From *Proc. Third Int. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Jan. 18-20, 1995, Durham, NC.

THE EFFECT OF WORKLOAD ON THE PERFORMANCE AND AVAILABILITY OF VOTING ALGORITHMS

Muhammad A. Qureshi and William H. Sanders
Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
The University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA
qureshi@crhc.uiuc.edu and whs@crhc.uiuc.edu

Abstract

Voting algorithms are a popular way to provide data consistency in replicated data systems. By maintaining multiple copies of data on distinct servers, they can increase the data's availability, as perceived by a user. Many models have been made to study the degree to which replication increases the availability of data, and some have been made to study the cost incurred in maintaining consistency. However, little work has been done to evaluate the time it takes to serve a request, accounting for server and network failures, or to determine the effect of workload on these measures. The effect of workload can be significant, since failures of system components are not important unless they are needed to deliver a service, and requests can force updates on data that would otherwise be outdated. In this paper, with the help of stochastic activity networks, we determine the availability and mean time to respond to write requests as a function of the number of replicated copies and workload offered to the system. The results illustrate that it is indeed possible to determine such measures analytically and that workload, as well as the number of copies, is an important determinant of availability and response time.

Key words: Replicated Data Systems, Voting Algorithms, Availability, Response Time, Stochastic Activity Networks, Stochastic Petri Nets.

1 Introduction

Dependable access to data is important for many applications. This dependability is most often achieved by replication of data on multiple nodes of a system. In this way, the failure of some subset of the nodes can be tolerated, maintaining the availability of data to a user of the system. However, increased availability comes at a cost in performance, since the user must be guaranteed to see the most current replicated copy which reflects the latest changes in data. Replica control algorithms provide such data consistency. The most popular replica control algorithms are based on the idea of voting which was first presented by Thomas [1]. In such schemes, each node is assigned a vote v_i. Read and write quorums (numbers of votes of r and w, respectively) are then chosen such that only one write operation can be performed at any time, and no read operations can be performed while a write operation is being performed. This behavior can be achieved by requiring that $r+w$ be greater than the total number of votes assigned to the nodes and w be greater than half of the total number of votes assigned to the nodes. Several variants of the basic voting algorithm have been introduced to improve availability and performance. These variants are generally classified as either static or dynamic. Algorithms which assign a fixed number of votes to each node are called "static" voting algorithms (for example, see [2, 3, 4]). Dynamic algorithms assign a dynamic number of votes to each host, depending on the state of the system. They increase the availability of data by reassigning votes, after a component failure among the remaining functioning hosts (e.g., [5, 6, 7]).

Most of the studies related to the evaluation of voting algorithms have focused on determining the availability or reliability of particular algorithms, or the "cost" associated with implementing the voting algorithm. While these studies provided useful information concerning the subject protocols, they were limited in several respects. First, by basing the criteria for proper operation (i.e., an available system) on the structural configuration of the system, they do not provide a user-oriented view of availability which bases the proper/improper service criteria on the ability of a system to deliver proper service *when it is requested.* The impact of workload on fault-tolerant system dependability is evident from earlier work (e.g., see [8, 9]). Second, most studies have been limited to studying the impact of a particular protocol and network on the availability of the data, without simultaneously considering the performance delivered by the system. Failures of system components can significantly affect the performance of a replicated data system, as perceived by a user. In many applications performance measures such as the mean response time can be useful in determining the suitability of particular voting algorithms. This fact has been recognized by several researchers (e.g., see [10, 11, 12]), which underscores the importance of determining the performance, as well as availability of voting algorithms.

In this paper, we use stochastic activity networks (SANs), a variant of stochastic Petri nets, to evaluate the performance and availability of particular static and dynamic voting algorithms, taking into account the effects of workload and component failures. Stochastic Petri nets were first applied to the evaluation of voting algorithms by Dugan and Ciardo [13], who evaluated the effect of changing the number of copies on data availability. They did not, however, study the effect of workload on availability or determine the performance of the algorithm in the presence of faults. In particular we consider both server and network failures. Our measure of performance is the mean response time of a write request to the system, keeping in mind that a request may have to wait until the number of copies required to form a quorum become available. The results are significant for two reasons. First, they illustrate that one can indeed build accurate and detailed models to analytically evaluate the performance and availability of replicated data systems. Second, they provide interesting information regarding the performance and availability of the specific voting algorithms and hardware configuration studied. Among other things, we show that the mean response time of the voting algorithms considered decreases with an increase in the number of copies employed, and an *increase* in workload. This result is non-intuitive and linked to the particular voting algorithms we considered.

2 Network, Voting, and Workload Model

2.1 Network Architecture Model

We consider a networked LAN environment where a single broadcast-capable link, i.e., "network", connects the hosts together. A host on the network behaves as either a client or a server. Each server carries a replicated data object. Clients generate requests for the replicated data, while the servers service the requests.

Furthermore, we assume that the number of replicated data objects is equal to the number of servers and each replicated data object resides on a different server. A data object becomes unavailable if either the server it is on or the network fails. For simplicity, we further assume that each replicated data object consists of a single file copy and all the data objects carry the copy of the same file. Since each server carries a replicated copy of the file, it is sufficient to monitor a server's status to keep track of the status of the file copy.

Servers and the network fail independently of each other and can fail at any time other than when the replicated file system is in reconstruction. Since reconstruction time is very short relative to the failure rates of the network and servers, it is reasonable to assume that failures do not occur during reconstruction. We assume that failure and repair times are exponentially distributed with fixed rates. Furthermore, we assume that a network failure prevents all communication on the LAN. In this case, the replicated file system becomes unavailable and cannot provide any service (including completing service of a request in the system). As soon as the network is repaired, it becomes available and, if conditions are such that it can form a quorum, once again begins processing requests.

Repair starts as soon as a server or the network fails. We take a conservative approach as suggested in [13], and always consider a repaired server as outdated. How outdated a repaired copy is depends upon the number of write requests it has missed. However, in order to model the replicated file system with a reasonably small state space, we treat all outdated copies similarly and ignore the degree to which they are outdated.

2.2 Voting Algorithms

Many voting algorithms could have been used for this study. We consider one static and one dynamic voting algorithm. Since they only differ in the requirement for quorum formation, we describe the static voting algorithm and highlight the differences as necessary. Both algorithms are related to the work by Jajodia and Mutchler [7].

The protocols make use of two types of version numbers associated with each copy of data, one "physical" and the other "logical." The *physical* version number is an indication of the version of the data at the server. The *logical* version number is the version that was written during the last quorum in which the copy participated. Note that the physical version number may be less than the logical version number for a copy. This indicates that the copy participated in a quorum formation, but (as is possible in the protocol) did not have its data updated. Given these two version numbers, there are three classes a copy can fall into. The first class is where both the physical and logical version numbers indicate that the copy participated in the most recent quorum formation. These copies are called *physically current* copies. How this will be determined is discussed in the following.

The second class is where the logical version number indicates the copy participated in the most recent quorum. However, the physical version number is less than the logical version number, indicating that the copy is out of date relative to the most recent update. We call these copies *logically current* copies. The third and final case is when the logical version number indicates that the copy did not participate in the last quorum formation. In this case the data of the copy is also outdated, since the physical version number must be less than or equal to the logical version number. We call these copies *outdated* copies. Both algorithms, static and dynamic, specify that a write request can be executed if there exists at least one physically current copy and the number of physically or logically current copies participating in quorum formation is greater than half of some number X. The selection of this number X is what differentiates the static and dynamic algorithms. The static algorithm specifies that X is equal to the total number of copies, while the dynamic algorithm requires that it must be equal to the number of copies that took part in the service of the last request.

More precisely, each file f has an associated version number, VN, which is initially set to 0 and then incremented with each write request on the file. Each copy f_i of the file f has a physical version number, PN_i, to keep track of the physical status of the copy, and logical version number LN_i to keep track of the last quorum that copy participated in. A copy f_i of the file f is said to be physically current if $PN_i = VN$. It is logically current but not physically current, if $LN_i = VN$ but $PN_i < VN$. Otherwise, it is outdated. Furthermore, each copy f_i has an associated variable SC_i to keep track of the number of copies that took part in the service of the last request. (Note that variable SC_i is only required in dynamic voting). Under normal operation (when a quorum can be formed), the two phases *voting* and *commit* are executed in response to a write request:

Voting After generating a write request for a file f, the client enters the voting phase. In this phase, the client transmits messages to the servers carrying copies of the file. Each server S_i locks the local copy f_i and sends the physical and logical version numbers, PN_i and LN_i, back to the client. (Note that in case of dynamic voting each server also sends variable SC_i back to the client). After receiving messages from the servers, the client first determines the greatest logical version number among the responses it received. It then checks to see whether it received a number of responses with this logical version number greater than half of the number of copies either in the system for static voting or that took part in the service of the last request for dynamic voting. It then checks to see if at least one of the copies with $LN_i = VN$ has $PN_i = VN$. If so, then the client has obtained a physically current copy, and can proceed to do the update and commit the write. If not, the client sends messages to the servers to release their locks and the file system is forced into reconstruction.

Commit If a quorum could be formed during the voting phase, the system enters the commit phase. In this phase, the client increments the VN associated with the file f and then it sends the updates to all servers that responded to the request for a quorum formation. Each receiving server S_i updates its local copy f_i, makes the associated $PN_i = LN_i = VN$, and then releases the lock. Also, in case of dynamic voting, each server updates the variable SC_i equal to the number of copies that took part in the quorum formation before releasing the lock. Note that in the variant of the algorithm studied in this paper, we physically update all servers that responded to the last quorum formation request. We do this because we assume a broadcast-capable network where all servers can be updated in a cost equal to that of updating a single server. When the cost to physically update multiple servers is greater than a single server (as it is on a non-broadcast-capable network), it may not be beneficial to update all servers. The algorithm itself only requires that we increment the logical version number of more than half the servers, and physically update at least one server.

Reconstruction is only invoked if the system is unable to form a quorum in the voting phase. During the reconstruction phase, all failed servers are first repaired and then all the servers, including the repaired ones, are physically updated. The write request which causes the invocation of reconstruction waits during the reconstruction and after the reconstruction triggers voting again.

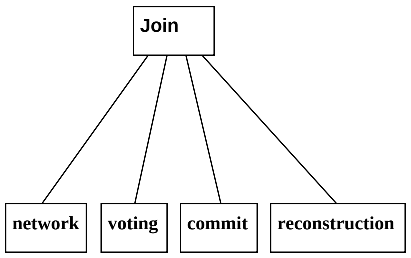

Figure 1: Composed Model: *file system model*

2.3 Workload Model

The workload model captures the demands placed on the data. In a real system, requests for the data may come from many sources, and result in contention for the data. The file system can only service one write request at a time, so requests must be queued and the system services them sequentially. Since our goal is to understand the effect of such a workload on the protocol, it suffices to model the workload as a simple generator of requests, with an exponentially distributed time between the completion of one request and the receipt of the next. This point of view simplifies the model as there is no need to represent the clients individually. Furthermore, since the workload model insures that there is no more than one write request at a time in the system, we do not explicitly need to model the locking mechanism which insures that there is at most one write request in the system.

3 SAN Representation

We use stochastic activity networks, a stochastic extension of Petri nets, to represent the replicated file system model. Space does not permit to review the theory of stochastic activity networks, but interested readers are encouraged to consult [14, 15].

To aid in understanding, the static and dynamic replicated file system models are divided into four submodels: *network*, *voting*, *commit*, and *reconstruction*, representing the file system architecture, the workload and voting phase, the commit phase, and the reconstruction process, respectively. The four submodels are then joined by connecting certain of their places together to generate a complete model of the replicated file system. In SAN terminology, this model is referred as a *composed model*, and is given in Figure 1. To illustrate how SANs work, the SAN submodel, *network*, is presented in the remainder of the section. The complete SAN model is not presented due to space limitations, but can be found in [16].

Figure 2 is a SAN model of the network architecture of the replicated file system. It specifies, precisely, how server and network failures and repairs occur, and the effect these failures and repairs have on the replicated file system. Tables 1-2 define the components of the SAN. As depicted in Figure 2, the marking of places *pc_servers*, *lc_servers*, *o_servers*, and *f_servers* represent the numbers of physically current servers, logically current servers, outdated servers, and failed servers, respectively. Initially, the marking of place *pc_servers* is set to the number of servers (copies) in the system, while the markings of places *lc_servers*, *o_servers* and *f_servers* are set to zero. Place *r_servers* is a temporary holding place for servers, and represents the number of servers that are repaired while the system is in reconstruction. Place *link* represents the status of the network. As discussed in the previous section, the network is non-degradable, i.e., it either is functioning, and can deliver messages, or has failed, and cannot provide communication. A marking of one in this place indicates the network is functioning; a marking of zero indicates that the network has failed.

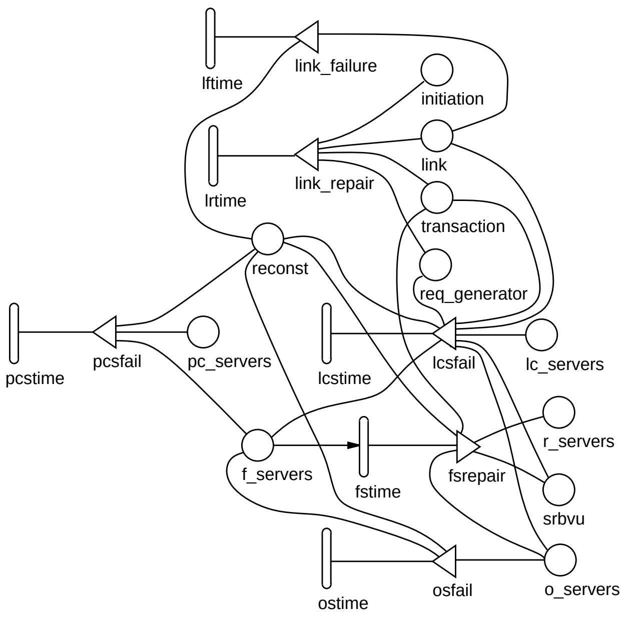

Figure 2: SAN Model: *network*

Four different failure situations need to be considered: failure of a physically current server, failure of a logically current server, failure of an outdated server, and failure of the network. The failure of servers and the network is modeled using input gates and timed activities. In particular, activity *pcstime* and input gate *pcsfail* represent the failures of physically current servers. Input gate *pcsfail* specifies the enabling condition for the timed activity *pcstime*. In particular, as shown in table 1, activity *pcstime* is enabled when there is at least one physically current server and the system is not in reconstruction. All timed activities are assigned exponential rates with per hour units.

Activity *lcstime* and input gate *lcsfail* represent the failures of an logically current server in a manner similar to that of physically current servers. However, since we allow servers to fail during the commit phase, these failures must be handled in a manner somewhat different than the failure of physically current servers. This complication need not be considered for physically current servers, since all physically current servers become logically current at the beginning of the commit phase (they have not yet been made physically current by the current write transaction), and become physically current at the end of the commit phase, since the network has broadcast capability. Servers can fail during the commit phase, however, and if they do, cannot be made physically current. Input gate *lcsfail* thus specifies that the failure of all remaining logically current servers during the commit phase ends the phase by setting marking of place *transaction* to zero. If the network is available, another request is now allowed, and a token is placed in place *req_generator* to indicate that it should be generated (see the SAN workload representation discussed in the next section).

The failure of the network is represented by activity *lftime* and input gate *link_failure*. As specified in table 2, we suppose that network failures are exponentially distributed with rate of 0.002 per hour. Failure of the network will block the completion of service of an ongoing transaction, and hence negatively impact its response time.

Server and network repairs begin as soon the respective component has failed, and are modeled by activities *fstime* and *lrtime*, respectively. Output gate *fsrepair* specifies what happens when a server is repaired (upon completion of activity *fstime*). Specifically, if a server is repaired during the reconstruction phase (i.e., the marking of place *reconst* is one), it joins the set repaired servers that need to be made physically current (i.e., the marking of place *r_servers* is incremented). If instead, the server is repaired during the commit phase then it is temporarily held in the place *srbvu* so that it can be distinguished from the available servers which formed the quorum in the voting phase, and hence are not made physically current. Finally, if the server was repaired when the system was not in the reconstruction or commit phase, it joins the set of outdated servers (marking of *o_servers* is incremented).

Activity *lrtime* specifies that the time to repair of the network is exponentially distributed, and has a repair rate of 2 per hour. Input gate *link_repair* specifies what happens when the network is repaired. If a

Table 1: Gate Definitions for SAN Model *network*

Input Gate: *pcsfail* Predcate: *MARK(reconst)==0 && MARK(pc_servers)>0* Function: *MARK(pc_servers)− =1; MARK(f_servers)+ =1;*
Input Gate: *link_failure* Predicate: *MARK(link)==1 && MARK(reconst)==0* Function: *MARK(link)=0;*
Input Gate: *link_repair* Predicate: *MARK(link)==0* Function: *MARK(link)=1;* *if(MARK(initiation)==0 && MARK(req_generator)==0* *&& MARK(transaction)==0) MARK(req_generator)=1;*
Input Gate: *osfail* Predicate: *MARK(reconst)==0 && MARK(o_servers)>0* Function: *MARK(o_servers)− =1; MARK(f_servers)+ =1;*

Input Gate: *lcsfail*
Predicate:
MARK(reconst)==0 && MARK(lc_servers)>0
Function:
MARK(lc_servers)− =1;
if(MARK(transaction)==1 && MARK(lc_servers)==0) {
   MARK(o_servers) + = (MARK(srbvu) + 1);
   MARK(srbvu) = 0; MARK(transaction) = 0;
   if(MARK(link)==1) MARK(req_generator)=1; }
else MARK(f_servers)+ =1;

Output Gate: *fsrepair*
Function:
if(MARK(reconst)==1)
   MARK(r_servers)+ =1;
else if(MARK(transaction) == 1)
   MARK(srbvu)+ =1;
else MARK(o_servers)+ =1;

Table 2: Activity Time Distributions for SAN Model *network*

Activity	*Dist*	*Rate*
pcstime	expon	*0.01*MARK(pc_servers)*
fstime	expon	*1*MARK(f_servers)*
lftime	expon	*0.002*
lrtime	expon	*2*
ostime	expon	*0.01*MARK(o_servers)*
lcstime	expon	*0.01*MARK(lc_servers)*

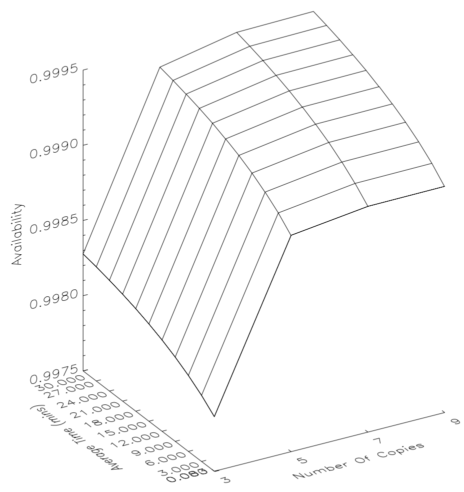

Figure 3: Effect of increasing the number of copies and decreasing workload on availability using static voting

transaction was blocked due to the failed link, it can now continue.

4 Results

Given the SAN model described in the previous section, the stochastic process representations for the static and dynamic models are automatically constructed using *UltraSAN* [15]. The construction procedure results in a continuous-time, discrete-state ergodic Markov process for each set of input parameter values. Depending on the number of servers considered, the state space sizes for static and dynamic models range from 66 (3 copies) to 375 (9 copies) states and 120 (3 copies) to 2640 (9 copies) states, respectively. The resulting Markov processes can then be solved (via *UltraSAN*), using successive over-relaxation. Solution time is quick, on the order of a few seconds, so we can evaluate the algorithm for a wide variety of parameter values. In particular, we determine the availability and mean response time of the system to write requests when:

1. The mean time between the completion of one request and the arrival of the next takes on values from .0833 to 30 minutes (denoted as "Average Time" in Figure 3).

2. The number of copies (i.e., servers) takes odd values from 3 to 9.

Availability can be determined directly from the model by measuring the fraction of time the system is not in reconstruction. Mean response time is determined by calculating the average number of requests in the system, the average arrival rate of requests to the system, and applying Little's result. Since at most one request can be in the system, the average number of requests in the system is equivalent to the probability that there is one request in the system. The rest of the section presents the results for availability and mean response time of replicated file system with static and dynamic voting.

4.1 Static Voting

The graph in Figure 3 depicts the effect of the change in workload and number of copies on availability. It shows that either an increase in the number of copies, decrease in workload, or both increases availability.

We consider first the increase in availability as the number of copies is increased for a fixed workload. This trend is well understood, and has been reported in the context of many voting algorithms. To understand it in terms of the SAN model consider the different structural configurations in which the system can be. By *configuration*, we mean a distribution of servers among physically current, logically current, outdated, and failed classes. Each configuration can be categorized as proper or improper, depending on whether a quorum can be formed in the configuration. Since each server carries a single copy of the replicated file, an increase in the number of copies implies an increase in the number of servers. Each additional server demands more server failures before the system enters an improper configuration. As shown by the model, the number of proper configurations increases and the probability of an incoming request finding the system in a proper configuration increases. Availability thus increases with an increase in the number of copies.

Looking at the graph, one can see that the increase in availability is significant when the number of copies is increased from 3 to 5, but becomes negligible when the number of copies is increased more than 5. Specifically, for a mean time of 0.0833 minutes between the completion of one request and the arrival of the next, the increase in availability is 1.08×10^{-3}, when the number of copies are increased from 3 to 5, while the increase in availability is in the order of 10^{-5}, when the number of copies is increased from 5 to 7 or 7 to 9.

The increase in availability with a decrease in workload is not so well known, or intuitive. Recall that the system is available whenever it is not in reconstruction. A decrease in workload could thus, in principle, push the availability of the system in two different directions. On one hand, since the system is forced into reconstruction only if a request arrives that cannot be serviced, a decrease in rate of requests would imply

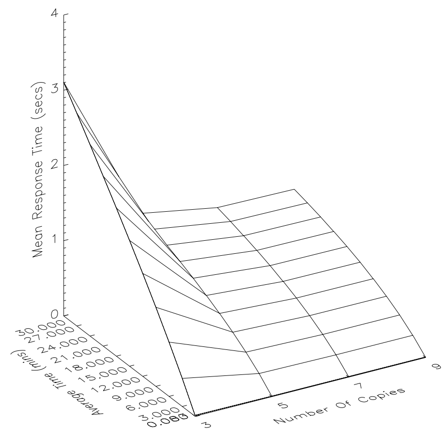

Figure 4: Effect of increasing the number of copies and decreasing workload on response time using static voting

that the rate at which events occur which can *possibly* cause the system to go into reconstruction decreases, hence availability might increase. On the other hand, a longer time between requests implies it is more likely that a server fails and is repaired between requests leaving it in an outdated state, and hence unable to help in forming a quorum. Thus a decrease in workload implies that the probability of entering reconstruction on a per arrival basis increases. The "possibility" of reconstruction becomes more probable for each request that occurs, thus availability might decrease with decreasing workload.

Which trend dominates depends on the particular workload, failure rates and repair rates considered. For the range of model parameters considered, the first trend dominates and, as seen in the graph of Figure 3, availability increases with decreasing workload. Specifically, for the failure and repair rates considered, the increase in availability is on the order of 10^{-4} over the range of workload studied. Furthermore, the increase in availability with the increase in time between requests is slightly more significant for higher numbers of copies. In particular, for an increase in the mean time between the completion of one request and the arrival of next from 0.0833 to 30 minutes, the increase in availability for the number of copies equal to 3, 5, 7, and 9 is 4.15×10^{-4}, 4.51×10^{-4}, 4.90×10^{-4}, and 4.96×10^{-4}, respectively.

The graph in Figure 4 illustrates that the mean response time decreases with an increase in the number of copies, but increases with a decrease in workload.

Figure 5: Distribution of the probability of being in structural configurations with different conditional mean response times

Recall that the mean response time is defined to be the average time it takes the system to respond to a write request, including the time waiting for the system to become available, if forced into reconstruction.

The decrease in response time with an increasing number of copies can be attributed to the same mechanism that caused an increase in availability with an increase in the number of copies. Simply put, an increase in the number of copies increases the probability of being in proper configurations, and since proper configurations have a much shorter conditional mean response time (on the order of milliseconds as compared to thousands of seconds, if reconstruction must be invoked), this increase results in a decrease in mean response time with an increasing number of copies. Note that this argument relies on the fact that the expected response time in proper configurations does not increase with an increasing number of copies. This is true for our scenario (since updates can be done via a broadcast), but would not be true if updates must be done in a serial fashion on the network. In this case, depending on values given to model parameters, mean response time might increase with an increasing number of copies. Further work must be done to determine exactly when this will be the case.

As was the case with availability, the decrease in mean response time is greatest when the number of copies is increased from 3 to 5 and is still considerable when the number of copies is increased from 5 to 7. It becomes negligible when the number of copies is decreased from 7 to 9. Specifically, for an average time of 3.0 minutes between the completion of one request and arrival of the next, the decrease in the response time for an increase in the number of copies from 3 to 5, 5 to 7, and 7 to 9 is 0.195 seconds , 0.012 seconds, and 4.82×10^{-4} seconds respectively.

The graph in Figure 4 also illustrates the non-intuitive result that for a fixed number of copies the average response time increases with a decrease in workload. This fact can be understood by considering the second trend discussed when considering the effect of workload on availability. The probability an incoming request finds the system in an improper configuration increases with decreasing workload. In particular, recall that as the mean time between the completion of one request and the arrival of another increases,

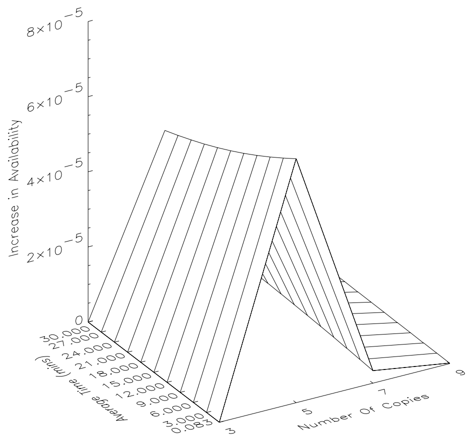

Figure 6: Increase in availability when the static voting is replaced by dynamic voting

the probability that a host will become outdated, by failing and being repaired increases. Higher probabilities of configurations with more outdated hosts lead to higher probabilities of improper configurations. Requests that arrive when the system is in an improper configuration take much longer to be serviced, since the system must first undergo reconstruction.

This relationship can be seen quantitatively by looking at the probability of seeing different expected response times conditioned on being in particular structural configurations. Figure 5 shows this for a system with 3 replicated copies. On this histogram, the parameter α indicates a specific workload (mean time between completion of one request and an arrival of the next), and the β indicate possible expected response times, conditioned on being in particular structural configurations. Note that it is easy to calculate an expected response time conditioned on being in a structural configuration, since the configuration defines precisely the set of steps the transaction must take to be serviced. Since the expected time to complete each of these steps is known, the total expected response time conditioned on being in each structural state can be calculated. $\beta 1$ and $\beta 2$ correspond to configurations where service is proper, while $\beta 3$-$\beta 6$ correspond to configurations that are improper. The graph shows that the probabilities associated with improper configurations increase slightly with decreasing workload. But even this slight increase in probability significantly effects the (unconditioned) expected response time, since the conditional expected response times within these configurations are so large.

4.2 Dynamic Voting

The results obtained for dynamic voting suggest that availability and mean response time increases either with an increase in the number of copies or a decrease in the workload. These trends are similar to the one we have discussed for static voting and also, stem from the same reasons. However, compared to the results from static voting, the results for dynamic voting show a higher availability and a lower mean response time for a given workload and a fixed number of copies. The reason for higher availability with dynamic voting has already been explained in many previous papers. Due to lack of space, we avoid to elaborate here.

The graph in figure 7 shows a decrease in mean response time when the static voting algorithm is replaced by the dynamic voting algorithm in a replicated file system. This behavior can be explained by realizing that for a given number of copies, certain configurations which are improper for the static voting algorithm become proper for the dynamic voting algorithm. We name such configurations *dual-behavior* configurations. The static voting invokes reconstruction when an incoming request finds the system in any of the dual-behavior configurations. In contrast the dynamic voting does not require reconstruction for dual-behavior configurations. Resultingly, as the graph shows, the average response time decreases for number of copies equal to 5, 7, or 9 when the static voting is replaced by the dynamic voting. However, for the number of copies equal to 3, the number of proper configurations remains the same for both static and dynamic voting algorithms. Therefore, there is no change in the response time.

Furthermore, by looking more carefully at the graph of figure 7, one notices that the maximum decrease in the response time is for the number of copies equal to 5. This behavior can be understood by realizing that for higher number of copies the probability of finding the system by an incoming request in any of the dual-behavior configurations decreases since the system requires more failures to go into a dual-behavior configuration. Hence, the decrease in response time is smaller for high numbers of copies.

5 Conclusion

As stated in the introduction, the results of this paper are significant for two reasons. First, they illustrate that it is indeed possible to construct analytic models of voting protocols that account for the effect of workload and protocol parameters values on performance and availability. Second, they provide interesting results regarding the effect of workload on

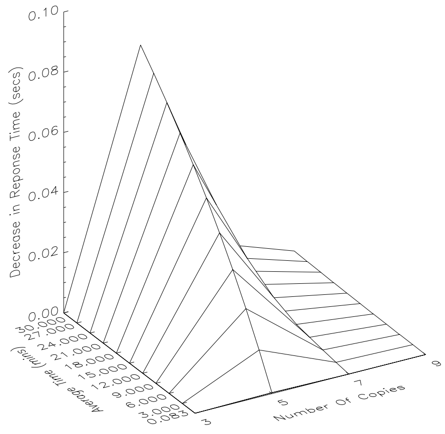

Figure 7: Decrease in response time when the static voting is replaced by dynamic voting

the performance and availability of the particular voting algorithm studied. Regarding the algorithm itself, the results obtained for availability were as expected. The availability increases either with an increase in the number of copies or a decrease in the workload or both. However, the results obtained for average response time were surprising. The average response time decreases with an increase in the number of copies while it increases with a decrease in the workload.

While the results are interesting in their own right, they also suggest further work could be done. For example, factors other than workload and, the number of copies may also affect a voting algorithm's performance and availability. If communication on the network is non-broadcast, then an important factor would be the number of physical updates performed within the commit phase. Seemingly, the availability would increase with increasing numbers of updates, but the behavior of mean response time is not intuitive. Interesting performance/availability tradeoffs may exist that could be investigated using a model similar to the one presented in this paper. Finally, the results suggest that comparisons could be made between different network architectures and voting schemes, taking into account the effect of workload on performance and availability.

References

[1] R. H. Thomas, "A majority consensus approach to concurrency control for replicated databases," *ACM Transactions on Database Systems*, vol. 4, no. 2, pp. 180–209, 1979.

[2] D. Agarwal and A. E. Abbadi, "The generalized tree protocol: an efficient approach for managing replicated data," *ACM Transactions on Database Systems*, vol. 17, no. 4, pp. 689–717, 1992.

[3] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Optimizing vote and quorum assignments for reading and writing replicated data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 3, pp. 387–397, 1989.

[4] J. Paris, "Voting with witnesses: a consistency scheme for replicated files," *Proceedings of 6th Int. Conf. on Data Engineering, Cambridge, MA*, pp. 606–620, 1986.

[5] D. Barbara, H. Garcia-Molina, and A. Spauster, "Increasing availability under mutual exclusion constraints with dynamic vote reassignment," *ACM Transactions on Computer Systems*, vol. 7, no. 4, pp. 394–426, 1989.

[6] M. Herlihy, "Dynamic quorum adjustment for partitioned data," *ACM Transactions on Database Systems*, vol. 12, no. 2, pp. 170–194, 1987.

[7] S. Jajodia and D. Mutchler, "Dynamic voting algorithms for maintaining the consistency of a replicated database," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 230–280, 1990.

[8] R. K. Iyer, D. J. Rossetti, and M. C. Hseuh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Transactions on Computer Systems*, vol. 4, no. 3, pp. 214–237, 1986.

[9] J. F. Meyer and L. Wei, "Analysis of workload influence on dependability," *Proceedings of 18th IEEE Int. Symposium on Fault Tolerant Computing, Tokyo*, pp. 84–88, 1988.

[10] M. Ahamad and M. H. Ammar, "Performance characterization of quorum-consensus algorithms for replicated data," *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 492–496, 1989.

[11] S. Y. Cheung, M. H. Ammar, and M. Ahamad, "The grig protocol: A high performance scheme for maintaining replicated data," *Proceedings of 6th Int. Conf. on Data Engineering, Cambridge, MA*, pp. 438–445, 1990.

[12] A. Kumar, "Performance analysis of a hierarchical quorum consensus algorithm for replicated objects," *Proceedings of 10th IEEE Int. Conf. on Distributed Computing Systems*, pp. 378–385, 1990.

[13] J. B. Dugan and G. Ciardo, "Stochastic Petri net analysis of a replicated file system," *IEEE Transactions on Software Engineering*, vol. 15, no. 4, pp. 394–401, 1989.

[14] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application," *Proceedings of International Workshop on Timed Petri Nets, Torino, Italy*, pp. 106–115, 1985.

[15] W. H. Sanders, W. D. Obal, M. A. Qureshi, and F. K. Widjanarko, "*UltraSAN* modeling environment," *Accepted for publication, Performance Evaluation Journal, Special issue on Performance Evaluation Tools*, 1994.

[16] M. A. Qureshi and W. H. Sanders, "The effect of workload on the performance and availability of static and dynamic voting algorithms," *Technical Report, Center of Reliable and High-Performance Computing, University of Illinois, Urbana-Champaign*, 1994.