

Voting strategies for Thomas's Majority Consensus Method
by William Jay Hutchison

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Montana State University
© Copyright by William Jay Hutchison (1992)

Abstract:
This paper investigates the performance of several strategies for ordering votes during update synchronization for transactions accessing replicated or multiple copy databases managed by Thomas's Quorum Consensus Method. Strategies for fixed order voting, random order voting, and shortest paths voting are presented and evaluated. Two methods for refreshing rejected transactions are evaluated and compared. A simulation of a distributed replicated database system residing on a network with update transaction access managed by Thomas's Method is described. A suite of workloads is developed to exercise the simulation facilitating the evaluation of the refresh methods and vote order strategies. The results from the simulation are presented. The average vote probe count,, transaction response time, and transaction throughput achieved by each strategy are examined.

Fixed order voting is shown to have the property of fast conflict detection and resolution. No advantage to either refresh method is found.

VOTING STRATEGIES FOR THOMAS'S MAJORITY CONSENSUS METHOD

by

William Jay Hutchison

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 1992

T7378
H975c

APPROVAL

of a thesis submitted by

William Jay Hutchison

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

12/18/92
Date

Chairperson, Graduate Committee

Approved for the Major Department

December 18, 1992
Date

Head, Major Department

Approved for the College of Graduate Studies

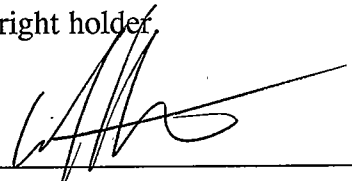
12/21/92
Date

Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under the rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature 

Date December 18, 1992

ACKNOWLEDGMENT

The work reported in this paper was supported by the Hewlett-Packard Company.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES viii

ABSTRACT ix

1. INTRODUCTION 1

2. REPLICATED DATABASES AND MUTUAL EXCLUSION 4

Replicated Databases 4
Mutual Exclusion 5
Distributed Transaction Management 7
Master/Slave Strategies 7
Distributed Access Control 8
Thomas's Majority Consensus Method 9
Gifford's Quorum Consensus Method 10
Hierarchical Quorum Consensus 12
$\sqrt{N}$ 14
Tree 16
Comparison of Methods 17

3. VOTING STRATEGIES FOR THOMAS'S METHOD 20

Thomas's Majority Consensus Method 20
Voting Strategies 25

4. THEORY OF OPERATION OF THE SIMULATION 28

The Replicated Database System 28
The Communications Network 30
Daisy Chain Selection 30
The Workload 31
The Network Topology 32
Database and Transaction Sizes 33
The Mean Transaction Interarrival Time and the Number of Transactions ..34

TABLE OF CONTENTS—Continued

Page

Workload Summary 35
The Data Collected 36

5. ANALYSIS OF SIMULATION RESULTS 39

Simulation Behavior 39
Rejected Transaction Refresh Method Results 42
Vote Order Strategies 43

6. CONCLUSION 48

REFERENCES CITED 50

LIST OF TABLES

Table Page

1. Comparison of quorum consensus voting algorithms 18
2. LAN simulation series workloads 35
3. WAN simulation series workloads 36
4. Summary of data points collected 37
5. Maximum interarrival rate for LAN workloads 41

LIST OF FIGURES

Figure Page

1. MC example of intersecting simple majorities 10
2. QC examples of adjusting quorum size to favor reading. 11
3. HQC example with 27 copies in 3 levels 12
4. HQC example of intersecting quorums 13
5. $\sqrt{N}$ example of 7 copies organized in 7 intersecting coteries 15
6. Tree example of 15 copies with quorum of 4 16
7. Tree example of quorum size degradation with single copy failure 17
8. Tree example of quorum size degradation with multiple copy failures 17
9. MC example of a successful update transaction 25
10. Simulation network models LAN and WAN 34
11. Average Probes/Transaction vs. τ for WAN workloads 40
12. Average LAN Response Time vs. Throughput for LAN workloads 41
13. Average LAN 15% Probes/Transaction vs. τ showing refresh methods 42
14. Average Probes/Transaction vs. τ for LAN 10% workload 44
15. Response Time vs. Throughput for LAN 10% workload 45
16. Average Probes/Transaction vs. τ for WAN 10% workload 46
17. Response Time vs. Throughput for WAN 10% workload 46

ABSTRACT

This paper investigates the performance of several strategies for ordering votes during update synchronization for transactions accessing replicated or multiple copy databases managed by Thomas's Quorum Consensus Method. Strategies for fixed order voting, random order voting, and shortest paths voting are presented and evaluated. Two methods for refreshing rejected transactions are evaluated and compared. A simulation of a distributed replicated database system residing on a network with update transaction access managed by Thomas's Method is described. A suite of workloads is developed to exercise the simulation facilitating the evaluation of the refresh methods and vote order strategies. The results from the simulation are presented. The average vote probe count, transaction response time, and transaction throughput achieved by each strategy are examined.

Fixed order voting is shown to have the property of fast conflict detection and resolution. No advantage to either refresh method is found.

CHAPTER 1

INTRODUCTION

As distributed systems gain acceptance in universities and businesses, replicated file and database systems are becoming important. The widespread use of networks to interconnect computers naturally leads to the desire to share information through access to files and databases located throughout the network. Many techniques have been developed to provide this type of access. Among them is the use of duplicates or replicates of a database spread through the network. Distributed replication allows continued access to the database in case of a database server host failure. Improvements in access performance are also provided by distributing the access workload among multiple computers and by shortening the communications distance between any user and a copy of the database.

Access to the database replicates must be managed in such a way as to guarantee consistency among the copies while allowing multiple users to access the database. This is generally achieved by applying some scheme to enforce serial equivalence among concurrent read and update database transactions. Serial equivalence requires that the state of the database after any two concurrent transactions complete be equivalent to the state the database would achieve if the two transactions were entirely serial. If two transactions conflict, serial equivalence cannot be achieved and one of the transactions must be aborted. Serial equivalence does not specify in which order the transactions would execute, only that the database be left in one of the two possible states after both transactions complete.

The two primary methods of managing access to replicated databases are master/slave and distributed access. Master/slave systems have a single primary copy of the database with all changes made to the primary and then promulgated out to the replicates. While quite efficient in normal operation, the loss of a database master server cripples the ability to update the database. Distributed methods are more robust because there is no single point of control. Distributed management prevents a single server failure from halting database update operations.

Distributed methods usually require an updating process to gain permission from each member of some subset or quorum of the copies to update the database. Permission is acquired by having the copies vote to allow the updating process to continue. If the updating process cannot assemble the required quorum of affirmative votes, permission is denied and the process must retry later. This voting technique was first described by R. H. Thomas in [Thomas, 1979][1]. The technique is generally known as Thomas's Majority Consensus Method. In order to update a replicated database, an application process must gather a simple majority of assenting votes from the database copies. The method does not describe any particular strategy for organizing or ordering the voting to quickly resolve update permission.

The majority of published research into distributed voting methods for managing access to replicated files and databases has concentrated on reducing the size of a quorum needed to gain permission to update a file or a database. The various papers on the topic have described organizations of copies that result in significantly smaller voting sets while maintaining most of the desirable attributes of a replicated database system including consistency, reliability and performance. Some of this research will be described in Chapter 2.

No research has been found which explores ordering voting to reduce the time and number of operations required to achieve a quorum. As noted above, Thomas did not

specify any particular order or strategy for gathering votes. While later methods may provide some improvement in quorum sizes, Thomas's original approach is elegant and straightforward and has been a mainstay of the literature regarding replicated file and database management. Thus the research reported here will concentrate on enhancing Thomas's Method by exploring several strategies for ordering the voting to improve the performance of update synchronization for a replicated database system.

The need for concurrent access control and several methods from the literature will be discussed. Thomas's Majority Consensus Method will be described in some detail. The system performance aspects of voting organization in the operation of this method will be singled out for further investigation. To quantify the performance effect of several possible voting strategies, a simulation of Thomas's Method will be developed.

The results of the simulation will be analyzed and conclusions about the applicability of the strategies will be drawn.

CHAPTER 2

REPLICATED DATABASES AND MUTUAL EXCLUSION

Replicated Databases

While concurrent access management methods have general application to data set and item locking in distributed databases, this paper will focus on transaction management for replicated databases. A database system with copies of each data set on separate machines is generally known as a *distributed replicated database service*. A number of independent computers collaborate to maintain consistent copies of critical or frequently accessed databases presenting a fast, reliable and cohesive database system service to client processes. In most cases, each computer bears partial responsibility for controlling access to the set of duplicates.

The objectives of replicating databases and files around a network are improved reliability and performance. Having multiple copies of an important database located on independent but cooperating computers allows continued access in the event of one or more system or network component failures. As long as a client is able to reach a server hosting a current copy of the target data set, processing may continue.

Where a database is accessed by processes running on many distributed systems and is frequently read but is only occasionally updated, multiple copies can be used to reduce communications overhead and read response time. Locating a copy on or topologically near a client process system reduces the number of network transactions required and shortens the average distance each transaction must travel. In addition, the client request load is concurrently shared by the file servers, improving file system

throughput. However, other methods of improving reliability and performance may provide better results where a database is frequently written. Sophisticated transaction concurrency control and serialization techniques such as distributed two phase commit protocols may be required.

Mutual Exclusion

Mutual exclusion is the requirement that access to a shared resource be limited to a single process at a time. Mutual exclusion is necessary for a shared resource if sharing processes will interfere with each other's use of the resource. Mutual exclusion is used in operating systems and in other applications that manage shared resources. A shared resource may be a piece of physical hardware such as a printer where interleaved access would result in confused output or a logical data structure such as a table or linked list where several contiguous operations must made without interference to maintain a consistent and correct structure from access to access. If two or more processes are attempting to simultaneously access a resource for which mutual exclusion is required, the process' access must be serialized. One process must be allowed to complete its activity with the resource before another is allowed to begin. Among the low level locking facilities often used to implement mutual exclusion for more than two processes are test-and-set locks, semaphores[2], and monitors.

A test-and-set lock is bit in memory that is accessed using a hardware test-and-set operation. A process attempting access to a lock controlled resource does a test-and-set on the lock. This operation notes the value of the bit, sets it and returns the original value. If the original value was reset, the executing process did the set and is allowed to continue. If the original value was set, the process is denied access and must try again. Once access is gained, the process uses the resource and resets the lock bit

allowing other processes access to the resource. This is a very low-level operation usually implemented in hardware.

A semaphore is a protected variable that can only be accessed and altered by two atomic software operations, P and V, named for Dutch railroad traffic controls, and an initialization routine. A process needing access to a resource controlled by semaphore executes P. P tests the value of the semaphore. If the value is positive, the process decrements it, gaining access to the controlled resource. If the value is not positive, the process enters a wait queue and is suspended. Once a process has completed its controlled activity with the resource, it increments the semaphore value and, if there are any, removes a suspended process from the wait queue. The revived process should now be able to decrement the semaphore and gain control of the shared resource. Semaphores in one form or another are the usual basis for most software locks.

A monitor is an abstract data type often implemented in concurrent programming languages and system libraries. A monitor is similar in operation to a semaphore but consists of a monolithic program object. A process attempting to use a resource does not implement the access and release operations as with a semaphore. Instead, the process uses a system or application specified monitor to arbitrate access and to manage the process wait queues.

Any of these three increasingly abstract facilities can be used as the building blocks for a high level local lock manager for distributed systems. All that is needed is a way to make the managers available to remote cooperating processes.

Databases are sets of data elements that can be read and written by processes. Two or more processes can safely read all or part of a database simultaneously because each process sees an unchanging, consistent set of elements. However if one or more processes are updating a database, some kind of access control is required to assure

serial equivalency. Two updating processes may not interleave their changes such that the database is left in an inconsistent state. A writing process should not be changing any data being read by other processes as the reading processes may not see a consistent set of data elements. Some elements may be stale and some may be new. Thus some type of mutual exclusion or transaction management is necessary to manage access to databases.

The same access conflicts that occur within a database also occur with a set of replicated databases. If processes are writing to a replicated database, the changes to all the copies must be managed in such a way as to guarantee that all readers see current and consistent database contents and to guarantee that multiple updates do not interfere with each other.

Distributed Transaction Management

The two general approaches to organizing transaction management for replicated databases are *master/slave strategies* and *distributed access control.*

Master/Slave Strategies

Master/slave strategies, such as that used by the Sun Network Information System also known as Sun Yellow Pages Service (YAP)[3], have one primary server and multiple secondary servers for each replicated file or data set. The primary server is responsible for all write access and update transaction serialization. Initial versions of a file or data set and subsequent modifications are propagated by the primary server to the secondary servers. Database service clients may read data elements from any copy but must direct all updates to the master copy. While this method has the advantage of being straightforward with a single point of update control, there are several disadvantages. A reliance on a master copy for update serialization allows a single

failure to preclude further changes. Modifications to the master copy are usually made without locking slave copies. Updates are distributed to slave copies after the master update is complete. Clients may read a stale slave copy until change notification is complete. Master/slave strategies are most appropriate for stable or slowly changing files and databases.

Distributed Access Control

Distributed access control is a more general approach in which there is no single master copy. Access arbitration generally involves several copies. While other mechanisms can be used to manage concurrent access and data set suite consistency, distributed access control allows processing to continue with the loss of some number of copies.

Following are some examples of distributed algorithms for managing access to a set of replicated files and databases.

- Majority Consensus (MC) [Thomas, 1979].
- Quorum Consensus (QC) [Gifford, 1979][4].
- Hierarchical Quorum Consensus (HQC) [Kumar, 1991][5].
- $\sqrt{N}$ [Maekawa, 1985][6] .
- Tree [Agrawal, 1989][7].

All of these techniques involve gaining access permission from each member of some subset of the database suite before reading or updating any copy of a data set. Mutual exclusion is achieved by guaranteeing that any two properly selected subsets will contain at least one member in common and that all members give permission only if the requested access will not interfere with any other active access to that member's data sets. Thus no two conflicting processes will each succeed in simultaneously gaining permission from a complete subset. The various techniques mentioned above

focus on organizing subset selection to reduce the size of a subset while guaranteeing at least one copy intersection between any two allowed subsets.

A database update transaction consists of a set of base elements, data items read from the current database from which the updates are computed, and a set of update elements, the data items to be changed.

Thomas's Majority Consensus Method

Majority consensus (MC) is the simplest and oldest of the algorithms discussed in this paper and is a mainstay of the literature. The concept is fairly straightforward. Each copy of a database in a database suite is given a vote on whether a process may have access to the suite or not. A process attempting to update the database suite must assemble at least a simple majority of copy votes agreeing to the action. Because a majority includes more than half of the files in the suite, every majority will include at least one common copy. The intersecting copy will arbitrate access to the database suite. An update transaction receives a OK vote only if its base elements are current and if the requested updates will not interfere with any other currently active transaction. The algorithm will be further detailed in Chapter 3.

For n copies and any two majorities m_1 and m_2:

$$m_1 + m_2 > n \qquad 2m > n \qquad m \geq \left\lceil \frac{n+1}{2} \right\rceil$$

Figure 1 depicts an example with 5 copies and two processes named *a* and *b*. If process *a*'s transaction has achieved a majority and is accessing the database suite, process *b*'s transaction will not be able to assemble the necessary number of OK votes if the two transactions conflict.

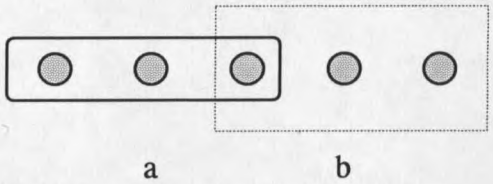


Figure 1. MC example of intersecting simple majorities.

When a majority is achieved, a reading process must inspect the version numbers of all members of the majority and read from the most recent copy. An accepted update must eventually be applied to all members of the database suite. These two requirements guarantee that a reader always gets the most current version of the file and that the database converges to a consistent and up to date state.

The failure or loss of a number of copies up to one less than a majority does not affect the operation of the algorithm except that processes may be delayed while waiting for a response from a failed copy. If the number of available copies drops below a majority, the entire distributed system must redefine the size of a majority.

The main problem with the MC algorithm is that a process must communicate with over half of the copies to establish a majority. A simple extension to Majority Consensus addresses this issue and is discussed in the next section.

Gifford's Quorum Consensus Method

The Majority Consensus algorithm can be easily improved by allocating different copies different numbers of votes. This algorithm is known as Quorum Consensus (QC) and is generally applied to suites of replicated files. As long as the vote allocation is consistent, mutual exclusion can be maintained. Each copy of a file in a suite is given a number of votes. In addition, the number of votes required to read the file is set to a count of votes called the read quorum, q_r. The number of votes required to write the file is set to a potentially different count called the write quorum q_w, The

two quorum values need not be the simple majority, but can be adjusted to reflect probable read/write access patterns as long as both the sum $q_r + q_w$ and the product $2q_w$ guarantee at least one overlapping copy. The intersecting copy will arbitrate write access to the database suite.

$$\text{For } n \text{ total votes:} \quad q_r + q_w > n \qquad 2q_w > n \qquad q_w \geq \left\lceil \frac{n+1}{2} \right\rceil$$

If the access workload is dominated by reads with the occasional write, the read and write quorum size can be adjusted to allow multiple reads. Figure 2 shows example read and write quorums with 5 total votes.

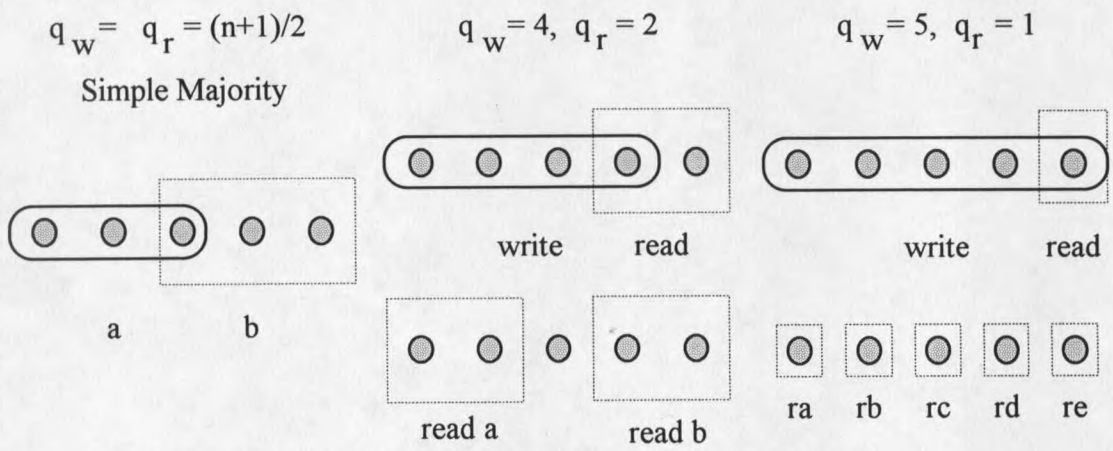


Figure 2. QC examples of adjusting quorum size to favor reading.

The number of votes assigned to a particular copy can be adjusted to reflect the importance of that copy to the file suite. For instance, a particularly fast or centrally located copy may be given more votes to increase the likelihood it participates in a quorum. A copy that is frequently backed up may be favored so that a recent version of the file is usually saved. The vote assignment and quorum sizes must be prearranged and known to all components of the distributed system and all of the accessing processes.

As with majority consensus, a reading process that has achieved a quorum must inspect the version numbers of all copies constituting the quorum and read any most current copy and writes must update all copies in the quorum. All quorums are guaranteed to start with at least one up to date copy.

A missing copy can be tolerated although a skewed assignment of votes can lead to one copy being more important than another and its loss more damaging.

The communications problem observed with majority voting remains an issue for QC although the read quorum size can be adjusted to reduce the impact at the expense of writers.

Hierarchical Quorum Consensus

The communications issue seen with both Majority Consensus and Quorum Consensus is addressed by finding voting organizations that reduce the number of participating copies necessary to achieve mutual exclusion. One approach to this is the Hierarchical Quorum Consensus (HQC) algorithm. For a file suite of size n, the n copies are organized into a tree of depth m where each copy is a leaf at level m and each interior node at level i, $0 \leq i \leq m$, has l_i offspring. Figure 3 shows an example with 27 copies organized in three levels.

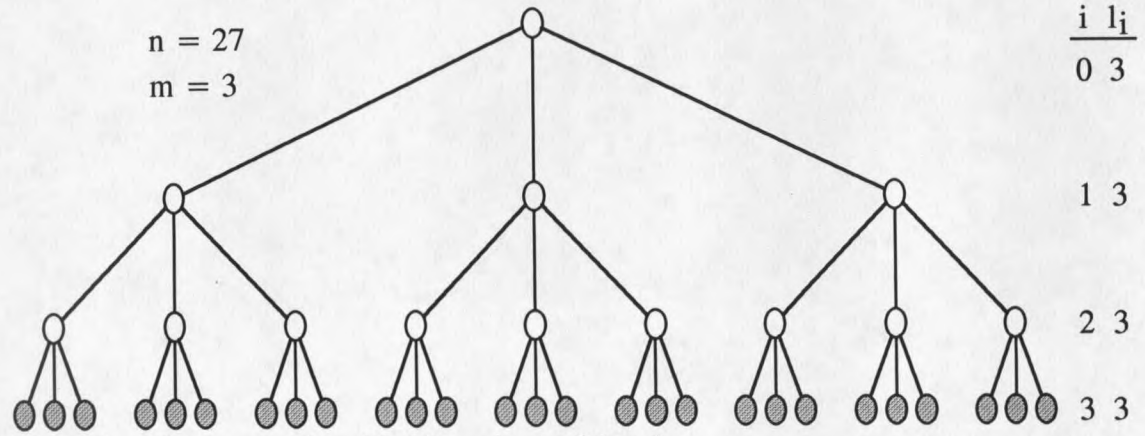


Figure 3. HQC example with 27 copies in 3 levels.

Nodes at level *m-1* represent logical groups of copies. Nodes at level *m-2* represent groups of groups, etc. Each level *i* has quorum sizes for groups at that level such that:

r_i = read quorum $\quad w_i$ = write quorum $\quad$ at level i

$q_r = r_1 \times r_2 \times \ldots \times r_m \quad q_w = w_1 \times w_2 \times \ldots \times w_m$

$r_i + w_i > l_i, \quad 2w_i > l_i, \quad 1 \le i \le m$

Access to a group at level *i-1* requires a quorum at level *i*.

Any pair of read and write quorums or pair of write and write quorums will share at least one physical (leaf) copy as shown in Figure 4.

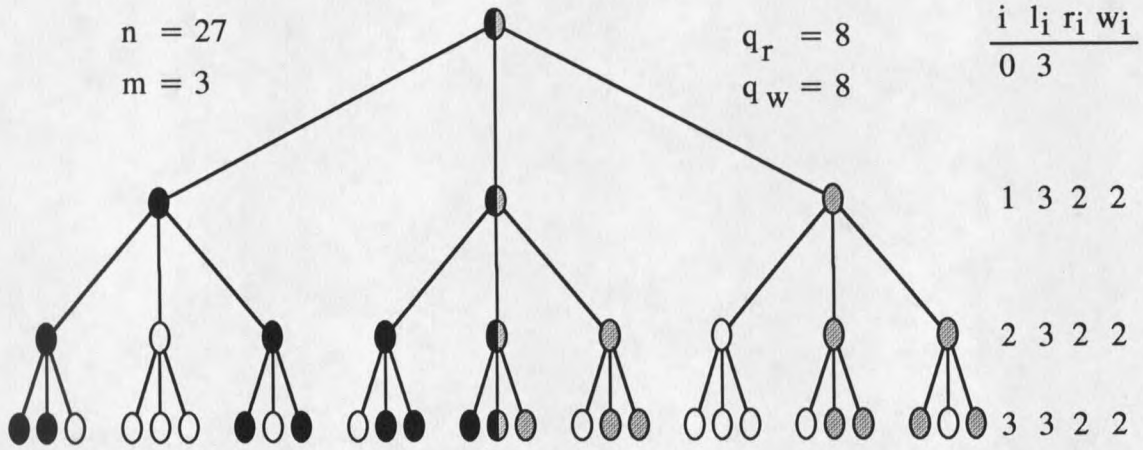


Figure 4. HQC example of intersecting quorums.

Since any two quorums must intersect at each level, the size of a quorum at any level is minimum when it is a simple majority of subgroups at that level:

$$maj(n) = \left\lceil \frac{n+1}{2} \right\rceil = \begin{cases} \frac{n}{2} + 1, & n \text{ even} \\ \frac{(n+1)}{2}, & n \text{ odd} \end{cases}$$

Read and write quorums may be adjusted to favor writes by setting:

r_i, w_i such that $r_i + w_i > l_i, \quad 2w_i > l_i, \quad 1 \le i \le m$

Read and write quorums may be adjusted to favor writes by setting:

r_i, w_i such that $r_i + w_i > l_i, \quad 2w_i > l_i, \quad 1 \le i \le m$

From [Kumar, 1991], it can be shown that the ideal number of groups at each level is 3. At each level, choosing a minimum quorum size that is a simple majority of the number of groups at that level leads to a set of bounds on the size of a quorum. This is shown in the relationships below. As the number of copies grows, the advantage in quorum size relative to Majority Consensus grows.

$$\text{If } n = 3^k \text{ and } k \in Z^+ \text{ then } q = 2^{\log_3 n} \cong n^{.63}$$

$$\text{If } n \neq 3^k \text{ then } n^{.63} < q << majority(n) \text{ and } n > 10$$

The major improvement HQC offers over Quorum Consensus is that the size of the quorum scales sublinearly. HQC retains the ability to adjust the relative sizes of read and write quorums. The loss of a single file copy does not affect the operation of the remainder of the file suite. However, the loss of several copies in one group can degrade the performance of the system by making entire groups unavailable at each level. One administrative problem is that all accessing processes must use the same hierarchy leading to initialization and maintenance overhead.

$\sqrt{N}$

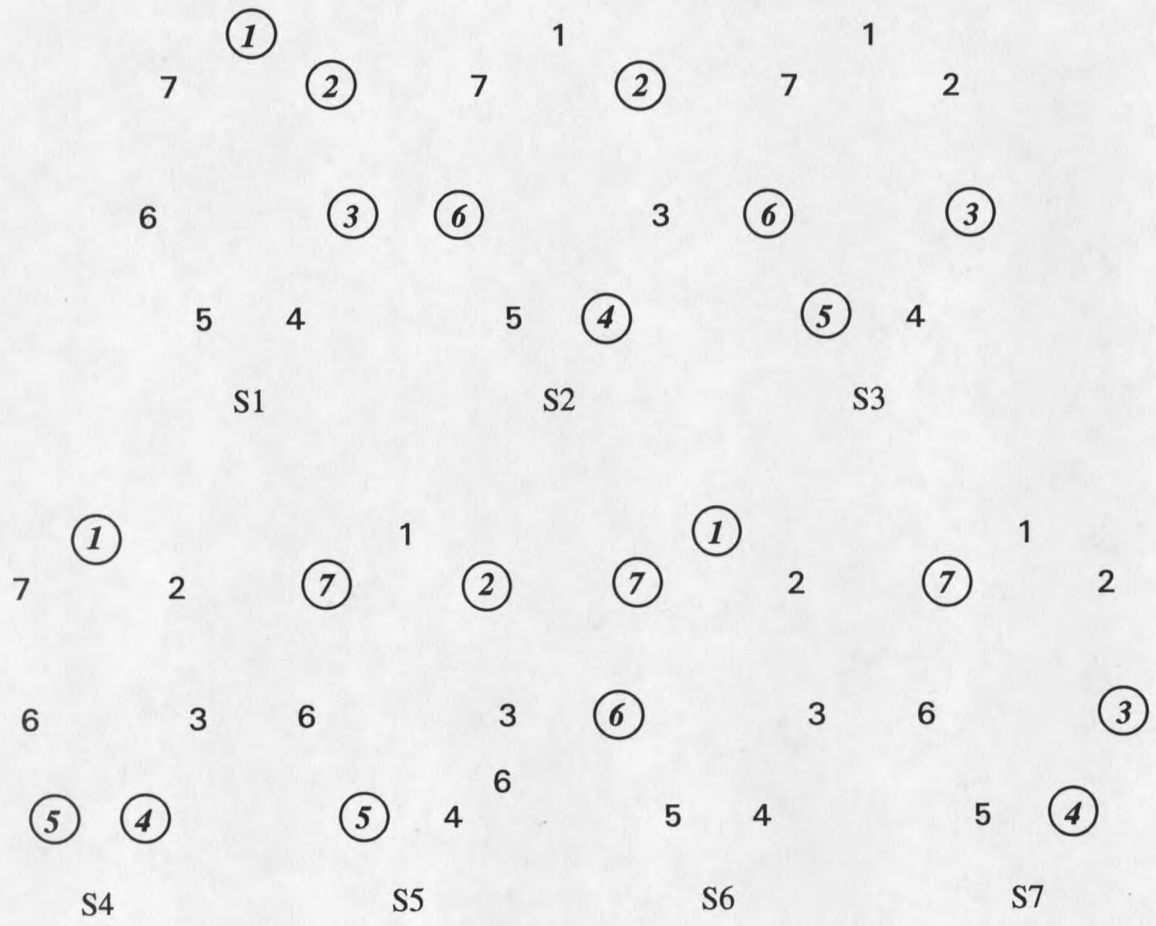
The $\sqrt{N}$ algorithm further reduces the size of a quorum by organizing the file suite into a set of equal sized coteries or cliques. All coteries are guaranteed to have at least one member in common. As with the previous methods, each copy has a vote. A process must get all the members of any coterie to agree to allow the process to access the file suite. No member of a coterie can vote for more than one process at a time.

For n copies, there are n groups or *coteries* and quorum size, $q \approx \sqrt{n}$:

$n = k(k-1)+1$, where $k = |\text{coterie}|$ and k is minimized

$k \approx \sqrt{n}$: For n copies and n coteries, each copy $\in \sqrt{n}$ coteries, $|\text{coterie}| = \sqrt{n}$

For example, if there are 7 copies numbered 1 through 7, there are 7 coteries also numbered 1 through 7. Each coterie contains 3 copies as shown in Figure 5.



Node i locks all members of S_i

Figure 5. $\sqrt{N}$ example of 7 copies organized in 7 intersecting coteries.

Note that each coterie shares one member with every other coterie. If all members of a coterie agree to allow a process access to the file suite, then no other coterie can grant access. Each coterie is managed by one copy. Thus a process need only contact one copy to arrange the coterie.

$\sqrt{N}$ scales sublinearly in the number of copies of a file slower than the Hierarchical Quorum Consensus algorithm. However, the size of a read quorum must be the same as the size of a write quorum. This prevents multiple simultaneous readers. The loss of a single copy causes the loss of $\sqrt{n}$ coteries and the loss of $\sqrt{n}$ copies could, in the

worst case, damage all coteries. The latter would render the system inoperative. As with Hierarchical Quorum Consensus, the organization of the coteries must be prearranged and known to all accessing processes.

Tree

The last mutual exclusion method reduces the size of a quorum for a suite of n copies to $\log_2 n$ by organizing the copies in a full binary tree of $\log_2 n$ levels with a copy at each interior and leaf node. To gain access to the file suite, a process must lock all copies on some path from the root of the tree to a leaf as shown in Figure 6. While the size of a quorum is now small and scales very slowly, the copies higher up in the tree are more important and must participate in more quorums.

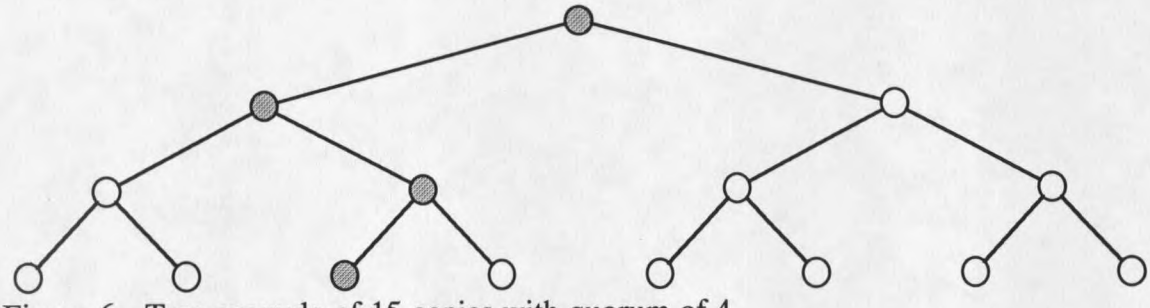
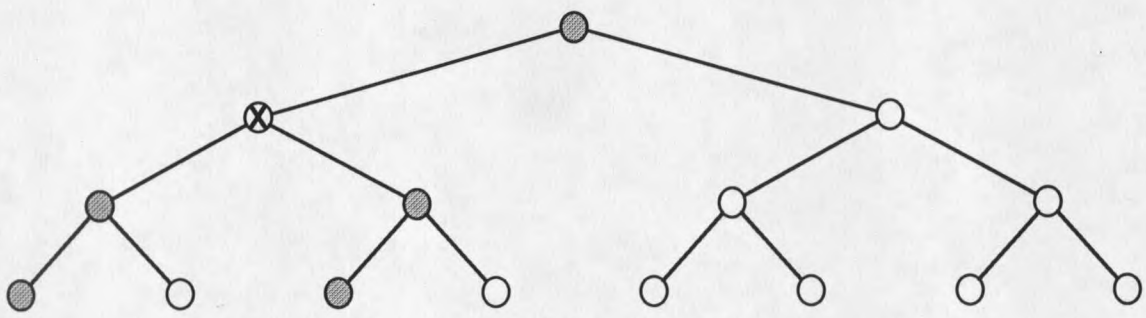


Figure 6. Tree example of 15 copies with quorum of 4.

While this scheme achieves the smallest quorum size yet, it is subject to degradation if one copy fails. If a copy is missing, the process must gain the votes of copies along two paths starting at the children of the missing copy as demonstrated in Figures 7 and 8. The degradation is graceful in that the algorithm will withstand multiple failures and that the number of copies needed to substitute for the missing copy increases slowly. However, in the worst case, the number of copies needed grows to one half the total.

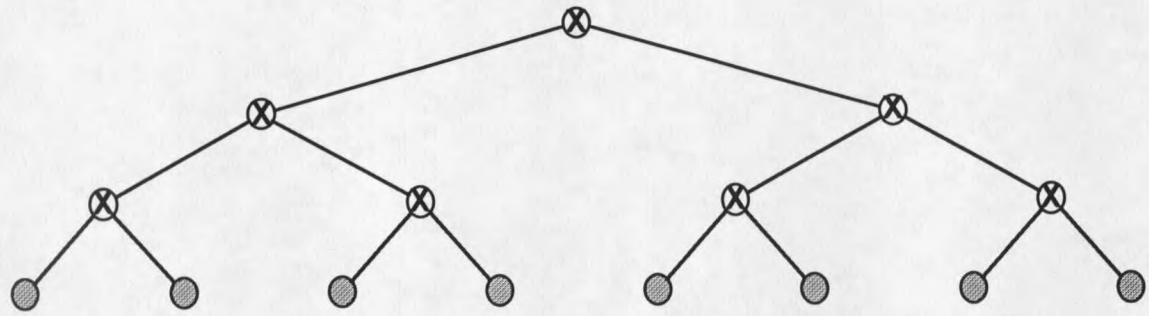
The main advantage the Tree algorithm offers is the quorum size, which is log of the number of copies. In normal operation, the root copy is the actual arbiter of access. If there is a failure, the path locking algorithm allows continued operation by

keeping a subset of the file suite up to date. Any access to the file suite must encounter at least one member of this up to date subset and so is able to work with a current copy of the file.



1 copy unavailable, quorum = 5

Figure 7. Tree example of quorum size degradation with single copy failure.



7 copies unavailable, quorum = 8 = (n+1)/2

Figure 8. Tree example of quorum size degradation with multiple copy failures.

Comparison of Methods

Table 1 on Page 18, taken largely from [Kumar, 1991], compares several key attributes of the various algorithms.

The communications performance of the 5 algorithms must also be considered. A rough lower bound to the number of messages required to access the file suite can be implied from the size of the quorum. In the best case, each member of the quorum must be contacted to request a vote and to relinquish the vote. The voting process can

be piggy backed on the request if daisy chaining is used. A voting member receives a request, concatenates its vote to the request and passes the request on to another voting member. When a member votes OK, the request is checked for resolution. If that vote resolved the access question, no further voting is necessary and the last voter can return the results to the accessing process or the managing member depending on how the system is organized with updates completed asynchronously. Consequently the number of messages scales almost exactly with the quorum size.

	MC	QC	HQC	$\sqrt{N}$	Tree
Best case quorum size	$\lceil \frac{n+1}{2} \rceil$	$\lceil \frac{n+1}{2} \rceil$	$n^{0.63}$	$n^{0.5}$	$\log_2 n$
Worst case quorum size	$\lceil \frac{n+1}{2} \rceil$	$\lceil \frac{n+1}{2} \rceil$	$n^{0.63}$	$n^{0.5}$	$\lceil \frac{n+1}{2} \rceil$
Is algo. fully distributed?	yes	yes	yes	yes	no
Worst cost of one node failure	1	1	1	$n^{0.5}$	$\log_2 n$
Can q_r and q_w be varied?	no	yes	yes	no	no

Table 1. Comparison of quorum consensus voting algorithms.

The communications performance of the 5 algorithms must also be considered. A rough lower bound to the number of messages required to access the file suite can be implied from the size of the quorum. In the best case, each member of the quorum must be contacted to request a vote and to relinquish the vote. The voting process can be piggy backed on the request if daisy chaining is used. A voting member receives a request, concatenates its vote to the request and passes the request on to another voting member. When a member votes OK, the request is checked for resolution. If that vote resolved the access question, no further voting is necessary and the last voter can return the results to the accessing process or the managing member depending on how the

system is organized with updates completed asynchronously. Consequently the number of messages scales almost exactly with the quorum size.

The concurrency control methods described in this chapter attempt to reduce communications overhead and improve replicated file or database access performance by reducing the size of a vote quorum required to achieve mutual exclusion, The remainder of this paper will look at how the order of voting relates to system performance and will concentrate on Thomas's Majority Consensus method.

CHAPTER 3

VOTING STRATEGIES FOR THOMAS'S METHOD

In [Thomas, 1979], Robert H. Thomas introduced the concept of copies of a file or a database voting on transaction access permission. While the concept of voting has been much refined since this seminal work, Thomas's method remains a standard in the literature on concurrency control and replicated database management. His algorithm is elegant and straightforward. The Majority Consensus Method provides a good base for exploring the effect of various vote gathering strategies in a networked environment.

Given a database replicated around a wide area network or a local area network, some method has to be found to manage concurrent reads and updates from multiple processes to ensure database consistency as described in Chapter 2. First, Thomas's Method will be described in greater detail. Then several approaches to voting daisy chain organization will be introduced. The following chapter will describe a simulation of Thomas's Method used to quantify the system performance differences between the daisy chain organizations. The results of the simulation will be analyzed in Chapter 5.

Thomas's Majority Consensus Method

Thomas describes a system environment consisting of two types of entities, Database Managing Processes or DBMPs and Application Processes or APs. DBMPs operate as database copy servers, each managing one copy of a replicated database. As a group, DBMPs control access to the entire database suite to ensure mutual copy

consistency between AP transactions. APs are database clients submitting read queries and update transactions to a DBMP for application to the database. A database consists of a set of data elements and a version number or time stamp for each element. An update transaction consists of a set of base elements previously queried from the database along with their respective time stamps and a set of update elements which must be a subset of the base elements.

The DBMPs must make sure that each update transaction's base elements are current and that the transaction will not interfere with any other active transaction before allowing the transaction updates to be applied to the database. This is done with the voting process explained below. When an update is applied to the database, the value of each update element becomes the current value of the respective database element and the time stamp of the transaction becomes the time stamp of each updated database element.

Database read queries are simply satisfied by submitting a list of elements to a DBMP which returns the values and the time stamps of those elements from its copy of the database.

Update transactions are more complicated. The need for concurrency control among all the database copies and accessing processes requires that a majority of DBMPs inspect each transaction and vote to accept it. In [Thomas, 1979], Thomas proposes a six step process for implementing update transaction synchronization.

1. *Query Database.* The AP queries any DBMP for a set of base element values and time stamps.
2. *Compute Update.* The AP computes the new values of the update elements which must be a subset of the base elements.
3. *Submit Request.* The AP submits the lists of base elements and time stamps and the list of update elements to a DBMP.

4. *Synchronize Update.* The DBMP initiates a voting process among the DBMP set to accept or reject the request.
5. *Apply Update.* Each DBMP applies an accepted update to its copy of the database. Rejected updates are removed from the system.
6. *Notify AP.* Some DBMP notifies the originating AP of the transaction outcome. A rejected update may be recomputed from a fresh set of base elements and resubmitted.

Thomas's Majority Consensus algorithm consists of five rules defining DBMP behavior. The rules implement a replicated database access control algorithm which guarantees serial equivalency among concurrent database update transactions.

1. *DBMP/DBMP Communication Rule.* This rule specifies the communications process used by the DBMP set to resolve a transaction request. From among Thomas's suggested options, this paper will concentrate on pure daisy chaining. A list, or daisy chain, of all DBMPs is generated. The first DBMP in the chain receives the transaction request and votes on the request. If the vote resolves the request, the DBMP notifies the originating AP and the other DBMPs of the outcome and, if the transaction is accepted, applies the update to its own copy of the database. If the vote does not resolve the request, the vote is appended to the transaction and the transaction is forwarded to the next DBMP in the chain for consideration.
2. *Voting Rule.* This rule specifies the transaction accept/reject decision process and consists of 5 steps.
   a. Compare the transaction base element time stamps to those of their respective database elements in the local copy of the database.
   b. Vote REJECT if any base element is obsolete.

c. Vote OK and mark the request as pending at that DBMP if all base elements are current and the set of base elements of the new transaction does not intersect the set of update elements of any other transaction pending at the DBMP. An intersection identifies a transaction conflict.

d. Vote PASS if the base elements are current but the transaction conflicts with a pending transaction with a more recent time stamp.

e. Defer voting by temporarily storing the transaction at the current DBMP if the transaction base elements are all current but the transaction conflicts with a pending transaction with an older time stamp or if a base element time stamp is more recent than the time stamp of that element in the database copy. This second condition occurs when a transaction base element is initialized by a DBMP which has applied a previous transaction's updates but the current DBMP has yet to apply that update.

3. *Request Resolution Rule.* This rule determines the action following the vote at each DBMP. The rule has two parts.

   a. After a DBMP votes on a request:

      1. If the vote was OK and a majority of OK votes has been accumulated, the originating AP and all DBMPs are notified that the transaction has been accepted.

      2. If the vote was REJECT, immediately notify the originating AP and all DBMPs that the transaction was rejected.

      3. If the vote was PASS, decide if a majority consensus is still possible. If not, reject the transaction, notifying the originating AP and all DBMPs.

      4. If not resolved by the previous three rules, forward the transaction and its accumulated votes to the next DBMP in the daisy chain.

b. When a DBMP learns the resolution of a request:

1. If the transaction is accepted, apply the update to the local copy of the database and reject any transactions deferred because of conflicts with the accepted transaction. Reconsider any transactions deferred because a base element was too current.
2. If the transaction was rejected, use the voting rule to reconsider any transactions deferred because of conflicts with the accepted transaction

4. *Update Application Rule.* This rule determines how an update is applied to each copy of the database. The time stamp of the transaction is compared with the time stamp of each transaction update element's corresponding database element. If the database element is older, the database element is assigned the value of the update element and the database element time stamp is assigned the transaction's time stamp. If the database element is newer, that element is not updated. Thus particular element updates may be skipped if the DBMP has already applied an update from a more recent transaction.
5. *Time Stamp Generation Rule.* This rule specifies that each transaction be assigned a time stamp by the receiving DBMP when the transaction is first submitted for update synchronization and that all time stamps generated in the system be unique and serial.

In [Thomas, 1979], the six steps and five rules enumerated above are demonstrated to guarantee a mutually consistent replicated database and to guarantee that no two conflicting update transactions will both be accepted. The algorithm is also shown to be deadlock free due to the PASS and deferred voting options. However, the algorithm does not eliminate transaction starvation.

Figure 9 shows an example of a successful update transaction originating at AP0 with six DBMPs in the daisy chain. The first four DBMPs vote OK allowing DBMPd to resolve the update request.

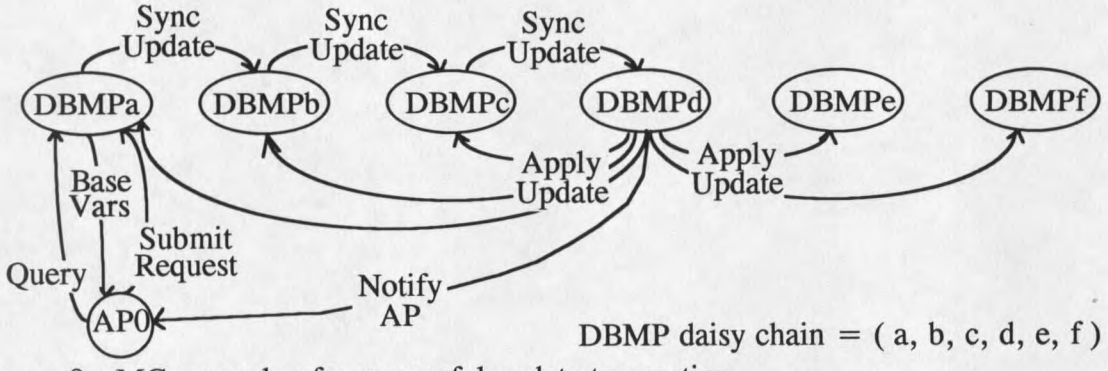


Figure 9. MC example of a successful update transaction.

Voting Strategies

In the absence of any previous investigation of vote ordering, the three obvious strategies for ordering daisy chains will be evaluated for system performance. Recalling that a daisy chain is the path through the DBMPs followed by a transaction as each DBMP votes on the acceptability of the transaction. The daisy chain is followed until some DBMP vote resolves the transaction causing either acceptance or rejection.

1. *Fixed Path Voting Order.* In this strategy, all Application Processes will use the same daisy chain to order the DBMP voting. The daisy chain will essentially be a list of all DBMPs in the system ordered by some system identifier. Consequently all APs will submit transactions to the same DBMP which will vote and pass the transaction to the same second DBMP and so on.
2. *Random Path Voting Order.* Each update transaction will follow a randomly generated daisy chain. The daisy chain will be some randomly selected permutation of the set of DBMPs.

3. *Shortest Paths Voting Order.* At system startup, each Application Process will determine the order of all DBMPs which provides the shortest path, in terms of communications latency, from the AP through the DBMPs. All transactions initiated by that AP will follow a daisy chain containing that order of DBMPs. This strategy only makes sense when the communications latency between any two DBMPs is not the same.

In addition to the ordering of DBMPs in the transaction, the handling of the resubmittion of rejected transactions will be investigated. There are two immediate possibilities here. Recalling Step 1 of the six step update process on Page 21, an AP first queries a DBMP for the values and time stamps of some set of base elements from which the transaction update element values will be computed. A common cause for rejection of an update is that the base elements are obsolete or out of date. This occurs when the DBMP responding to the AP's query for base values has not applied some accepted transaction and returns values that are out of date or become out of date before the AP's transaction can be accepted.

When a transaction is first created, the originating AP queries the first DBMP in the daisy chain for the transaction base values. If the transaction is then submitted and rejected, the AP must recompute the update using fresh base values. These values may be queried from the same first DBMP, the *Query First* method, or from the DBMP which rejected the transaction, the *Query Rejecter* method. Unless the first DBMP has updated its database copy, it will return the same stale values as before. The rejecting DBMP may possibly return more current values. A transaction may also be rejected for conflicts with other transactions accepted in the interim. In this case, the transaction was rejected because its base values become obsolete when the conflicting transaction is accepted and applied. Querying the rejecting DBMP may again result in fresher base values. This strategy still does not guarantee a transaction will be accepted

when resubmitted. Additional transactions may be accepted while the originating AP recomputes and resubmits the rejected transaction. None of these strategies will guarantee that a transaction will not starve.

CHAPTER 4

THEORY OF OPERATION OF THE SIMULATION

In order to evaluate the relative merits of the three daisy chain vote order strategies, a simulation of a replicated database system operating on a network was developed. The simulation was implemented using Hewlett-Packard's C++ language system on an HP Series 900 Model 375 Computer. The simulation made extensive use of the Task Class Library from AT&T. This library provides a set of co-routine and simulation tools which facilitate the development and operation of very complex simulations in C++ without the need to resort to a specialized and expensive simulation package. In particular, the co-routine or task scheduling facility and the simulation clock facility provide a surprisingly powerful platform for system simulation.

The Task Class Library provides for the simulation of a system consisting of any number of independent and concurrent process threads. Simulation time, as implemented by the Task Class Library, is measured in an arbitrary unit which shall be called a Tic. Simulation time advances only when a task specifically incurs a latency. Thus a simulated system has complete control over how much work can occur at any moment in simulation time.

The Replicated Database System

The replicated database system consists of a set of Application Processes and a set of Database Managing Processes interconnected by a network and communicating by the exchange of transaction messages. At system initialization, each AP and each

DBMP is assigned an identification number and a message input queue. The identification numbers are used to manage the daisy chain voting process.

A transaction consists of a transaction time stamp, a set of base element names, a set of base element time stamps, a set of update elements, a daisy chain list of DBMP identifiers, an OK vote count, a PASS vote count, and total vote probe count. The total probe count accumulates the number of DBMPs voting on the transaction's update request over the entire life of the transaction, including retries after rejection.

When the system is ready to operate, the simulation clock is started and the APs begin launching transactions. The six steps in the update transaction process and the five rules implementing the concurrency control algorithm specified by Thomas and introduced in Chapter 3 are faithfully implemented.

Each AP shares the workload equally. There is a transaction interarrival time management mechanism for establishing the interval between each transaction start. Each AP launches its next transaction at the time specified by the interarrival time manager.

At instantiation, a transaction randomly selects a specified percentage of the database elements to be the transaction base elements and then randomly selects a specified percentage of the base elements to be the transaction update elements. A daisy chain of DBMP identifiers is built. The daisy chain is an ordering of all DBMPs in the system and specifies the path the transaction will follow through the network as the DBMPs vote on the transaction update request.

The transaction is then passed to the network for delivery to the first DBMP for the base element query step. The transaction is returned to the initiating AP for the compute step and then submitted along the daisy chain for the update synchronization step. The first DBMP in the daisy chain assigns a time stamp to the transaction, votes on the transaction's update request and passes the transaction along to the next DBMP

in the daisy chain. If a DBMP votes REJECT, the transaction is immediately aborted, and the originating AP and all other DBMPs are notified. The AP may then restart the transaction with the same base and update element sets, but must acquire fresh base values and time stamps by querying a DBMP.

If a majority of DBMPs vote OK, the transaction is immediately accepted and the originating AP and all other DBMPs notified. All DBMPs apply the update as they receive the acceptance message. Each DBMP database element in the transaction update element set is assigned the transaction's time stamp.

At any time, there may be many transactions active in the system.

The Communications Network

There is a network object which provides message passing services between APs and DBMPs and between DBMPs. In addition, the network implements service latencies or delays in communications in order to simulate the behavior of a replicated file system operating on a network.

At system initialization, a network connection cost table specifies the service latencies between each pair of nodes in the network. The latency value is the sum of a base or minimum component and a probabilistic component. This simulates the behavior of a network with minimum transmission time plus traffic or routing related delays.

Daisy Chain Selection

As noted in the previous chapter, three strategies for selecting DBMP daisy chains are implemented. A daisy chain specifies the order in which DBMPs vote on a transaction's update request. Transactions use the same daisy chain when retrying a rejected transaction.

The first is Fixed Order Voting. All transactions in the system use the same DBMP daisy chain. This path is simply the list of DBMPs ordered by the network identifier of the DBMP.

The second is Random Order Voting. At instantiation, each transaction creates a randomly ordered list of all DBMPs for its daisy chain.

The last strategy is Shortest Paths Voting. At system startup, each AP computes the shortest paths from itself though the set of DBMPs. If there is more than one path in the shortest paths list, each transaction randomly selects its daisy chain from the list.

The Workload

The set of simulation variables includes:

1. *The Network Topology.* This is the table of network communication latencies between APs and DBMPs. It includes the number of APs, the number of DBMPs, and the expected network latency between each pair of nodes. The latency a transaction encounters on the path between each network node is the sum of the base value for the source/destination pair and a probabilistic value based on a randomly generated exponential number. The mean of the probabilistic exponential values generated for each source/destination pair is also specified in the network topology table.
2. *The Vote Order Strategy.* This is one of Fixed, Random, or Shortest. Shortest is only used when the network topology includes non-uniform communications latencies.
3. *Rejected Update Refresh Method.* This is either Query First or Query Rejecter. When an AP retries a rejected transaction, it can query either the first DBMP in the daisy chain, as it did on the initial attempt, or the AP can query the DBMP which rejected the transaction update request.

4. *The Replicated Database Size.* This is the number of elements in the database.
5. *The Transaction Size.* This is the average percentage of database elements randomly selected by a transaction for use as base elements. It also includes the average percentage of base elements randomly selected by a transaction for use as update elements.
6. *The Mean Transaction Interarrival Time or* τ. τ is an exponential random variable specifying the mean time between the initiation of transactions. The value of τ is the mean of the exponential distribution of interarrival times generated by the transaction interarrival time management mechanism.
7. *The Number Of Transactions Executed.* The number of transactions specifies the length of the simulation. The simulation completes when the last transaction has completed and any outstanding updates have been applied to all database copies.

Two replicated database models were developed and network topologies were defined to represent them. A standard database size was selected. Series of simulation runs were made for each model. In each series, the vote order strategy, the rejected transaction refresh source, and the number of transactions was specified. The values of τ and the transaction size were varied over the series. Because the behavior of each run of the simulation is governed by a number of random variables, the simulation was repeated at least 10 times for each model configuration. The data for each entry in the series simulation results is the average of the data for the repeated runs for that entry.

The Network Topology

For this investigation, two network topologies were selected. The first, named LAN, is intended to simulate the inter-node communications behavior of a replicated

database system residing entirely on one Local Area Network. In this case, the average communications latency between any two nodes should be identical. The database system consisted of two APs and six DBMPs.

The second network topology, named WAN, is intended to simulate the inter-node communications behavior of a replicated database system residing on two LAN-type clusters connected on a Wide Area Network. Thus the communications latency between the nodes located on the same LAN should, on average, be the same. However, the two LANs need not have similar characteristics. There will also be a set of values describing the latency encountered by transactions passed between two nodes residing on different LAN clusters. Each LAN cluster consists of one AP and three DBMPs. The DBMPs are allocated evenly but randomly between the two clusters.

The values for the base and probabilistic components were selected by the repeated use of the UNIX ping facility to characterize communications between actual computer nodes on each of several LAN networks and also between geographically distant nodes connected by the Internet. The Local Area Networks used were the Montana State University network and the Carnegie-Mellon University network. The Internet measurements were taken between nodes located at Montana State University, at Carnegie-Mellon University, at Stanford University and at Hewlett-Packard in Boise, Idaho. The collected data suggested the values shown in Figure 10.

Database and Transaction Sizes

The database size selected for the simulation was 200 elements. Four transaction sizes were used. 5%, 10%, 15%, and 20% of the database elements were selected as transaction base elements. In all cases, 25% of the transaction base elements were selected as transaction update elements.

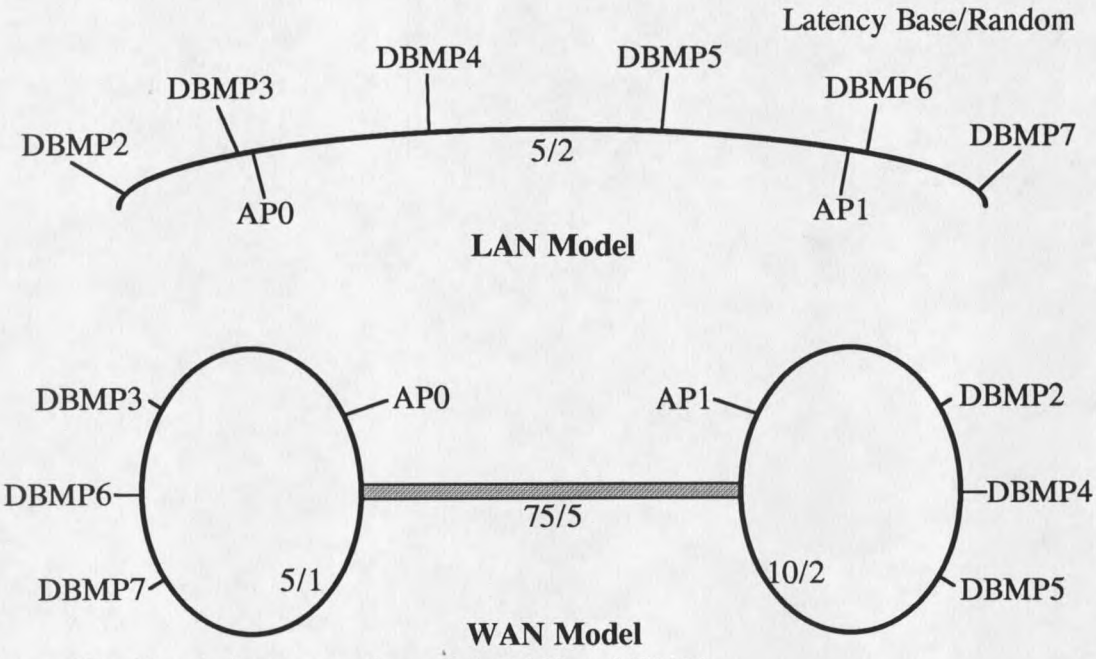


Figure 10. Simulation network models LAN and WAN.

The Mean Transaction Interarrival Time and the Number of Transactions

The value of τ was varied over a series to establish the performance of the selected voting strategy over a range of workloads. The low value of τ was limited by constraining the number of concurrent transactions in the system. This makes sense because the average transaction interarrival time must not exceed the average transaction response time. If the response time is exceeded on average, the system will be overrun with active transactions and will become unstable. More transactions will enter the system than leave the system. This is not a realistic situation.

The high value for τ was selected such that most transactions completed without encountering a conflict. For each entry in the series, τ was incremented by a value selected to give a reasonable range of data points over the interval $\tau_{low}..\tau_{high}$.

The simulation was run over one thousand transactions. This number provided a balance between simulation execution wall time and simulation accuracy. One

thousand transactions provided enough workload to allow the simulation to achieve stability, reducing the startup and termination affects on the data collected.

Workload Summary

Table 2 and Table 3 on Page 36 summarize the simulation workloads for which data was collected.

Database Model	Trans. Size	Vote Strategy	Refresh Method	$\tau_{low}..\tau_{high}$, inc
LAN	5%	Fixed	Query first	4 .. 14, 1
			Query rejecter	4 .. 14, 1
		Random	Query first	5 .. 15, 1
			Query rejecter	5 .. 15, 1
	10%	Fixed	Query first	10 .. 50, 2
			Query rejecter	10 .. 50, 2
		Random	Query first	20 .. 50, 2
			Query rejecter	18 .. 50, 2
	15%	Fixed	Query first	18 .. 38, 2
			Query rejecter	18 .. 38, 2
		Random	Query first	44 .. 74, 2
			Query rejecter	44 .. 74, 2
	20%	Fixed	Query first	25 .. 75, 5
			Query rejecter	25 .. 75, 5
		Random	Query first	75 .. 140, 5
			Query rejecter	75 .. 140, 5

Table 2. LAN simulation series workloads.

Because the messages on a LAN should experience similar communications latencies, the Shortest Paths daisy chain vote strategy is not applied to the LAN replicated database model. The Shortest Paths daisy chain strategy is applied to the WAN database model.

Database Model	Trans. Size	Vote Strategy	Refresh Method	$\tau_{low}..\tau_{high}$, inc
WAN	5%	Fixed	Query first	22 .. 54, 2
			Query rejecter	22 .. 54, 2
		Random	Query first	36 .. 66, 2
			Query rejecter	36 .. 66, 2
		Shortest	Query first	30 .. 70, 2
			Query rejecter	30 .. 70, 2
	10%	Fixed	Query first	60 ..200, 5
			Query rejecter	60 ..200, 5
		Random	Query first	140 ..200, 5
			Query rejecter	140 ..200, 5
		Shortest	Query first	95 ..200, 5
			Query rejecter	95 ..200, 5
	15%	Fixed	Query first	120 ..220, 5
			Query rejecter	120 ..220, 5
		Random	Query first	280 ..380, 5
			Query rejecter	280 ..380, 5
		Shortest	Query first	160 ..260, 5
			Query rejecter	160 ..260, 5
	20%	Fixed	Query first	180 ..380, 10
			Query rejecter	180 ..380, 10
		Random	Query first	450 ..600, 10
			Query rejecter	450 ..600, 10
		Shortest	Query first	210 ..360, 10
			Query rejecter	210 ..360, 10

Table 3. WAN simulation series workloads.

The Data Collected

Each simulation run collected data summarizing the response time and throughput behavior of the system for the selected network model, transaction size, vote strategy, rejected transaction refresh method and average transaction interarrival time. The simulation time measurements are made in arbitrary units called Tics. Tics may be related to any actual time unit and are intended in this case to correlate roughly with

millisecond time units. The specific data points collected and their units are summarized in Table 4.

Data Point	Units
τ	Tics
Transaction Throughput	trans/kTic
Average Transaction Response Time	kTic/trans
Average Number of Vote Probes per Transaction	Probes
Maximum Transaction Concurrency	Transactions
Simulation Time for 1000 Transactions	Tics
Wall Time for Simulation Run	Seconds

Table 4. Summary of data points collected.

τ, the average transaction interarrival time, is an input and is echoed to specify the data ordinate when charting results.

Transaction Throughput is the number of transactions completed per unit simulation time. Over the one thousand transaction simulation run, average transaction throughput = 1000/simulation time. For reasons of readability, throughput is reported in transactions per thousand Tics (trans/kTic).

Average Transaction Response Time is the average length of simulation time required to complete a transaction during the simulation run. Average response time = Σ response times/1000. Again for reasons of readability, average response time is reported in thousand Tics per transaction (kTic/trans).

Average Number of Vote Probes per Transaction is the average number of DBMPs that vote on a transaction. All votes on a transaction's update request are totaled. This includes any votes leading to a rejection and all votes during retries of a rejected transaction.

The Maximum Transaction Concurrency is the high water mark for the number of transactions active in the system at one time during the simulation run.

Simulation Time is the total number of simulation Tics required to complete one thousand transactions and is reported only as a check value.

Wall Time is the actual time required to complete a run and is reported simply to help manage the data collection effort.

The analysis of results will focus on the average number of probes for votes a typical transaction required before acceptance for each workload. In addition, the typical system performance indices of response time and throughput will be examined.

CHAPTER 5

ANALYSIS OF SIMULATION RESULTS

The simulation was run with the workloads shown in Tables 2 and 3 in Chapter 4. The results will first be reviewed for tests of "reasonableness"—does the simulation behave as expected? Then the performance of the rejected transaction refresh methods and of the vote order strategies will be examined. Finally, the performance of the vote order strategies will be considered. The results from the two network topology models were quite consistent over all workloads. Accordingly, charts of selected results are presented and discussed.

Simulation Behavior

With any simulation driven by a probabilistic artificial workload based on exponential interarrival times for resource using activities, any measure of system performance should, when plotted against the interarrival time τ, show near exponential behavior. This was the case with all workloads tested on this simulation. For example, Figure 11 shows the results for the average number of vote probes experienced by WAN workload transactions for each transaction size over a range of τ for the Fixed Path vote order chain strategy. Recall that the workloads used 5%, 10%, 15%, and 20% of the database respectively, as transaction base variables.

With any database simulation, fixing the transaction complexity (the number of base and update variables), and varying τ should result in a family of curves showing

performance decreasing with complexity. Figure 11 also demonstrates this type of behavior.

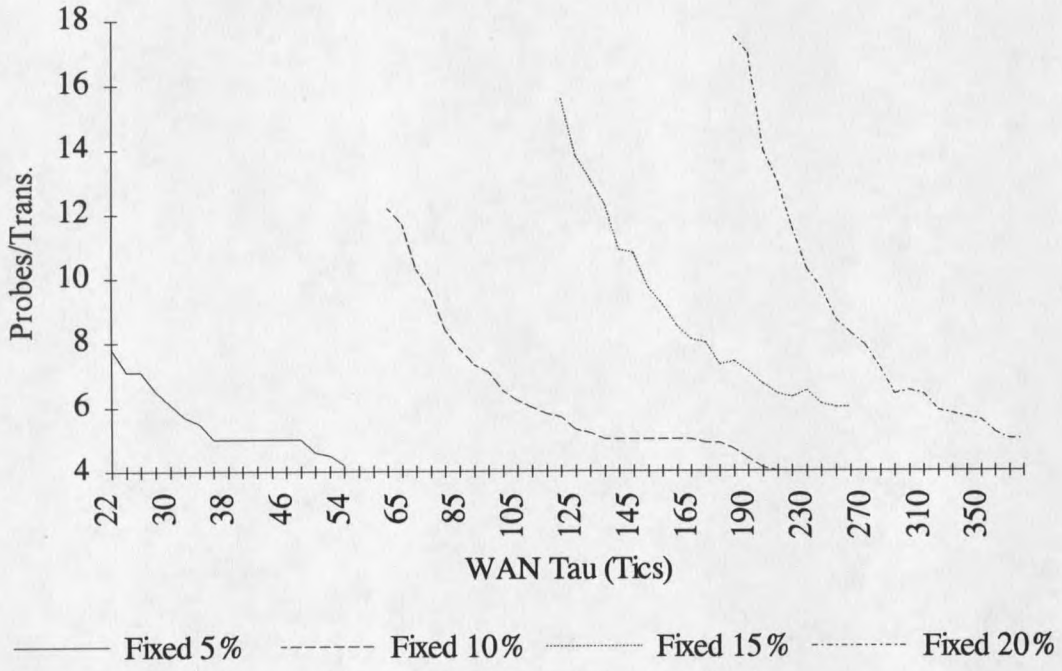


Figure 11. Average Probes/Transaction vs. τ for WAN workloads.

In addition, with exponential service times such as the network latencies used in this simulation, there should be an exponential relationship between transaction response time and transaction throughput over a range of τ. When throughput is plotted against response time, the curve should be vertically asymptotic to the maximum interarrival rate, λ_{max} where $\lambda_{max} = 1/\tau_{min}$. Figure 12 shows the results for the average number of vote probes experienced by LAN workload transactions for each transaction size over a range of τ for the Random Path vote order strategy. Table 5 shows the values of λ_{max} for each transaction complexity. The curve for Random 10% clearly shows a reasonable approximation of the expected asymptotic behavior.

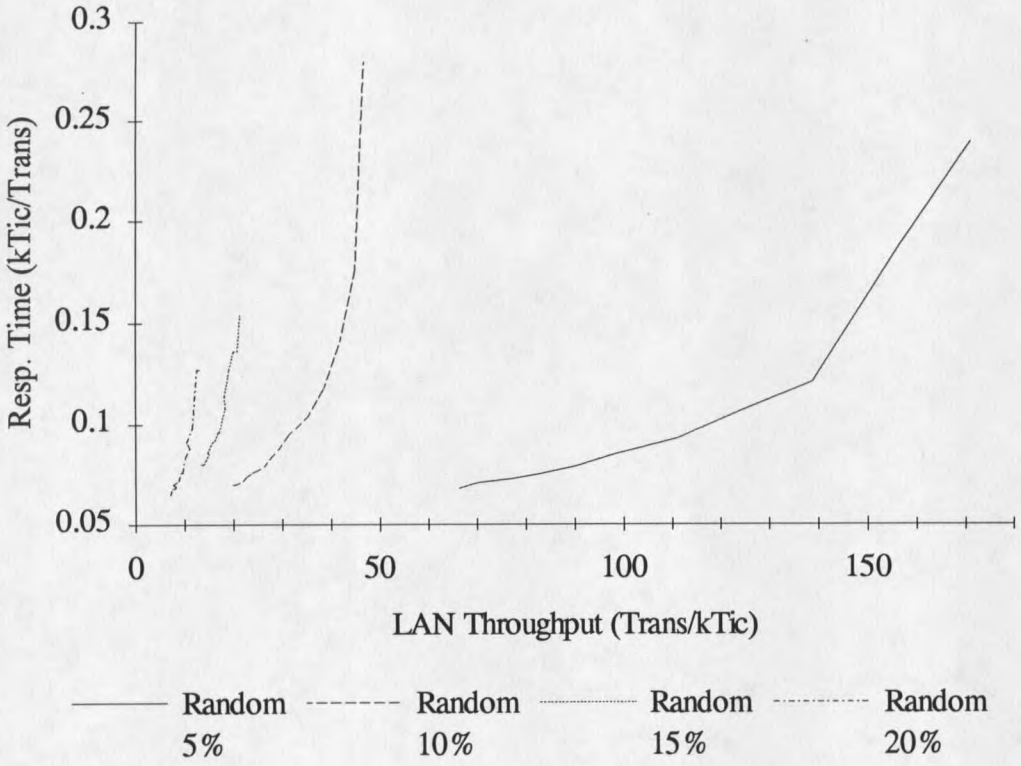


Figure 12. Average Response Time vs. Throughput for LAN workloads.

Workload	λ_{max}	Workload	λ_{max}
5%	200.0	15%	22.7
10%	50.0	20%	13.3

Table 5. Maximum interarrival rate for LAN workloads.

Figures 11 and 12 are examples of the behavior of the simulation under various workloads. A review of all the results shows similar behavior improving confidence in the usefulness of the simulation. The results demonstrate several points regarding the relative merits of the two rejected query refresh methods and the three vote order strategies evaluated.

Rejected Transaction Refresh Method Results

Figure 13 is typical of the results obtained when comparing the two transaction refresh methods. The method of refreshing base variables before resubmitting a rejected transaction has little effect on the number of vote probes required to complete the average transaction. It makes little difference whether the initiating AP queries the initial DBMP or the rejecting DBMP for fresh base variable values.

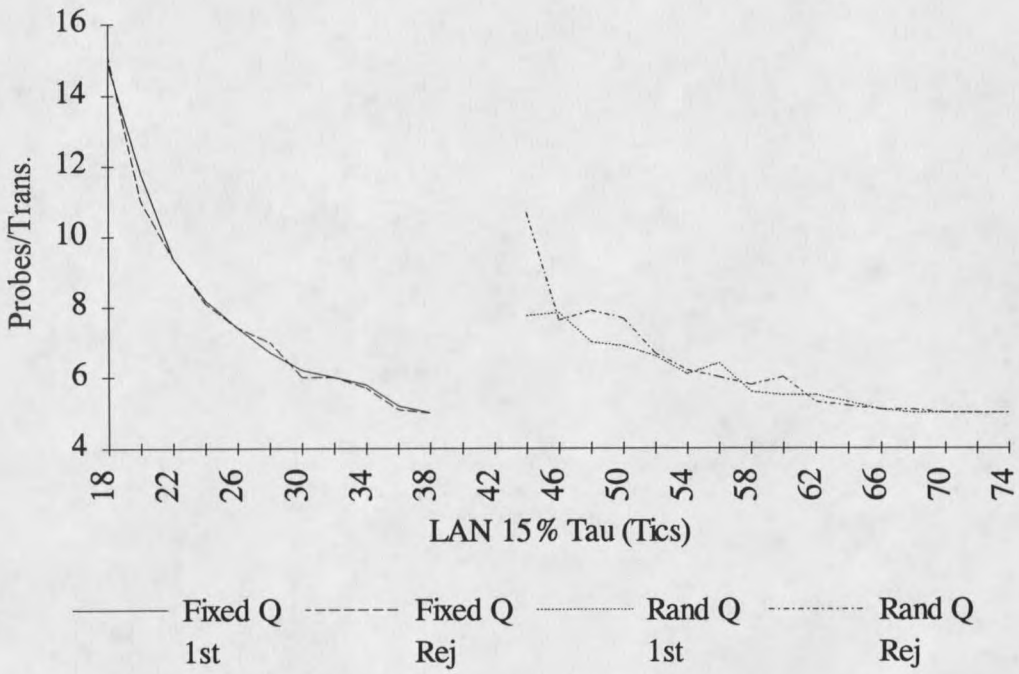


Figure 13. Average LAN 15% Probes/Transaction vs. τ showing refresh methods.

Figure 13 shows the results for LAN 15% workload with the Fixed Path and the Random Path vote order strategies and both transaction refresh methods.

As τ increases, the system is less busy and there are fewer opportunities for conflicting transactions to be rejected. Thus there should be little difference at the higher interarrival rates. As τ decreases, there will be more concurrent transaction activity in the system resulting in more conflicts and rejections. With a busy system,

the rejecting DBMP would be expected to have more recent database time stamps than the initial DBMP. A review of several simulation traces shows that the particular communication latencies implemented by the two simulated network topologies allowed outstanding transaction updates to reach the initial DBMP before the originating AP could resubmit a rejected transaction. Thus the initial DBMP usually had current database elements for the requery. Different network topologies may give different results.

Because there is no demonstrated advantage to either rejected transaction refresh method, the remainder of the analysis will consider only results from the refresh method in which the originating AP requerys the first DBMP in the daisy chain for current base element values and time stamps and will ignore the method which queries the rejecting DBMP.

Vote Order Strategies

With many distributed applications, workload balancing is a primary consideration. By spreading system activity evenly over a set of processors, improved system throughput and response time is achieved. In the case of replicated databases and files systems managed by some form of quorum consensus, this is not the case.

The potential for conflicting concurrent transactions leads to rejections and transaction restarts. The sooner a conflict is detected and one transaction aborted, the sooner the rejected transaction can be restarted. The three vote order strategies evaluated here show different conflict detection performance.

Random Path order voting would seem to follow the workload distribution tenant mentioned above. However, with each transaction taking a different random path through the set of DBMPs, conflicts may not be detected until late in the update synchronization process. In addition, Random Path voting makes no use of

communication latency knowledge. The average transaction will expect average communications performance at best. The worst case communications latency is the longest path.

Fixed Path order voting cause every transaction to follow the same path through the network. DBMPs vote on each transaction in the same order. This means that the same DBMP usually casts the deciding majority vote for all transactions. This same DBMP will usually have the most recent set of database elements and will be in the best position to detect and resolve transaction conflicts. There will be exceptions to this due to variance in communications latency but, in general, it will hold. While it seems that the workload is concentrated at this one DBMP, a majority of DBMPs will have participated in the decision process and all DBMPs will have processed accepted and rejected transactions. The additional work of processing the deciding vote could be managed by placing additional computing resources at the host of the DBMP in the majority position.

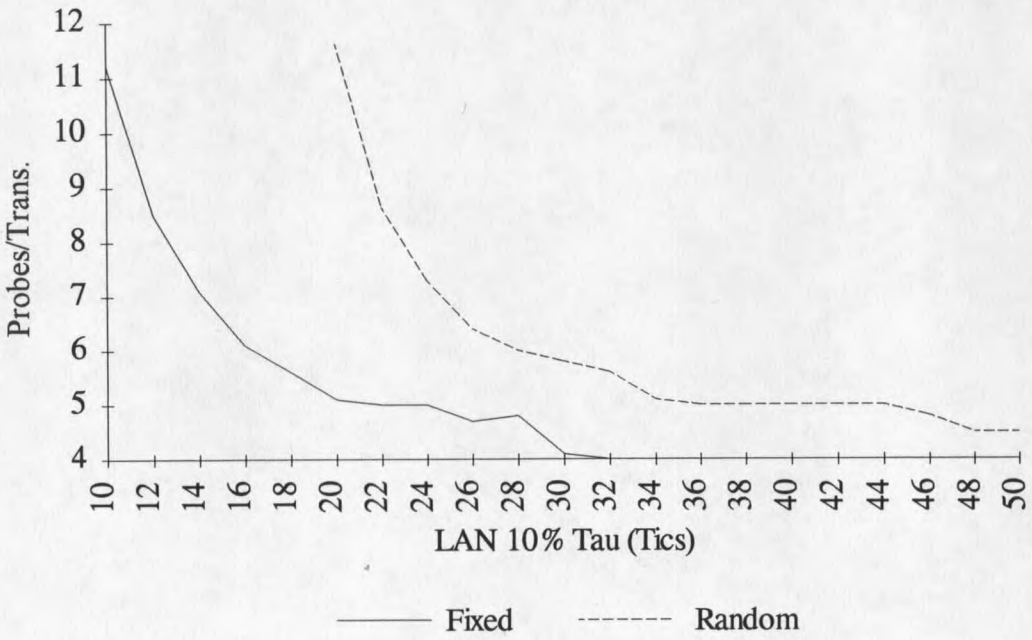


Figure 14. Average Probes/Transaction vs. τ for LAN 10% workload.

Figures 14 and 15 demonstrate the advantage of Fixed Path order voting over Random Path order voting for the LAN workloads. Both the average vote probes per transaction and the response time and throughput indices are better for the Fixed Path voting.

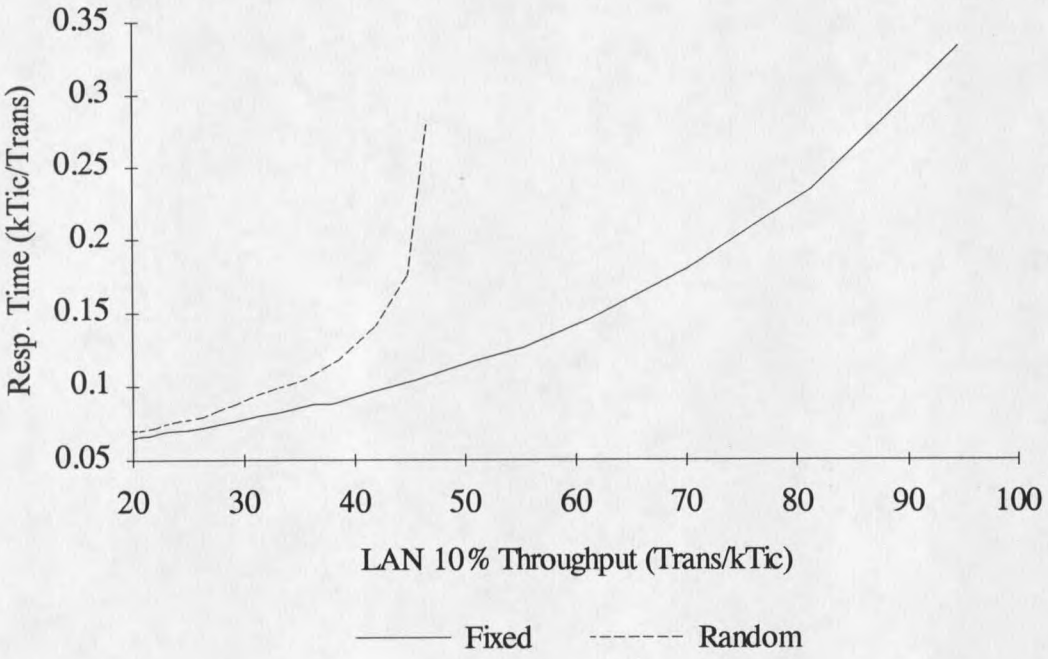


Figure 15. Response Time vs. Throughput for LAN 10% workload.

As Figures 16 and 17 show, the Fixed Path order voting and Random Path order voting strategies have the same relative performance on the WAN network topology 10% workload as was seen on the LAN workloads. The Shortest Paths vote order strategy gives mixed results. While the average transaction requires more probes for votes than the average Fixed Path transaction, the improvement in average communications latency gives generally better response time and throughput performance.

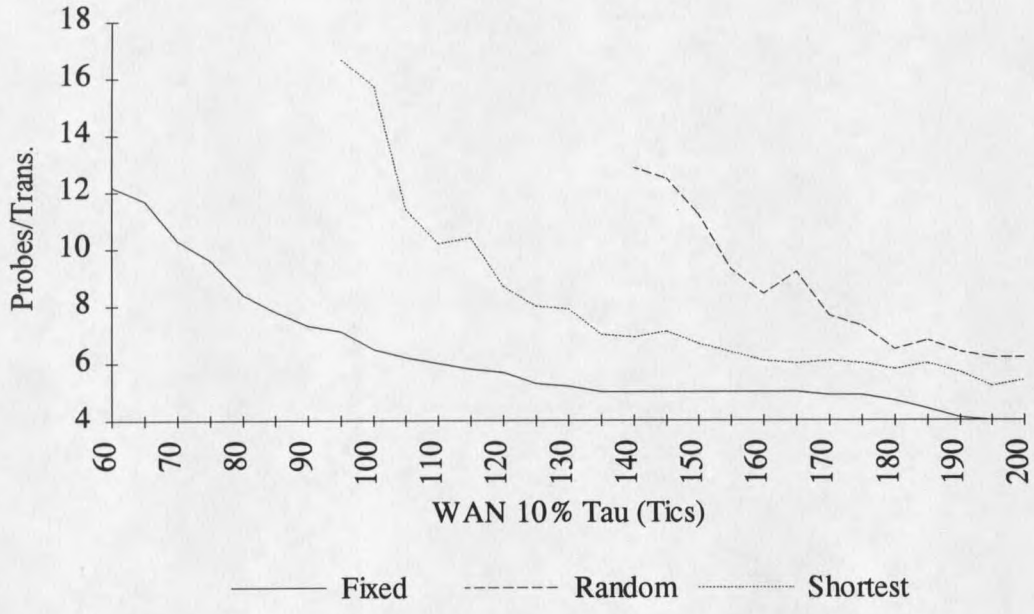


Figure 16. Average Probes/Transaction vs. τ for WAN 10% workload.

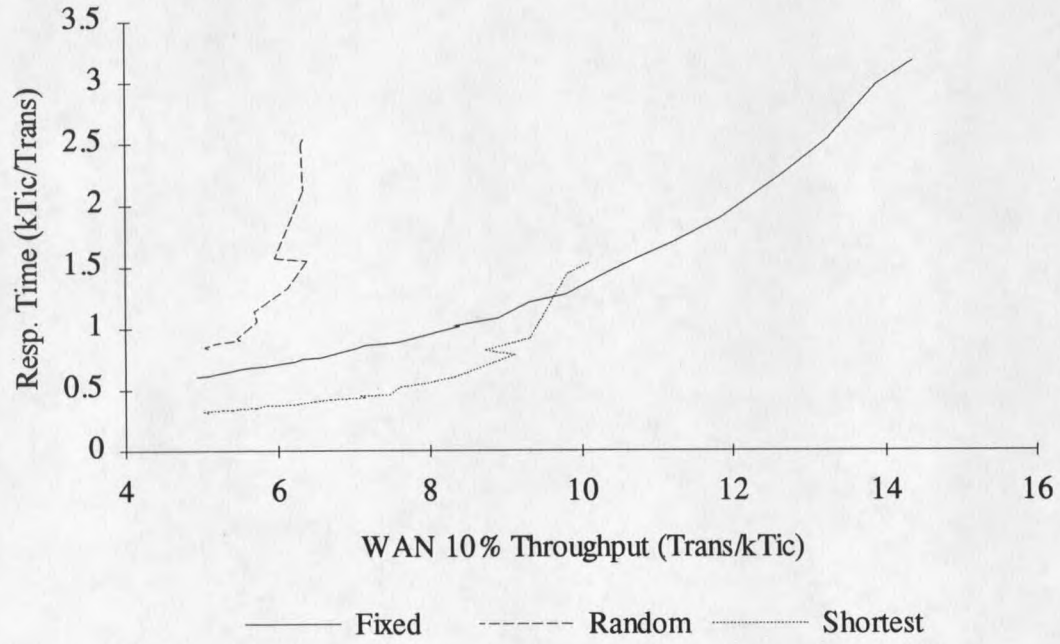


Figure 17. Response Time vs. Throughput for WAN 10% workload.

Each transaction using the Shortest Paths strategy may randomly select one daisy chain from a set of similarly performing paths for each transaction. If this was changed such that all transactions originating at the same AP always used the same shortest path, the strategy should perform closer to Fixed Path voting in terms of average number of probes for votes per transaction. This should also result in additional improvements in response time and throughput for the strategy by reducing conflict detection time.

Due to the variable communications latencies of various paths through the WAN network topology, the Random Path order strategy and the Shortest Paths order strategy give erratic results over the range of τ. This may be improved if more simulation runs were averaged for each data point.

The selected findings shown here are characteristic of the results of all the workloads when executed on the simulation.

CHAPTER 6

CONCLUSION

This paper presented three strategies for ordering votes when synchronizing updates to a replicated database system managed by Thomas's Quorum Consensus. Fixed Path voting requires all update transactions to follow the same daisy chain of voting copies during update synchronization. Random Path voting allows each transaction to arbitrarily select an ordering of database copies. Shortest Paths voting allows a transaction to identify and follow the path with the lowest expected communications latency.

The three strategies were evaluated for average vote probes per transaction, transaction response time, and transaction throughput. The evaluation was done by means of a distributed replicated database simulation. The simulation modeled several network topologies and transaction workloads allowing the three strategies to be tested in a range of environments.

The results demonstrate that the usual advantages of evenly distributing work in a distributed system are overweighed by the need for early update conflict detection and resolution. A fixed transaction path through the set of voting copies was shown to have this property. If all transactions follow the same vote order, the database copy in the majority position is able to effectively arbitrate access to the replicated database suite. Rejected transactions may be quickly restarted by the originating process.

Opportunities for additional investigation include evaluating the performance of an optimized fixed path. This strategy would specify a single vote order which is

optimized for all active application processes. Optimized would mean that the sum of the expected communications latency of the DBMP daisy chain plus the expected latencies from each AP to the first DBMP in the chain is minimized. The optimized fixed path strategy would combine the conflict detection advantages of the arbitrary Fixed Path strategy examined in this paper with the response time performance of the Shortest Paths strategy. The optimized fixed path strategy should perform well in both the average number of vote probes needed to accept a transaction (early conflict detection) and in average transaction response time. Reducing the number of vote probes reduces both the system workload and the average number of communications required to complete a transaction. These two measures determine, in large part, a distributed database system's throughput capability.

Two methods for refreshing rejected transactions before resubmission were also evaluated. The first was requerying the initial database copy for fresh base variables. The second method was querying the database copy which rejected the transaction. There was no advantage to either method in the two simulation environments used in the evaluation. Further work may find a network configuration favoring one or the other.

REFERENCES CITED

[1]R. H. Thomas, "A majority consensus approach to concurrency control," *ACM Trans. Database Syst.*, vol. 4, no. 2, June 1979, pp. 180-209.

[2]E. W. Dijkstra, "Cooperating Sequential Processes," *Technical Report EWD-123*, Technological University, Eindhoven, The Netherlands, 1965.

[3]Sun Microsystems, *Sun Network File System (NFS) Reference Manual*, Sun Microsystems, Mountain View, CA, 1987.

[4]D. K. Gifford, "Weighted voting for replicated data," *Proc. 7th ACM SIGOPS Symp. Oper. Syst. Principles,* Pacific Grove, CA , Dec. 1979, pp 150-159.

[5]A. Kumar, "Hierarchical quorum consensus: a new algorithm for managing replicated data," *IEEE Trans. Comput.*, vol. 40, no. 9, Sept. 1991, pp. 996-1003.

[6]M. Maekawa, "A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Trans. Comput. Syst.*, vol. 3, no. 2, May 1985, pp. 145-159.

[7]D. Agrawal and A. El Abbadi, "An efficient and fault-tolerant algorithm for distributed mutual exclusion," *Proc. Eight Annu. ACM Symp. Principles Distributed Comput.*, 1989.

MONTANA STATE UNIVERSITY LIBRARIES
3 1762 10101790 1

HOUCHEN
BINDERY LTD
UTICA/OMAHA
NE.