The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data*

Shun Yan Cheung *Mostafa H. Ammar* *Mustaque Ahamad*

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

We present a new protocol for maintaining replicated data that can provide both high data availability and low response time. Existing protocols are designed primarily to achieve high availability by updating a large fraction of the copies which provides some (although not significant) load sharing. In the new protocol, transaction processing is shared effectively among nodes storing copies of the data and both the response time experienced by transactions and the system throughput are improved significantly. We present an analysis of the availability of the new protocol and use simulation to study the effect of load sharing on the response time of transactions. We also compare the new protocol with a voting based scheme.

1 Introduction

A distributed system can replicate data to improve availability and performance. Replication introduces the problem of maintaining consistency among the different copies of the same data. Many replica control protocols have been proposed with the chief design goal being to maximize data availability. Such protocols generally involve a relatively large fraction of nodes in the execution of an operation. As a result, the number of requests that arrive at a particular node is not much smaller than the total requests arriving to the entire distributed system. Thus, the response time is not improved significantly because the load is not shared between the nodes storing the data replicas. In this paper, we propose a new replica control method for maintaining replicated data for reading and writing that can achieve both increased system performance by sharing load and high data availability. The proposed method involves a smaller number of nodes and has low response time.

A popular method for maintaining consistency of replicated data is weighted voting [1] which is a generalization of the majority consensus method presented in [2]. Each replica is assigned a number of votes and read and write operations need to obtain a predefined quorum of r and w votes respectively. To ensure correctness, typically $r + w$ is set to one more than the total number of votes. Performance and availability of data for operations of a type (read or write) are enhanced by the use of a low quorum value. However, the performance and availability of the other operation is reduced accordingly because a higher quorum value must be used. Dynamic schemes derived from voting, as those presented in [3, 4, 5, 6, 7, 8], can achieve very high data availability but the operational cost of these protocols may be of the same order as voting. The methods presented in [9] and [10] were designed to reduce storage requirement in voting, but they compromise some data availability. Other techniques such as available copies [11] have a high overhead for write operations.

Performance analysis of voting is especially difficult because of the large number of vote and quorum assignments. In [12], Barbara and Garcia-Molina analyzed the availability of voting for mutual exclusion. In a related work [13], the authors studied unweighted voting for reading and writing. To analyze voting in general, the authors presented in [14] an enumeration method for all vote and quorum assignments which can be used in finding the optimal assignment. A direct method for finding the optimal vote assignment for mutual exclusion is presented in [15].

In [16] a scheme is described that achieves mutual exclusion in a distributed system using only $O(\sqrt{N})$ messages. Our new method uses a similar technique to execute both read and write operations and it achieves higher load sharing than existing methods. Although the primary goal in the design of replica control protocols is to increase data availability, in this work, we investigate how load sharing in a replicated data system can be used to enhance performance. In particular, we aim to reduce the response time experienced by transactions arriving to the system. A transaction makes a number of read and write access requests to nodes and the response time consists of queueing and service delays at nodes which depend on the arrival rate of requests into the nodes. Queueing delays can be reduced when load is shared.

Our protocol is designed to increase load sharing among

*This work was supported in part by NSF grants NCR-8604850 and CCR-8806358.

the nodes storing replicas and still maintain high availability. However, our protocol needs to replicate data at a higher number of nodes as compared to voting for achieving a certain level of availability. Notice that load sharing cannot be increased for *both* types of operations when voting is used because higher degree of replication requires that at least one of the operations be executed by more nodes. We also show how the protocol can be efficiently implemented in a broadcast bus local area network (LAN) environment by exploiting the support for multicast communication in such networks.

The paper is organized as follows. In Section 2 we describe the grid protocol for synchronizing reading and writing of replicated data. We also present an implementation using multicast addressing in a broadcast network and an analysis of the data availability. Section 3 discusses load sharing and in Section 4, we present numerical examples and compare our method with voting. We conclude the paper in Section 5.

2 The Grid Protocol

We consider a distributed system in which data is fully replicated and we allow two types of operations on the replicated data, namely read and write. A read operation returns some value and a write operation installs a new value. Proper synchronization is achieved if read operations return the value installed by the last write operation, and read and write operations and two write operations are not executed concurrently. Each node uses a centralized concurrency control scheme, such as two-phase locking or timestamp ordering, to synchronize accesses to its local copy and a replica control protocol to coordinate accesses to the various replicas.

The basic principle in maintaining consistency of replicated data is to require conflicting operations to lock at least one common copy. A read and write group are *minimal* groups of copies that must be locked for a read and write operation to proceed to completion, respectively. A read and write *coterie* are the sets of all read and write groups, respectively. (See [17] and [14] for similar definitions.) The conditions given above for proper synchronization are satisfied if each read and write group and two write groups have a non-empty intersection.

There are many ways to realize the intersection property of read and write groups and voting is the best known example. Garcia-Molina and Barbara showed in [17] that voting cannot be used to provide all possible read/write coteries. However, voting has the advantage of being easily implementable. We will describe a coterie based replica control protocol that can be efficiently implemented using multicast. The nodes that store copies of data are logically arranged in a grid topology and read and write operations are required to lock rows and columns of nodes such that conflicting operations request locks from a common node. A read coterie used in our method contains groups of nodes containing exactly one node in each column and the write coterie consists of groups that contain nodes of a read group and all nodes of a column. The grid topology is a conceptual tool used to describe our method. Later in this section, we briefly discuss an efficient implementation of our method using multicast addressing in broadcast bus networks. Notice that such a logical arrangement is also used in [16] to achieve mutual exclusion in distributed systems but fault tolerance and availability issues were not addressed.

2.1 Protocol Description

We consider a set of nodes that store data replicas which are *logically* arranged into a grid network (see Figure 1). We assume version numbers are used to identify the current copy of data. The number of nodes in a row and a column of the grid are M and N respectively and we assume that node failures are fail-stop.

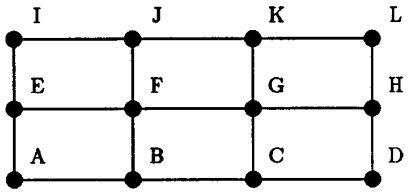

Figure 1: A Grid Network

We number the rows and columns of the grid network as 1, 2, .., M and 1, 2, .., N, respectively, and each node knows its position in the grid. We define the operation R-cast(i, (*replyset*,*msg*)) as the sending of a multicast message (*replyset*,*msg*) to nodes in row i, where *msg* is the message content and *replyset* is the set of nodes (subset of nodes in row i) from which a reply is requested. Similarly, C-cast(j, (*replyset*,*msg*)) is a multicast message that is sent to nodes in column j where *replyset* and *msg* are the same as defined in the R-cast operation. Multicast communication mechanisms that allow a message to be delivered to a group of nodes are supported by many distributed operating systems. It will be seen later that the delivery of a multicast message need not be reliable.

A set G of nodes is a *C-cover* if each column intersects with G, e.g., in Figure 1, the set {A, B, G, H} is a C-cover. Synchronization requirements given in the last subsection, can be achieved by locking a C-cover when reading and a C-cover and all nodes in a column when writing. (Read locks are sharable and write locks are exclusive.) The details of the algorithms for reading and writing are presented in Figures 2 and 3, respectively. In the protocol executed by a read operation a random permutation π_R of 1, 2, ..., M is selected. The randomization is intended to distribute the load evenly over the nodes. The permutation π_R is the order in which the rows are interrogated to read-lock a C-cover of nodes. To reduce the num-

```
π_R := a random permutation of 1, 2, ..., M;
COLUMN-COVER := φ;
i := 0;
do
   i := i + 1;
   REQUEST-RESPONSE := {nodes in row π_R[i] for
                        which column ∉ COLUMN-COVER};
   R-cast(π_R[i], (REQUEST-RESPONSE,R-Lock));
   receive RESPONSES; (* until timeout *)
   if (RESPONSES contains an abort message) then
      Abort;
   if i = 1 then
      COLUMNS := RESPONSES;
      (* Record responses for writing *)
   COLUMN-COVER := COLUMN-COVER ∪
                   {columns of nodes in RESPONSES};
until (COLUMN-COVER = {1,2,...,N}) or (i = M)

if (a C-cover is obtained) then
   return the most recent value in the C-cover
else
   Abort; (* System is unavailable *)
```

Figure 2: Protocol for Read

ber of messages used, the multicast operation includes a REQUEST-RESPONSE set of nodes indicating that only nodes in this set are to respond. The REQUEST-RESPONSE set is computed from the COLUMN-COVER variable which contains the columns that have some node locked. A multicast message is sent to the next row of nodes and responses are collected in the variable RESPONSES. If a C-cover is found, i.e., when COLUMN-COVER is equal to {1, 2, ..., N}, the value associated with the latest version at the nodes in the C-cover is returned and the operation completes successfully. If an abort message is received or no C-cover can be found after searching all rows of the grid network then the operation is aborted. A node will respond with an abort message if it detects an error condition in the concurrency control procedure. For instance, if the concurrency control method used is locking, a node will send an abort message if deadlock is detected. The information recorded in the variable COLUMNS is not used by the read protocol.

The write protocol is required to synchronize a write operation with both read operations and other write operations. Synchronization with read operations is accomplished by locking a column and with write operations by locking a C-cover. First a C-cover is locked using the read protocol and the columns that responded to the *first* multicast message are recorded in a variable COLUMNS. The purpose of this variable is to increase efficiency in the remaining part of the write protocol as we use this information to exclude columns with failed nodes. If the data have been locked for reading, this step of the write protocol can be omitted. When a C-cover is locked successfully, the execution proceeds to lock all nodes in a column in the order given by the random permutation π_C of COLUMNS. If all nodes in some column are locked, the execution can terminate successfully by updating copies in that column by using a distributed commit protocol, otherwise, the operation aborts.

```
Using algorithm in Figure 2 lock a C-cover;

π_C := a random permutation of COLUMNS;
j := 0;
do
   j := j + 1;
   REQUEST-RESPONSE := {nodes in column π_C[j]}
   C-cast(π_C[j], (REQUEST-RESPONSE,W-Lock));
   receive RESPONSES; (* until timeout *)
   if (RESPONSES contains an abort message) then
      Abort;
until (RESPONSES = {all nodes in some column })
      or (j = |COLUMNS|)
if (RESPONSES = {nodes in some column }) then
   update copies in the complete column
else
   Abort; (* System is unavailable *)
```

Figure 3: Protocol for Write

The protocol executed by a read operation is less involved than the one by a write operation because we believe reading will be predominant. Also, the data is more available for reading. In general, the grid protocol uses different read and write coteries than voting and in most grid networks, the read and write coteries used cannot be obtained from vote assignments. There are two limiting cases of grid networks that have voting equivalents. When the grid network consists of a single row of nodes, the grid protocol is equivalent to voting using read all/write one quorum. The other case is a grid network consisting of a single column which corresponds to voting using read one/write all quorum.

The following property assures that read and write operations and two write operations are properly synchronized.

Property:

Two write operations cannot be executing concurrently. Furthermore, concurrent execution of read and write operations is not possible and a read operation will return the value installed by the last write operation.

Proof:

When two write operations are executing concurrently, each locks a C-cover and all nodes in a column. If the two operations have each locked a C-cover, neither can obtain write-locks from all the nodes in any column. This is due to the fact that no node in the C-cover locked by the other

operation can respond positively. Hence concurrent writes are not possible. A similar argument can be used to show that concurrent execution of read and write operations is not possible. Also, since a C-cover contains at least one copy of each column, it must contain a value written by the last write operation because it updates copies at all nodes in the column.□

2.2 Implementation

In a broadcast bus LAN (e.g., Ethernet), each node is connected to the bus via a node interface and each interface can recognize its own address and a number of multicast addresses. Node interfaces that recognize a common multicast address form a multicast group. When the network interface hears a transmission on the bus and recognizes the address, the packet is delivered to the node that is connected to the interface.

The grid protocol can be implemented by having each row and column of nodes in the grid form a multicast group. Thus, each node belongs to two multicast groups corresponding to its row and column. Also, only nodes that store a copy need to be in the grid but other nodes not in the grid can still read and write data. We can implement the grid by using the multicast addresses $1, 2, \ldots, M$ and $M+1, M+2, \ldots, M+N$ for rows $1, 2, \ldots, M$ and columns $1, 2, \ldots, N$, respectively.

2.3 Availability Analysis

We now present an availability analysis of the protocol presented in the previous section. Let M and N be the number of nodes in each row and column respectively. We will assume node failures to be independent and all nodes are identical. Let p be the availability of a node, i.e., p is the steady state probability of a node being operational. The availability of an operation is the probability that the system is in a state that would allow the operation to succeed assuming the state does not change while the operation is in progress.

A column is *covered* if some node in that column has been successfully locked. Let $\alpha^R_{m,n}$ be the probability of locking a C-cover when n columns are uncovered and m rows are unsearched. Thus, $\alpha^R_{M,N}$ is the availability of read operations. We can compute $\alpha^R_{m,n}$ by using a recurrence relation. Let ℓ be the number of operational nodes that respond to a request for a C-cover which is sent to the subsequent row. Clearly, ℓ is a binomially distributed random variable with parameters n and p and hence the likelihood of this event is,

$$\binom{n}{\ell} p^\ell (1-p)^{n-\ell}$$

Now consider the remaining unsearched grid: the number of uncovered columns is $n-\ell$ and the number of unsearched rows is $m-1$. The data is available for reading if the remaining $n-\ell$ columns can be covered by searching in the remaining $m-1$ rows. The probability of success is equal to $\alpha^R_{m-1,n-\ell}$ and by the law of total probability, we have the relation:

$$\alpha^R_{m,n} = \sum_{\ell=0}^{n} \binom{n}{\ell} p^\ell (1-p)^{n-\ell} \alpha^R_{m-1,n-\ell} \quad (1)$$

for $m = 2, 3, \ldots, M$ and $n = 1, 2, \ldots, N$. We define $\alpha^R_{m,0} = 1$ for $m = 1, 2, \ldots, M$. To lock a C-cover for n columns in one row, all n nodes must be operational and therefore,

$$\alpha^R_{1,n} = p^n, \qquad \text{for } n = 1, 2, \ldots, N \quad (2)$$

Let $\alpha^W_{M,n}$ be the probability of locking a C-cover and a column when n columns are unsearched, then $\alpha^W_{M,N}$ is the write availability. Using a technique similar to one above (see [18]) we find,

$$\alpha^W_{M,n} = p^M \sum_{\ell=0}^{n-1} (1 - p^M - (1-p)^M)^\ell \alpha^R_{M,n-\ell-1} \quad (3)$$

where we define $\alpha^R_{M,0} = 1$.

Iterative algorithms have been implemented that make use of Equations (1) and (3) to evaluate read and write availability.

3 Sharing Load

An advantage of replicating resources is reduced response time for requests due to load sharing. In a distributed system where data is replicated, a user request can be processed by only a subset of nodes and all nodes do not need to process the request. When only a small number of nodes need to process a request, the effect of sharing is a reduced arrival rate to a particular node which can have a significant impact on delay. We must distinguish carefully between the *external* rate of request arrivals and the rate of arrival to a *particular* node which we will call the *effective arrival rate*. An external arrival will trigger simultaneous arrivals to a subset of nodes and the effective arrival rate will depend on the external rate and the manner in which the subset of nodes is chosen. For instance, if an external arrival would trigger arrivals to half the nodes chosen at random as it might be done in a voting scheme using majority quorum, the effective arrival rate to a particular node would be half the external rate of arrival and the load reduction is about one half. In general, the smaller the fraction of nodes used, the lower the effective arrival rate and better the response time.

Consider a voting based system consisting of K nodes, each allocated one vote, that uses the read and write quorum of r and w votes ($r + w = K + 1$), respectively. The load reduction for reading and writing are approximately $\frac{r}{K}$ and $\frac{w}{K}$, respectively (exact if nodes are reliable) if the nodes used for reading and writing are chosen at random. Assuming that reading is predominant, reduction of read request arrivals would be maximized using the lowest read quorum $r = 1$, however, the write quorum must be set

to K. The resulting system will have poor availability for write operations which may be unacceptable. To achieve good write availability, the read and write quorums are set to approximately half the number of copies and the resulting reduction factor is about half. Notice that increasing the degree of replication will not significantly reduce the load because the quorums must be increased accordingly. Furthermore, a problem that arises with a large number of replicas is communication cost. Since $r + w = K + 1$, either the read or write protocol in voting will use a large number of messages and this can cause network congestion which can result in long delay.

The load reduction achieved in the scheme proposed by this paper depends on the topology of the grid and it is approximately equal to $\frac{1}{M}$ and $\frac{M+N}{MN}$ for reading and writing, respectively (it is exact if nodes are reliable). For $M \approx N$, *both* load reduction for read and write operations are of the order $\frac{1}{\sqrt{K}}$ ($K = MN$) which can be reduced by increasing the degree of replication. Unlike voting, where $r + w = K + 1$, the number of nodes that must execute read and write operations is smaller than K which leads to low communication cost. Thus, by exploiting the logical structure, it is possible to reduce the cost of accessing replicated data.

To demonstrate how load reduction can decrease response time, we have written a detailed simulation for both a voting based system and the grid system. We use simulation because the relationship between load reduction and response time is very complex. The simulation model and results obtained from it are reported in the next section.

4 Numerical Examples

This section presents some numerical examples that compare our protocol and voting. The measures used are availability and response time. Let K be the number of nodes in the system, p be the reliability of the nodes, r be the read quorum and $w = K + 1 - r$ be the write quorum. Since all nodes have the same availability, the optimal vote assignment is one where each node is assigned the same number of votes and without loss of generality, we use the uniform vote assignment where each node is assigned one vote. Let $\alpha^V_{K,t}$ be the availability of operations that requires t votes using voting with K copies, we have that,

$$\alpha^V_{K,t} = \sum_{\ell=t}^{K} \binom{K}{\ell} p^\ell (1-p)^{K-\ell}$$

The read and write availability are respectively $\alpha^V_{K,r}$ and $\alpha^V_{K,w}$.

4.1 Availability

We will start by considering the data availability of the grid protocol described in Section 2. Table 1 shows the availability of a number of systems that use the grid protocol where each node is 0.95 available. The number of nodes in the system ranges from 9 to 36 in a 3x3 and a 6x6 grid respectively. The value for M is fixed in each row of the table while the value for N is increased from 3 to 6. M also ranges from 3 to 6. Each column is subdivided into 2 columns where the left and right columns contain the read and write unavailability values respectively.

Number of Rows	Number of Columns							
	3		4		5		6	
	read	*write*	*read*	*write*	*read*	*write*	*read*	*write*
3	374.95	3268.59	499.91	912.25	624.84	683.60	749.77	758.14
4	18.75	6400.56	25.00	1208.75	31.25	250.82	37.50	78.23
5	0.94	11577.66	1.25	2620.12	1.56	594.00	1.87	135.90
6	0.05	18590.32	0.06	4924.78	0.08	1304.67	0.09	345.69

Table 1: Grid Unavailability x10^{-6} with $p = 0.95$

The data availability for reading and writing are very high and it can be affected by altering the grid topology. In general, when the number of rows in the grid is increased while the number of columns remain fixed, the read availability increases while the write availability decreases. Increasing the number of columns while the number of rows are unchanged has the opposite effect. This gives us a degree of flexibility in designing for system performance that is similar to choosing the read and write quorum in voting.

In the design of a highly available and high performance system, a trade off must be made to obtain better performance for other measures. The dimensions of the grid will affect read and write availabilities and design decisions are made based on the mix of user transactions and tolerance levels set for read and write availability. Consider for instance two 30 node systems arranged into an 5x6 and 6x5 grid. We can see in Table 1 that the read availability of both systems is very high and may be assumed satisfactory. The write availability in the 5x6 grid is better than in the 6x5 grid. However, reading in the 6x5 grid uses on the average one less message (there is one less column) and the load reduction for reading is better (there is one more row). The 6x5 grid is therefore better suited when the proportion of read operations is very high and write availability is not critical. However, if write availability were of utmost importance we would decide to use the 5x6 grid in spite of read inefficiency.

We have compared the availability of the grid protocol with voting. As described earlier, we assume one vote is assigned to each node. Since the cost of reading increases when the read quorum is larger, we will use the lowest read quorum that satisfies predetermined availability requirements. We use nodes with 0.95 availability to design voting systems that achieves a predetermined read and write availability of $1 - 10^{-6}$ and 0.9955 respectively. The smallest voting system that achieves the minimum availability requirements is a 10 node system with read and write quorums of 4 and 7 respectively. Table 2 contains voting sys-

	$n = 10$		$n = 12$		$n = 14$		$n = 16$	
	$r = 4$	$w = 7$	$r = 4$	$w = 9$	$r = 4$	$w = 11$	$r = 5$	$w = 12$
α	0.08	1028.50	0.00	2236.40	0.00	4173.24	0.00	857.31
	$n = 18$		$n = 20$		$n = 22$		$n = 24$	
	$r = 5$	$w = 14$	$r = 5$	$w = 16$	$r = 5$	$w = 18$	$r = 6$	$w = 19$
α	0.00	1546.44	0.00	2573.94	0.00	4022.34	0.00	962.35
	$n = 26$		$n = 28$		$n = 30$		$n = 32$	
	$r = 6$	$w = 21$	$r = 6$	$w = 23$	$r = 6$	$w = 25$	$r = 7$	$w = 26$
α	0.00	1510.56	0.00	2268.70	0.00	3282.49	0.00	868.50

Table 2: Voting Unavailability x10^{-6} with $p = 0.95$

tems that uses the lowest read quorum to achieve the predefined availability requirements. The number of nodes ranges from 10 to 32 with an increment of 2 nodes.

To satisfy the given availability requirement above, a system using the grid protocol needs 30 nodes arranged into a 6x5 grid. As we discuss in the next subsection, the 30 node grid system has much lower response time and higher capacity than the 10 node voting system.

4.2 Response Time

The advantage of the grid system is lower response time due to higher level of load sharing. A read request can be processed by any one row of nodes in the grid and a write request by any row and column. The arrival rate of requests made to individual nodes is reduced resulting in lower response time and higher system capacity. Increasing the number of nodes in voting does not result in higher load sharing as in the grid system. This is due to the fact that the write quorum must be increased accordingly. For instance, a thirty copy voting system that satisfies the predetermined availability requirements given above uses read and write quorum equal to 6 and 25 respectively.

To show the effect of load reduction on the response time of the transactions, we have performed a simulation study. The model uses similar ideas to the one presented in [19] for a centralized computing system and is intended to model both a voting-based system and one that uses our grid protocol. We have not modeled network delay in the simulation because we are interested primarily in the reduction of response time through load sharing. The concurrency control method used is two-phase locking and if a lock request is denied, the transaction is blocked. The system maintains a global wait-for graph to detect deadlock and if a deadlock is discovered, the youngest transaction involved in the deadlock is aborted. The aborted transaction will release its locks and restart immediately making the same requests again. The commit protocol used is two-phased. The first phase directs the nodes in the write quorum to record the modified granules and the log. The second phase makes the modifications permanent.

Figure 4 shows the model of a node. It consists of two units, the concurrency control unit (CC) and the disk unit (Dsk). Each unit has a queue associated with it that contains arriving requests. Requests made by transactions arrives in the CC-queue and are serviced in a first come, first serve (FCFS) basis. The CC-unit checks if the request can be granted. If it can, the request is passed to the disk unit for processing and otherwise it is put in the block queue and the global wait-for graph is updated. Transactions are unblocked when the lock holder releases the lock as it commits or aborts. Priority for unblocking is given to write requests in FCFS order. The global wait-for graph is updated whenever a request is unblocked. The delay in the CC-unit is assumed to be negligible. The disk unit also uses the FCFS service discipline and services read, write and commit requests as follows. A read request transfers a granule from the disk to a private buffer in memory, a write request modifies the buffer and a commit request flushes the log pages and the dirty buffers to disk. The log is written serially and the processing time is assumed negligible compared to the random access delay for writing the buffers. The delays experienced by transactions are blocking delay and random access disk delay.

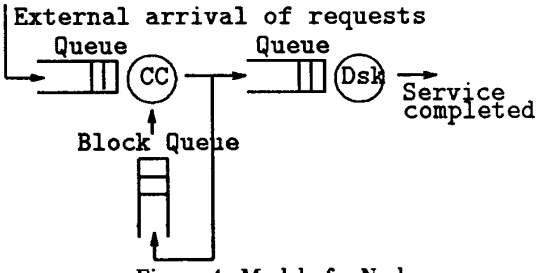

Figure 4: Model of a Node

In the simulation, the arrival of transactions is modeled by a Poisson process. Each transaction has a unique identification number which is assigned upon arrival and it is increasing in time. A transaction chooses a random number of different data granules to be read and with a probability of WRITEPROB the granule read will also be updated. The read requests are generated first and then the write requests are processed. A transaction submits its requests to the system one at a time. A request (other than the first) is generated only after the previous one has completed successfully. The transaction sends its request to several nodes simultaneously, choosing the nodes according to whether the voting or the grid protocol is being used. First the transaction attempts to read-lock the first data item at a read quorum set of nodes. The nodes are selected in an arbitrary manner to distribute load uniformly. It sends requests to the nodes and waits for replies. If the transaction is successful before the timeout period expires, the remaining data will be accessed using the same quorum set. Although we have not modeled node failures, in reality a transaction must recover by timeout. In the sim-

ulation, a transaction can timeout due to excessively long delays in congested nodes. If a transaction times out during the quorum collection phase, it will continue, if possible, to complete the required quorum. In the grid system, the procedure is described in Section 2. In voting, the transaction will send requests to a number of unsearched nodes that is equal to the number of votes to complete the read quorum. If the transaction is unable to lock a read quorum after searching all nodes, it aborts and restarts immediately. When the transaction restarts, the read quorum set is randomized again to avoid submitting requests to the same set of nodes. If the transaction has successfully locked a read quorum, the subsequent read requests are sent only to the nodes in the quorum. A transaction can also timeout during these requests and in this case, we abort the transaction. The transaction would restart again immediately.

After reading the data, the transaction may update a subset of the data read. Each granule is updated independently of other granules with a probability of WRITEPROB. Like reading, the transaction must find a write quorum when it submits the first write request. In a grid system, this procedure is given in Section 2. In a voting system the transaction will supplement its read quorum with the number of nodes that is equal to the difference between the write and the read quorum. The procedure to collect a write quorum is similar to the one for a read quorum. If after searching all nodes, the supplement set found is less than $w - r$ nodes or any of the nodes in the read quorum set has not responded positively, the transaction aborts and restarts immediately. Otherwise, the subsequent write requests are made to the write quorum set found and if the transaction would time out during these write requests, it is aborted and restarted immediately. The timeout period for read and write requests is RESPONSETIMEOUT. After making all write requests, the transaction will commit the updates using a two-phase protocol. The timeout value for the first commit phase is COMMITTIMEOUT and it is larger than RESPONSETIMEOUT. This is because the processing delay for commit is longer than for reading or writing. In this phase, the nodes in the write quorum record the log data and the updates made by the transaction on stable storage (disk). The transaction can terminate successfully if all nodes in the write quorum respond positively in the first phase. It will then send a commit message to all nodes that hold locks on behalf of the transaction. The transaction does not need to wait for responses from the nodes in the second phase as the nodes can determine the outcome of the commit decision among themselves. If a transaction would timeout during the first phase, it aborts and restarts immediately. Table 3 summarizes the system parameters.

The systems studied in the simulation are a 6x5 grid system and three voting systems, namely the systems of 10, 20 and 30 nodes in Table 2. These systems satisfy the requirements that read and write availability are at least

Parameter	Value
Number of data granules	10000
Minimum number of granules accessed	10
Maximum number of granules accessed	20
Access pattern	Uniform between min. and max. value
WRITEPROB	0.2
RESPONSETIMEOUT	10 sec
COMMITTIMEOUT	30 sec
Disk random access time	Uniform with 30 msec mean

Table 3: Simulation Parameters

$1 - 10^{-6}$ and 0.9955 respectively and the system availabilities are given in Table 4. The systems in the table *do not* achieve the maximum possible system availability, for instance, the maximum availability of a 30 node voting system is higher than that of a 10 node system, but in Table 4, the 30 node system is less available. The settings of the quorums in voting are such that they satisfy the minimum required read and write availability *and* maximize load sharing. The availability of the 30 node system can be increased, but the response time will also increase. We have also simulated a system that does not use data replication. This system does not meet the requirements and is included as a control experiment.

	Voting			*Grid*
f	*10 nodes*	*20 nodes*	*30 nodes*	*6x5 grid*
0.8	0.9998	0.9995	0.9993	0.9997

Table 4: System availabilities

Figure 5 shows the average response time of the transactions versus the average external arrival rate when the parameters are set as given in Table 3. The non-replicated system has lower response time for low arrival rates since a request is serviced as soon as the node finishes processing. In replicated systems, a request is serviced only if *all* nodes in the quorum finish processing. The non-replicated system has a lower system capacity (maximum throughput) than the other systems that use data replication.

We have compared the grid and voting systems. At low arrival rates, all these systems have similar response time values, but at higher rates, the performance of the grid system is much better than the voting systems. Also, the system capacity of the grid system is higher than all the voting systems studied. In the simulation, we find that a thirty node 6x5 grid system can handle about twice the offered load of a thirty node voting system using a read quorum of 6 and a write quorum of 25.

The above example shows the strength and weakness of both protocols. Voting achieves the required availability

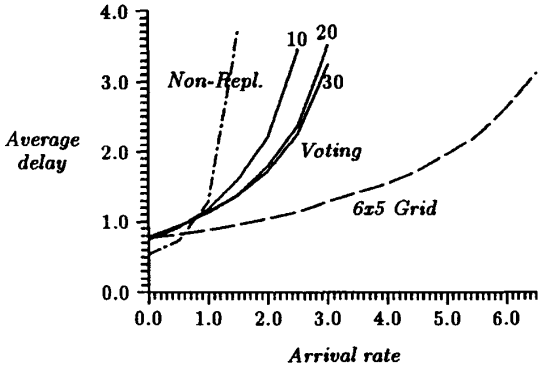

Figure 5: Average delay versus arrival rate

threshold with a relatively small number of copies but in order to do so, the read and write quorum must be set to about half the total number of copies. The likelihood that a node will be searched by a request is about $\frac{1}{2}$ and the effective arrival rate to any node is not reduced significantly. Therefore, voting achieves good data availability but provides little load sharing. The grid protocol has better load sharing and response time for higher arrival rates. In order to achieve these benefits, data is replicated over a larger number of nodes. Also, increasing the number of nodes in a voting system does not decrease response time significantly as would the use of a grid protocol.

5 Concluding Remarks

In this paper, we have presented a new data replication scheme in which the nodes storing copies of the data are arranged in a logical grid. The scheme is designed to perform effective load sharing among the nodes storing copies, as well as provide an increase in data availability over a single copy system.

The performance of the grid scheme was compared to that of a voting scheme. It was found that voting will achieve the same level of data availability using less copies. However, the load sharing feature of the grid protocol provides a significant decrease in the response time experienced by transactions using the system. It was also shown that a voting scheme cannot be made to achieve the level of load sharing of the grid protocol even by increasing the number of nodes.

References

[1] H. Gifford, "Weighted voting for replicated data," in *Proceedings of 7th Symposium on Operating Systems*, pp. 150 –162, ACM, 1979.

[2] R. H. Thomas, "A majority consensus approach to concurrency control for multiple copy databases," *ACM Transactions on Database Systems*, vol. 4, pp. 180–209, June 1979.

[3] D. Eager and K. Sevcik, "Achieving robustness in distributed database systems," *ACM Transactions on Database Systems*, vol. 8, no. 3, pp. 354–381, 1983.

[4] M. Herlihy, "Dynamic quorum adjustment for partitioned data," Tech. Rep. CMU-CS-86-147, Carnegie-Mellon University, Pittsburgh PA 15213., 1987.

[5] D. Davcev and W. Burkhard, "Consistency and recovery control for replicated data," in *Proceedings of 10th Symposium on Operating Systems Principles*, pp. 87 – 96, ACM, 1985.

[6] S. Jajodia and D. Mutchler, "Dynamic voting," in *Proceedings of SIGMOD-87*, pp. 227 – 238, ACM, 1987.

[7] D. Barbara, H. Garcia-Molina, and A. Spauster, "Protocols for dynamic vote reassignment," in *Proceedings of Principles of Distributed Computing*, pp. 195 –205, ACM, 1986.

[8] A. E. Abbadi and S. Toueg, "Maintaining availability in partitioned replicated databases," in *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pp. 240–351, ACM, 1986.

[9] J. Paris, "Voting with witnesses: A consistency scheme for replicated files," in *Proceedings of the 6th International Conference on Distributed Computing Systems*, pp. 606–612, IEEE, 1986.

[10] D. Agrawal and A. E. Abbadi, "Reducing storage for quorum consensus algorithms," in *Proceedings of Very Large Databases Conference*, pp. 419 – 430, 1988.

[11] P. Bernstein and N. Goodman, "An algorithm for concurrency control and recovery in replicated distributed databases," *ACM Transactions on Database Systems*, vol. 9, no. 4, pp. 596–615, 1984.

[12] D. Barbara and H. Garcia-Molina, "The reliability of voting mechanisms," *IEEE Transactions on Computers*, vol. 36, no. 10, pp. 1197–1208, 1987.

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

[14] S. Y. Cheung, M. Ahamad, and M. H. Ammar, "Optimizing vote and quorum assignments for reading and writing replicated data," *To appear in IEEE Transactions on Knowlegde and Data Engineering*, 1989. (A preliminary version appeared in the Proceedings of 5th International Conference on Data Engineering, pages 271 – 279, IEEE, 1989.).

[15] Z. Tong and R. Y. Kain, "Vote asignments in weighted voting mechanisms," in *Proceedings of the 7th Symposium on Reliable Distributed Systems*, pp. 138 – 143, IEEE, 1988.

[16] M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Transactions on Computer Systems*, vol. 3, pp. 145–159, May 1985.

[17] H. Garcia-Molina and D. Barbara, "How to assign votes in a distributed system," *Journal of ACM*, vol. 32, no. 4, pp. 841–860, 1985.

[18] S. Y. Cheung, M. H. Ammar, and M. Ahamad, "The grid protocol: A high performance scheme for maintaining replicated data," Tech. Rep. GIT-ICS-89/22, Georgia Institute of Technology, Atlanta, GA., 1989.

[19] R. Aggrawal, M. Carey, and M. Livny, "Models for studying concurrency control performance: Alternatives and implications," *Proceedings SIGMOD Conference*, pp. 108–121, 1985.