PERFORMANCE EVALUATION OF REPLICA CONTROL STRATEGIES IN A DISTRIBUTED SYSTEM

by

Simon John Marwood

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

C Copyright by Simon John Marwood 1994

ABSTRACT

Performance Evaluation of Replica Control

Strategies in a Distributed System

Master of Science, 1994

Simon John Marwood

Graduate Department of Computer Science

University of Toronto

In a distributed system, users at many different sites may need to access the same data concurrently. Through the development of object replication, accessibility of this data has been greatly enhanced. Inherent with this replication is a need to control access to the data to ensure correctness. Correctness is most difficult to maintain in the presence of network partitions. Replica control strategies have been developed to manage this access using a variety of techniques, which involve availability and timing trade-offs.

In this study, the Gifford, Missing Writes, Herlihy and Hierarchical replica control strategies are implemented in a simulation of a transaction processing system. The strategies are subjected to identical input parameters, generating timing and availability statistics under both partitioned and non-partitioned operation. The strategy with the best overall performance is Herlihy's, unless network partitions are frequent and timing is crucial, in which case Gifford's method is superior.

Acknowledgements

Special thanks to Gene Davis, Scott Graham, Vassos Hadzilacos,

Maurice Herlihy, Kariné Kalayjian, Akhil Kumar, François Pitt,

Karen Reid, Ken Sevcik and Sam Toueg.

Table of Contents

1 Introduction 1

1.1 Model of Transaction Processing 7

1.2 Model of Partitioned Behaviour 10

2 Related Work 14

2.1 Replica Control Strategies 14

2.1.1 Gifford 15

2.1.2 Missing Writes 16

2.1.3 Dynamic Quorum Adjustment 19

2.1.4 Views 24

2.1.5 Hierarchical Quorum Consensus 29

2.1.6 Dynamic Ordering of Transactions 31

2.1.7 Multi-Class Replication-Control Protocol 32

2.1.8 Regeneration with Virtual Copies 35

2.2 Availability Bounds 39

3 Simulation Model 44

3.1 Simulation Inputs 45

3.2 Transaction Behaviour 52

3.2.1 Transaction Generation and Arrivals 52

3.2.2 Transaction Queuing and Processing 57

3.2.3 Transaction Departures 59

3.3 Simulation Outputs 60
3.4 Statistical Validity 63

4 Results 65
4.1 Overview 66
4.2 Replica Control Strategy Analysis 73
4.2.1 Gifford 73
4.2.2 Missing Writes 75
4.2.3 Herlihy 77
4.2.4 Hierarchical 77
4.3 Additional Experiments 83
4.3.1 Output Sensitivity 84
4.3.2 Centralized Operation 86
4.3.3 Inter-arrival time variation 88
4.3.4 Transaction Mix Variation 89

5 Conclusion 94
5.1 Overview 94
5.2 Future Work 97

References 99

Appendix A 102

Appendix B 127

List of Tables

Table 1: % Increase in Average Residence Times over Base Method (Non-partitioned operation) 67

Table 2: % of Transactions Completing Under Partitioned Operation 69

Table 3: % Increase in Average Residence Times over Base Method (Partitioned operation) 70

Table 4: Ranking of Strategies under Partitioned/Non-Partitioned Operation and Importance of Availability/Average Residence Time . . . 71

Table 5: Availability Bounds 72

Table 6: % of Transactions Completing Under Partitioned Operation For Hierarchical Method and 15-site Replication 79

Table 7: Average Residence Times and Availability for 1-copy replication 88

Table 8: Average Residence Times (Non-Partitioned Operation) for new Transaction Mix 90

Table 9: Availability for new Transaction Mix 92

1 Introduction

In a distributed system, users at many different sites may need to access (query and update) the same data concurrently, often data which is stored in a database. Through the development of concurrency control methods for centralized databases, this has been made possible without violating mutual exclusion rules.

However, centralized databases accessed via distributed networks are highly susceptible to both communication and site failures. In the event that the site which stores the database is cut off from the rest of the system due to a failure, the data becomes inaccessible. For this reason, the development of distributed databases with entity replication has been desirable.

In theory, replication has two main advantages: the greater likelihood of accessibility in the presence of network failures, and a decrease in the number of expensive cross-network accesses for queries, as there is a greater chance of the particular entity being stored nearby.

However, the replication of data does entail problems of its own: *mutual consistency* problems (i.e., ensuring that all copies of the entity have the same value), *integrity* problems (i.e., ensuring that the data meets certain integrity constraints), and *availability* problems (i.e., ensuring maximal access to data during network failures). The most difficult types of failures to deal with are *network partitions*, where some set of sites in the network are unable to communicate with some other disjoint set of sites. It is

desirable that the above problems are dealt with despite network partitions.

One of the tradeoffs that must be made in a distributed database in the presence of failures is that of correctness versus availability. In the ideal world both could be maximized simultaneously, but in the presence of network partitions, this is not possible.

Suppose a network partition took place: all transactions could be permitted to continue operation within their partitions (assuming that the data is available), and then compensating transactions may be run upon recovery, to bring all sites back to a consistent state. This technique would be acceptable for some applications where availability of the data is more important than a guarantee of correctness. Such a policy may lead to highly undesirable external results, however. Davidson et al. [1] present several practical examples of this issue.

On the other hand, correctness can be guaranteed simply by suspending all transactions in all of the partitions but one. The changes to replicated data made in this one active partition can then be forwarded to the other sites when the network recovers. Internally, consistency could thus be guaranteed, and it would also be ensured at the other sites, assuming a correct failure recovery algorithm. For some applications, it is essential that only the most up-to-date, correct information is ever seen by a transaction, which would be guaranteed using such a strategy. However, availability of data would be severely restricted. This situation may be impractical; if many sites nationwide are down in a commercial operation, customers may be lost to the firm.

Replica control strategies have been developed which ensure one-copy serializability even in the face of network partitions, while maintaining a high degree of availability. Many such replica control schemes have been proposed; they may be distinguished according to the trade-offs that are made between consistency and availability.

Pessimistic strategies limit the availability of data objects by restricting the transaction processing which can occur in the system while the network is partitioned. One-copy serializability is ensured by guaranteeing that no transaction in one partition is able to execute operations on replicated data which could potentially conflict with operations in another partition. When the partition is repaired, any changes successfully completed to replicated data in one partition can easily be transferred to the other partitions, with a guarantee that the resulting database state will be consistent.

Optimistic strategies are concerned more with maximizing the amount of processing which can continue even when the network is partitioned. Consistency is maintained within individual partitions, as a concurrency control protocol for non-partitioned data can be used to ensure one-copy serializability within the partition. However, consistency across different partitions is often violated by optimistic protocols, which forces more complex remedies to be applied when the partition is repaired, in order to bring the database back to a globally consistent state. Typical strategies to resolve such inconsistencies include rolling back transactions which have been committed in only one partition and applying compensating transactions which nullify or change the effects on replicated data of previously committed transactions. If significant activity outside of the

database has occurred because of a transaction which needs to be rolled back or corrected, it may be difficult to repair or undo. Hence, optimistic strategies, while providing greater availability during network partitioning, must be used with care.

This thesis deals only with pessimistic replica control strategies. A transaction processing system simulation is presented, and four proposed replica control strategies are implemented subject to a uniform context. The simulation is run for failure-free and partitioned network states, to model the behaviour of each strategy under both conditions. Additionally, simulation runs are performed for three different degrees of object replication. Availability and timing results are derived from the output, and these statistics provide insight into the effectiveness of each strategy.

The thesis is organized as follows: the remainder of Chapter 1 will provide a set of common assumptions and definitions relating to distributed database systems. Chapter 2 will examine background material in the area of pessimistic replica control strategies. Chapter 3 presents the transaction processing system model, and discusses its input parameters, transaction behaviour within the system, the performance measures that the system generates, and the validity of the results. Chapter 4 presents an overview of the results obtained from the simulation, and also discusses any particularly interesting findings for each strategy. Finally, Chapter 5 contains concluding remarks and suggestions for further work in this area.

1.1 Model of Transaction Processing

A *distributed system* is a collection of *n* processing sites numbered *1* through *n* connected by a set of bidirectional communication links. A *distributed database* is a collection of logical data objects distributed over this system, with each object potentially replicated at different sites. The system is subject to a variety of potential failures, including site crashes, asynchronous communication delays and communication link failures. Network sites and communication links are considered to have *independent failure modes*, so that a failure at one system component will not induce other failures elsewhere. A *network partition* occurs when some set of sites, a *partition group*, is unable to communicate with another set of sites for some significant period of time. A network which partitions into two partition groups is called a *simple partition*. Any larger number of resulting partition groups is a *multiple partition.* If adequate precautions are not taken, network partitions can cause multiple updates to occur to the same logical data item in different partitions, thereby violating correctness constraints on the database. It is also desirable to limit the amount of extra processing which must take place to guarantee correctness without adversely affecting system performance in the absence of failures.

In a distributed database system, a logical data item *x* is implemented by replicating the object at different network sites. These replicas are the *physical copies* of *x*. A *transaction* is a program which executes a set of reads and writes on database items. Each transaction has a *home site*, which is the network node at which it resides. The transaction eventually either successfully completes (commits), or fails to complete all of

its operations (aborts). Some transactions are read-only, and perform no database writes. These are known as *queries*. Other transactions *update* the database. The set of entities that a transaction reads is referred to as its *read set*. A transaction's *write set* is the set of entities that it writes. Similarly, a *read (write) quorum* is the number of physical copies of an object which must be available for a transaction to complete a read (write) request. It is assumed that transactions are internally correct: if executed independently, the database would be transformed from one correct state to another correct state. Writes performed by a transaction are not installed into the physical copies of an object until the transaction successfully commits.

When a transaction has completed its operations, a decision must be made as to whether the updates can be successfully installed without violating consistency constraints. In many transaction processing algorithms, this is performed in two phases. In the first phase, permission is granted by all of the entity copies to update data, and the copies are locked so that no other transaction can disrupt the atomicity of the update. Entity copies' sites may also reject the update request, which will cause the transaction to either re-start or abort. This first phase is often called the *pre-commit* phase. In the *two-phase commit protocol (2PC)*, the value of the update is also written onto secure storage at the remote sites. In the event that the update intention was accepted at all sites, the transaction *commits*. At this point, the transaction may not be aborted or restarted, and it is considered to have completed. The application which created the transaction can assume that all updates have been performed, and that all output is correct. The home site then sends a commit message to every site, and the update is permanently installed. In the 2PC protocol, this simply entails applying the update already present in the remote sites' stable

storage. In the event of failures, there may be an arbitrarily long delay between the update occurring at the transaction's home site and its application at the remote entity copies.

Transactions interact with each other by *conflicting*. Two transactions conflict if they both perform operations on a data object, and at least one of the operations is a write. Operations which conflict are important because, if not properly controlled, they may lead to inconsistencies in the final database state.

A centralized database system is assumed to be correct if the following two properties are met. These two conditions present to the user the illusion of operations on a virtual, centralized copy of the database; it appears that the operations are indivisible and are operating independently. The conditions are:

- a transaction's execution is atomic. It either runs correctly to completion, with all of its operations running successfully, or it is aborted, and no changes to the database take place (the property of *atomic commitment*).

- any execution of several concurrent transactions can be duplicated by some serial execution of the same transactions (the property of *serializability*).

Some database systems also impose *integrity constraints*, which are application-dependent (e.g., a bank balance having to be non-negative at the end of the transaction). It is generally assumed that these constraints are dealt with by the transaction (i.e., it aborts if it realizes that the constraints will not be met).

Distributed databases entail another constraint: that of *one-copy serializability*. This property ensures that any execution of transactions in a distributed database which affects multiple physical copies of the same logical entity has the same effect on these copies as a serial execution of the same operations would have in a non-replicated environment. The protocols that guarantee that this condition is satisfied are known as *replica control* algorithms.

1.2 Model of Partitioned Behaviour

As discussed above, one major problem facing distributed databases is that of network partitions. The model of partitioned behaviour to be used assumes that one-copy serializability is the correctness criterion.

If the network is partitioned, decisions must be made on which transactions can be run in the disjoint partitions without violating the correctness criterion. Thus, it must be ensured that correctness is maintained among sites in the partition where the transaction is run, and that the actions in one partition do not conflict with what may be occurring in another partition, thereby maintaining correctness across the entire database.

The first problem is fairly easy to solve, if it is guaranteed that a transaction's home site is capable of detecting a network partition, and determining which sites belong to its partition group. Upon detecting such a failure, the site simply adapts the replica control algorithm for the entire network to work only within its partition group. Trivially,

one-copy serializability is thus ensured within the particular partition.

Ensuring one-copy serializability across the entire network is more complex: simply running the replica control algorithm in each partition does not ensure correctness across the whole network, as conflicting transactions may be perfectly acceptable amongst individual partitions, but may well violate one-copy serializability across the system as a whole.

Another problem faced by replica control algorithms in the presence of network partitioning is that of the *dangling pre-commit* problem. There may be transactions operating at the time of the partitioning of the network, and these transactions may access physical copies of an object in several partition groups. From the model of transaction processing, when a physical copy pre-commits itself to a transaction, the copy becomes locked, and is unavailable to other transactions. At the time of the partitioning, a pre-committed object may be unable to determine whether it should commit or abort the update for which it has been pre-committed. It cannot commit the update, as the transaction may have been rejected at the pre-commit stage elsewhere, or the transaction may have been aborted for some reason. Similarly, it cannot simply abort the update, as it does not know whether other sites may have already applied the updates to their physical copies. In either event, the wrong action would result in a violation of mutual consistency.

It may be possible to alleviate this problem slightly, however, depending on how the partition has taken place. Once a site *S* with a physical copy which has pre-committed

detects a network partition, it can communicate with other sites to determine what knowledge they may have of the transaction's status. There are three possible cases to consider:

• some site(s) in the partition group of *S* need to be pre-committed before the transaction can be committed, and such a pre-commit has not yet taken place.
ACTION: inform the site(s) that a network partition has taken place, and to reject any pre-commit messages that they receive for the transaction in question. *S* can then release locks on its physical copies of the pre-committed object.

• some site(s) in the partition group of *S* know that the transaction has either aborted or committed.
ACTION: if the transaction has committed, perform the update and release locks on physical copies of the object. If the transaction has aborted, then simply release the locks.

• no sites in the partition group of *S* have any knowledge of the transaction's status, and all have pre-committed.
ACTION: in this case, no action can be taken. The transaction is blocked, the locks must remain in place on the physical copies at all sites in the partition group, and the dangling pre-commit problem remains.

Upon resolution of the partition problem, the dangling pre-commits may be resolved using standard database failure recovery procedures, and any necessary updates

may be made.

The problem with queries during network partitions is that they will not necessarily yield the most current entity value. If availability is of prime importance in the system, however, the replica control protocol may permit read-only transactions to *run in the past* so long as one-copy serializability is not violated. As such transactions have no other effects, it may not matter that the value read is out of date. Transactions running in the past which read an object's value are serialized *after* the update whose value they read, but *before* any further updates which may change the value of the object.

2 Related Work

This chapter presents background material in the field of replica control strategies. A number of strategies are introduced and summarized, with some references to improvements or shortcomings compared to the other strategies. Additionally, previous work in determining availability bounds is presented. The added consideration of write-only transactions allows an expansion of these bounds, as shown in Chapter 4.

The chapter is organized as follows: Section 2.2 examines each of the protocols in turn, Section 2.3 examines upper bounds on availability, and Section 2.4 presents some concluding remarks.

2.1 Replica Control Strategies

This section examines a number of proposed replica control protocols, and the restrictions that they place on transaction processing. It is assumed that a correct concurrency control protocol exists, which will guarantee one-copy serializability within individual partitions, and that no transactions are in progress at the time of the partitioning. Any transaction which is in progress at the time of the failure and which has not yet committed may be aborted. Transactions which have committed but whose updates are not complete at all sites will block while the sites are unavailable. In-progress transactions will not be dealt with further.

The determination by a site that a partition has occurred is an as yet unsolved problem. This difficulty is due to an inability to differentiate between the situation where a site can no longer reach another site (due to a partition) and the situation where there is no partition, but extremely lengthy communication delays in the network. Detection of recovery is another difficult problem, as partitioning may re-occur due to another communication link failure around the same time that the initial failure is repaired.

2.1.1 Gifford

The Gifford method [2] assigns some number of votes to each replicated copy of an object. At the time that a transaction wishes to perform a read or write operation on the object, it must gather a ceratin number of these votes, denoted a *quorum*. The read quorum, r, and the write quorum w must satisfy the *quorum invariant* $r + w > v$, where v is the total number of votes assigned to object copies.

This invariant ensures that there is a non-empty intersection between every read quorum and every write quorum in the system, guaranteeing that any read operation will access the most current version of an object. Version numbers are used to determine which of the accessed copies has the most recent version. Most interpretations of the Gifford method also use the additional requirement that $w > v/2$ so that writes to the same object in different partitions are forbidden.

Weighted voting is not considered in this study, as it has been shown [3] that not

all valid quorum assignments can be achieved using weighted voting.

2.1.2 Missing Writes

Gifford's method requires that read and write quorums be obtained for the read and write sets of a transaction in order for these operations to be undertaken. Eager and Sevcik [4] note that while these conditions ensure consistency in the presence of failures, they are not necessary in all instances, and in fact may seriously degrade performance when there are no failures. Their work strives to achieve *maximal partial operability*, which provides the highest level of transaction activity possible when failures occur, without seriously impacting on system performance at other times.

In the model that is presented in the paper, transactions normally only interact with the database to perform read and write operations. Only when failures have been detected is additional interaction required. This interaction, which includes additional use of quorums and special posting operations, is detailed later. Transactions therefore operate in two distinct modes: *normal* mode, when no failures have been detected by the transaction, and *failure* mode, when failures have been detected.

To model conflicts between transactions, the concept of *precedence graphs* is used. These graphs are used to model the ordering of transactions, and to check for serializability across network partitions. Vertices in the precedence graphs denote transactions in the system, whereas edges represent conflicts between transactions.

Transactions in different partitions conflict with each other if a transaction reads the value of an item which is updated by a transaction in another partition. A transaction's read set is assumed to include its write set, hence write-write conflicts are modelled as a pair of read-write conflicts.

Transactions running within a partition are modelled as a serial history of their execution, their read sets and their write sets. Transaction execution within a partition is trivially serializable, so such a history is guaranteed to exist. Let the vertex T_{ij} denote the j^{th} transaction to execute in the serial history of partition i. Edges (T_{ij},T_{ik}) between transactions in the same partition show precedence in that a transaction T_{ij} read a copy of some entity that was later written by T_{ik}, or vice-versa. Due to the serializability of transactions within a partition, this part of the graph is guaranteed to be acyclic. However, conflicts between transactions in different partitions must also be accounted for in this model. Edges (T_{ij},T_{kl}) in the precedence graph represent the situation where T_{ij} reads some entity that is updated by some transaction T_{kl} in another partition. T_{ij} will thus have read an "older" value of the entity than the value present after T_{kl}'s update. Again, this interference signals that there must exist a serial execution of the transactions which will maintain the ordering of the edges.

If there are cycles in the precedence graph, then one-copy serializability is violated. However, if the graph is acyclic, there exists some serial schedule which emulates the operations of the concurrent transactions running in different partitions.

In normal mode, transactions are unaware of any failures, and may therefore use

a read-one, write-all approach when accessing replicated data items. If a write operation fails, the transaction switches to failure mode, which will require the posting of *missing update information* at the sites it accesses (this concept is explained below). Any transaction that subsequently attempts to read or write an object located at a site at which missing update information is posted must restart in failure mode, unless the network failure has been repaired, and the site is able to receive its missing updates. Read-only transactions, however, may be permitted to *run in the past*, which allows sites that have become isolated from the rest of the network to continue to execute queries on old values of shared data in normal mode. Such execution is permitted so long as the transactions that run in the past do not create cycles in the precedence graph of committed transactions.

In failure mode, transactions have become aware of a missed update at some site, and thus of a network failure. Transactions running in failure mode must obtain quorums for every object in their read and write sets in order to commit. In addition, these transactions must pass the missing update information of which they are aware to any transactions that depend on them. This missing update information contains:

- which entity copies have not been updated
- which transactions created the missing updates
- which transactions the information should be passed to (i.e. what operations in a transaction that visits the site should cause the missing update information to be delivered to that transaction)

In most systems, a transaction does not know *a priori* which other transactions depend on its actions. Hence, the information is not explicitly passed to other transactions, but is simply flagged to be passed to those transactions that execute specific operations on entity copies at the site. It is these operations that conflict with the original transaction's actions, and cause the dependence.

Eventually, the network partition will be repaired, and the missing update information will propagate to the sites that originally missed being updated (via transactions which successfully access these sites), and the updates can then successfully be made. The relevant missing update information is then removed from the other sites at which it was posted, and transactions can return to normal mode operation.

2.1.3 Dynamic Quorum Adjustment

Both Gifford's method and the Missing Writes algorithm use static read and write quorums which must be defined *a priori*. Gifford's method always uses the quorums when accessing shared data, while Eager and Sevcik's method only requires read quorums in the presence of network failures. The next method presented introduces the concept of dynamic quorums, which allow a replicated object's read and write quorums to adjust when the state of the underlying network changes [5]. This method permits some transactions that would be stalled under one quorum assignment to proceed to commitment using some other assignment, without violating one-copy serializability. Herlihy's paper centres around two techniques for adjusting quorum sizes: quorum

inflation and deflation, which will be discussed after the basic model of transaction processing is examined.

In this model, each object is assigned a set of read and write quorums, its *quorum assignment*. Each of the quorums in the set is assigned to a particular *level* of transactions. Transactions determine their level when they are created. Serializability is ensured by a concurrency control method, which serializes transactions at the same level using a standard timestamping method, while transactions on different levels are serialized using *ratchet locks*. These locks ensure that transactions with higher levels are serialized after transactions with lower levels. Objects in the system consist of two parts: repositories and front ends. Repositories are the actual replicated data values of the object, whereas front ends interface with the transactions. Front ends are static and can thus be replicated at will. Each repository stores the most recent value it knows about at each level, and associates a timestamp with this value.

Ratchet locks are associated with each copy of a replicated object, and they record the highest level of any transaction that has read that copy. Subsequent write attempts by transactions with a level lower than that recorded at the lock are rejected. In this manner, we are guaranteed that higher level transactions are always guaranteed to have seen the value of the copy after all successful lower level transactions have committed, ensuring serializability. Writes to an object are *enabled* at some level n if none of the ratchet locks on copies of the object exceed n.

To guarantee correctness, all quorums must satisfy the *quorum intersection*

invariant:

> If writes to an object are enabled at level *n*, then all write quorums at level *n* must intersect every read quorum of level *n* and greater.

This invariant ensures that transactions at different levels will be serialized as described above, as any write at a lower level will necessarily encounter some copy of the object which has been read at a higher level, and thus be prohibited from performing its update to that object.

To perform a read operation, an *n*-level transaction sends the read request to a front-end for the object, which forwards it to each of the copies in the level *n* read quorum. The objects respond with the highest time-stamped value that they have for level *n* or less, and the associated timestamp. The front end collects these values, and returns the value with the highest level and timestamp to the client transaction. Upon commitment of the transaction, each ratchet lock in the quorum is set to the maximum of the reader's level and its current value.

To perform a write operation, the transaction sends its new value to a front-end, together with the transaction's time stamp and level. The front-end accesses the write quorum, ensures that the transaction's level is greater than each of the ratchet locks in the quorum, and then forwards the updates, which become permanent when the transaction commits.

Quorum inflation is a technique used by transactions when they are unable to commit at their current level. Transactions may have to abort at their original level for a variety of reasons, including an inability to acquire quorums for operations they wish to perform, or due to conflicts with ratchet locks. It may be possible to find a higher level that has an attainable quorum, or at which all of the objects on which the objects wishes to perform updates have writes enabled. Quorum inflation makes it easier to perform updates on objects when the network is partitioned, and so long as the quorum intersection invariant is not violated, permits assignments that allow writes to the same object to occur in different partitions.

Quorum deflation is a technique used by objects to change the quorum assignments for particular levels. Each object has a preferred quorum assignment (usually the original level 1 assignment). Objects may find that all transactions have increased their levels to some threshold t due to a network partition, but that the network partition has now been repaired. Availability is usually maximized at level 1, but as transactions can only increase their levels, they have no way of returning to this optimal level of execution. Hence, an object may *deflate* the quorum assignment at level t to reflect the new state of the network, thereby increasing availability. The protocol ensures that the new quorum assignments are seen by all transactions, by ensuring the new quorum assignments are posted at a set of sites, the *co-quorum*, which intersects with every quorum for that object. The strategy that determines when objects deflate is not fixed; it may be aggressive or lazy, depending on the needs of the system. Quorum deflation allows transactions which have inflated to higher levels to acquire write quorums (at the expense of read operations) to return to a quorum assignment that is optimal for that

object, thereby enhancing availability of both read and write operations.

This method of transaction processing has a number of advantages:

- any transaction that can obtain a read quorum for an object is always permitted to perform read operations on it.
- some quorum assignments permit writes to the same object to be executed in different partitions without violating one-copy serializability.
- dynamic quorum assignment allows some transactions to complete which might otherwise have aborted due to their inability to secure static quorums.
- dynamic quorum assignment allows quorums to change based on the actual state of the network, enhancing availability of both read and write operations.

2.1.4 Views

This method [6] is similar to Herlihy's Dynamic Quorum Consensus protocol, but does not require quorum assignment tables to be maintained, and is hence not restricted by a quorum assignment invariant. Instead of using levels as the serialization technique, this paper introduces the concept of *views*.

A site's view is a list of other sites in the network. The policy for inclusion of a

site in this list is arbitrary, and may be different from one site to another. In the basic protocol, sites may only read or write copies of objects that are at other sites which have an identical view (determined by a totally ordered *view_id*). Optimizations to the protocol presented later in the paper allow different read and write strategies that permit reading and writing at some sites with different *view_id* identifiers. One-copy serializability in this model is again two-fold: a standard concurrency protocol ensures one-copy serializability within a view, and transactions executing in sites with a lower *view_id* are serialized before those executing in sites with a higher *view_id*. A transaction is said to be executing in a particular view *v* if it was created at some site with view *v*.

The concept of quorums, which was used in the other papers, is again used with the view protocol, but at two levels. At the system level, *accessibility thresholds* determine the number of copies of an object that must be accessible in a view in order to read or write the object in that view. The read and write accessibility thresholds of an object *x* are denoted $A_r[x]$ and $A_w[x]$, respectively. These thresholds must satisfy the normal quorum invariant seen in the Gifford method:

$$A_r[x] + A_w[x] > n[x]$$

(where $n[x]$ is the number of copies of *x* in the system). This invariant guarantees that any set of accessible copies read will intersect with every possible set of accessible copies written, ensuring that a read operation will access the most recent version of an object. As this is the only invariant which deals with the accessibility thresholds, no restrictions are placed on transactions updating the same object in different partitions, so long as this

invariant holds. Trade-offs naturally ensue when determining the accessibility thresholds, as decreasing one threshold (to make that operation more available) forces an increase of equal magnitude in the other threshold. At the view level, in each view v an object x is assigned read and write quorums $q_r[x,v]$ and $q_w[x,v]$ respectively. These quorums must then meet the following invariants:

$$q_r[x,v] + q_w[x,v] > n[x,v]$$

$$2q_w[x,v] > n[x,v]$$

$$1 \leq q_r[x,v] \leq n[x,v]$$

$$A_w[x] \leq q_w[x,v] \leq n[x,v]$$

(where $n[x]$ is the number of copies of x in view v). These conditions guarantee the one-copy serializability of transactions within a view.

Transactions under this protocol are divided into two types: *user transactions* and *update transactions*. A user transaction corresponds to the intuitive model of a transaction: it is initiated by a client, and accesses, reads and writes objects. Update transactions are used to revise sites' views, and will be described later.

In order to read an object x under this protocol, a transaction executing in view v must:

- ascertain from its local view information that x is read accessible in v.
- access $q_r[x,v]$ copies of x residing on sites located in v which also have view v.

• determine the maximum version number of the accessed copies.

• read a copy with the given maximum version number.

To write an object x under this protocol, a transaction executing in view v must:

• ascertain from its local view information that x is write accessible in v.

• access $q_w[x,v]$ copies of x residing on sites located in v which also have view v.

• determine the maximum version number of the accessed copies.

• update all of the selected copies and increase their version numbers to be one greater than the previous maximum version number.

Under this basic model of transaction processing, any user transaction initiated in view v which attempts to access an object at a site which has some different view u is aborted.

Update transactions are executed when a site decides to change its view. Views may be changed in two ways: a site may decide to change its own view based on its own information, or a site may see a transaction from a site with a different view, and decide to inherit that view.

An update transaction will first create a new unique *view_id*, *new_view_id*, for the new view. The *new_view_id* is larger than any other *view_id* that the site which initiates the transaction has seen, with the site's unique identifier appended to ensure uniqueness.

The site must then update all of its local object copies. It does so by accessing $A_r[x]$ copies at different sites, thus ensuring that it will see the most recent version of the object, and reading the value of the object with the highest version number. It installs this value in its local copy. If the update transaction encounters a site which has a *view_id* > *new_view_id*, it re-starts with a new *new_view_id* > *view_id*. The new view is said to be *installed* if the update transaction successfully completes. Any accessed site which has a *view_id* < *new_view_id* attempts to inherit the new view by running its own update transaction with the same *new_view_id* as that which it has just seen.

The frequency with which sites change their views does not alter the correctness of the protocol. Performance may be drastically affected, however. High-priority transactions which execute under one set of views when there are no failures in the network may be stalled if the views do not change when the network partitions. There are a variety of potential *tracking policies* that control the way in which sites decide to change their views. *Aggressive tracking* attempts to match sites' views to the actual state of the network. *Lazy tracking* allows views to change only if some high-priority objects are unavailable under an old view, but would become available again under the new view. *Demand tracking* causes a change in views only when a high-priority transaction is unable to execute in an old view but could commit in some new view.

As stated earlier, the basic model used in this protocol does not allow transactions to access objects that are in a view different from the one in which they were initiated. In fact, this restriction can be eased without violating one-copy serializability.

Let *v[T]* denote the view in which transaction *T* was initiated. The *relaxed write rule* allows writes to be expanded so that a transaction updates $q_w[x,v]$ copies of an object *x* with *view_id*'s less than or equal to *v[T]*. The *alternative read rule* allows reads to be expanded so that a transaction accesses $q_r[x,v]$ copies of an object *x* with *view_id*'s less than or equal to *v[T]*. Any subsequent writes by a transaction *T'* to a copy of *x* updated or accessed by *T* must have $v[T'] \geq v[T]$, or they will be rejected. Additionally, subsequent views installed at the site must have $view_id > v[T]$. These policies clearly use the same approach as ratchet locks in Herlihy's work.

Another optimization presented in the paper reduces the amount of work necessary in updating views by associating views with copies of objects instead of sites. When an update transaction is initiated for object *x*, it must therefore only access $A_r[x]$ copies of *x*, rather than having to do the same for each object stored at the site.

When the alternative read rule and relaxed write rule are added to the basic model presented in this work, the same advantages as the Quorum Consensus model are realized. In addition, as noted earlier, no quorum assignment model needs to be maintained, and the quorum invariant does not have to be maintained across such a table.

2.1.5 Hierarchical Quorum Consensus

A novel approach to the replica control method is undertaken with the hierarchical quorum consensus protocol [7,8,9]. These papers observe that many early replica control

strategies required a majority of sites to be available in order for updates to be allowed (i.e. $2q_w > n$), and notes that for large networks it becomes less likely that this goal can be reached. The protocol presented uses a hierarchical approach with the goal of reducing the number of sites (and hence the average number of messages) required to form a quorum.

The algorithm is based on an organization of the replicated data objects into a tree structure of depth n, with the root represented as level 0. The physical copies of the data objects are stored at level n. Intermediate nodes are thought of as logical placeholders. Each logical object at level l has l_{l+1} sub-objects at level $l+1$. Hence, the tree has $l_1 * l_2 * ... * l_n$ physical objects at level n.

Quorums for this algorithm are defined on a per-level basis. Read and write quorums at level l are defined as the number of objects which must be accessible at level l in order to gather read and write quorums at level $l+1$. By recursively traversing the tree in this manner, quorums are eventually obtained on the physical replicas stored at level n. To ensure one-copy serializability, the usual quorum constraints are modified to a per-level basis. Hence, at level l:

$$r_l + w_l > l_l$$

$$2w_l > l_l$$

Note that these constraints do not permit multiple concurrent write quorums to be held on a level. Hence, unlike the Dynamic Quorum Consensus Protocol and the Views

method, concurrent writes to the same object by transactions in different partitions are forbidden.

This protocol is compared [7] to majority voting and dynamic voting using a simulation study on two 9-site networks. The performance criteria evaluated are average number of messages and availability, where the former is defined as the average number of additional sites needed to form a quorum, and the latter as the probability that at least one site is able to execute a transaction requiring a quorum to be formed. The simulations are run over both low and high connectivity networks, and different levels of communication link reliability, with mixed results. The Hierarchical Quorum Consensus method required fewer messages on average, but was generally inferior to dynamic voting in terms of availability except for low connectivity networks with highly reliable communication links.

2.1.6 Dynamic Ordering of Transactions

This protocol uses another new approach to ensure that one-copy serializability is met, by dynamically ordering transactions [10]. The numbering of the transactions corresponds to an arbitrary order imposed on the replicated objects in the system. This algorithm also restricts write operations on a replicated object to at most one partition, while attempting to maximize read access. The underlying access method used is that of the Missing Writes algorithm presented earlier.

2.1.7 Multi-Class Replication-Control Protocol

This protocol strives to minimize transaction latency by decreasing the distance between a transaction and the objects it accesses, and eliminating transaction blocking while updates commit [11]. The basis for this protocol is the majority consensus protocol combined with 2PC, using some asynchronous broadcast to eliminate blocking.

Two assumptions are made in this protocol which are instrumental in its design:

- concurrent conflicting operations by different clients on an object are very rare.
- objects that are currently being accessed by a client for the first time will be accessed again by the same client in the near future.

Furthermore, the concept of *locality* is defined. *Locality(x,t)* is the set of sites from which transactions accessing *x* originated in the time interval [t-δ,t] for some constant value δ. From the assumptions above, the locality of an object being written will usually consist of only one site.

Replicas are divided into three distinct classes. *Metareplicas* are replicated lists of up-to-date sites for a replicated data object. *Metareplicas* allow multiple transaction-operations to execute concurrently at a single site, reducing communication and replication costs. *Pseudoreplicas* are temporary instances of replicated data objects, which

permit localized operations on the objects, further reducing costs. *True replicas* are the traditional copies of a data object stored at multiple sites in the network. Each of these three classes will be described in some detail below.

The protocol described maintains for each object a list of sites that have current versions of the object. These lists are in turn replicated, and are known as metareplicas. By consulting a metareplica, a transaction can determine which sites can correctly perform transaction operations. The client selects one of these sites to be the *leader*, and sends the relevant operations to it. The leader gathers all the quorums, acquires all necessary locks and performs the operations, thus freeing the client.

Metareplicas have two main characteristics, *update insensitivity* and *partial staleness*. Update insensitivity states that even if an object is updated somewhere in the system, the metareplica remains current unless some failure or recovery in the network causes the set of up-to-date sites to grow or shrink. Partial staleness states that generally, only a small amount of the data in the metareplica will become out-of-date. These characteristics are based on the assumption that network failures are rare, and affect only a small percentage of the network sites. As a consequence, most transactions that access a metareplica will receive current information, even if the metareplica is stale. Furthermore, only read operations need access a current copy of the data item. Hence, it is not important to keep all metareplica copies consistent, so long as it is possible to detect out-of-date objects in the unlikely event that they are accessed before the metareplica is brought up to date.

Communication costs in the system can clearly be reduced if a transaction can access a local copy of an object. In some instances, transactions will be able to access true replicas at their home sites. In other cases, it is possible to provide a temporary copy of an object at the home site, particularly if the locality is of long duration. These copies are called pseudoreplicas. The replica control protocol is able to create pseudoreplicas, and adapt the local metareplica to ensure that these "new" replicas are the copies accessed. Again, due to locality and the assumption that concurrent writes are rare, it is unlikely that a transaction will read an out-of-date pseudoreplica. Only true replicas must be updated synchronously to ensure correctness. Hence, out-of-date pseudoreplicas may be invalidated by asynchronously broadcasting cancellation messages which remove them from the remote sites. Again, the protocol must ensure that stale pseudoreplicas are not accessed in the meantime.

The final class of replicas are true replicas, which are the traditional copies of a data object distributed throughout the system. These copies must be kept consistent, and are thus the most expensive to maintain. The protocol ensures that one-copy serializability is conserved in the database by requiring a quorum of true replicas to participate in read/write operations.

The replica control protocol is divided into three stages: the *pre-operation stage*, which obtains replica location information and selects a leader; the *operation stage*, where the leader executes the operation and returns the result to the client; and the *2PC stage*, where the true replicas either commit or reject the transaction and the client is informed of the decision to commit or abort the transaction.

This protocol was designed to reduce delays in transaction processing, and the authors show a performance study [11] which compares it to a generic replicated protocol and a primary-site protocol. The Multi-Class Replication-Control protocol outperforms the other two protocols increasingly as the probability of a local pseudoreplica at a client site increases, for two different transaction sizes and two different networks. The protocol does suffer from the same restriction as earlier strategies, as concurrent writes to the same object are forbidden in different partitions.

2.1.8 Regeneration with Virtual Copies

This protocol diverges from the other strategies presented, as it uses a technique known as dynamic regeneration to improve availability of objects [12] and uses a different definition of majority partition. This is accomplished by combining dynamic voting with *Virtual Copies*. Virtual copies contain only state information about an object, but are permitted to participate in read and write quorums, so long as at least one real copy is included. A *generation threshold* is defined, which represents some pre-defined optimal number of real copies of an object to be maintained in the system. If the number of real copies falls below this generation threshold, some virtual copies are converted to real copies, and the reverse occurs when the generation threshold is exceeded. This process of changing a copy's type is known as *regeneration*, and it adapts to changes in the network to ensure that the optimal number of real copies is maintained when possible. Furthermore, by implementing regeneration, the last copy to fail will always be a real

copy. An allocation table is maintained which indicates the sites that have copies of replicated objects, and represents the optimal allocation of copies.

One of the objectives of the protocol was to design it so that it would be applicable to Wide Area Networks (WANs). Clearly a protocol running on such a network should attempt to minimize instances of large data transfer, due to the costs and latency involved. To ensure consistency, each update or change to a set of copies of a replicated object will involve a large amount of data transfer. Hence, regeneration should be kept to a minimum. The algorithm therefore uses a policy of selective regeneration by attempting to stay at the generation threshold.

The protocol is implemented using four major modules: DO_READ, DO_WRITE, DO_FAILURE, and DO_RECOVER. The first two modules implement read and write operations on objects. DO_FAILURE is executed by a site when it discovers another site has failed. DO_RECOVER is executed by a failed site upon recovery. The other subprogram used is a function called "majority" which returns true if the calling site is in the current majority partition with respect to the object being accessed, and false otherwise. Under this protocol, a site is considered to be in the majority partition with respect to an object if either a majority of the sites which participated in the object's last update are also in the partition, or in the case of a tie, the tied partition which contains the lexicographically highest ordered object. This definition ensures that a majority partition can always be found, and thus mutual exclusion is guaranteed.

The four modules used by the protocol will now be described. When a site wishes

to perform a read operation on an object *O*, it calls the DO_READ module. This module determines whether the site is in the current majority partition with respect to *O*. If so, a quorum containing at least one real copy of *O* can be obtained, and the real copy's value is read. Otherwise, the requesting transaction is aborted.

When a site wishes to perform a write operation on an object *O*, it calls the DO_WRITE module. This module determines whether the site is in the current majority partition with respect to *O*. If so, a quorum containing at least one real copy of *O* can be obtained, and all of the real copies have their values updated and their state information changed. Virtual copies only have their state information changed. Otherwise, the requesting transaction is aborted.

When a site detects a failure of another site, it calls the DO_FAILURE module. The module checks the allocation table to determine which objects had real copies stored at the failed site. If the generation threshold for the object has been crossed, it is regenerated at another site. The site at which the conversion occurs depends on the application.

When a site recovers from a failure, it calls the DO_RECOVER module. For each object stored at that site, a check is made to see if the site is in the current majority partition for the object. No changes are made if the site is not in the current majority partition. Otherwise, if the object is a real copy, and it is in the allocation table for the site, the copy is made current by reading from another copy in the partition, and its state information is updated. If the object/site combination is not in the allocation table, then

the real copy is reduced to a virtual copy, although the state information is still updated from an accessible copy of the object. Finally, if the object is a virtual copy, only the stet information is updated.

Additionally, if a site detects that the generation threshold of an object has been exceeded in the majority partition, other real copies of the object are converted to virtual copies, in a manner which attempts to reach the optimal file allocation set out in the allocation table.

This algorithm permits read and write operations on an object to occur during network partitions, even when the number of available sites in the majority partition is as low as one. This is done simply by ensuring that these operations cannot execute concurrently on the object in different partitions. Mutual exclusion and one-copy serializability are ensured. However, while transactions may proceed in minimal majority partitions, this definition of majority partitions combined with this restriction causes the overall availability of transactions in this protocol to be lower than in the other protocols examined.

2.2 Availability Bounds

Some upper bounds on availability of transactions when networks partition have previously been derived. Coan et al. [13] derive an availability bound for the case of *simple partitions*, *full replication* and *uniform data access*, and show that the bound is

tight by examining some of the well-known protocols presented earlier. Shah and Marzullo [14] build on these results by also examining the bounds when *partial replication* and *multiple partitions* are built into the model. Multiple partitions are not dealt with in this study. Also, for simple partitions under partial replication, Shah and Marzullo depend on a site being able to tell if it is in a partition with the majority or minority of sites. The number of copies replicated within the partition is irrelevant. This dependence is counter-intuitive, and there are no replica control strategies that follow this model. Furthermore, it is unclear whether a non-responsive site is the result of a failure or an extreme communication delay. This problem is unsolvable, akin to the earlier discussion of determining whether a network partition has occurred. Under the expanded definitions of majority and minority partitions for partial replication given below, the upper availability bounds will be the same for partial replication as they are under full replication, as full replication is simply a sub-case of generalized degrees of replication. This leads to a much cleaner solution. In Chapter 4, upper availability bounds are given subject to these definitions which include write-only transactions. Coan et al. and Shah and Marzullo do not consider blind writes (write operations on an object by a transaction that does not first read the object). Hence, write-only transactions are not considered in the work in this section.

Uniform data access depends on the *uniformity assumption*, which states that the transaction mix is the same at each site in the network. Dominant local access depends on the assumption that the distribution of transactions at a site is skewed towards accessing data stored at local and nearby sites rather than at distant sites. Shah and Marzullo give a possible definition of dominant local access in their paper. Some further

comments and suggestions regarding future work and experimentation with dominant local access are given in Chapter 5.

Before the bounds can be examined, some definitions must be presented. A database is *fully replicated* if each object in the database is replicated at every site in the network; it is *partially replicated* otherwise. Under full replication, a simple partition usually divides the network into two partition groups, the *majority partition,* which includes a majority of the sites, and a *minority partition*. Note that if there are an even number of sites in the network, it is possible for both partitions to have an equal number of sites. In this case, the choice of the majority partition is arbitrary. The definition of majority partition, combined with the uniformity assumption, ensures that most of the work initiated in the system is in the majority partition. A multiple partition creates more than two partition groups, possibly none of which is a majority partition. For partial replication, we expand the definition of majority and minority partitions to be transaction-dependent rather than network-dependent. Under partial replication, a transaction is operating in a majority partition if it can access a majority of copies of all objects in its read and write sets.

The performance measure being tracked is availability, which is defined as:

$$availability = \frac{number\ of\ transactions\ successfully\ completed}{number\ of\ transactions\ initiated\ in\ system}$$

Additionally, the following parameters are defined:

r_{maj} = read-only transaction in majority partition.

u_{maj} = update (read/write) transaction in majority partition.

r_{min} = read-only transaction in minority partition.

N_{maj} = number of sites in majority partition.

q_w = write quorum for an object.

T = number of transactions initiated in the system during network partition.

For full replication and simple partitions, to ensure one-copy serializability, it is clear that updates to an object cannot be performed in both partitions. Thus, for the best availability, updates will only be permitted in the majority partition. Even in the worst case, however, read-only transactions should be able to complete in any partition. Hence, the upper bound derived by Coan et al. for availability is $(u_{maj} + r_{maj} + r_{min}) / T$. The Gifford, Missing Writes and Views protocols have availability bounds derived as shown below. The upper availability bound is shown to be tight under the assumption of no blind writes, as both the Missing Writes and Views protocols' availability limits achieve it.

• Gifford Protocol

$$\text{Availability} = \begin{cases} (r_{maj} + r_{min}) / T & (\text{if } N_{maj} < q_w) \\ (u_{maj} + r_{maj}) / T & (\text{otherwise}) \end{cases}$$

The Gifford method either permits all transactions in the majority partition, or read-only transactions in both partitions to commit. Under most interpretations of the Gifford method, concurrent writes to an object in both partitions are not permitted.

• Missing Writes Protocol

$$\text{Availability} = (u_{maj} + r_{maj} + r_{min}) / T$$

The Missing Writes protocol expands on the Gifford method's availability, by permitting read-only transactions to *run in the past* in the minority partition.

• Views Protocol

(aggressive tracking)

$$\text{Availability} = (u_{maj} + r_{maj}) / T$$

(lazy tracking)

$$\text{Availability} = (u_{maj} + r_{maj} + r_{min}) / T$$

The Views protocol also allows read-only transactions to *run in the past* when using a lazy tracking protocol, whereas an aggressive tracking protocol would only allow majority transactions to commit, as quickly updated views would reflect the partitioned state of the network.

3 Simulation Model

This chapter describes the simulation model used to derive the performance measures for each of four replica control strategies. The model consists of a transaction processing system in which read-only, write-only and read-write (update) transactions arrive at the system for service. The system determines whether the transactions are permissible under the constraints of the replica control strategy, and then enqueues the transactions and acquires and releases the necessary locks for the operations to take place.

The four replica control strategies evaluated in this simulation were the Gifford, Missing Writes, Dynamic Quorum Consensus (Herlihy), and Hierarchical Quorum Consensus (Hierarchical) approaches. These protocols were chosen both for their historical significance, and for the significantly different techniques that each one uses to manage the replicas. The strategies also made similar assumptions about the underlying network and replication strategies, which allowed them to fit easily into the simulated transaction processing system.

Section 3.1 describes the input parameters to the simulation, Section 3.2 examines transaction behaviour within the system, Section 3.3 identifies the performance measures that the system produces, and Section 3.4 has a brief discussion on the validity of the results.

3.1 Simulation Inputs

The simulation model was designed to be as flexible as possible in order to realistically model a transaction processing system. This flexibility requires a large number of input parameters. In the final application, this number was reduced to 15 parameters, thus striking a balance between realism and practicality. Each of these input parameters will be described in turn. The values shown below were those used in the experimentation with the system, leading to the results displayed in Appendix A and discussed in Chapter 4. Variations to some of the parameters, and their effects on the simulation's output are also discussed in Chapter 4.

For many of the costs, the absolute values of the parameters are not particularly important; the key is their magnitude relative to the other parameters. For instance, intuitively, writing an object should be significantly more expensive than reading it. With the parameter values used in the simulation, the cost of writing an object will always be at least double the cost of reading it, due to higher overhead costs for writing and the larger average write quorum sizes. Similarly, the restart cost should be much greater than the cost of simply reading or writing an object, as it involves aborting the initial invocation of the transaction, resetting its quorums, and re-submitting the transaction to the system.

For read and write costs, the system assumes a large amount of concurrency in propagating updates, consulting ratchet locks and missing write information, and so on. By taking advantage of this concurrency, the read and write costs are a combination of

a large fixed overhead cost, and a much smaller variable lock acquisition cost. The overhead costs are the same for all operations of the same type, regardless of the number of copies of an object which the transaction must access. Any small changes in the operation's costs due to differing quorum sizes are considered to be built into the variable costs associated with lock acquisition. Some discussion on the results of relaxing this assumption is presented in Section 4.3.1.

Although the system is based on a 29-node network and 10 data objects, additional experiments not presented here make it reasonable to assume that the results can be extended to much larger systems. It is the behaviour of the strategies relative to each other, and not the absolute values generated by the simulation which are used in drawing conclusions. A system of this size is more manageable and reasonable for simulation, in terms of both run time and ease of checking that the system behaviour is correct.

The input parameters are as follows:

- Replica Control Strategy:

  { Gifford, Missing Writes, Herlihy, Hierarchical}

  These were the four replica control strategies implemented in the simulation system.

- Number of network nodes: 29

  The number of sites in the network underlying the distributed system. Each site is capable of holding any

number of replicated objects. This number gives a reasonable sized network which allows for some differences from site to site in terms of partial replication, but is not so large as to become unmanageable. This value also causes interesting results in the Hierarchical method for full replication, by creating an uneven hierarchy.

- Number of database objects: 10

  The number of data objects in the distributed system, each of which may be partially or fully replicated throughout the network sites. Choosing 10 objects yields reasonably sized transactions when using a uniform distribution. It also yields a good degree of conflict between transactions, as they share many of the same objects in their read/write sets.

- Degree of replication: { full, 7-site, 15-site }

  The degree of replication of the data objects throughout the system. The degree of replication of one object is independent of the others in the simulation. However, for simplicity, all objects had the same degree of replication in the experiments. The placement of each of the object copies was performed randomly with a uniform distribution over the nodes. However, at most one copy of an individual object can be placed at a single network site.

- Mean transaction inter-arrival time: 31 time units

  The transactions arrive in the system with interarrival times having a negative exponential distribution with a mean of 31 time units. Given the standard settings of other parameters, 31 was the lowest integer mean that could be chosen which was larger than the expected service time of a transaction (thus avoiding possible system saturation). The effects on average residence time of transactions by altering this value are shown in Section 4.3.3.

- Run time: 31000 time units

  The simulation was run for 31000 time units, with the goal of generating approximately 1000 transactions in a run.

- Partition location: 21

  The first network site in the second partition, if the simulation is running in partitioned mode. In the experiments, the network had partitions {1..20} and {21..29} when running in partitioned mode. This choice of partition location gives a clear majority/minority split in the network for full replication, at about 2:1. Furthermore, this selection leads to poor results for the Hierarchical model, which shows its unreliability with respect to availability. Partition location does not have any effect on the results of

the other strategies.

- Partitioned operation: { True, False }

  As described in Chapter 1, the system can run either in non-partitioned mode, where each network site can communicate with all of the other sites, or partitioned mode, whereby sites in one partition cannot communicate with those in another. In this simulation, only simple partitions (i.e., creating two partitions) were considered.

- Read (Write) fixed overhead cost: 2 (4) time units

  The cost to a transaction of performing a read (write) action on a data object, regardless of how many copies of the object are physically accessed in performing the operation.

- Read (Write) variable locking cost - remote: 0.01 (0.02) time units

  The combined cost to a transaction of obtaining and relinquishing a read (write) lock on a physical copy of a data object at a site other than the transaction's home site. The total variable cost of acquiring and relinquishing remote read (write) locks for a transaction is hence the product of this cost and the number of remote read (write) locks required.

- Read (Write) variable locking cost - local: 0.001 (0.002) time units

  The combined cost to a transaction of obtaining and relinquishing a read (write) lock on a physical copy of a data object that resides at the transaction's home site. This cost was set to be a factor of ten less than the remote read (write) lock cost due to the absence of communication delays in the lock acquisition process. The combined variable cost of acquiring and relinquishing local read (write) locks for a transaction is hence the product of this cost and the number of local read (write) locks required.

- Restart cost: 10 time units

  The cost to a transaction of aborting its current operation and restarting with different read and/or write quorums. The value of 10 time units gives other transactions a reasonable chance of arriving and being serviced before the transaction restarts than a lesser value would allow, without delaying the restarting transaction unreasonably, which a larger restart cost would cause. The effects of lowering this restart cost are examined in Section 4.3.1.

3.2 Transaction Behaviour

This section describes the behaviour of transactions in the simulation model. Transactions are generated by having certain attributes determined (pseudo-randomly or deterministically) and are then checked for acceptability under the replica control strategy. Those transactions which fail to meet the requirements of the strategy are aborted; the remainder are enqueued for service. The transaction queue follows a first-fit, first-served protocol. When enough copies of the transaction's read and write data objects are available, the transaction acquires the required locks and performs its operations. Upon completion, the transaction is enqueued on a time-ordered departure queue and the relevant performance measures are gathered. Section 3.2.1 deals with transaction generation and arrivals, Section 3.2.2 deals with transaction queuing and processing, and Section 3.2.3 deals with transaction departures. The performance measures calculated for the system are discussed further in Section 3.3.

3.2.1 Transaction Generation and Arrivals

Transactions arrive into the system with interarrival times having a negative exponential distribution with a mean of 31 time units. This choice causes the load on the system to be heavy but sustainable. When transactions are generated, the following attributes are pseudo-randomly determined:

- home site: the network site at which the transaction originates.

This attribute is particularly important for determining which copies of data objects the transactions is able to access. The home site is generated from a uniform distribution across the network sites.

- number of operations: the total number of read/write operations that the transaction performs on data objects. This attribute is generated from a uniform distribution from [1..2*number of data objects]. In this way, at most a transaction can read and write every data object exactly once.

- read/write sets: the operations performed by the transaction and the data objects accessed by them. Each operation has a 50% chance of being a read and 50% chance of being a write. The set of objects accessed is generated using uniform distributions across all data objects not yet read/written by the transaction. As mentioned above, a transaction can read or write an object at most once. Additional simulation runs using a transaction mix containing a higher proportion of read-only and write-only transactions were also constructed. The results of this experiment

are discussed in Section 4.3.4.

Based on these pseudo-randomly determined attributes, the remainder of the transaction's fields can be determined:

• The transaction's operation list is first re-ordered so that a read operation on a data object always precedes a write operation on the same object. Although not essential for the simulation's implementation, this ordering makes more intuitive sense.

• The operation list is then examined, and based on the operations performed, it can be determined whether the transaction is read-only, write-only, or an update (read/write). This information is recorded for purposes of statistics gathering.

• Based on the home site field, current state of the network and location of read and write set object copies, the transaction is marked as being in either the majority or minority partitions. When there is no partition in the network, the transaction is always considered to be in the majority partition. When there is a partition and full replication, the transaction is in the majority partition if the majority of the network sites are in the same partition, or if exactly 50% of the sites are in the same partition, along with site #1. In the case of partial replication, the transaction is in the majority partition if it can access more than 50% of the copies of each of its read/write objects (or exactly 50% of the copies *and* it can

access network site #1).

• Finally, the initial read and write quorums are selected, based on the replica control strategy in effect. The initial quorums are as follows:

- Gifford read quorum : ceiling (number_copies / 2)
  write quorum: ceiling ((number_copies + 1) / 2)
- Missing Writes read quorum : 1 copy
  write quorum: all copies
- Herlihy read quorum : 1 copy
  write quorum: all copies

The initial quorums for the Hierarchical strategy are determined by a fairly complex algorithm [9].

• Other miscellaneous housekeeping fields such as the Transaction ID and start time are also recorded.

Once the attributes have been determined, the next arrival is scheduled, and the transaction is submitted to the Lock Manager.

The Lock Manager determines whether or not an arriving transaction meets the criteria required by the replica control strategy. It first ensures that the initial read and write quorums meet the quorum invariants specified by the strategy. A check is then made

to see whether the transaction is able to physically reach the number of copies specified by its quorums, and whether there is additional information (Missing Update Flags, Ratchet Locks) being stored at the network sites which may impede the transaction. In addition, as described in Chapter 2, the Hierarchical replica control strategy enforces a tree structure on the object copies, and the required number of copies in specific groups must all be accessible in order for the transaction to be eligible. If the transaction is able to reach the required number of copies immediately, and there is no extra information to block it, it is serviced by the Lock Manager according to the protocol given in Section 3.2.2. The status of the object copies (locked or not locked) is irrelevant at this stage. This check essentially determines whether the transaction will *ever* be able to attain locks on the object copies with the current quorums, based on its capacity to physically reach the required number of copies in the network.

If unable to proceed with the current quorums, a transaction under the Missing Writes or Herlihy methods is forced to change its quorums and restart. The transaction has its quorums reset as specified in Chapter 2, and is placed on a restart queue, to be awakened after the restart delay specified in Section 3.1. A transaction running under the Missing Writes protocol is allowed to be restarted once. A transaction executing according to the Herlihy strategy may restart as many times as there are quorum levels defined. Transactions unable to access the required number of copies executing under the Gifford or Hierarchical strategies, or transactions which exceed the maximum number of restarts allowed by the Missing Writes or Herlihy methods are aborted, and placed on the departure queue. They have no further interaction with the system. The departure queue is discussed further in Section 3.2.3. Restarting transactions that are able to reach the

required number of object copies under their new quorum assignments, and are not blocked by site information, are also serviced in the method described in Section 3.2.2.

3.2.2 Transaction Queuing and Processing

Once a transaction has been deemed eligible for processing, as described in Section 3.2.1, the Lock Manager checks to see whether it is able to begin executing immediately. If not, the transaction is enqueued in a first-fit first-served queue. This queuing protocol means that the queue is traversed from the head to the tail, and a transaction is only removed from the queue when it is able to obtain *all* of its locks and begin executing. In this manner, no candidate transactions are needlessly delayed by others which were able to acquire some (but not all) of their required locks and blocked for the remainder. Deadlock is also avoided. As described in Section 3.2.1, arriving transactions are checked for eligibility immediately. Transactions in the queue are checked for eligibility when a transaction that is being serviced releases its locks and departs the system. In this manner, the immediate servicing of eligible arriving transactions does not violate the first-fit first-served queuing protocol, as it is guaranteed that there are no other transactions already waiting which could have executed instead, as any such transaction would already have started after the most recent departure.

To check for a transaction's eligibility, the Lock Manager examines each object in the transaction's read and write sets. If any accessible copy of a read set object already has a write lock on it, or any accessible copy of a write set object already has a read or

write lock on it, the transaction is currently ineligible to begin execution. The eligibility check for the transaction is thus discontinued after the first such lock is discovered. In the case of an arrival, the ineligible transaction is subsequently enqueued. Otherwise, the eligibility check was made because of a departure from the system, and the next transaction in the queue is considered. Again, in the case of the Hierarchical strategy, its particular grouping and locking rules must also be satisfied.

Once a transaction has been selected for service, it first uses its read and write quorums, along with the read, write and locking costs to determine and schedule its completion time. It subsequently acquires the necessary locks. For objects in the transaction's read set, a lock is placed on each accessible copy of the object until the read quorum has been reached. If a copy of the object resides at the transaction's home site, it is guaranteed to be one of the copies locked, which reduces the overall service time required by the transaction due the difference between remote and local locking costs. For objects in the transaction's write set, a lock is placed on every accessible copy of the object. The latter policy ensures that all accessible copies of the object are updated together at the time that the transaction commits, eliminating the need for propagating the updates later. As with the eligibility checks above, the locks placed under the Hierarchical strategy must also meet the grouping requirements as discussed in Chapter 2.

When the locks have been placed on the transaction's object copies, it is placed on the departure queue, which is ordered according to departure time. It then "goes to sleep" until it is awakened at its scheduled departure time, as discussed in the next section. Departure time is simply the arrival time of the transaction into the system plus

the locking, queuing, restart and read/write costs the transaction requires.

3.2.3 Transaction Departures

Departures from the system occur according to the ordering of transactions in the departure queue, which is ordered according to completion time. When a transaction is removed from the queue at its scheduled time, it releases any locks that it acquired. At this point, transactions are deemed to have committed if they performed their operations successfully, or aborted otherwise. Statistics are gathered based upon the transaction's attributes and activity while in the system, for committing transactions only. The number of aborted transactions is compiled for purposes of computing availability, but otherwise such transactions' quorums and run times are not tabulated. Once the transaction's locks have been released, other transactions may be eligible to begin executing, as discussed in Section 3.2.2. The Lock Manager is signalled to begin searching for new transactions to activate.

3.3 Simulation Outputs

The simulation model generates a table of 96 output figures per (strategy, degree of replication, partitioned or non partitioned operation) triple. Departing transactions contribute statistics at both the macro and micro levels. At the macro level, all transactions are included together in the output statistics. At the micro level, transactions are classified into read-only, write-only and update transactions. Additionally, at both

levels, statistics for majority and minority partitions are reported separately. The statistics gathered are as follows:

- Number of transactions initiated:

  The number of transactions generated in the simulation. This figure is used, along with the number of transactions committed and aborted to compute availability for the replica control strategy.

- Number of transactions committed:

  The number of transactions successfully completing all of their operations in the simulation.

- Number of transactions aborted:

  The number of transactions that aborted their execution in the simulation, because of a failure to acquire the number of locks on object copies required by the replica control strategy in effect. As mentioned above, this statistic is the only one kept for aborting transactions.

- Average read quorum size:

  The average number of locks required by a committing transaction to access a read object. Calculated by taking the number of read locks obtained by committing transactions

and dividing by the total number of read objects accessed by committing transactions.

- Average write quorum size:

  The average number of locks required by a committing transaction to access a write object. Calculated by taking the number of write locks required by committing transactions and dividing by the total number of write objects accessed by committing transactions. Recall that the number of write locks *obtained* in this case will always be the total number accessible by the transaction, and is thus greater than of equal to the number of locks *required*.

- Average number of restarts:

  The average number of restarts performed by a committing transaction for purposes of changing read and/or write quorum sizes. Calculated as the total number of restarts by committing transactions divided by the total number of transactions committed.

- Average service time:

  The expected total time spent by a committing transaction being serviced by the transaction processing system. This time includes the time to acquire all read and write locks

and to perform the read and write operations. Calculated as the total service time by all committing transactions divided by the total number of transactions committed.

- Average residence time:

  The expected total time spent by a committing transaction in the system. Residence time includes queuing time, restart delays and service time. Calculated as the total time spent in the system by all committing transactions divided by the total number of transactions committed.

3.4 Statistical Validity

Any simulation requires that the user have confidence in its operation and subsequent output results as a successful model of reality. Some discussion of the validity of the results is thus warranted.

For the 15-site replication, partitioned mode results displayed in Appendix A, confidence interval testing was performed on the data for the average residence and service times. It was possible to assert with 90% confidence that the actual means for these performance measures would fall within 3.5% and 1.0%, respectively, of the calculated means of the aggregate test data for the Missing Writes strategy. The other strategies' confidence intervals showed similar tightness. Except for some expected overlapping with other strategies in some cases (due to similarities in quorum size and

hence service times), the confidence intervals for the strategies remain distinct at both the aggregate and micro levels. Due to the tightness of these bounds, and the fact that all of the other trials were run for the same period of simulated time, and using the identical transaction processing model, we assume that the output data from the remainder of the cases is similarly indicative of the behaviour of the simulated system. In addition, some of the performance measures (e.g., average quorum sizes, number of restarts) were verified theoretically, and the availability measures of transactions committing and aborting are also as expected.

4 Results

This chapter presents a discussion of the results generated by the simulation model. The raw output data from the simulation can be found in the tables in Appendix A. Section 4.1 presents a broad overview of the results and expanded availability bounds for the four replica control strategies studied. The majority of the remaining sections in this chapter deal with results from each of the four strategies evaluated in the simulation: Section 4.2.1 deals with the Gifford strategy, Section 4.2.2 with the Missing Writes strategy, Section 4.2.3 with the Herlihy method, and Section 4.2.3 with the Hierarchical protocol. Each of these sections provides a more in-depth analysis of interesting results relating to each of these strategies under the various replication, partitioning and transaction type cases. Section 4.3 provides some additional results for extensions of the original simulation model.

The availability bounds in this chapter use the same variables as in Chapter 2, with the addition of the symbols w_{maj} and w_{min} to denote write-only transactions in the majority and minority partitions, respectively.

The simulation model was written in the Turing programming language and run on both an IBM PC and a Unix-based SUN network. The code is 2216 lines long, and has an execution time of approximately 20-30 minutes per trial on the SUN network, and one hour on the IBM PC. Each trial consists of the simulation of a single strategy for one of the degrees of replication / partitioning combinations.

4.1 Overview

This section provides a broad overview of the methods used for evaluating the results of the simulation model, as well as a discussion of these results.

Each of the four replica control strategies was implemented for three degrees of replication: full (29-site) replication, 7-site partial replication and 15-site partial replication. The strategies were tested under each of these cases for both non-partitioned and partitioned operation. In the latter case, the network was partitioned into two groups of sites, with 20 sites in the first partition and 9 sites in the second. Results from the simulation were also broken down by transaction type (read-only, write-only, or update). Furthermore, as discussed in Section 3.2.1, transactions were deemed to be in either a majority or minority partition with respect to their read and write sets, and results were gathered for majority, minority and aggregate operation. All references to majority (minority) partition in this chapter refer to the partition in which transactions operate containing the majority (minority) of the copies of the objects in their read and write sets.

In addition to the four replica control strategies, a simulated "base" strategy was introduced into the model for output comparison purposes. This strategy was deemed to have the highest availability possible without violating one-copy serializability, so all transactions in the majority partition and read-only and write-only transactions in the minority partition were permitted. Additionally, transactions were not required to spend any time in the system acquiring locks, nor were they ever forced to restart. Hence, service time was reduced to the cost of performing the physical read and write operations

on objects. The time requirement of each transaction was thus minimized, and provided a solid base case against which to compare the service and residence times of the four replica control strategies.

When the network is not partitioned, all of the strategies clearly have the same availability, as all transactions are able to commit. There is also not a wide margin of difference between the strategies at the aggregate level when comparing the average residence times of transactions in the system. The Missing Writes and Herlihy methods do show the best average residence times, however, due to their lower quorum requirements. Table 1 shows the percentage increase in average residence times over the base method for the strategies under all three degrees of replication. Much greater differences can be found when the network partitions, however.

Strategy	Full	15-site	7-site
Gifford	39.6%	19%	8.5%
Missing Writes	34.7%	16.9%	7.6%
Herlihy	34.7%	16.9%	7.6%
Hierarchical	35.6%	17.2%	7.6%

Table 1: % Increase in Average Residence Times over Base Method (Non-partitioned operation)

In terms of availability under partitioned operation, there is not a lot of difference between the strategies under the costs, transaction distribution, and protocol followed by this system. The availability measures are shown in Table 2.

Strategy	Full	15-site	7-site
Base	75.4%	75.4%	30.4%
Gifford	72.9%	72.9%	26%
Missing Writes	74.3%	74.3%	27.5%
Herlihy	75.4%	75.4%	30.4%
Hierarchical	72.9%	14.5%	4.6%

Table 2: % of Transactions Completing Under Partitioned Operation

Clearly, the Hierarchical method falls far short of the others in terms of availability. The reasons for this shortfall will be discussed in Section 4.2.4. The Herlihy method, however, achieves the same availability percentages as the base method under all three replication schemes. It does so, however, at far greater cost than the Gifford method, while achieving only a 2.5% higher transaction commitment rate under full and 15-site replication, and 4.4% under 7-site replication. Table 3 shows the percentage increase in average residence times over the base method for each strategy under partitioned operation. The Hierarchical method's average residence times are not considered, because the availability of transactions when the network is partitioned is so low as to make the results insignificant.

Strategy	Full	15-site	7-site
Gifford	19%	10.4%	9%
Missing Writes	40.6%	32.1%	61.7%
Herlihy	37.3%	29.1%	58.4%

Table 3: % Increase in Average Residence Times over Base Method (Partitioned operation)

The table clearly shows that the gains in availability achieved by the Herlihy method are made only with much higher average residence times. While the Gifford method has greater or equal read and write quorums than the Herlihy method in all cases, the restarts by transactions under the Herlihy method dramatically counterbalance the (only slightly) smaller average quorum sizes. The small increase in availability does not seem to justify the large increase in average residence time, which would suggest that the Gifford method is in fact a better strategy when the network is partitioned, unless maximizing availability is imperative. The Missing Writes strategy has the greatest increase in average residence times for all three degrees of replication, due to its requirement that all transactions that cannot access some object copy's site must restart.

For the parameter values and transaction mix generated by this simulation, the best strategy to pick can be selected according to the likelihood of a network partition. Additionally, if the frequency of network partitions is high, the strategies can also be ranked according to whichever is considered more important of the timing or availability trade-off. The results are summarized in table 4, ranking the strategies under each situation.

Situation	Gifford	Missing Writes	Herlihy	Hierarchical
Low Partition Rate	4	1*	1*	3
High Partition Rate Availability	3	2	1	4
High Partition Rate Timing	1	3	2	4

* The Missing Writes and Herlihy methods perform identically when the network is not partitioned.

Table 4: Ranking of Strategies under Partitioned/Non-Partitioned Operation and Importance of Availability/Average Residence Time

In a system where there were larger numbers of read-only and write-only transactions however, the Herlihy method would certainly be preferable to the others in both partitioned and non-partitioned situations, as demonstrated in Section 4.3. Transactions which were able to determine *a priori* which level to operate at would also lead to the Herlihy method being the best choice, as restarts would be minimized. The Herlihy method requires smaller average quorums than the other methods, and has the optimal availability. The Gifford method is not able to process any transactions which are in the minority partition without sacrificing all update transactions in the network, and neither the Missing Writes nor Hierarchical methods can process any such transactions with write operations. In addition, both of these strategies are unpredictable with respect to minority partition read-only transactions in the case of partial replication, and the Hierarchical method's availability measures are entirely dependent on the initial division of network sites into the hierarchy it enforces, and the location of the network partition with respect to this hierarchy. As a result, this strategy's availability is highly unpredictable when the network is partitioned, and is hence quite unreliable.

These results clearly show that factors such as the transaction mix, restart and locking costs, and frequency of network failures are extremely important in determining the best replica control strategy for a particular environment. A discussion of the sensitivity of these results to changes to individual parameter values and transaction mixes is included in Section 4.3.

As mentioned in Chapter 2, the inclusion of write-only transactions allows an expansion of the availability bounds originally studied by Coan et al. [13] for the replica control strategies. These bounds were derived theoretically, and were found to be tight when the strategies were simulated. In the cases of the Gifford and Herlihy methods, the bounds are always met. The availability bounds for each strategy are shown in Table 5.

Strategy	Availability Bound
Gifford ($N_{maj} < q_w$)	$(r_{maj} + r_{min}) / T$
Gifford (otherwise)	$(u_{maj} + r_{maj} + w_{maj}) / T$
Missing Writes	$(u_{maj} + r_{maj} + w_{maj} + r_{min}) / T$
Herlihy	$(u_{maj} + r_{maj} + w_{maj} + r_{min} + w_{min}) / T$
Hierarchical	$(u_{maj} + r_{maj} + w_{maj} + r_{min}) / T$

Table 5: Availability Bounds

These bounds are simply an extension of those originally derived by Coan et al. [13] with the inclusion of write-only transactions. Only the Herlihy method is able to execute write-only transactions in the minority partition. Additionally, the Hierarchical method was not included in the original study. In the case of this strategy, it should be noted that the actual availability measured by the simulation often fell far short of this

bound, as discussed in Section 4.2.4.

4.2 Replica Control Strategy Analysis

This section will consider each of the strategies in greater depth, discussing those situations for which each strategy has interesting results, and providing explanations for any non-intuitive results.

4.2.1 Gifford

The Gifford method, as mentioned above, has the lowest availability of the four methods, as no transactions in the minority partition are able to commit. Furthermore, the read and write quorums for objects accessed by the strategy are always the minimal majority set. The Gifford method performs very strongly for average residence time under partitioned operation, as detailed above. When the network is not partitioned, however, the Gifford method's average residence times are uniformly worse than all of the other strategies considered, both at the aggregate and micro levels. This result is due to the method's requirement to always lock a majority of the copies of all objects in a transaction's read and write sets.

4.2.2 Missing Writes

The Missing Writes strategy generally provides middle-of-the-road availability and long average residence times. When the network is partitioned, it is able to process all transactions in the majority partition, and some or all read-only transactions in the minority partition, depending on the degree of replication, as discussed below. Average residence times under Missing Writes are the worst of all of the strategies considered when the network is partitioned, as transactions which discover the failure are forced to restart and to lock a majority of the copies of their read and write sets.

Most previous analysis of the Missing Writes strategies considers systems with full replication, and thus assumes that read-only transactions would always be able to complete in the minority partition. Under full replication, such transactions would only have to access the local copy of each of the objects in their read sets, thus making them unaware of the partition, and avoiding the restart and higher quorum requirements which would cause them to fail. Under partial replication, as shown in the results, read-only transactions in the minority partition may in fact fail. There are two reasons for this:

- A transaction may not be able to find *any* copies of a read set object in the partition. In this case, it is clearly impossible for it to read that object, and the transaction must be aborted.

- A previous update or write-only transaction, operating in a majority

> partition with respect to its read and write sets, may have left missing writes information at some of the network sites which are later read by the read-only transaction. This transaction is then forced to restart and lock a majority of the copies of each of its read set objects. As the transaction is operating in a minority partition with respect to its read set, it is clearly unable to reach the majority of the object copies for some of its read set objects, and is thus forced to abort.

The results generated by the simulation showed that under 15-site replication, no read-only transactions had to be aborted, whereas under 7-site replication, only 75% of minority read-only transactions (and hence 88% of the read-only transactions in the system) were able to commit. Despite this possible failure of minority read-only transactions, it is not surprising that no read-only transactions under 15-site replication failed. Dealing with the first potential reason for aborting, any transaction in the 20-site physical partition would be guaranteed to have at least six copies of each object available to it, and there is only a 0.02%[1] chance of an object having no copies in the 9-site partition. Transactions aborting for the second reason are also unlikely, as there is only a 0.95% chance of an object having the majority of its copies in the 9-site partition. The likelihood of aborting under 7-site replication is much greater, with a 5% chance of having no copies of an object in the 9-site partition, and a 10.8% chance that the majority of copies of an object will be in the 9-site partition. Both of the aforementioned reasons for aborting are thus more likely to occur. In addition, there is a 0.2% chance that no

[1] See Appendix B.

copies of an object will be present in the 20-site partition.

4.2.3 Herlihy

The Herlihy method is clearly the best strategy in terms of availability, as it always matches the base method's availability measures of all transactions in the majority partition with respect to their read and write sets, and all read-only and write-only transactions in the minority partition with respect to their read and write sets to complete. When the system is not partitioned, it also has the best average residence times (along with the Missing Writes strategy) for all types of transactions. However, due to its high restart costs, and the requirement of inflating its quorums to new levels under partitioned operation, it has much poorer average residence times than the Gifford method when the network is partitioned. The Herlihy method's results are straightforward other than instances where the average residence time is less than that of the base method, as described earlier.

4.2.4 Hierarchical

The Hierarchical method warrants the most discussion of the four methods, because of its poor and somewhat unpredictable performance when the network partitions. These results are due to the hierarchy imposed on the network sites, how the grouping of object copies into these sites is performed, and how the network partitions relative to this hierarchy. The hierarchies for full, 15-site and 7-site replication, and the quorums required

at each level are shown in Figures 1, 2, and 3 respectively.

In the simulation, this strategy seemed to perform as well as the Gifford method, as all transactions in the majority partition are able to commit, but none are able to execute in the minority partition. However, as shown in Figure 4, a slight change in the partitioning of the sites would have resulted in a complete inability to perform write operations anywhere in the system, but would have permitted all read-only transactions in both partitions to commit. This situation is exactly what occurred in the case of 7-site replication, which has the somewhat extreme requirement that all copies of an object be accessible before write operations are allowed on it.

In the case of the 15-site hierarchy, the availability measures are extremely hard to predict. The availability for each class of transaction was as follows:

Transaction Class	Majority Availability	Minority Availability	Total Availability
Read-only	100%	35.7%	79.5%
Write-only	32.5%	0%	26%
Update	14.5%	0%	10.5%
All Transactions	19.2%	1.9%	14.5%

Table 6: % of Transactions Completing Under Partitioned Operation For Hierarchical Method and 15-site Replication

This result is highly undesirable. In the vast majority of cases, a transaction is able to access a majority of copies of all objects in its read and write sets, but is still unable

fig 1

fig 2,3

fig 4

to commit. In fact, it is possible under the hierarchy derived for 15-site replication for 13 of the 15 copies (87% of the copies) of an object to be available, but for transactions to still be unable to write to it. Similarly, in the case of full replication, the same result could occur even when 25 of the 29 (86%) copies are available.

As claimed, this algorithm does generally reduce the size of quorums, but it clearly places an undue restriction on which sites may form these quorums. If the network partitions, even when a dominant majority of copies are available to a transaction, it is likely to be forced to abort. As a result, unless the probability of a partition is very small, or its location predictable (in which case the hierarchy could be structured accordingly), this algorithm would appear to be ineffective in managing replicated data.

4.3 Additional Experiments

In order to concretely determine the sensitivity of these results to changes in various input parameters, a series of additional experiments were constructed. These experiments will be described in this section. It should be noted that none of the changes altered the conclusions established earlier in this chapter. These experiments therefore back up the assertion that very large changes to (possibly multiple) input parameters would have to be made in order for the conclusions about the relative merits of the proposed strategies to change.

In addition, the four replica control strategies being studied were examined under single-copy (centralized) replication, and these results are discussed briefly. Also, a set

of experiments was performed to compare average response times for each strategy over a range of mean transaction inter-arrival times. Finally, an experiment was run with the Herlihy and Gifford methods using a transaction mix with a higher percentage of read-only and write-only operations. This experiment's results back up the assertion made in Section 4.1 that the Herlihy method would be preferable with such a mix.

4.3.1 Output Sensitivity

Two separate sets of experiments were constructed to determine the sensitivity of the system to changes in both input parameter values and object locking philosophy. The first set of tests involved changing the initial values of the restart and locking costs. The second set of tests involved eliminating the fixed locking costs associated with read and write operations for transactions, by distributing this fixed overhead across the variable costs associated with locking and accessing each object copy.

In the first case, several additional runs were constructed with different values for the restart costs and locking costs. The restart costs were lowered to 5 time units in one test, and all of the locking costs were increased by a factor of 10 in the second case. The Gifford and Missing Writes strategies were re-evaluated and compared under these conditions, for partitioned operation and all three degrees of replication. The results showed that altering one parameter, even by this degree, did not affect the original model's conclusions. In the case of the alteration to the restart time, the average residence time for transactions tended to drop by approximately 5 time units (as expected), but still remained approximately 5 time units longer for the Missing Writes method than for the

Gifford method, backing up the assertion that any significant restart cost for the Missing Writes and Herlihy methods would more than compensate for the higher average quorum sizes (and hence larger locking costs) of the Gifford method, thus continuing to lead to better average residence times for the latter strategy. It seems intuitive that the overhead to a transaction required to restart and re-initialize its quorums and re-enter the locking queue should be greater than the overhead cost to a transaction of a single write operation. As the latter cost is 4 time units, 5 time units would appear to be the lowest reasonable restart cost attainable. Thus, simply reducing the restart cost parameter, even this significantly, will not lead to a change in the above conclusions. When the locking costs were increased ten-fold, the average residence times increased by factors of 5.4, 1.9 and 1.2 for the Gifford method, and 4.7, 1.8 and 1.2 for the Missing Writes method, for full, 15-site, and 7-site replication respectively. Although the increase was greater for the Gifford method, the average residence times for transactions under this method remained approximately 10 time units less than for the Missing Writes method, making it clear that a much more significant increase in locking costs, or a combined increase in locking costs *and* a large drop in restart cost would be necessary for the Missing Writes or Herlihy methods to obtain better results in average residence time for transactions than the Gifford method.

In the second case, the fixed overhead locking costs for transactions were eliminated, but the average total locking cost was maintained by distributing this fixed cost across the cost for locking and accessing individual object copies. The total locking cost thus becomes linear based on the number of object copies accessed, allowing different quorum sizes amongst the replica control strategies to have a larger effect on the

transaction's total costs. Although this change made a significant difference in the *absolute* values for average residence times when tested for the Gifford and Herlihy methods, it still did not change the conclusions reached earlier. The Herlihy method continued to achieve significantly lower average residence times when the network was not partitioned, whereas the Gifford method was superior under partitioned operation. The cause for the Herlihy method's longer average residence times was again its high restart costs. To change this result, the Herlihy method would seem to require that this cost somehow be eliminated, or that transactions be "smart" in choosing the level they wish to operate at when they are generated so as to avoid having to restart.

4.3.2 Centralized Operation

As discussed in Chapter 1, a centralized database is highly susceptible to site and communication failures, as the site storing the only copies of the data objects can become partitioned from many or all of the other sites. Distributing the single object copies over different sites alleviates the severity of a possible single site failure, but will still result in extremely low availability. The following results for availability and average residence times were achieved for the four replica control strategies with single copy replication.

Strategy	Residence Time Non-Partitioned	Availability Non-Partitioned	Residence Time Partitioned	Availability Partitioned
Gifford	68.51	100%	10.61	11.9%
Missing Writes	68.51	100%	10.61	11.9%
Herlihy	68.51	100%	10.61	11.9%
Hierarchical	4.02	4.6%	3.17	1.98%

Table 7: Average Residence Times and Availability for 1-copy replication

The average residence times for the first three strategies are identical, as there are no longer any differences in quorum sizes or restart costs between the protocols. Transactions operating under either of the three strategies are either able or unable to access the sole copy of the object immediately, so no further processing is required, meaning that the strategies work identically in this situation. The Hierarchical model performs poorly in terms of availability (even when the network is *not* partitioned) because it does not specify how to construct hierarchies and/or quorums for single copy or 2-copy degrees of replication.

4.3.3 Inter-arrival time variation

For this set of experiments, the mean transaction inter-arrival time was varied between its current setting of 31 and 100 for non-partitioned operation, and mean times of 21 and 100 for partitioned operation. Under non-partitioned operation the average residence time increases slowly for each of the four strategies until it reaches the 40-50

range. For partitioned operation, it increases slowly until it reaches the 20-30 range. Figures 5 and 6 graph the average response times versus average inter-arrival time for non-partitioned and partitioned modes respectively. It is interesting to note the result that transactions have a faster average response time when the system is partitioned, *ceteris paribus*. Clearly, this effect is due to the large number of transactions that abort and the fact that transactions operating in different partitions do not interfere with each other's operation.

4.3.4 Transaction Mix Variation

As discussed in Section 4.1, the Herlihy method would likely be the best choice for a replica control strategy even in the face of high partition rates if the transaction mix included a higher percentage of read-only and write-only transactions than were present under the initial conditions of the simulation. The final set of experiments conducted involved generating a transaction mix for the Gifford and Herlihy methods whereby an update transaction had a 10% chance of being changed to be a read-only transaction, and another 10% chance of being changed to a write-only transaction. The results were as follows:

Strategy	29-site	15-site	7-site
Gifford	66.9	58.22	53.7
Herlihy	64.9	57.29	53.3

Table 8: Average Residence Times (Non-Partitioned Operation) for new Transaction Mix

fig 5,6

Under non-partitioned operation, the results are similar to those determined using the original transaction mix. Although the absolute average residence time values are much lower (due to the smaller average size of read-only and write-only transactions), the conclusions remain the same. The Herlihy method has better average residence times, but not by a wide margin. Under partitioned operation, however, the Herlihy method's availability measures show a substantial gain:

Strategy	29-site	15-site	7-site
Gifford	73.6%	73.6%	28.6%
Herlihy	80.1%	80.1%	42.2%

Table 9: Availability for new Transaction Mix

The Herlihy method shows availability gains of 4.7%, 4.7% and 11.8% for the three degrees of replication using this transaction mix, whereas the Gifford method has gains of only 0.7%, 0.7% and 2.6% with the change. The ability of the Herlihy method to successfully commit over 80% of submitted transactions for 29 and 15-site replication, and over 42% for 7-site replication represents a huge advantage. Originally, the reason for selecting the Gifford method over the Herlihy method as the strategy of choice when chance of network partitions was high was due to the much higher average residence times of the Herlihy method while achieving limited availability gains. These gains are clearly much more substantial when there are a greater number of read-only and write-only transactions in the system. Also, as shown in Table 9, the differences in average residence times between the two strategies using such a transaction mix are very similar to those found in the original experiments. Hence, substantial availability gains have been made without a significant increase in average residence times. This result backs up the

earlier assertion that a system with a larger proportion of read-only and write-only transactions would be best served by the Herlihy strategy.

Strategy	29-site	15-site	7-site
Gifford	39.8	37.1	14.0
Herlihy	46.0	43.6	23.1

Table 10: Average Residence Times (Partitioned Operation) for new Transaction Mix

5 Conclusion

This chapter presents an overview of the purpose and results of this study in Section 5.1, and suggests some possible future extensions of this work in Section 5.2.

5.1 Overview

This study has examined a 29-site transaction processing system where transactions perform read, write, or both operations on data objects. A number of *pessimistic* replica control strategies were discussed, which guarantee one-copy serializability while attempting to permit the highest possible number of transactions to commit. Four of these strategies - the Gifford, Missing Writes, Herlihy and Hierarchical methods - were implemented in a simulation study of a distributed transaction processing network. The system was tested using simulation under both partitioned and non-partitioned operation, and with full, 15-site and 7-site replication. The results of this study provided insight into the availability and average processing time of transactions under each strategy, and examined the effects of altering some of the key input parameters and system assumptions. The conclusions of the thesis are robust with respect to these changes in assumptions and parameter values.

This study found that under non-partitioned operation, the availability was 100% for all four replica control strategies, but that the Herlihy and Missing Writes methods had the lowest average residence time for all three degrees of replication under non-partitioned operation. However, under partitioned operation, for all three degrees of replication, the

Herlihy method had the best availability measures, whereas the Gifford method had the best average residence time. The differences in average residence time between the Gifford method and the Herlihy and Missing Writes methods were substantial, but the gains in availability achieved by the latter two methods were relatively small. This result demonstrated the classical trade-off of availability versus cost, and showed that the likelihood of failures in the system, the expected transaction mix and the differing costs assigned to each strategy were important factors in choosing the best replica control method for a particular system. Additionally, the simulation study showed that the Hierarchical method performed very poorly under partitioned operation, as its availability measures were both unpredictable and often extremely low.

Several other replica control strategies were examined in Chapter 2, but not implemented in the simulation either because of their similarities to the simulated strategies [6,10] or due to major differences in system or transaction processing assumptions [11,12]. The El-Abbadi Views protocol [6] achieves the same degree of availability as the Herlihy method when the relaxed read and write rules are applied, while eliminating the expensive restart costs that the latter method entails when the network is partitioned. The Views protocol does have its own expensive overhead however, due to the update transactions that are initiated to change the views in the system. In effect, these transactions would have a similar effect to transactions restarting, as they would delay user transactions which are queuing for service. The approach of ordering transactions dynamically [10] and Multi-Class Replication Protocol [11] have the same availability bounds as the Missing Writes strategy, and also have fairly substantial overhead costs which compare with the restart costs generated in the Missing Writes

protocol. The Virtual Regeneration Protocol [12] ensures that *some* site will always be able to perform operations on a data object, even if only one copy of the object is available. This strategy is at the expense of overall system availability however, as it is possible that permission to access the object will be limited to some small partition of the sites, while transactions operating in a much larger majority partition would be forced to abort. Failure recovery elsewhere in the network could permit all other copies of the object to be available to such a transaction, but the transaction would be forced to abort because these copies did not participate in any recent updates. This result implies very poor availability results for this strategy, and there are substantial overhead costs involved in generating new copies of objects.

5.2 Future Work

In any simulation study, there are always innumerable possible variations to the system being modelled. Obvious candidates for further experimentation are changes to the size of the system, input parameters and costs, and different alternatives for queuing and execution policies.

In terms of this study specifically, there are two additional directions which should be explored. Firstly, the current implementation encompasses transactions operating under the equilibrium partitioned and non-partitioned states. The behaviour of the system under the different replica control strategies during network breakdown and recovery is also an interesting case to examine.

Secondly, a system in which *dominant local access* exists may provide substantially different availability and timing measures for the various replica control strategies. This assumption is more realistic than the uniform data access model, as it assumes that a majority of transactions accessing an object will originate at a site at which a copy of the object exists, or where numerous copies of the object exist at nearby sites. Dominant local access combined with partial replication would permit a greater number of transactions to be classified as operating in the majority partition, and thus result in higher availability levels than under uniform access, even for low degrees of replication. The average cost to a transaction of accessing an object would also decrease, as a greater number of transactions would be able to take advantage of accessing local copies at relatively low communication costs. As the availability *bounds* would not change under this assumption, the relative performance of the strategies studied in this thesis would remain the same, as the gains in availability would be due to larger numbers of transactions falling into a majority partition. The strategies with the higher bounds would therefore achieve greater *absolute* availability gains from this access policy than those with lower bounds. An interesting goal for future research would be to determine good read / write set object distributions under the dominant local access assumption and to study the absolute availability and timing gains realized.

References

[1] S.B. Davidson, H. Garcia-Molina, D. Skeen. "Consistency in Partitioned Networks". Computing Surveys, 17(3):341-370, September 1985.

[2] D.K. Gifford. "Weighted Voting for Replicated Data". In Proceedings of the 7th Symposium on Operating Systems Principles, pages 150-159, 1979.

[3] H. Garcia-Molina, D. Barbara. "How to assign votes in a distributed system". Journal of the ACM, 34(2):841-861, October 1985.

[4] D. Eager, K.C. Sevcik. "Achieving Robustness in Distributed Database Systems". ACM Transactions on Database Systems, 8(3):354-381, September 1983.

[5] M. Herlihy. "Dynamic Quorum Adjustment for Partitioned Data". ACM Transactions on Database Systems, 12(2):170-194, June 1987.

[6] A. El-Abbadi, S. Toueg. "Maintaining Availability in Partitioned Replicated Databases". ACM Transactions on Database Systems, 14(2):264-290, June 1989.

[7] A. Kumar. "Performance Analysis of a Hierarchical Quorum Consensus Algorithm for Replicated Objects". In Proceedings of the 10th International Conference on Distributed Computing Systems, pages 378-385, 1990.

[8] A. Kumar. "Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data". IEEE Transactions on Computers, 40(9):996-1004, September 1991.

[9] A. Kumar, K. Malik. "Optimizing the Cost of Quorum Consensus through Hierarchies". To Appear. 1993.

[10] A. Sandoz. "Achieving High Availability in a Replicated File System by Dynamically Ordering Transactions". In Proceedings of the 10th International Conference on Distributed Computing Systems, pages 432-439, 1990.

[11] P. Triantafillou, D. Taylor. "Using Multiple Replica Classes to Improve Performance in Distributed Systems". In Proceedings of the 11th International Conference on Distributed Computing Systems, pages 420-428, 1991.

[12] N.R. Adam, R. Tewari. "Regeneration with Virtual Copies for Replicated Databases". In Proceedings of the 11th International Conference on Distributed Computing Systems, pages 429-436, 1991.

[13] B.A. Coan, B.M. Oki, E.K. Kolodner. "Limitations on Database Availability when Networks Partition". In Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing, pages 187-194, 1986.

[14] A. Shah, K. Marzullo. "Trade-offs Between Replication and Availability in Distributed Databases". Technical Report TR 89-1065, Department of Computer Science, Cornell University, December 1989.

Appendix A Simulation Results

This appendix contains the results of running each of the transaction processing strategies through the simulation engine. The tables of results are broken down by strategy, degree of replication and partitioned or non-partitioned operation. All of the trials were conducted on a 29-node network containing 10 data objects. Identical transaction sets were used in each simulation. In the case of partial replication, the data object arrangement was also identical across the various strategy/partitioning combinations.

For all of the tables in this appendix, the following abbreviations apply:

T = number of transactions initiating.

C = number of transactions successfully committing.

A = number of transactions aborting.

RQ = average read quorum size.

WQ = average write quorum size.

NR = average number of restarts.

RT = average residence time.

ST = average service time.

Trial 1
Strategy: Gifford
Degree of Replication: Full
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	15	15	0	94.47	26.92
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	15	15	0	94.47	26.92
R/O								
Maj. :	44	44	0	15	0	0	39.41	4.28
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	15	0	0	39.41	4.28
W/O								
Maj. :	50	50	0	0	15	0	58.80	11.04
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	15	0	58.80	11.04
Updates								
Maj. :	865	864	0	15	15	0	99.34	28.99
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	15	15	0	99.34	28.99

Trial 2
Strategy: Missing Writes
Degree of Replication: Full
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	1	29	0	91.14	26.63
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	1	29	0	91.14	26.63
R/O								
Maj. :	44	44	0	1	0	0	37.80	4.00
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	1	0	0	37.80	4.00
W/O								
Maj. :	50	50	0	0	29	0	55.60	11.04
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	29	0	55.60	11.04
Updates								
Maj. :	865	864	0	1	29	0	95.91	28.68
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	1	29	0	95.91	28.68

Trial 3
Strategy: Herlihy
Degree of Replication: Full
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	1	29	0	91.14	26.63
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	1	29	0	91.14	26.63
R/O								
Maj. :	44	44	0	1	0	0	37.80	4.00
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	1	0	0	37.80	4.00
W/O								
Maj. :	50	50	0	0	29	0	55.60	11.04
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	29	0	55.60	11.04
Updates								
Maj. :	865	864	0	1	29	0	95.91	28.68
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	1	29	0	95.91	28.68

Trial 4
Strategy: Hierarchical
Degree of Replication: Full
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	4	16	0	91.80	26.69
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	4	16	0	91.80	26.69
R/O								
Maj. :	44	44	0	4	0	0	37.21	4.06
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	4	0	0	37.21	4.06
W/O								
Maj. :	50	50	0	0	16	0	56.57	11.04
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	16	0	56.57	11.04
Updates								
Maj. :	865	864	0	4	16	0	96.62	28.75
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	4	16	0	96.62	28.75

Trial 5
Strategy: Gifford
Degree of Replication: 15-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	8	8	0	80.56	25.69
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	8	8	0	80.56	25.69
R/O								
Maj. :	44	44	0	8	0	0	33.68	4.15
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	8	0	0	33.68	4.15
W/O								
Maj. :	50	50	0	0	8	0	47.89	10.38
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	8	0	47.89	10.38
Updates								
Maj. :	865	864	0	8	8	0	84.84	27.67
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	8	8	0	84.84	27.67

Trial 6
Strategy: Missing Writes
Degree of Replication: 15-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	1	15	0	79.10	25.54
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	1	15	0	79.10	25.54
R/O								
Maj. :	44	44	0	1	0	0	32.62	4.01
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	1	0	0	32.62	4.01
W/O								
Maj. :	50	50	0	0	15	0	46.85	10.38
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	15	0	46.85	10.38
Updates								
Maj. :	865	864	0	1	15	0	83.33	27.52
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	1	15	0	83.33	27.52

Trial 7
Strategy: Herlihy
Degree of Replication: 15-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	1	15	0	79.10	25.54
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	1	15	0	79.10	25.54
R/O								
Maj. :	44	44	0	1	0	0	32.62	4.01
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	1	0	0	32.62	4.01
W/O								
Maj. :	50	50	0	0	15	0	46.85	10.38
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	15	0	46.85	10.38
Updates								
Maj. :	865	864	0	1	15	0	83.33	27.52
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	1	15	0	83.33	27.52

Trial 8
Strategy: Hierarchical
Degree of Replication: 15-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	2	10	0	79.31	25.56
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	2	10	0	79.31	25.56
R/O								
Maj. :	44	44	0	2	0	0	32.77	4.03
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	2	0	0	32.77	4.03
W/O								
Maj. :	50	50	0	0	10	0	46.98	10.38
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	10	0	46.98	10.38
Updates								
Maj. :	865	864	0	2	10	0	83.55	27.54
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	2	10	0	83.55	27.54

Trial 9
Strategy: Gifford
Degree of Replication: 7-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	4	4	0	73.43	24.98
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	4	4	0	73.43	24.98
R/O								
Maj. :	44	44	0	4	0	0	31.91	4.08
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	4	0	0	31.91	4.08
W/O								
Maj. :	50	50	0	0	4	0	44.88	10.01
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	4	0	44.88	10.01
Updates								
Maj. :	865	864	0	4	4	0	77.20	26.91
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	4	4	0	77.20	26.91

Trial 10
Strategy: Missing Writes
Degree of Replication: 7-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	1	7	0	72.83	24.92
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	1	7	0	72.83	24.92
R/O								
Maj. :	44	44	0	1	0	0	31.47	4.02
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	1	0	0	31.47	4.02
W/O								
Maj. :	50	50	0	0	7	0	44.46	10.01
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	7	0	44.46	10.01
Updates								
Maj. :	865	864	0	1	7	0	76.58	26.85
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	1	7	0	76.58	26.85

Trial 11
Strategy: Herlihy
Degree of Replication: 7-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	1	7	0	72.83	24.92
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	1	7	0	72.83	24.92
R/O								
Maj. :	44	44	0	1	0	0	31.47	4.02
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	1	0	0	31.47	4.02
W/O								
Maj. :	50	50	0	0	7	0	44.46	10.01
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	7	0	44.46	10.01
Updates								
Maj. :	865	864	0	1	7	0	76.58	26.85
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	1	7	0	76.58	26.85

Trial 12
Strategy: Hierarchical
Degree of Replication: 7-site
Partitioned: no

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	959	958	0	1	7	0	72.83	24.92
Min. :	0	0	0	0	0	0	0	0
All :	959	958	0	1	7	0	72.83	24.92
R/O								
Maj. :	44	44	0	1	0	0	31.47	4.02
Min. :	0	0	0	0	0	0	0	0
All :	44	44	0	1	0	0	31.47	4.02
W/O								
Maj. :	50	50	0	0	7	0	44.46	10.01
Min. :	0	0	0	0	0	0	0	0
All :	50	50	0	0	7	0	44.46	10.01
Updates								
Maj. :	865	864	0	1	7	0	76.58	26.85
Min. :	0	0	0	0	0	0	0	0
All :	865	864	0	1	7	0	76.58	26.85

Trial 13
Strategy: Gifford
Degree of Replication: Full
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	699	0	15	15	0	50.03	26.63
Min. :	260	0	260	0	0	0	0	0
All :	959	699	260	15	15	0	50.03	26.63
R/O								
Maj. :	30	30	0	15	0	0	20.93	4.57
Min. :	14	0	14	0	0	0	0	0
All :	44	30	14	15	0	0	20.93	4.57
W/O								
Maj. :	40	40	0	0	15	0	31.32	10.85
Min. :	10	0	10	0	0	0	0	0
All :	50	40	10	0	15	0	31.32	10.85
Updates								
Maj. :	629	629	0	15	15	0	52.61	28.68
Min. :	236	0	236	0	0	0	0	0
All :	865	629	236	15	15	0	52.61	26.68

Trial 14
Strategy: Missing Writes
Degree of Replication: Full
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	699	0	15	15	1	60.21	26.63
Min. :	260	14	245	1	0	0	3.43	3.43
All :	959	713	245	14.88	15	0.98	59.10	26.17
R/O								
Maj. :	30	30	0	15	0	1	36.50	4.57
Min. :	14	14	0	1	0	0	3.43	3.43
All :	44	44	0	11.18	0	0.68	25.98	4.21
W/O								
Maj. :	40	40	0	0	15	1	41.32	10.85
Min. :	10	0	10	0	0	0	0	0
All :	50	40	10	0	15	1	41.32	10.85
Updates								
Maj. :	629	629	0	15	15	1	62.54	28.68
Min. :	236	0	235	0	0	0	0	0
All :	865	629	235	15	15	1	62.54	26.68

Trial 15
Strategy: Herlihy
Degree of Replication: Full
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	699	0	14.69	15	0.96	59.20	26.61
Min. :	260	24	235	1	1	0.83	14.15	5.82
All :	959	723	235	14.57	14.89	0.95	57.71	25.92
R/O								
Maj. :	30	30	0	1	0	0	12.93	4.27
Min. :	14	14	0	1	0	0	3.43	3.43
All :	44	44	0	1	0	0	9.91	4.00
W/O								
Maj. :	40	40	0	0	15	1	40.81	10.85
Min. :	10	10	0	0	1	2	29.16	9.16
All :	50	50	0	0	12.46	1.2	38.48	10.51
Updates								
Maj. :	629	629	0	15	15	1	62.54	28.68
Min. :	236	0	235	0	0	0	0	0
All :	865	629	235	15	15	1	62.54	26.68

Trial 16
Strategy: Hierarchical
Degree of Replication: Full
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	699	0	4	16	0	49.18	26.41
Min. :	260	0	260	0	0	0	0	0
All :	959	699	260	4	16	0	49.18	26.41
R/O								
Maj. :	30	30	0	4	0	0	20.52	4.33
Min. :	14	0	14	0	0	0	0	0
All :	44	30	14	4	0	0	20.52	4.33
W/O								
Maj. :	40	40	0	0	16	0	30.60	10.85
Min. :	10	0	10	0	0	0	0	0
All :	50	40	10	0	16	0	30.60	10.85
Updates								
Maj. :	629	629	0	4	16	0	51.73	28.45
Min. :	236	0	236	0	0	0	0	0
All :	865	629	236	4	16	0	51.73	28.45

Trial 17
Strategy: Gifford
Degree of Replication: 15-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	699	0	8	8	0	46.39	25.66
Min. :	260	0	260	0	0	0	0	0
All :	959	699	260	8	8	0	46.39	25.66
R/O								
Maj. :	30	30	0	8	0	0	19.74	4.43
Min. :	14	0	14	0	0	0	0	0
All :	44	30	14	8	0	0	19.74	4.43
W/O								
Maj. :	40	40	0	0	8	0	27.99	10.35
Min. :	10	0	10	0	0	0	0	0
All :	50	40	10	0	8	0	27.99	10.35
Updates								
Maj. :	629	629	0	8	8	0	48.83	27.65
Min. :	236	0	236	0	0	0	0	0
All :	865	629	236	8	8	0	48.83	27.65

Trial 18
Strategy: Missing Writes
Degree of Replication: 15-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	699	0	8	8	1	56.58	25.66
Min. :	260	14	245	1	0	0	3.44	3.44
All :	959	713	245	7.94	8	0.98	55.53	25.23
R/O								
Maj. :	30	30	0	8	0	1	35.36	4.43
Min. :	14	14	0	1	0	0	3.44	3.44
All :	44	44	0	6.09	0	0.68	25.20	4.11
W/O								
Maj. :	40	40	0	0	8	1	37.99	10.35
Min. :	10	0	10	0	0	0	0	0
All :	50	40	10	0	8	1	37.99	10.35
Updates								
Maj. :	629	629	0	8	8	1	58.77	27.65
Min. :	236	0	235	0	0	0	0	0
All :	865	629	235	8	8	1	58.77	27.65

Trial 19
Strategy: Herlihy
Degree of Replication: 15-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	699	0	7.84	8	0.96	55.63	25.66
Min. :	260	24	235	1	1	0.83	14.09	5.76
All :	959	723	235	7.79	7.95	0.95	54.25	25.00
R/O								
Maj. :	30	30	0	1	0	0	13.53	4.28
Min. :	14	14	0	1	0	0	3.44	3.44
All :	44	44	0	1	0	0	10.32	4.01
W/O								
Maj. :	40	40	0	0	8	1	37.52	10.35
Min. :	10	10	0	0	1	2	29.01	9.01
All :	50	50	0	0	6.73	1.2	35.82	10.08
Updates								
Maj. :	629	629	0	8	8	1	58.79	27.65
Min. :	236	0	235	0	0	0	0	0
All :	865	629	235	8	8	1	58.79	27.65

Trial 20
Strategy: Hierarchical
Degree of Replication: 15-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	699	134	565	2	10	0	11.67	11.47
Min. :	260	5	255	2	0	0	2.82	2.82
All :	959	139	820	2	10	0	11.35	11.16
R/O								
Maj. :	30	30	0	2	0	0	4.97	4.30
Min. :	14	5	9	2	0	0	2.82	2.82
All :	44	35	9	2	0	0	4.66	4.09
W/O								
Maj. :	40	13	27	0	10	0	7.10	7.10
Min. :	10	0	10	0	0	0	0	0
All :	50	13	37	0	10	0	7.10	7.10
Updates								
Maj. :	629	91	538	2	10	0	14.54	14.46
Min. :	236	0	236	0	0	0	0	0
All :	865	91	774	2	10	0	14.54	14.46

Trial 21
Strategy: Gifford
Degree of Replication: 7-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	249	249	0	4	4	0	17.84	16.50
Min. :	710	0	710	0	0	0	0	0
All :	959	249	710	4	4	0	17.84	16.50
R/O								
Maj. :	24	24	0	4	0	0	4.09	3.65
Min. :	20	0	20	0	0	0	0	0
All :	44	24	20	4	0	0	4.09	3.65
W/O								
Maj. :	27	27	0	0	4	0	8.18	7.59
Min. :	23	0	23	0	0	0	0	0
All :	50	27	23	0	4	0	8.18	7.59
Updates								
Maj. :	198	198	0	4	4	0	20.82	19.27
Min. :	667	0	667	0	0	0	0	0
All :	865	198	667	4	4	0	20.82	19.27

Trial 22
Strategy: Missing Writes
Degree of Replication: 7-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	249	249	0	3.99	4	0.99	27.83	16.50
Min. :	710	15	694	1	0	0	3.48	3.48
All :	959	264	694	3.88	4	0.94	26.45	15.76
R/O								
Maj. :	24	24	0	3.86	0	0.92	14.02	3.65
Min. :	20	15	5	1	0	0	3.48	3.48
All :	44	39	5	2.78	0	0.56	9.96	3.59
W/O								
Maj. :	27	27	0	0	4	1	18.18	7.59
Min. :	23	0	23	0	0	0	0	0
All :	50	27	23	0	4	1	18.18	7.59
Updates								
Maj. :	198	198	0	4	4	1	30.82	19.27
Min. :	667	0	666	0	0	0	0	0
All :	865	198	666	4	4	1	30.82	19.27

Trial 23
Strategy: Herlihy
Degree of Replication: 7-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	249	249	0	3.81	4	0.90	26.97	16.49
Min. :	710	43	666	1	1	1.07	19.80	8.82
All :	959	292	666	3.63	3.71	0.93	25.92	15.36
R/O								
Maj. :	24	24	0	1	0	0	4.80	3.60
Min. :	20	20	0	1	0	0	4.52	4.52
All :	44	44	0	1	0	0	4.67	4.02
W/O								
Maj. :	27	27	0	0	4	1	18.18	7.59
Min. :	23	23	0	0	1	2	33.09	12.56
All :	50	50	0	0	2.24	1.46	25.04	9.87
Updates								
Maj. :	198	198	0	4	4	1	30.86	19.27
Min. :	667	0	666	0	0	0	0	0
All :	865	198	666	4	4	1	30.86	19.27

Trial 24
Strategy: Hierarchical
Degree of Replication: 7-site
Partitioned: yes

	T	C	A	RQ	WQ	NR	RT	ST
Total								
Maj. :	249	24	225	1	0	0	3.60	3.60
Min. :	710	20	690	1	0	0	4.52	4.52
All :	959	44	915	1	0	0	4.02	4.02
R/O								
Maj. :	24	24	0	1	0	0	3.60	3.60
Min. :	20	20	0	1	0	0	4.52	4.52
All :	44	44	0	1	0	0	4.02	4.02
W/O								
Maj. :	27	0	27	0	0	0	0	0
Min. :	23	0	23	0	0	0	0	0
All :	50	0	50	0	0	0	0	0
Updates								
Maj. :	198	0	198	0	0	0	0	0
Min. :	667	0	667	0	0	0	0	0
All :	865	0	865	0	0	0	0	0

Appendix B Combinatorial Placement Calculation

In Section 4.2.2, probabilities were given for the likelihood of various combinations of data object placement given a partitioned network. These calculations assume independence of placement of copies of various objects and also equal likelihood of all possible partitions of a particular size.

The probability of having x copies of an object in a partition of size m and y copies of an object in a second, non-overlapping partition of size n, where there are a total of $x + y$ copies of the object, and $m + n$ network sites is:

$$\frac{\binom{m}{x} * \binom{n}{y}}{\binom{m+n}{x+y}}$$