual exclusion constraints with dynamic vote reassignment. *ACM Transactions on Computer Systems*, 7(4):394–426, 1989.

[6] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[7] Shun Yan Cheung, Mustaque Ahamad, and Mostafa H. Ammar. Optimizing vote and quorum assignments for reading and writing replicated data. Technical Report GIT-ICS-88/20, Georgia Institute of Technology, June 1988.

[8] Hector Garcia-Molina and Daniel Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, October 1985.

[9] D. K. Gifford. Weighted voting for replicated data. In *Proceedings 7th ACM SIGOPS Symposium on Operating Systems Principles*, pages 150–159, Pacific Grove, CA, December 1979.

[10] E. N. Gilbert. Random graphs. *Annals of Mathematical Statistics*, 30:1141–1144, 1959.

[11] Maurice Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Transactions on Database Systems*, 12(2):170–194, June 1987.

[12] Sushil Jajodia and David Mutchler. Dynamic voting. In *Proceedings of the SIGMOD Annual Conference*, pages 227–237. ACM, May 1987.

[13] Sushil Jajodia and David Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Transactions on Database Systems*, 15(2):230–280, June 1990.

[14] Donald B. Johnson and Larry Raab. Effects of replication on data availability. *International Journal of Computer Simulation*, 1991. to appear.

[15] Donald B. Johnson and Larry Raab. A tight upper bound on the benefits of replication and consistency control protocols. In *Proceedings of the 10th Symposium on Principles of Database Systems*. ACM, May 1991. to appear.

[16] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages*, 4(3):382–401, July 1982.

[17] Jehan-François Pâris and Darrell D. E. Long. Efficient dynamic voting algorithms. In *Proceedings of the 4th International Conference on Data Engineering*, pages 268–275. IEEE, February 1988.

[18] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes*. Cambridge University Press, 1986.

[19] R. D. Schlichting and F. B. Schneider. Fail stop processors: An approach to designing fault-tolerant computing systems. *ACM Transations on Computer Systems*, pages 222–238, 1983.

[20] R. Thomas. A majority consensus approach to concurrency control. *ACM Transactions on Database Systems*, 4(2):180–209, June 1979.

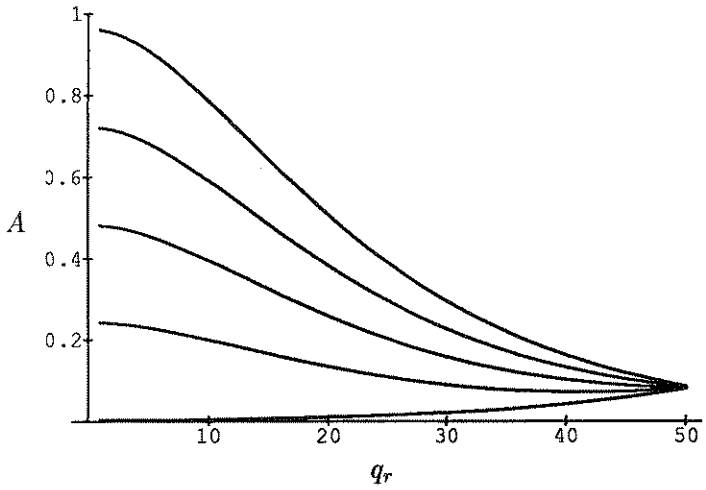

Figure 2: Topology 0

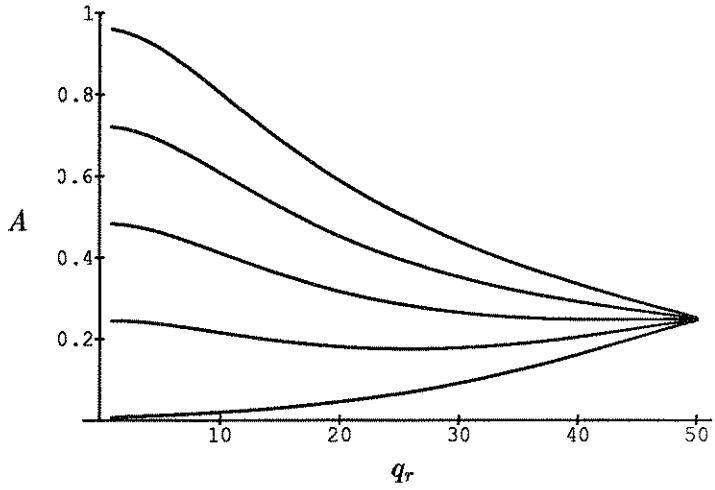

Figure 3: Topology 1

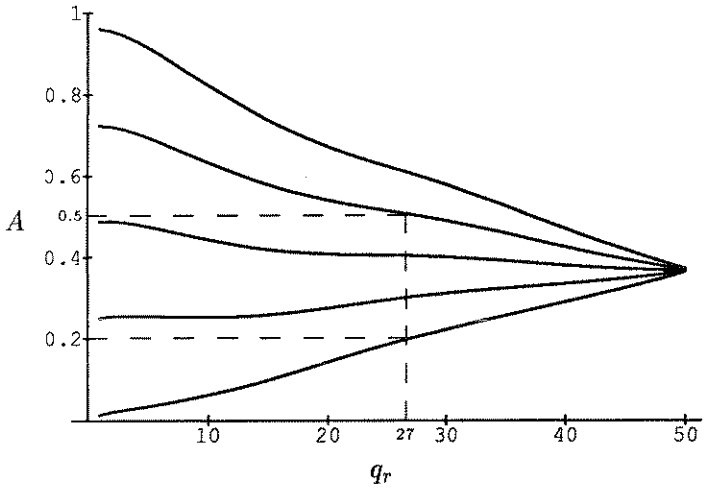

Figure 4: Topology 2

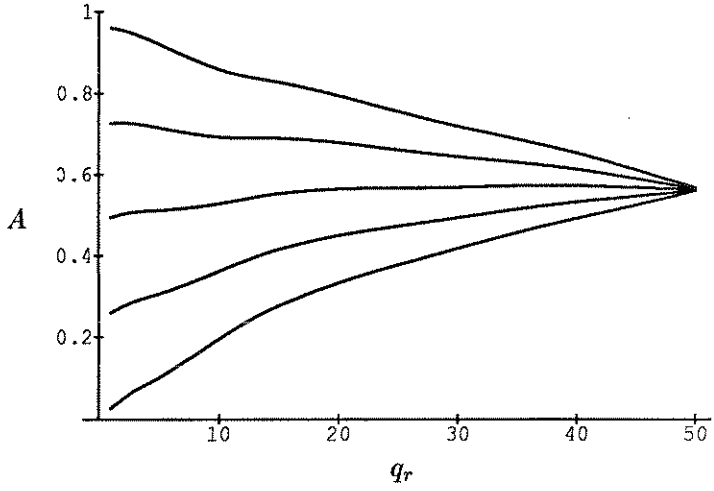

Figure 5: Topology 4

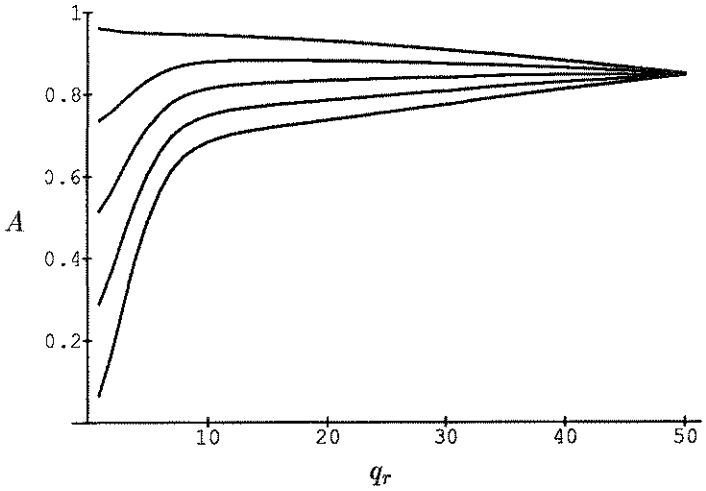

Figure 6: Topology 16

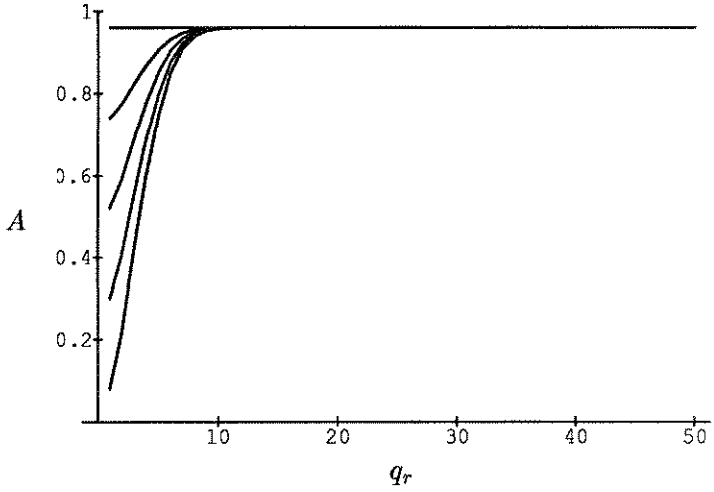

Figure 7: Topology 256

The curves of each figure represent, from bottom to top, $\alpha = 0$, .25, .50, .75, and 1. The parameter α is the fraction of access requests which are read requests.