Performance Study of Protocols in Replicated Database

By

Ching-Ting, Ng

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF PHILOSOPHY

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

THE CHINESE UNIVERSITY OF HONG KONG

JUNE 1996

Abstract

For the traditional protocols in replicated database, their transaction managements are centralized. That is, when there are a number of operations $o_1, o_2, o_3, ...o_n$ in a transaction and some operation o_i depends on the result of some previous operation o_j, some communication overhead is required to secure the result of o_j before o_i is started.

As communication cost is high compared to computational and data storage cost in many applications [Gra88], especially for the WAN, some researchers have recently proposed to eliminate this type of communication overhead by replication of transaction. In particular, Transaction Replication Scheme(TRS) [cFlC94] is shown, theoretically, to be efficient in terms of communication time and message. Instead of centralized control of each transaction, the transaction is broadcasted so as to reduce the message overhead involved. TRS only considers two kinds of data, shared private data and public data. Compared with other previous work on replicated distributed database systems which assumes only public data, it utilizes the semantics of node autonomy. Moreover, TRS also can handle partition failure and site failure.

As the practical performance characteristics of TRS has not been studied extensively. In this thesis, we will investigate the performance characteristics of TRS, that includes the parametric value of TRS period, the ratio of local to public transactions and the number of operations, etc.

Our approach is to build a simulation model to make a comparison study on both the traditional centralized transaction management protocols such as Majority Quorum Consensus and Tree Quorum Protocols, as well as the replication transaction management protocols (TRS).

Acknowledgement

I gratefully acknowledge the support and encouragement from my thesis advisor, Prof. Ada Wai-Chee Fu, without whom this thesis could not have been completed. She has been the most patient and kindest advisor. I thank my thesis committee, Prof. Chin Lu and Prof. John Lui for their efforts in making thoughtful comments on this thesis.

For these two years, I am glad that there are a number of friends supporting me. Sau-Ming Lau patiently told me about the techniques of simulation. Keith Hang-Kwong Mak shared the experience of using CSIM with me. Terry, Wai-Kwong Lau discussed with me about Transaction Replication Scheme. Johnson, Chiu-Fai Chong taught me a lot about Matlab.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Replicated Database is a database which stores data at multiple sites. Its objective is to increase fault-tolerance since copies of data continue to be available to applications in the event of local site failure or network failure; and to improve the performance as data are stored at all sites at which it is required rather than remotely across the network.

However, in order to achieve the above objectives, the system must ensure that all copies of replicated data items are consistent. One way to achieve this is to make sure that all sites at which replicas are stored to be operational and connected to the network; and all copies of each data item are updated by each logical update.

However, there are problems with this approach, they are:

1. in the event of network or site failure, it may not be possible to update copies of such replicated data items at all. Especially in the event of partition failure, data copies in one partition may be updated by one transaction while copies of data in another partition are subjected to a different update by another transaction. These two transactions are executed independent of one another since due to partitioning of network, no communication between two sites is possible. Hence if they are allowed to commit, then versions of replicated data can diverge resulting in consistency problem.
2. updating all copies of each data item instead of one copy of data item would counter the efficiency factor.

Therefore, the advantages of improved performance, availability, and site autonomy of Replicated Database mainly apply to read-only applications, and are jeopardized by the need for propagating updates to all sites.

There are other protocols which tackle these problems. These include:

1. Protocols tackling site failures, including Write All Available Approach [PVN87] and Directory Oriented Available Copies [PVN87],
2. Protocols tackling partition failure, including Primary site [PVN87], Quorum Consensus [PVN87], Missing Writes [PVN87] and Virtual Partition [AS89];
3. Protocols tackling the performance of updating all copies of each data item including: Distributing Writes Immediately [PVN87], Defer Writes Until Transaction Terminates [PVN87]

Recent Research is interested in protocols for enhancing the performance because for the real life applications such as flight reservation, the factors of performance and efficiency is very significant. For instance, it is unacceptable for a flight reservation system to become blocked or unavailable in case of a site failure or network partition. Therefore, current research focuses on delayed propagation of updates [SHKS95], transaction chopping [SLSV95] and transaction replication(TRS) [cFlC94]. Such recent research mainly emphasize on how to improve the performance and efficiency of replicated database protocols.

Nowadays, as an organization usually spans a large geographical area, the bottleneck of replicated database is mostly due to the long message transmission time in WAN, The situation cannot be alleviated even when applications are run on advanced HW platforms with plenty of physical resources.

The Transaction Replication Scheme (TRS) [cFlC94] has been proposed to address this problem. In this thesis, we investigate the performance characteristics of the Transaction Replication Scheme. Thus, we are motivated to build a simulation model and

simulate different types of system workload, different I/O requests, different amount of concurrent request conflicts and communication delay, etc.

The objective of our work is not only to evaluate the comparative performance of TRS, with other protocols such as Majority Quorum Consensus and Tree Quorum Protocols, but also to identify the factors that lead to its superior performance. The findings in our study can be used to pinpoint the aspects that require more attentions when designing new concurrency control algorithms. The ultimate aim is to give a guideline for database designers in choosing a concurrency control algorithm during the design of a database engine.

The basic contributions of our research are summarized as follows:

- Build a simulator to investigate the performance characteristic of different protocols under various database workloads
- identify the essential factors that lead to better performance
- compare traditional protocols and TRS.

This thesis is organized as follows. Chapter 2 introduces the motivation of our work and summarizes the background study of related protocols and the TRS protocol. The details of TRS can be found in Chapter 3. In Chapter 4, the simulation model of the system is introduced, followed by the simulation results and data analysis in Chapter 5. We conclude in Chapter 6.

Chapter 2

Background

Before we start our investigation on the Transaction Replication Scheme, we summarize the known protocols for replicated database in this chapter. We first describe the traditional protocols which are mainly used to tackle the site failure including Write All Available Approach [PVN87]; and Directory Oriented Available Copies [PVN87]. Next, the protocols mainly used to handle partition failure including Primary Site [PVN87]; Quorum Consensus [PVN87]; Missing Writes [PVN87] and Virtual Partition [AS89] are illustrated. Then, we summarize the protocols which are used for tackling the performance of updating all copies of each data item. The protocols include Distributing Writes Immediately [PVN87]; Defer Writes Until Transaction Terminates [PVN87]; Independent Updates and Incremental Agreement in Replicated Databases [SHKS95]. In addition, an overview of the Transaction Replication Scheme is given.

2.1 Protocols tackling site failure

1. Write-All Approach(Ideal World)[PVN87]
   Assume sites never fail, $Read(X)$ is translated into $Read(X_a)$, where X_a is any copy of data item X. $Write(X)$ is translated into $Write(X_{a1})$, ...$Write(X_{an})$, where X_{a1},...X_{an} are all copies of X. However, Write-All approach is unsatisfactory if any copy of X fails since it would have to delay processing $Write(X)$ until it could write all copies of X. Moreover, more copies of X implies a higher probability that one

copy is unaccessible. In this case, increased replication of data actually makes the system less available to update transactions.

2. Write-All Available Approach[PVN87]

   A fixed set of copies for each data item is known to every site. Each copy is assumed to be created once and can fail at most once. After creation and before failure a copy is available. Otherwise, it is unavailable. A write operation writes into all available copies. That is, it ignores any copies that are unavailable. However, this leads to the problem of correctness. Some copies of X may not reflect the most up-to-date value of X. This problem can be solved by preventing transactions from reading copies from sites that have failed and recovered until these copies are brought up-to-date.

   If the read operation of data copy X_a, $R(X_a)$ is rejected, a negative acknowledgment is returned and the transaction T_i is aborted. If $R(X_a)$ is accepted, but, if site A is down, it could submit $R(X_b)$ to another site B. If no copy of X can be read, T_i is aborted.

   Writes for which there is no response are called Missing Writes. If Missing Writes from all available sites are received, then the operation is rejected and the transaction is aborted. Otherwise, it is successful.

3. Directory-Oriented Available Copies[PVN87]

   It uses directories to define the set of sites that currently store the copies of an item. Unlike Write-All Available Approach, Directory-Oriented Available Copies can avoid transactions which attempt to update copies at down sites.

2.2 Protocols tackling Partition Failure

2.2.1 Primary site

Every data item has one copy (at one site) as the primary copy; all other copies are slave copies, each update is directed to the primary copy and then propagated to

slave copies. In the event of network partitioning, only the partitions with primary copies are available. If the primary site fails, it is possible to promote one of the slave copies and designate it as the new primary copy. A new primary copy cannot be elected if the network is partitioned due to communication failure because the original primary site may still be operational but the other partitions have no way of knowing this.

2.2.2 Quorum Consensus Protocol

One way to prevent conflicting transactions from executing in different partitions is to allow only one partition to process any transaction at all. Since the partitions cannot communicate with each other, each partition must independently decide whether it can process transaction. A quorum is a set of sites. With quorum consensus, a set of quorums which intersect each other is defined. For read and wirte operations, read quorums and write quorums are defined so that each write quorum intersects each other write quorum and each read quorum. Only the one partition which contains a quorum of sites can process a transaction.

(a) Majority Quorum[AA91]

It requires both read and write quorums to contain a majority of copies.

Let a Read Quorum be a set with $Q_r(X)$ copies and a Write Quorum be a set with $Q_w(X)$ copies. Let $N(X)$ be the total number of sites. We require that

$Q_r(X) + Q_w(X) > N(X)$, and

$2Q_w(X) > N(X)$

For instance, if there are 13 sites, i.e. $N = 13$

We can set $\{Q_w(X) = 7,\ Q_r(X) = 6\}$,$\{Q_w(X) = 8,\ Q_r(X) = 5\}$, etc.

(b) Tree Quorum[AA92]

The replicated sites are organized in the form of a logical tree for constructing

quorums. Given a set of N copies of an object X, we logically organize them into a tree of height h and degree d, that is each node has d children and the maximum height is h. We also assume that the tree is complete. A tree quorum is said to have length l and width w, or, it has dimensions $< l, w >$. The quorum is constructed by selecting the root and w children of the root, and for each selected child, w of its children, and so on, for a depth of l. If successful, this forms a tree quorum of height l and degree w.

We denote the dimensions of a read quorum Q_r by $< l_r, w_r >$;
and the dimensions of a write quorum Q_w by $< l_w, w_w >$.
The following constraints guarantee the nonempty intersection of read and write quorums, and of 2 write quorums:

$$l_r + l_w > h$$
$$w_r + w_w > d$$
$$2l_w > h$$
$$2w_w > d$$

Consider a replicated object with thirteen copies. We superimpose a ternary tree of height 3 on the copies as illustrated in Figure 2.1, with the sites numbered as shown.

For instance, $Q_r =< 1, 2 >$. As the length of a read quorum is one, so a read quorum contains only the root of the tree.
$Q_w =< 3, 2 >$, examples of write quorums are {1,2,3,5,6,8,9}, {1,2,4,6,7,12,13},etc

(c) Locking

The basic idea of locking is that whenever a transaction accesses a data item, it locks it, and that a transaction which wants to lock a data item which is already locked by another transaction must wait until the other transaction

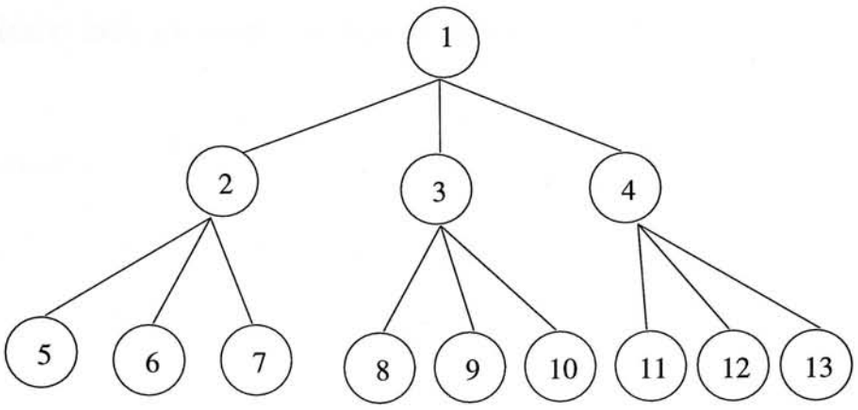


Figure 2.1: The diagram of Ternary Tree

has released the lock (unlock).

In fact, typical locking has the notion of a lock mode: a transaction locks a data item in a shared mode if it wants only to read the data item and in an exclusive mode if it wants to write the data item. A transaction is well-formed if it always locks a data item in shared mode before reading it, and it always locks a data item in exclusive mode before writing it. The following compatibility rules exist between lock modes:

- A transaction can lock a data item in a shared mode if it is not locked at all or it is locked in a shared mode by another transaction.
- A transaction can lock a data item in exclusive mode only if it is not locked at all.

Whether the protocol is Majority Quorum or Tree Quorum, the site first have to send the request lock message to a Read Quorum or a Write Quorum according to the read operation r_i or write operation w_i. When the sites receive the request lock message, if the lock is available, the site will send back the grant lock message to the site which requests the lock. However, if the lock is unavailable, it will typically queue up to wait for the lock for a time-out period. After the time-out period, if the transaction still cannot receive the grant lock message,then it will be aborted. If the transaction can successfully get the required number of lock messages, its transaction operation will be executed and at the end, the locks are released by

sending release lock messages to the quorum.

- Advantages
  - in the event of failure, quorum consensus provides greater data availability than Primary Copy and it involves less overhead to handle failures and recoveries.
  - Recoveries of copies require no special treatment as a copy of X that was down and therefore missed some writes will not have the largest version number. Thus, after it recovers, it will not be read until it has been written at least once.
- Disadvantages
  - A transaction probably access more than one copy of each data item it wants to read. This defeats one of the motivations for data replication. Since it involves more overhead to process transaction during periods in which no failures or recoveries take place as it requires consensus from a read or write quorum for every read or write operation.
  - It probably needs a large number of copies to tolerate a given number of site failures.
  - All copies of each data item must be known in advance. A known copy of X can recover, but a new copy of X cannot be created immediately.

2.2.3 Missing Writes

During normal operation since all copies are available in normal mode, the DBS (Database Management System) processes $Read(X)$ by reading any copy of X and $Write(X)$ by writing all copies of X. However, when failure is detected, the system changes to failure mode and voting (Quorum Consensus) strategy is used.

2.2.4 Virtual Partition Protocol

Each site maintains a view (a set of sites) which it believes it can communicate with. Within the view in which transaction T executes, DBS uses the approach of write-all and read any one copy. But, when a site detects a difference between its present view and the set of sites it can actually communicate with, it needs to execute a View Update Transaction.

- Advantages

  Compared with Majority Quorum Consensus, a transaction never has to access more than one copy to read a data item. Thus, the closest copy available to a transaction can always be used for reading.

2.3 Protocols to enhance the Performance of Updating

(a) Immediate Write

When a transaction issues $Write(X)$, the DBS is responsible for eventually updating a set of copies of X. It can distribute these Writes immediately at the moment it receives $Write(X)$ from the transaction.

- Advantages: Early commitment of transaction and Early detection of conflicts between operations.
- Disadvantages: Immediate Write tends to use more messages than deferred writing.

(b) Defer Writes on Replicated Copies until transaction terminates

When a transaction issues $Write(X)$, the DBS is responsible for eventually updating a set of copies of X. It can defer the Writes on replicated copies until the transaction terminates.

- Advantages: Since all replicated writes destined for the same site are put in a single message so as to minimize the number of messages required to execute a transaction. With deferred writing, the DBS delays the distribution of those Writes until after transaction T_i has terminated. If T_i aborts before it terminates, then the abortion is less costly than Immediate Write.

- Disadvantages: Performance is degraded as the commitment of a transaction is delayed compared with the Immediate Writing. The detection of conflicts between operation is delayed.

2.3.1 Independent Updates and Incremental Agreement in Replicated Databases

Transaction atomicity and serializability are major obstacles to the development of replicated databases. Many practical applications, such as automated teller machine networks, flight reservation, and part inventory control, do not require these properties. One approach is incrementally updating a distributed, replicated database without requiring multi-site atomic commit protocols. In [SHKS95] there are two main characteristics introduced for dealing with the update propagation. They are the progressive, and non-blocking characteristics. By progressive, we mean that the transaction's coordinator always commits, possibly together with a group of other sites. The update is later propagated asynchronously to the remaining sites. Non-blocking means that each site can make unilateral decisions at each step of the algorithm. Sites which cannot commit updates are brought to the same final state by means of a reconciliation mechanism. This reconciliation mechanism uses the history logs, which are stored locally at each site, to bring sites to agreement.

2.3.2 A Transaction Replication Scheme for a Replicated Database with Node Autonomy

Many replicated distributed database protocols manage execution of a given transactions at one site, accessing local and remote data copies for its operation, and organize the commit/abort of transaction from that site. This is referred to as centralized transaction management.

With centralized transaction management, if there are a number of operations O_1, O_2, ...O_n in a transaction and some operation O_i depends on the result of some previous operation O_j, some communication overhead is required (See Figure 2.2).

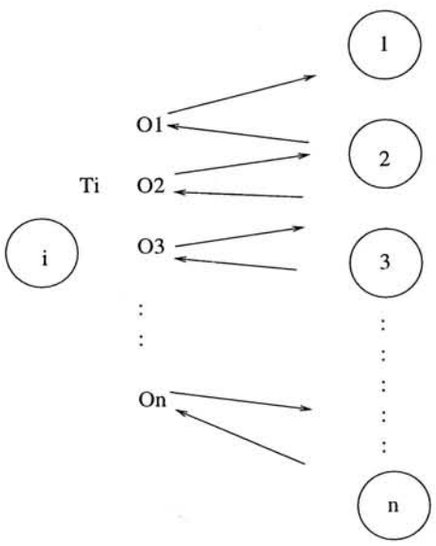


Figure 2.2: Centralized Transaction Management

To eliminate this type of overhead, a transaction must be executed entirely at one site which contains replication of a relevant data. One possible way to achieve this is to have the execution of entire transactions replicated at data replication sites.

We consider data of two types:

- shared-private data owned by a particular site and which only the owner site can modify,
- public data that all sites can modify

Thereby, two types of transactions are considered:

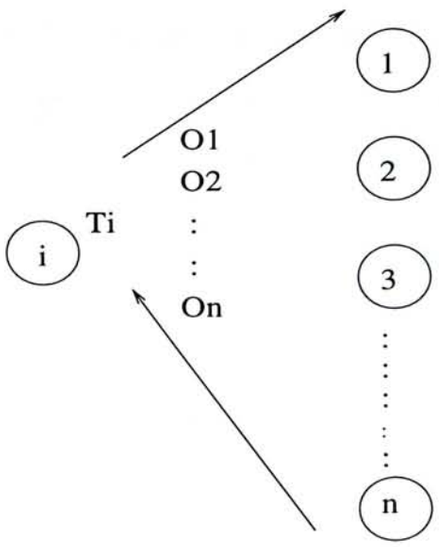


Figure 2.3: Replication of Transaction

- Local transaction - transaction initiated at site s that reads or writes only the data owned by site s. That is, it can write only the shared-private data.

- Public transaction - transaction can read both the public and /or shared-private data and can write only the public data.

In this scheme, transactions accessing only shared-private data can normally be executed and committed under a local concurrency control protocol.

Transaction Broadcasting Scheme:

A short period of time $[t1, t2)$ is considered and named TRS period. Let B be a batch of transactions submitted during such a period of time at all sites. B may contain both public and local transactions. Local transactions in B are executed immediately at their origin sites, while the public transactions in B submitted at each site are broadcast and executed on all replication sites. This means that local transaction and public transaction replicas in B are executed at different times.

This scheme requires each site to remember some old versions of the shared-private data it owns. The basic step of the protocol for each site can be summarized as follows:

- accumulate public transactions submitted at site s for a time period and broadcast

the public transaction at the end of the period. If no public transaction is accumulated, then a null messages is sent. A history is recoverable if each transaction commits after commitment of all transactions from which it reads. A site may keep multiple versions of shared-private data for the execution of global transactions. For the execution of local transactions, it needs only to consider the latest version at any time.

- At the end of each TRS period, site s broadcasts the latest committed values of shared-private data updated by local transactions committed in that period, together with a local batch of public transactions accumulated during that period at s.

- After a global batch arrives, site s examines the messages that have been received from the other sites, which may contain new versions of the senders' shared-private data. Those new versions of shared-private data are first written to the local copies of shared-private data. Site s then executes the public transactions of the global batch received.

Advantages

With TRS, the execution of public transactions normally incurs only 2 communication delays. Hence TRS is more efficient than centralized transaction management schemes in terms of communication delay, especially when the transaction consists of multiple interdependent operations.

Local concurrency control is adopted for local transactions that access only the shared-private data and it can also enhance the performance.

Disadvantages

If public transactions are not frequent, then TRS may generate a lot of wasteful null messages. In addition, TRS repeats the execution of each transaction at multiple sites, hence it incurs more computation overhead if the transaction's computational cost is high.

Besides, the storage for keeping multiple versions of shared-private data is required, if this becomes a problem, all data is made public data.

Application

As TRS generates null messages if no public transactions are received, it is more useful for busy system. Also, TRS transmits transactions instead of data items. In other words, it is especially more efficient if the size of transaction is smaller than that of the data. Moreover, TRS is better for the business applications which are more I/O-oriented than computation-oriented.

Chapter 3

Transaction Replication Scheme

3.1 A TRS for a Replicated Database with Node Autonomy

In the Transaction Replication Scheme (TRS), the execution of entire transactions is replicated at the data replication sites.

In TRS, we consider data of two types (Figure 3.1). The first type is shared-private data, which is owned by a particular site and only the owner site can modify. The second type is public data, which all sites can modify. Since shared-private data is modified by only one owner site, a simpler concurrency control is sufficient. In TRS, transactions accessing only shared-private data can be executed and committed under a local concurrency control protocol. Most previous work on replicated distributed database assume only public data. With the introduction of shared-private data, TRS can utilize the semantics of node autonomy to improve the overall performance of transaction execution.

3.1.1 Example

For example, in an airline database system, there may be an accounting site, a flight scheduling site and many sites for seat reservation. Reservation sites read flight schedules and policies determined by the scheduling site and the accounting site but they will not modify such data. Hence flight schedules and accounting policies are shared-private data.

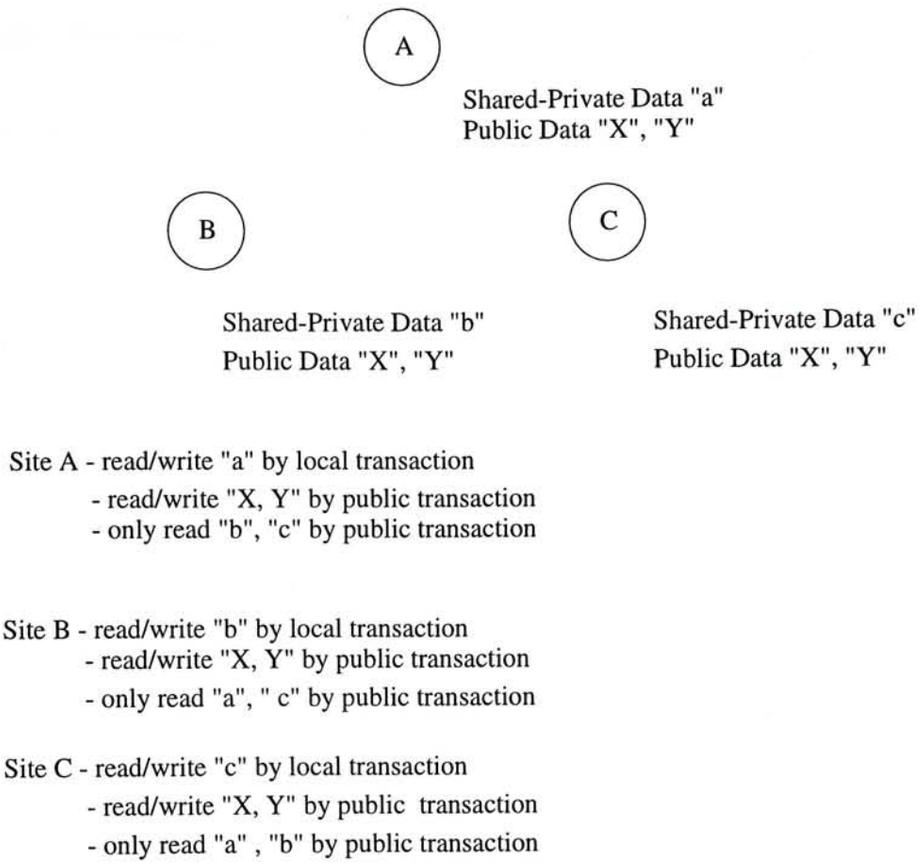


Figure 3.1: Public and Shared-private data

The seat plan of each flight is public data among reservation sites since each site can book seats and modify the data. Thus, the seat plan is considered public data.

3.1.2 Problem

As transactions are replicated at many sites, we have to ensure the essential (*serialized*) orderings of transaction execution are identical at all the replication sites. Therefore, the global timestamping and the Conservative Timestamping method are used to prevent aborts in normal operation.

3.1.3 Network Model

The system model includes a set of sites connected through a communication network. The sites store the replicated data and they are called replication sites.

The assumptions about the network and timing are as follows:-

1. Each replication site has a unique ID.

2. Messages are sent in FIFO order.

3. Each replication site has a clock. The clock are synchronized to within a small deriation from each other.

4. Each site may suffer from fail-stop failure. That means, when a site fails, it stops processing. So, the site is either working correctly (is operational) or not working at all (is down). The communication links are subject to failures. The failures result in loss of messages. Furthermore, we assume that the communication link is a 2-way connection so that if the link between Site A and Site B fails, then both communication from Site A to Site B and from Site B to Site A are disabled.

3.1.4 Transaction and Data Model

A transaction accesses data X by operations $READ(X, y)$ and $WRITE(X, v)$. A $READ(X, y)$ operation reads the value of data X and returns it in a variable y. $WRITE(X, v)$ updates the value of X to that of v. In addition, each transaction contains a $COMMIT$ or an $ABORT$ as its last operation.

The shared-private data and public data are defined in terms of transaction operations as follows:-

- Shared-private data X owned by site s - only transactions submitted by the owner site s of data X can perform $WRITE(X, v)$; and these transactions can only access the shared-private data at s; transactions submitted at other replication sites can only perform $READ(X, y)$.

- Public data X - transactions submitted at any site can perform $WRITE(X, v)$ and $READ(X, y)$.

  All shared-private and public data are assumed to be fully replicated at the replication sites.

Accordingly, we can identify two main types of transactions in TRS:-

- Local transaction - a transaction initiated at site s which reads or writes only logical data owned by site s.
- Public transaction - a transaction which may read public and /or shared-private data and can write only public data.

3.1.5 Histories and One-Copy Serializability

Let X_i denote the copy of a data object X at site i. A data object and its copies are called logical data object and physical data objects, respectively. When a transaction T_i is executed, the system uses a translation function τ_i to translate a logical operation into a set of physical operations. That is, a write operation W_i that writes X, $W_i[X]$, is translated into $W_i[X_a], W_i[X_b], ..., W_i[X_l]$, where $X_a, ..., X_l$ are copies of X and $R_i[X]$ is translated into $R_i[X_1], R_i[X_2], ..., R_i[X_m]$, where $X_1, ..., X_m$ are copies of X.

A replicated history is used to model the execution of a set of transactions with replicated data objects. A set T of transactions is a partially ordered set $\{T_i = (\Sigma_i, <_i)\}$ where Σ_i is the set of reads and writes issued by transaction i, and $<_i$ indicates the order in which those operations execute. A replicated history over such a set T is a partially ordered set $L = (\Sigma(T), <)$ such that

1. $\Sigma(T) = \cup_{i=0}^{f} \tau_i(\Sigma_i)$, where τ_i is the translation function for T_i;
2. for each i and any two operations p_i and q_i in Σ_i, if $a \in \tau_i(p_i), b \in \tau_i(q_i)$ and $p_i <_i q_i$, and if a and b operate at the same site, then $a < b$;
3. all pairs of conflicting physical operations are $<$ related; and
4. T contains two fictitious transactions T_o and T_f. T_o is translated into a set of physical write operations, one for each copy of each data object, and these precedes all other physical operations, T_f is translated into a set of physical read operations,

one for each copy of each data object, and these are preceded by all other physical operations.

In the replicated histories, the ordering of logical operations within a transaction is preserved by its physical replicas at each site. A committed transaction T_j reads X from another transaction T_i in a replicated history L=($\Sigma(T), <$) if there exists a copy X_a such that

1. $W_i[X_a]$ and $R_j[X_a]$ are operations in $\Sigma(T)$;
2. $W_i[X_a] < R_j[X_a]$; and
3. there is no $W_k[X_a]$ such that $W_i[X_a] < W_k[X_a] < R_j[X_a]$.

T_j may read X from two or more transactions, each physical read operation being performed at a different copy. A one-to-one read-from relation exists if for each transaction T and for each X that T reads, T reads X from exactly one transaction.

A replicated history L_1 is equivalent to another history L_2 if both L_1 and L_2 have the same read-from relation. A history H is serial if for any two transactions T_i, T_j that appear in H, either all operations of T_i appears before all operations of T_j or vice versa. A one-copy serial history is a serial history that consists only of logical operations. A replicated history is one-copy serializable if it is equivalent to a one-copy serial history over the same set of logical transactions. The TRS protocol will ensure one-copy serializability.

3.1.6 Transaction Broadcasting Scheme

Let the time at each site be divided into equal intervals called a TRS periods. Let B be the batch of transactions submitted during a TRS period from all sites. B may contain both public and local transactions. Local transactions in B are executed immediately at their origin sites, while the public transactions in B submitted at each site are essentially broadcast and executed on all replication sites. This means that the local transaction and the public transaction replicas in B are executed at different times.

Basic steps of the protocol for each site:-

- accumulate public transactions submitted at each site s for a TRS time period and broadcast the public transaction at the end of period. If no public transaction is accumulated, then a null message is sent.

- At the end of each TRS period, Site s also broadcasts the latest committed values of shared-private data updated by local transaction committed in that period, that is, together with the local batch of public transaction accumulated during that period at Site s.

- After a global batch has arrived, site s examines the messages that have been received from the other sites, which may contain the new versions of senders' shared-private data, these are first written to the local copies of the shared-private data. Site s then executes the public transactions of the global batch.

Definition 1: Clock values is a set of real numbers which can be divided into intervals of $[t_1, t_2)$, where t_2 - t_1 is a constant value equal to δ. Each of these intervals is called a TRS period. δ is the length of duration of a TRS period.

Definition 2: A batch of public transaction collected in a TRS period of $[t_1, t_2)$ at a site s is called a local batch of s at t_2. We define the global batch at time t_2 as the set of all the public transactions collected at each replication site in the period $[t_1, t_2)$.

After every δ time units, s starts the next period of global transaction accumulation and broadcast. For example, if $\delta = 0.5$, then each site broadcasts at times 0.5, 1.0, 1.5,..., of its local clock. If we concatenate the submission TRS time of each global transaction with the unique site ID, then we get a globally unique timestamp for each global transaction.

Once a site s has received broadcast messages from all sites at the same TRS time, it executes the global batch collected based on their timestamp order.

3.1.7 Local Transactions

Under the local concurrency control scheme, the sites can execute the local transactions immediately upon submission.

3.1.8 Public Transactions

Consider a public transaction T broadcast by a site s_2 at time t which is executed at site s_1, and suppose it reads a shared-private data object X. T should read the *virtual version* X_p of X , which was *implicitly broadcast* by some site s_3 at time t, where s_3 is the owner site of X. If a physical version X_p actually exists for a virtual version of a shared-private data item X at time t and a newer version also exists, then version X_p is discarded when the execution of the global batch at t is finished.

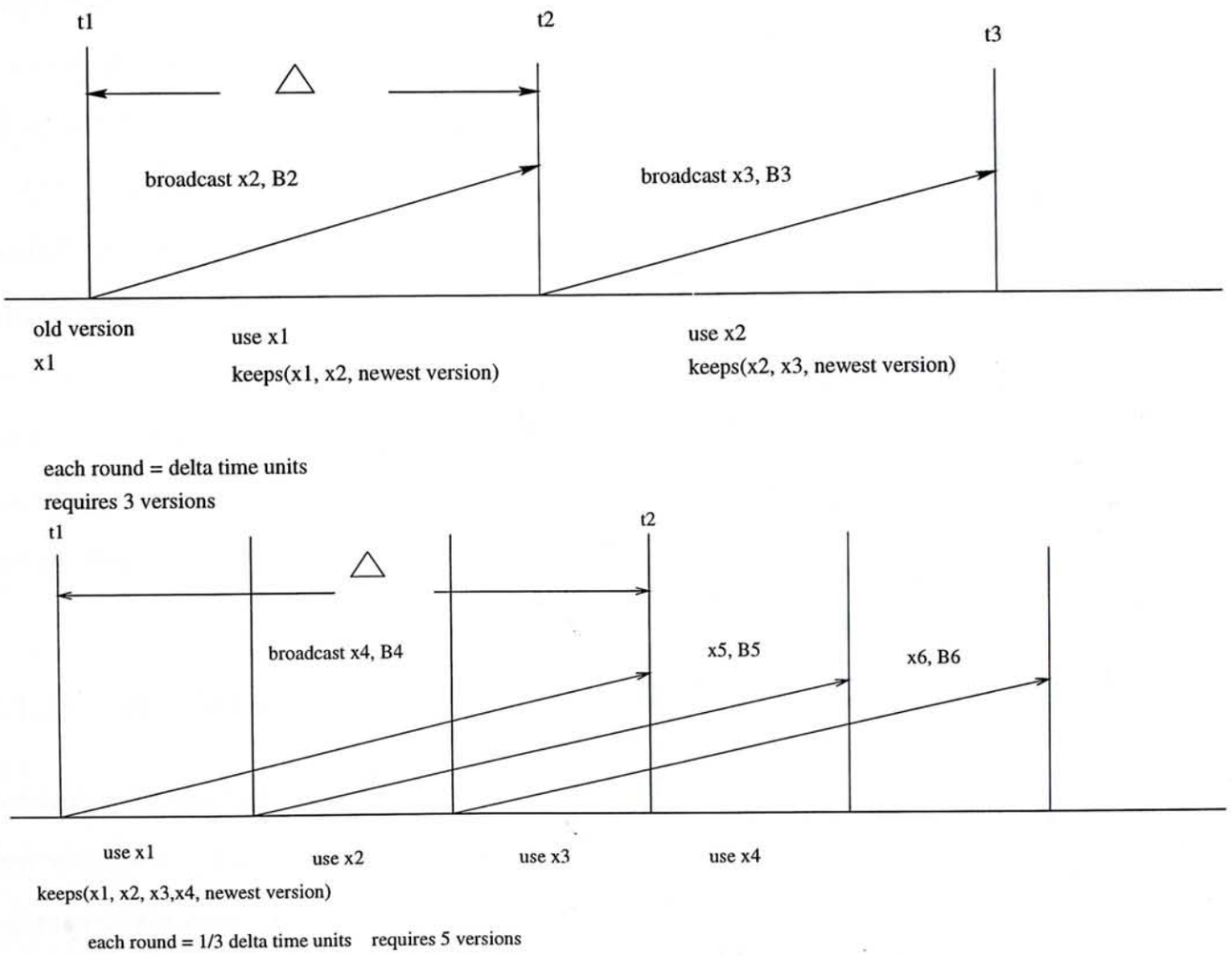


Figure 3.2: Transaction broadcast and multiple versions of shared-private data

In figure 3.2, we assume that a site s owns a shared-private data object X with consecutive versions, $x_1, x_2, x_3, x_4, x_5, x_6$. When site s broadcasts a new version x_i, it also broadcasts a new local batch of public transactions B_i submitted at s. When all the messages of a period are received from all sites, the execution can be started. For

example, in Figure 3.2, a site can start executing a global batch of public transactions B_2 at time t_2. In this figure, we assume that all message transmit take Δ time. Between t_2 and t_3, site s must keep x_2 since execution of B_2 uses x_2, it keeps x_3 because B_3 will need x_3, it also keeps the newest version of X which is used by the current local transactions. The lower half of the figure shows that for TRS periods of length $\Delta/3$, 5 versions of shared-private data have to be kept.

Generally, if a message sent at TRS time t at site s arrives at another site s' at TRS time t' where $t' > t$ and $t' = t+\Delta$, if each TRS period has length Δ/q, and if the execution of a global batch of transactions can be completed in δ time units, then in the worst cast, $\lceil q \rceil + 2$ versions of some local shared-private data are required at s'.

If $t' \leq t$ above, then at most 2 versions are needed for s' but more versions will be needed at s since the clock of s is ahead of that of s'. We assume that the system can handle the execution of a global batch within one TRS period, else the system is receiving more work than it can manage. If the average message delay D is less than δ and if clocks are closely synchronized, then only a maximum of 3 versions are required. If a shared-private data has not been updated in a period, then no extra version is needed for this period.

3.1.9 A Conservative Timestamping Algorithm

Concurrency control using conservative timestamping ordering does not require transaction abortion. The periodical broadcasting of transactions makes this approach easier because little waiting is necessary for a site to make sure that no transaction with older timestamps will be received from other sites. We make use of the assumption that each transaction T pre-declares its readset and writeset, denoted by $readset[T]$ and $writeset[T]$, respectively.

Our approach is to preprocess all the transactions in each global batch B in a view to detect read/write and write/write conflicts. Let $T_1, T_2, ...T_n$ be the transactions in B in timestamp order. We propose an algorithm that maintains two sets: $PRECEDE[T_i, X]$

and $INFORM[T_i, X]$. $PRECEDE[T_i, X]$ contains all transactions that access X and which should be executed before T_i. $INFORM[T_i, X]$ keeps the transactions that should wait for T_i to finish before using X.

We assume that a fictitious transaction T_0 with a timestamp smaller than any of the current transactions writes to all data items initially. For each data object X, preprocessing examines each transaction T_i that reads or writes X. If T_i writes X, then the algorithm looks for the latest preceding transaction T_j that writes X and puts it in $PRECEDE[T_i, X]$. All the transactions with timestamp between those of T_j and T_i that read X are also placed in $PRECEDE[T_i, X]$. For each transaction T_j in $PRECEDE[T_i, X]$, T_i is inserted into the set $INFORM[T_j, X]$, so that T_j can inform T_i about the completion of T_j when it finishes. If X is only read by T_i, then the algorithm looks for the closest preceding transaction T_j that writes X. T_j is then placed in $PRECEDE[T_i, X]$, and T_i is inserted into the set $INFORM[T_j, X]$.

After the preprocessing, we can start execution. A transaction manager (TM) carries out these operations locally at each site. At the beginning, we assume that T_0 has finished. When a transaction T_j finishes, TM examines each transaction T_i in $INFORM[T_j, X]$ and deletes T_j from $PRECEDE[T_i, X]$. A transaction T_i cannot access data object X unless $PRECEDE[T_i, X] = \emptyset$.

3.1.10 Decentralized Two-Phase Commit

A commit protocol is required to ensure that whenever a site decides to commit(abort) a public transaction, then every other site must also decide to commit(abort) the public transaction.

In TRS, we have two types of "commit/abort". The first type is the conventional commit/abort of individual transactions and the second type is the "commit/abort" of the global transaction batches. If the global batch aborts, then all transactions in the batch abort. Public transactions batches are committed in a two-phase consensus. A decentralized two-phase commit protocol for each global batch is illustrated in Figure 3.3.

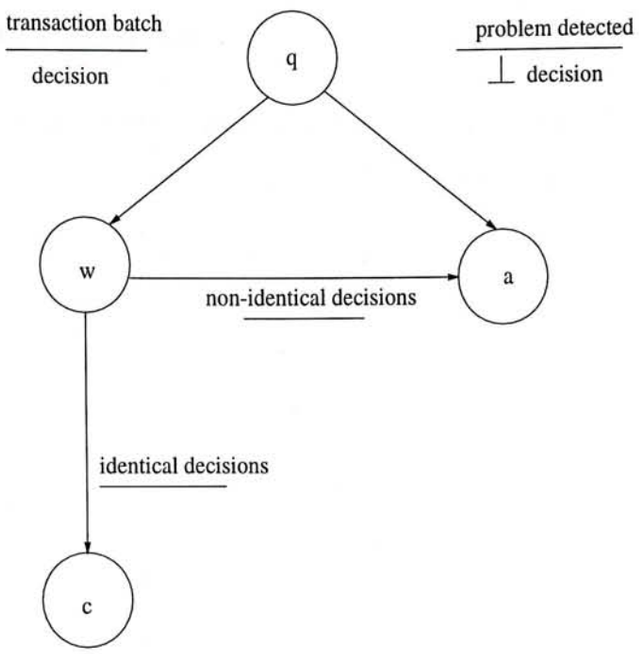


Figure 3.3: Two Phase Commit Protocol

[Phase 1]

Transactions are broadcast from all sites to all sites. After receiving a global batch, the site will execute transactions in the batch. Next, it will broadcast the tentative decision of the site on the commit/abort for each transaction in the batch. For example, if there are 3 ordered transactions T_1, T_2, T_3 in a batch, and the site S tentatively decide to commit T_1, T_2 but abort T_3, then its decision will be commit, commit, abort. We also define a special $\perp$ decision which is not identical with any other decision. If a site sends out $\perp$ decision, it can go to state (a) since the decisions are guaranteed not to be identical.

[Phase 2]

If a site receives identical decisions from all sites, it can commit the batch. That is, the transaction batch moves from state (w) to state (c). A site S decides to abort when it receives non-identical decisions from two or more sitee. The transaction batch moves from state (w) to state (a). Transaction batch aborts are triggered only by exceptional cases, such as problems at some site, site failures or partition failures.

3.1.11 Partition Failures

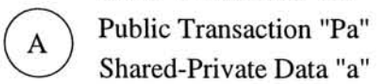
Under the failure conditions, a generalized version of the virtual parition protocol is used. Each transaction is executed under a view. The view of a site indicates what the site considers as the partition it currently belongs to, it is the "virtual partition" seen by the site. The major criteria of GVP is here:

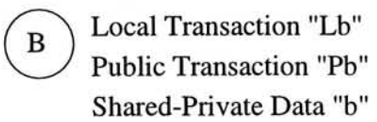
1. View-id: Each user transaction executes in a view. It is unique.

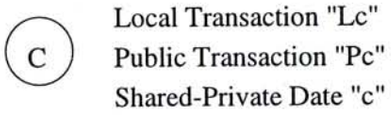
2. Global read quorums: For each data object X, a global read quorum set $RQ(X)$ is defined and it is a set of quorums of the replication sites of X.

3. View quorums: For each data object X, a view read quorum set $rq(X, V)$ and a view write quorum set $wq(X, V)$ are defined. Each view write quorum in $wq(X, V)$, if any, intersects each quorum in $RQ(X)$ and each view read quorum in $rq(X, V)$, if any. If $wq(X, V) \neq \emptyset$ ($rq(X, V) \neq \emptyset$), then X is said to be writable (readable) in V.

Step 1:

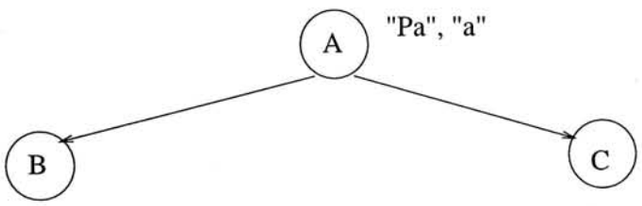






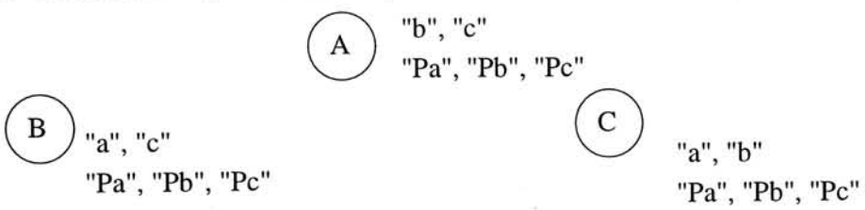
Local transactions are executed immediately.
Public transactions are collected.

Step 2: At the arrival of TRS period



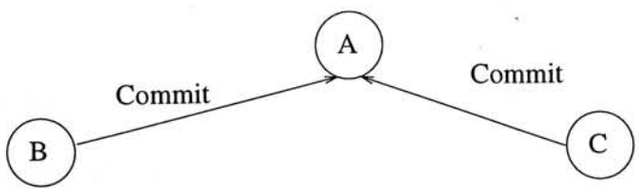
Site A broadcasts the batch of public transactions collected, "Pa" and the new version of shared-private data "a" to each sites. Similarly for site B and site C.

Step 3: After receiving all the batches,



each site will execute the batch of public transactions collected with the version of shared-private data received.

Step 4: After executing the batch of public transactions,



There is no need to keep the version of shared-private data and the "Commit" message will be sent back to the the owner site.
If the owner site receives all the "Commit" message from all the sites, the batch of public transactions will be committed. Else, they will be aborted.

Figure 3.4: Algorithm of TRS

Chapter 4

Simulation Model

4.1 Simulation Model

4.1.1 Model Design

A replicated Database Management System (DBMS) Simulation Model is developed for studying the performance of a variety of protocols including Transaction Replication Scheme (TRS), Majority Quorum Consensus(MQC) and Tree Quorum(TQ) protocol. Each site in the model has four components:

- a SOURCE, which generates transactions and also maintains transaction-level performance information for the site,
- a TRANSACTION MANAGER(TM), which models the execution behavior of transactions,
- a CONCURRENCY CONTROL MANAGER(CC Manager), which implements the details of a particular concurrency control protocol (i.e. TRS, MQC, TQ)
- a RESOURCE MANAGER, which models the CPU and I/O resources of the site,
- a NETWORK MANAGER, which models the behavior of the communication network.

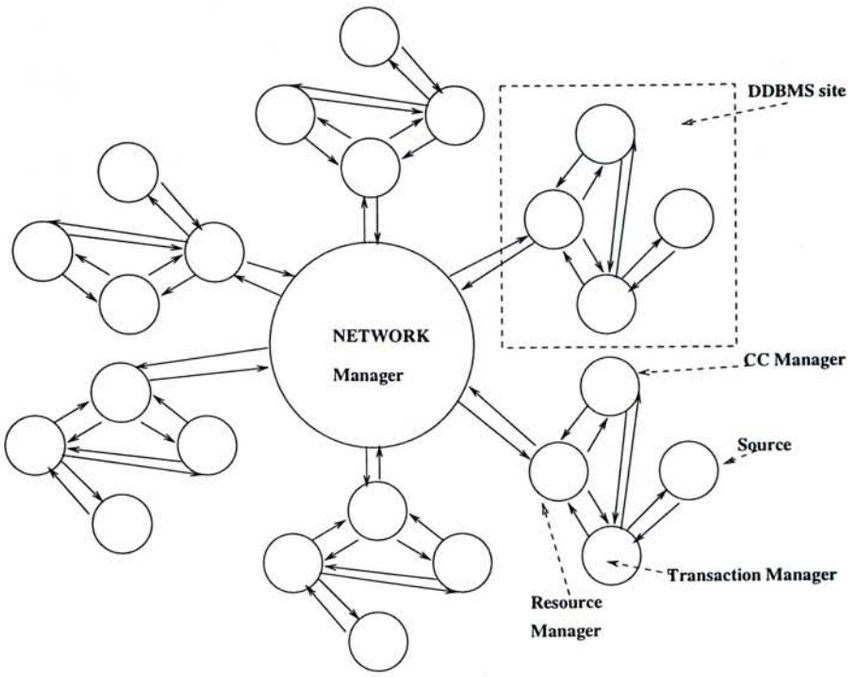


Figure 4.1: The diagram of Database Management System (DBMS)

These components are designed to support modularity, making it easy to replace the mechanism in any component (in particular, the Concurrency Control Manager) so as to implement different protocols for concurrency control (i.e. TRS, MQC and TQ) without affecting the others.

SOURCE

SOURCE is responsible to generate the workload for a site. It generates the transactions of different classes for the site and maintains the transaction-level performance information for the site. For example, it has to control the interarrival rate of transactions in order to simulate cases when the system is in a busy or idle state for different periods of time.

We shall try different ratios of local transaction and public transaction for TRS. Each transaction has a different number of operations and the ratio of read/write operations is varied. We can increase the conflicts among transaction operations by decreasing the size of the set of data accessed. We believe that TRS behaves well in conflicting cases since

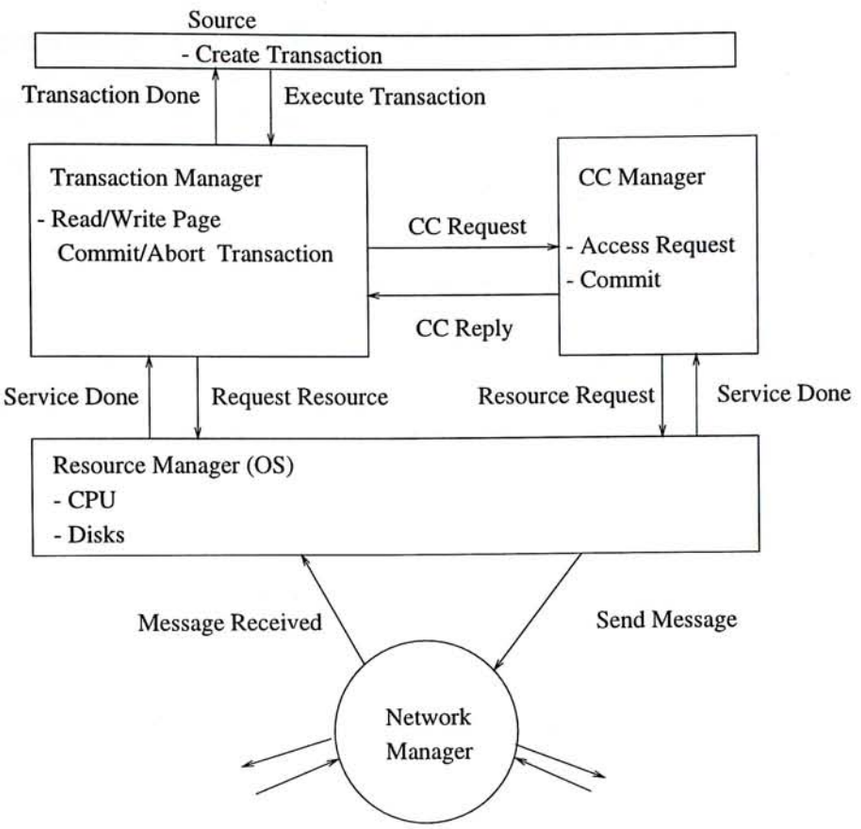


Figure 4.2: The diagram of a Closer Look at the DBMS model

locks on data are not required.

The transactions generated may access data object X by operation READ(X,y) and WRITE(X,v) for some variable x, y (see Section 3.1.4) and vary the number of data accesses so as to increase or reduce the data access conflicts.

As TRS performs better when computational cost is comparatively lower, we can vary the factors of the computational time and disk I/O time of different transactions so as to verify its performance.

TRANSACTION MANAGER

The Transaction Manager(TM) models the execution behavior of transactions and is responsible for accepting transactions from the SOURCE so as to model their execution.

For example, for read access, it involves a Concurrency Control (CC) request to get access permission, possibly followed by a disk I/O to read the page, followed by a period of

CPU usage for processing the operation. When the Concurrency Control request cannot be granted immediately, due to conflict, the transaction will wait until the request is granted by the CC Manager.

RESOURCE MANAGER

It is viewed as a model of a site's Operating System and Resource. It manages physical resources of site, including CPU and disks. It also provides message sending service (msg_cpu) because sending and receiving message involve the use of CPU resource. For each site, TM uses CPU and I/O service for read and write disk pages. The CC Manager also uses CPU service for processing CC request.

NETWORK MANAGER

It encapsulates a model of communication network. The main cost of sending messages in LAN is the CPU processing cost at sending and receiving sites. The bandwidth and propagation delay are the bottleneck in WAN [Gra88].

The time (delay) to send and receive a message is computed as follows:-

- Delay = Message Size/bandwidth + CPU+Transmit_Delay
  where Message Size is the size of message in bits, bandwidth is the speed(bits/second) of the communication media, and CPU is the processing time required to send and receive the message.

The following data is quoted from the paper [Gra88].

Parameter	LAN/WAN	Value
Transmit_Delay	LAN	0.00001s
	WAN	0.01s
Message_Size		100 bytes
Message header		32 bytes
Bandwidth	LAN	1×10^9 bytes/s
	WAN	1×10^3 bytes/s
Message CPU	LAN	0.000025s
	WAN	0.00012s
I/O time		0.035s
CPU time/ operation		0.005s

We can vary the above parameters to model different types of distributed system. A distributed system can be modeled as processes communicating via messages. This model can abstract three degree of distributed system: shared memory(shared memory multi-processes), local network(local network connecting several central nodes) and WAN(long haul network connecting several local network) in accordance with their differences in message transport cost and message transport reliability.

The time of message transmissions can differ by at least an order of magnitude at each degree of distribution. The reliability of message transmission can also differ by at least an order of magnitude at each degree of distribution. The message cost of a distributed algorithm is an important measure of its cost, especially for wide-area networks.

CONCURRENCY CONTROL MANAGER (CC Manager)

It is the only module that may change from protocol to protocol. It is responsible for handling the CC requests made by TM.

1. TRS

   After receiving both local and public transactions from the SOURCE, the SCHEDULER has to schedule the local transactions immediately and collect the public transactions in a batch within a TRS period. The local transactions will be scheduled to execute immediately. The batch of public transactions collected as well as

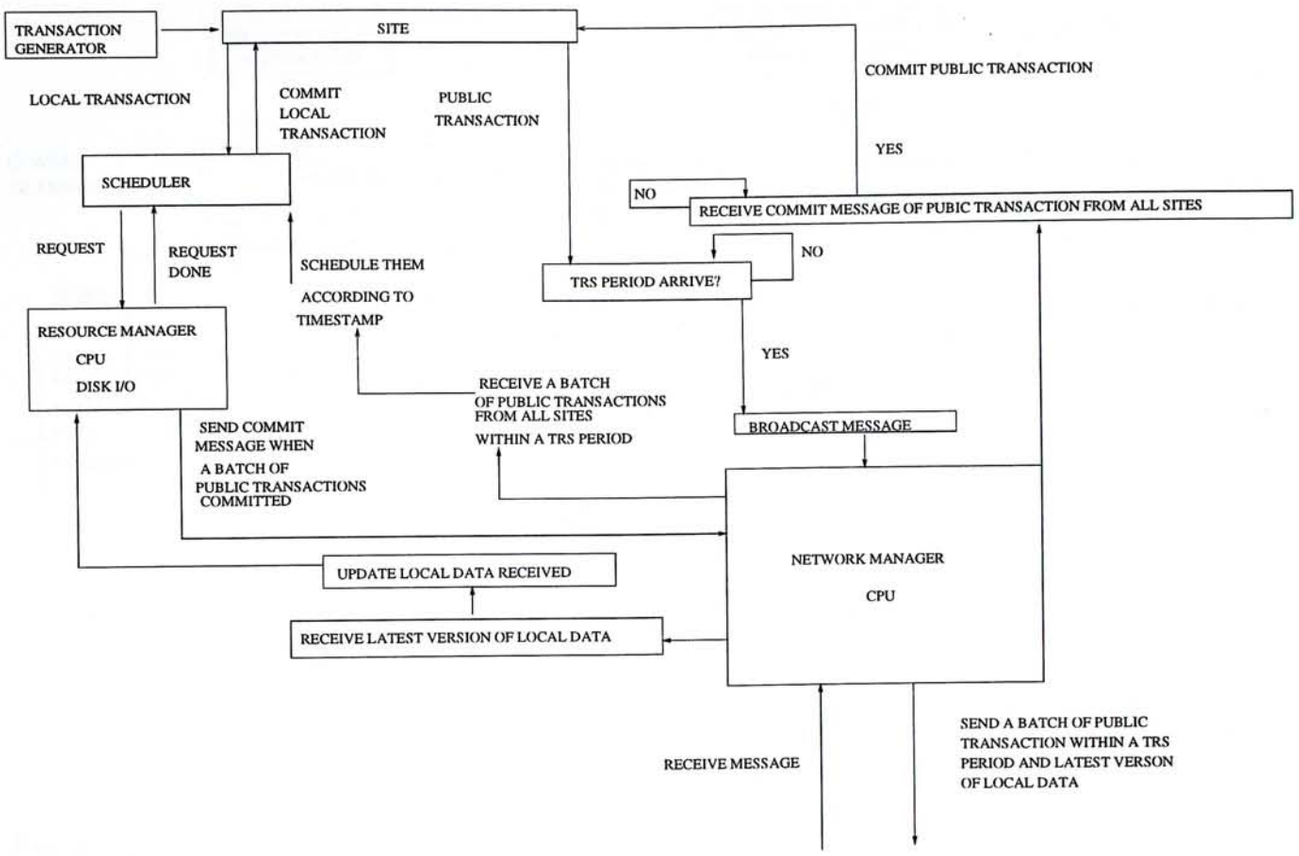


Figure 4.3: The diagram of a Closer Look at the TRS model

the latest version of shared-private data will be broadcast to other sites.

When the site receives the latest version of shared-private data from the other sites, it will immediately update the local data. After the site receives the batch of public transactions from all the sites within a TRS period, it will schedule these public transactions according to the Conservative Timestamp method and the SCHEDULER will schedule them to execute also.

The SCHEDULER has to request from the RESOURCE MANAGER for the use of CPU and Disk I/O in order to execute the transaction.

The local transactions can be committed or aborted by the site itself. For the public transactions, after executing the batch of public transactions. It will send the commit message to all the other sites. At the same time, the site will commit the batch of public transactions if and only if all the commit messages of that batch of public transactions are also received from all the sites and they are identical.

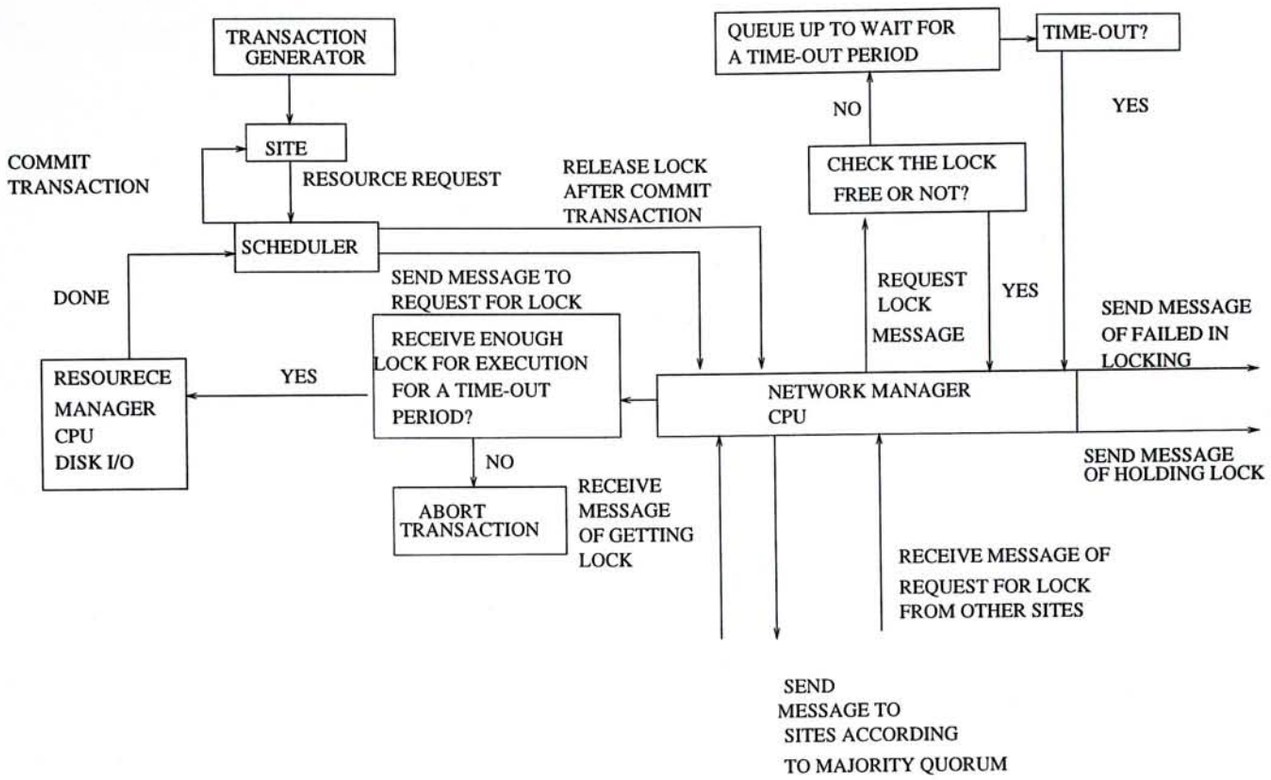


Figure 4.4: The diagram of a Closer Look at the Majority Quorum Consensus model

2. Majority Quorum Consensus

   After the site receives the transactions, the SCHEDULER has to send via the network manager the request message for the LOCK to a majority number of sites (selected randomly).

   When the site receives the request LOCK message, it will grant the LOCK to that sender and send a "Grant lock" message back if the LOCK is free. Otherwise, if the lock is unavailable, the request is queued up to wait until the lock is free for a time-out period. After time out, the request will not wait for the request lock and it will receive the failure message for granting the lock.

   If the site gets all the required lock messages, the transaction can be executed and it will request from the RESOURCE MANAGER for the use of CPU and disk I/O. After executing the transaction, the site has to release all the locks held and so it has to send "Release lock" messages to those sites it has locked. Then, a two-phase commit protocol [KS91] is carried out.

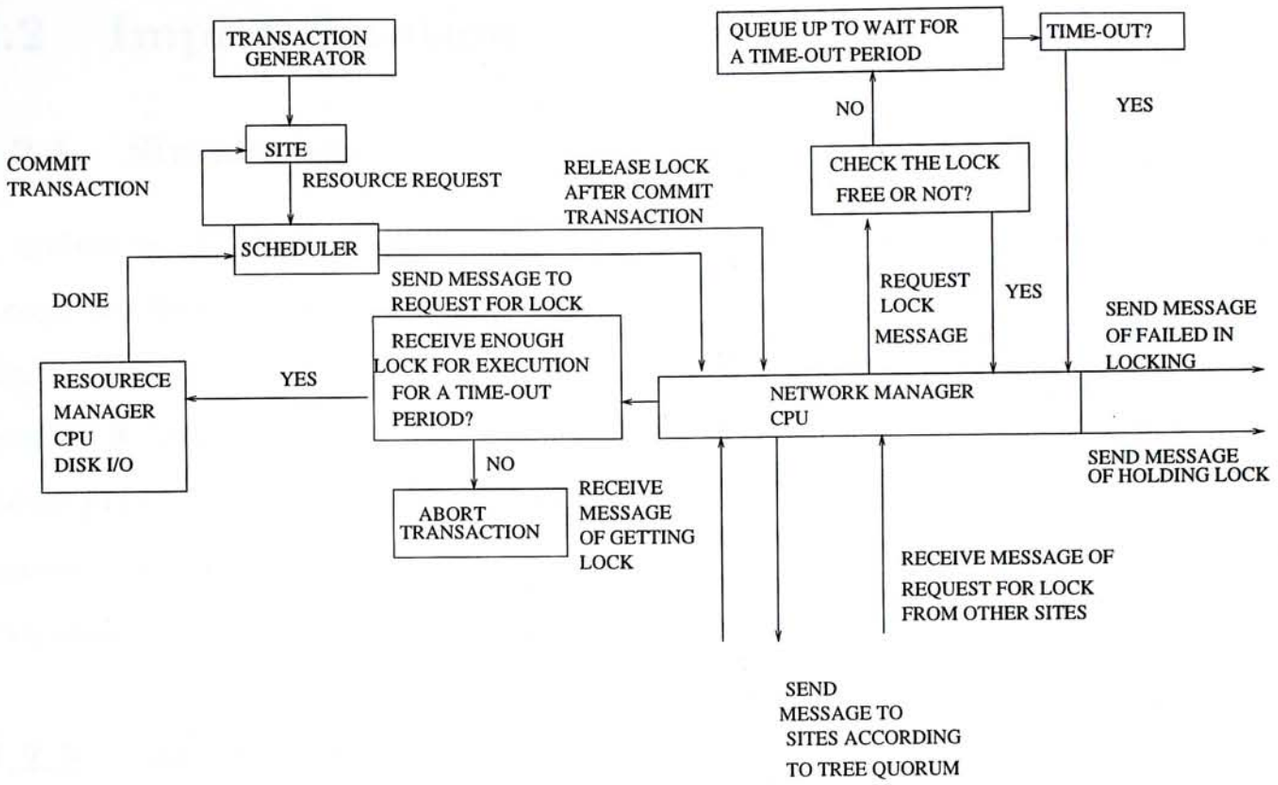


Figure 4.5: The diagram of a Closer Look at the Tree Quorum model

3. Tree Quorum

After the site receives the transactions, the SCHEDULER has to send the request message for the LOCK to the sites in accordance with the Tree Quorum[AA92]. When a site receives the request LOCK message, it will grant the LOCK to that sender and send the "Grant lock" message back if the LOCK is free. Otherwise, if the lock is unavailable, the request is queued up to wait until the lock is free for a time-out period. After time out, the requesting site will receive the failure message for granting the lock.

If the site gets all the required lock messages, the transaction can be executed and it requests from the RESOURCE MANAGER for the use of CPU and disk I/O. After executing the transaction, the site has to release all the locks held and so it has to send "Release lock" messages to the locked sites and then a two-phase commit protocol [KS91] is carried.

4.2 Implementation

4.2.1 Simulation

A system is modeled as a collection of resources together with a collection of processes competing for the use of these resource. A resource can be a CPU, Disk I/O or communication channel. A process can model the behavior of program execution in a computer system. A process-oriented simulation program declares the resources of the system; individual program segment and then mimics the behavior of processes as they visit first one resource and then another, until either the process leaves the system or the simulation terminates.

4.2.2 Simulation Language

The simulation language CSIM17[Sch] is employed and it is a library of routines, for use with C or C++[Fai] program, which allows us to create process-oriented, discrete-event simulation models. In the C++/CSIM library, all of the structures of CSIM are defined as classes, and most of CSIM verbs have become methods in these classes. The advantage of CSIM is that by using the inheritance features of C++, we can build new classes which are derived from the existing C++/CSIM base classes. Details of the implementation is given in Appendix A.

Chapter 5

Performance Results and Analysis

5.1 Simulation Results and Data Analysis

By using the simulation models for Transaction Replication Scheme, Majority Quorum Protocol and Tree Quorum, we can now compare their performance.

In the simulation, we have to consider various types of parameter settings of the system. First of all, we have to consider the variation of the message cost, since TRS is expected to perform better compared with the other Protocols when the message cost is high. The message cost is calculated by the factors of Message CPU (msg_cpu), Transmit Delay (transmit_delay) and Bandwidth (bandwidth), these are the input parameters for the system simulation.

Secondly, as TRS should perform better than traditional protocols when the number of operations per transaction is large, so the number of operations (maxoper) is also one of the parameter settings being studied.

We have to determine the length of the simulation time the simulation needs to arrive at a stable state, i.e. the results we obtained from the simulation jobs are steady.

Moreover, we have to consider the interarrival rate of transactions (lar_tm) to ensure that the system is not overloading. That is, about 90% of transactions generated can be completed within the simulation time.

The ratio of read/write operations per transaction (ratiorw) has to be studied as it will determine the read and write quorum of Tree Quorum Protocols.

The number of sites (numsite) within the system is also varied. We have to set up different number of sites to get different performance results.

The total number of data (numdata) and the frequently accessed data(accd) can affect the data conflict of the simulation.

In addition, the Disk I/O time (diskio) and the Computational Cost (op_cpu) are the time cost of executing a transaction operation.

The following are the parameters for the system:-

Parameter	Meaning
sim_time	Simulation Time of the System Model
numsite	number of site
numdata	number of data
accdata	number of data access
lar_tm	interarrival time of transaction
maxoper	maximum number of operation
ratiorw	ratio on the number of read write operation per transaction
msg_cpu	cpu time for processing receive or send message
transmit_delay	delay time for message transmission
bandwidth	speed of transferring number of bytes per second
diskio	disk io time
op_cpu	computational cost for a transaction

Table 5.1: TRS, Tree Quorum, Majority Quorum Consensus Models' Common Parameters

The Table 5.1 shows the common Parameters setting of TRS, Majority Quorum and Tree Quorum. However, for the Tree Quorum and Majority Quorum Model, we have to consider one additional parameter, time out period, i.e. the maximum period that a site will wait for a lock it requests. After timeout, if the site still cannot acquire the lock, it will assume that deadlock occurs and it will abort its transaction.

Parameter	Meaning
timingout	Maximum time a site will wait for a lock, when time is out,
	the corresponding transaction will abort

Table 5.2: Additional parameters for Tree Quorum and Majority Quorum Models

For TRS, there are also some other parameters to be considered. They are the TRS period, i.e. the fixed interval of time at which each site will broadcast a batch of public transactions and the latest version of shared-private data. Moreover, we also have to consider the number of public and local data, unlike the traditional protocols which just treat all the data as public type. The ratio of local transaction to public transaction is also one of the TRS parameters.

Parameter	Meaning
TRS	TRS period for broadcasting a batch of public transaction and latest version of shared-private data
numpublic	number of public data for all replicated sites
numlocal	number of local data each site
ratio	ratio of local transaction to public transaction

Table 5.3: Additional parameters for TRS Models

After formulating the simulation system, we have to consider the performance metrics of the simulation system for comparison and analysis. The measure of the performance of the system mainly bases on the response time of the transactions. In addition, the throughput of the system is monitored so as to find out the breakdown point of the system for TRS, Majority Quorum and Tree Quorum protocols. Moreover, the commit rate of the transactions against data conflict, site failure and partition failure is measured.

The following parameters would be measured from the simulation system:-

Performance Index	Meaning
throughput	Number of transactions completed per second
response time	average response time of transactions
Commit rate	number of transactions committed

Table 5.4: TRS, Tree Quorum and Majority Quorum Consensus Common Models' Metrics

For TRS, the overall response time of both public and local transactions, the response time of public transactions and the response time of local transactions are measured.

Performance Index	Meaning
overall response time	average response time of both local and public transactions
public response time	average response time of public transactions
local response time	average response time of local transactions

Table 5.5: Additional TRS Model Performance Metrics

A number of basic parameter settings are as follows:-

Performance Index	Value
Number of Operations per transaction	20
Disk I/O	0.007s
Read Write Ratio on Operation	6:4
Simulation Time	5000s
Interarrival Time of Transaction	20s
Ratio of Local to Public Transaction	1:1
Number of local data each site	100
Number of public data each site	1000
Transmit Delay	0.3s
Message Size	100 bytes
Message Header Size	32 bytes
Bandwidth	1×10^6 bytes/second
Message CPU time	0.000025s
Number of Sites	4, 13
Cache Hit	80%
CPU time for operation	0
Ratio of Local to Public Transaction	1:1

Table 5.6: Basic Model Metrics

The performance is studied by the following experiments. They include:

1. variation of TRS period,
2. variation of clock synchronization accuracy,
3. variation of the ratio of Local to Public Transactions,
4. variation of message transmit delay,
5. variation of interarrival rate of transactions,
6. variation of transaction computational cost,
7. variation of disk I/O time,
8. variation of cache hit ratio,
9. variation of number of data access,
10. variation of ratio of read/write operation,
11. variation of one particular failed site,
12. variation of number of sites available,

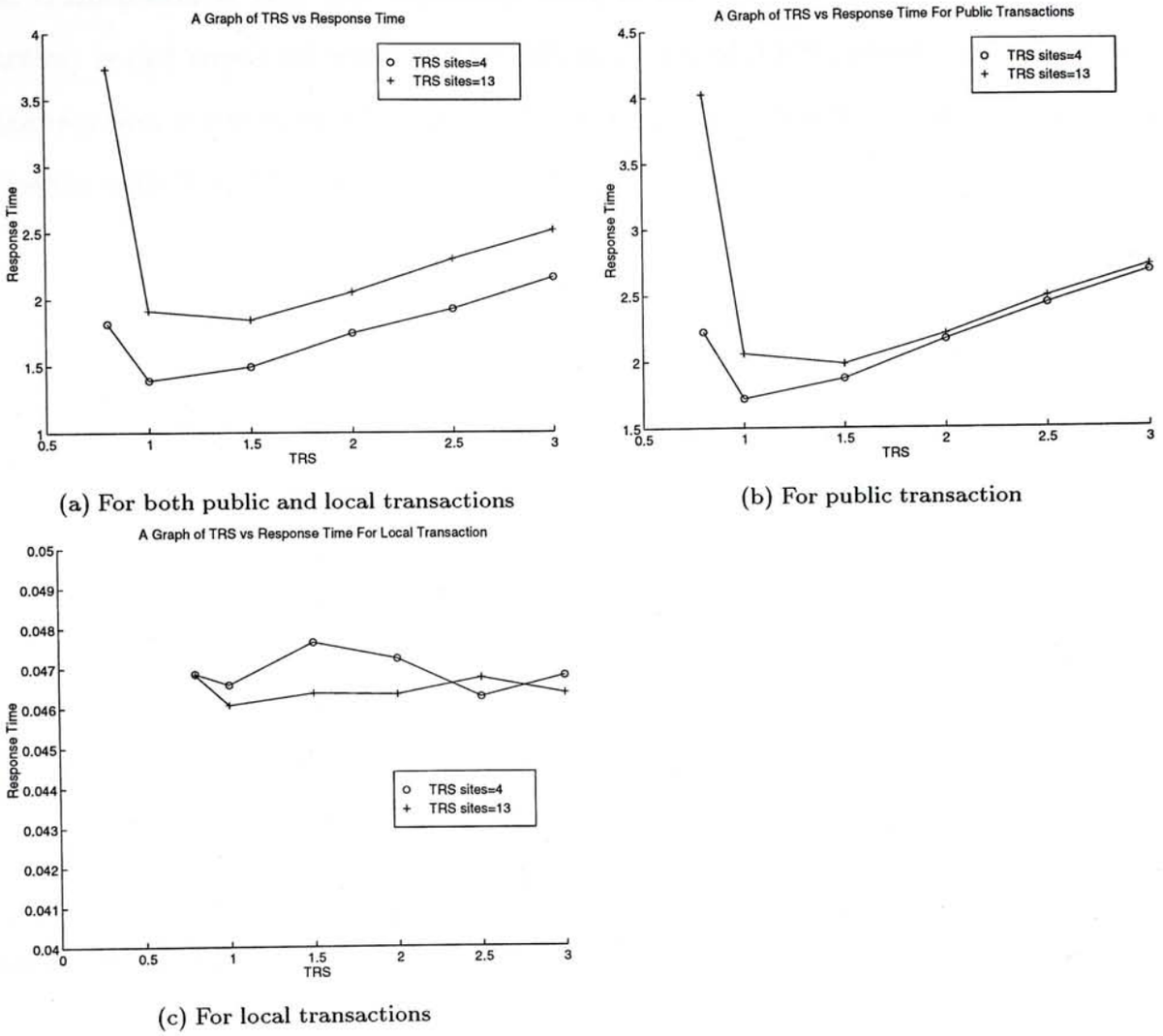
5.1.1 Experiment 1 : Variation of TRS Period

In this experiment, the simulation is done with 4 sites and 13 sites respectively. The length of simulation time is 5000s. Each transaction has 20 operations and each transaction has the ratio of read/write operation 6:4. The computational cost is 0s and the disk I/O time is 0.007s with the cache hit ratio of 80%. The interarrival time of transaction is at each site 20s. The ratio of local to public transaction is 1:1. The number of public data is 1000 and the local data for each site is 100. The message size is 100 bytes and the message header size is 32 bytes, the bandwidth is
1×10^6 bytes/second, the message transmit delay is 0.3s and the message CPU time is 0.000025s.

For different values of the TRS period, we find out the behavior of Transaction Replication Scheme by measuring its response time and the number of versions of shared-private data stored. The range of TRS period variation is from 0.8s to 3s.

Performance Index	Value
TRS	0.8, 1.0, 1.5, 2, 2.5, 3(second)

Table 5.7: Experiment 1 Varying TRS period



(a) For both public and local transactions

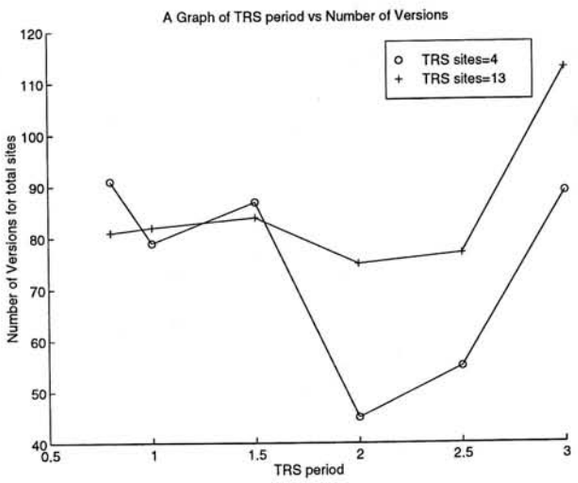
(b) For public transaction

(c) For local transactions

Figure 5.1: Experiment 1 The diagram of TRS period Vs Response time

Figure 5.1(a) shows that when TRS period is decreased from 3 second to about 1 second, the response time decreases correspondingly. However, after the minimum point (about 1 - 1.2 second), the response time will increase. This is because when the TRS period is too long, the public transactions have to wait for a longer time (greater TRS period) to be executed. Therefore, when the TRS period increases, the response time increases accordingly. On the other hand, if the TRS period is too short, it implies the public transactions have to be frequently broadcast. Moreover, the system may not always complete the execution of the batch of public transactions in a period. That is why the response time of A the transactions increases. In figure 5.1(c), the response time

of local transaction is very low compared with figure 5.1 (a). The response time of local transaction is not much affected by the different size of TRS period. This is because the local transaction are executed immediately under node autonomy and so there is no close relationship with the TRS period.



(a) TRS Vs number of version for all sites

Figure 5.2: Experiment 1 The diagram of TRS period Vs Maximum number of versions of shared-private data

From Figure 5.2(a), the number of versions of shared-private data we needed is about 40 to 110. This is the maximum number of old versions of all the shared-private data a site has to keep during the simulation time. The number of data that a site has to store is small. Therefore the data storage for shared-private data is not a problem in TRS.

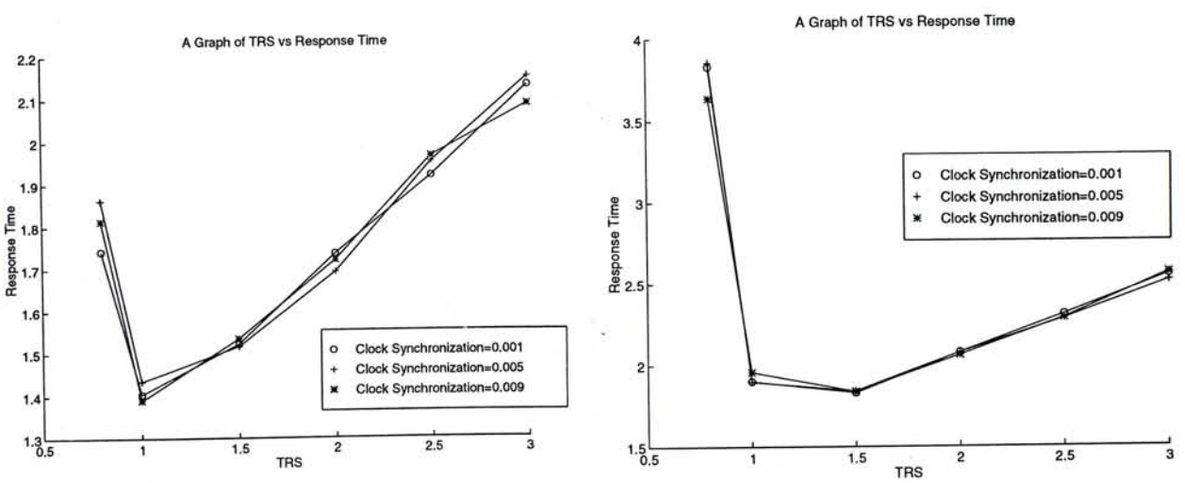
Conclusion

The choice of TRS period is rather significant. If the TRS period is too large, the response time becomes great. If the TRS period is too small, the response time becomes la rger too. In the following experiments, we will use a TRS period of 2.0 seconds.

5.1.2 Experiment 2 : Variation of Clock Synchronization

From [Mil88], the network time protocol NTP synchronizes clocks of nodes on geographically distributed networks. It does this at low cost and provides clocks that are synchronized to within a few milliseconds of one another. NTP is running on the internet today and is used to synchronize clocks of nodes throughput the United States, Canada, and various places in Europe. Since clock synchronization belongs to the lower levels of the system, and it is outside the scope of this thesis, we do not consider the overhead of clock synchronization.

As TRS has to broadcast messages periodically, that is according to the length o f TRS period. We do an experiment about the variation of clock synchronization to find out whether it w ill deteriorate the performance of TRS. We vary the clock synchronization from 0 to 9 milliseconds. As we can see from f igure 5.3(a) & (b), the accuracy of clock synchronization do not affect much the response time. This is because the optimal TRS period is about 1 second. This shows that the accuracy of cl ock synchronization is not a problem in TRS broadcasting.



(a) 4 sites

(b) 13 sites

Figure 5.3: Experiment 2 The diagram of TRS period Vs Response time by varying clock synchronization accuracy

Performance Index	Value
Clock Synchronization	0, 1, 5, 9 (milliseconds)

Table 5.8: Experiment 2 Varying Clock Synchronization

Conclusion

The accuracy of clock synchronization does not have any effect on the performance of TRS.

5.1.3 Experiment 3 : Variation of Ratio of Local to Public Transaction

In this experiment, we vary the ratio of local transactions to find out the impact on the performance by the introduction of the new transaction type, i.e. local transaction (unlike the traditional protocols). In this experiment, the interarrival time of transactions for each site is 20 second.

Performance Index	Value
TRS	2
Local to Public transaction Ratio	2:1,6:1,10:1,14:1,18:1

Table 5.9: Experiment 3 TRS Model: Vary Ratio of Local to Public Transaction

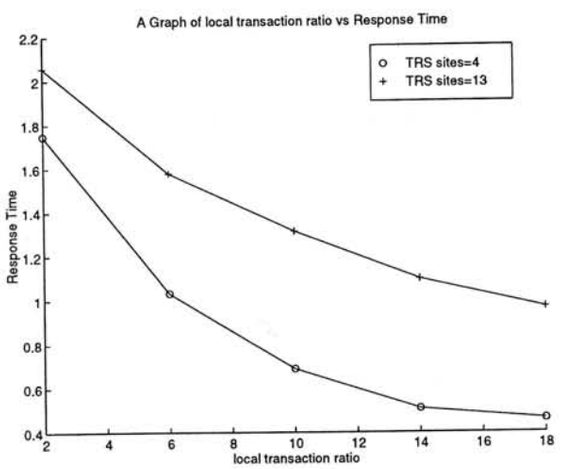
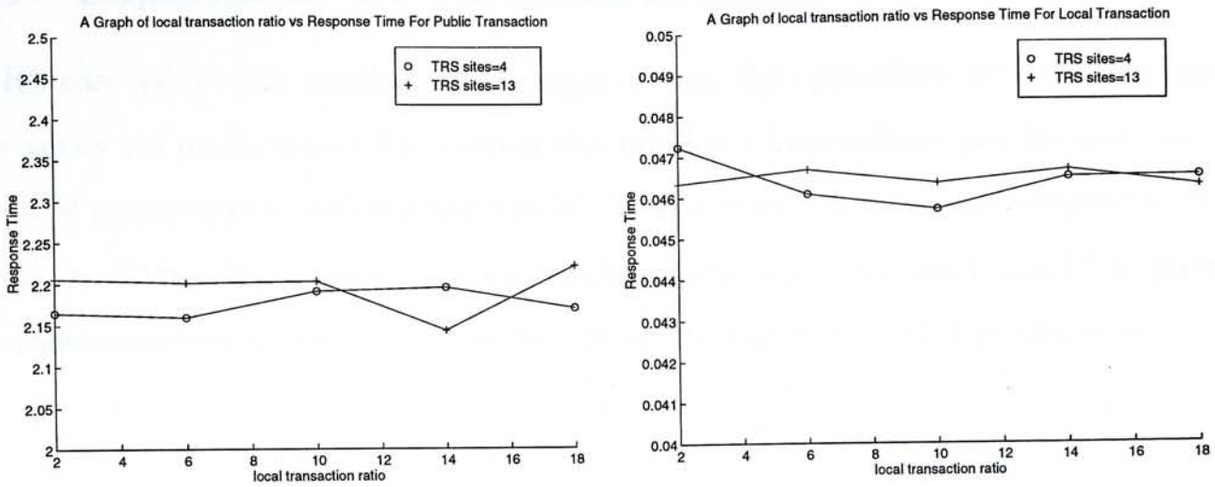


Figure 5.4: (a) For both Public and Local Transactions

From Figure 5.4a, we observe that when the ratio of local transaction increases, the overall response time of both public and local transactions decreases. This is because the local transaction can be executed and committed immediately, which is unlike public transactions which have to wait for the arrival of TRS period to execute.



(a) For Public Transactions

(b) For Local Transactions

Figure 5.5: (b) Diagram of Ratio of Local to Public Transaction Vs Response Time

Conclusion

TRS performs better when there is a higher ratio of local transactions.

5.1.4 Experiment 4 : Variation of Number of Operations

As TRS can reduce the number of messages among the operations within a transaction, so we verify its performance by varying the number of operations per transaction. The number of operations is varied from 5 to 25. In this experiment, as we compare with Tree Quorum and Majority Quorum protocols, the numbers of sites are 4 and 13 respectively since these numbers of sites can form the complete logical tree of Tree Quorum.

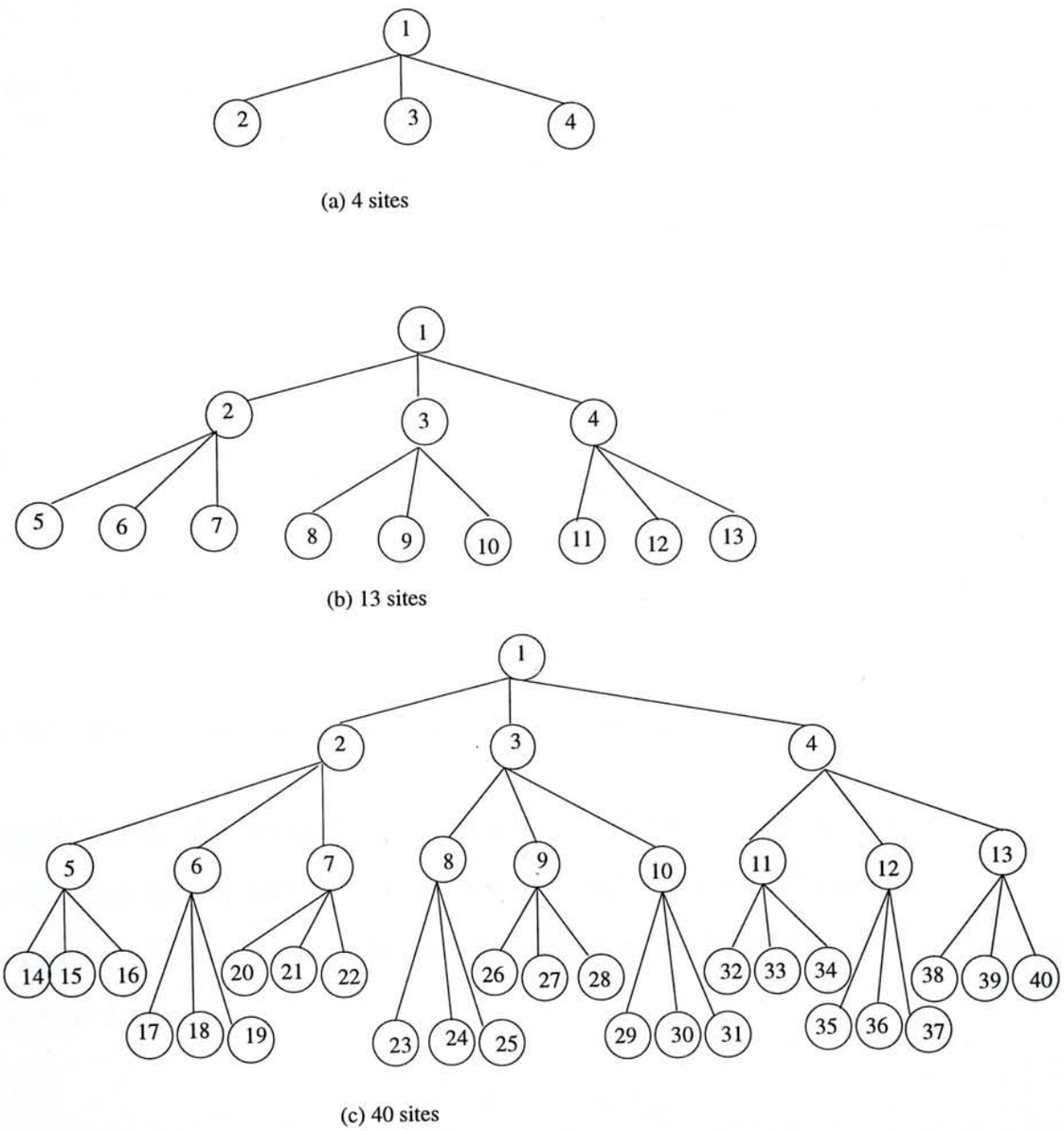
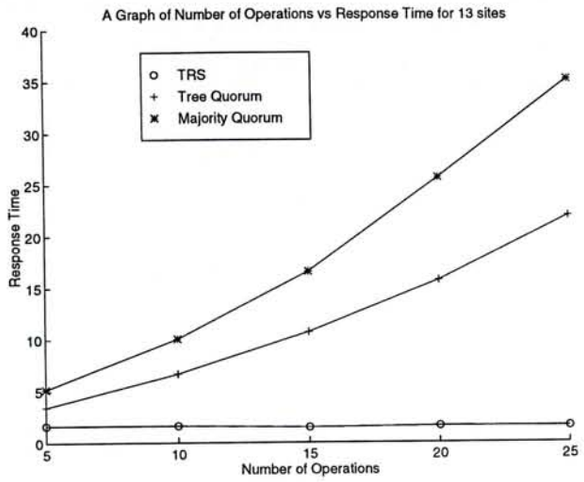
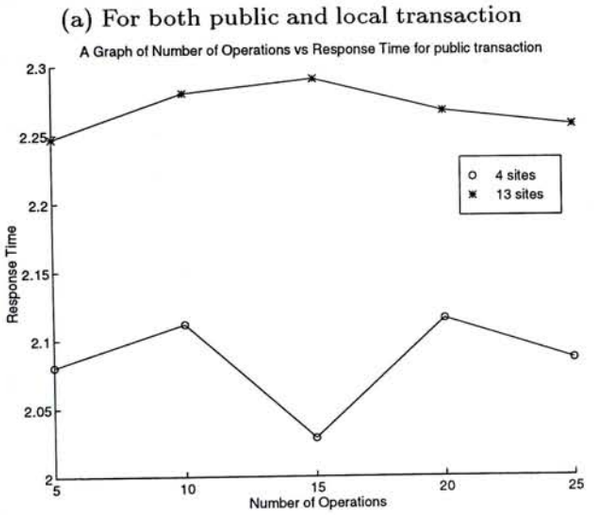


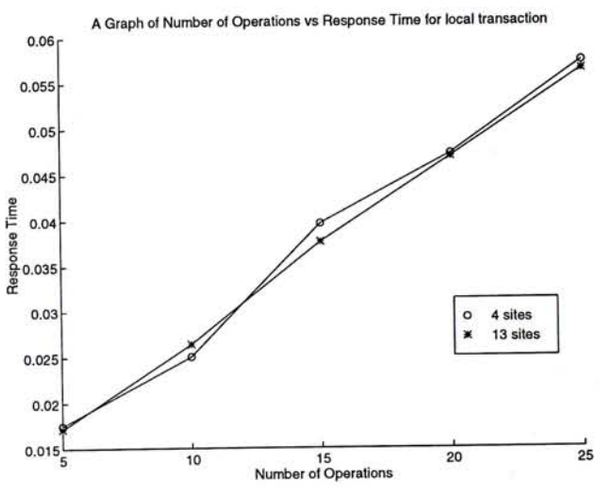
Figure 5.6: Logical Tree Structure of Tree Quorum



(a) For both public and local transaction



(b) for public transaction only



(c) for local transaction only

Figure 5.7: Experiment 4 The diagram of Number of Operations Vs Response time

In Figure 5.7(a), the overall response time of both public and local transaction increase with the increase in the number of operations within a transaction. This is the same for local transaction (Figure 5.7c). In this experiment, we have the ratio of local to public transaction of 1:1.

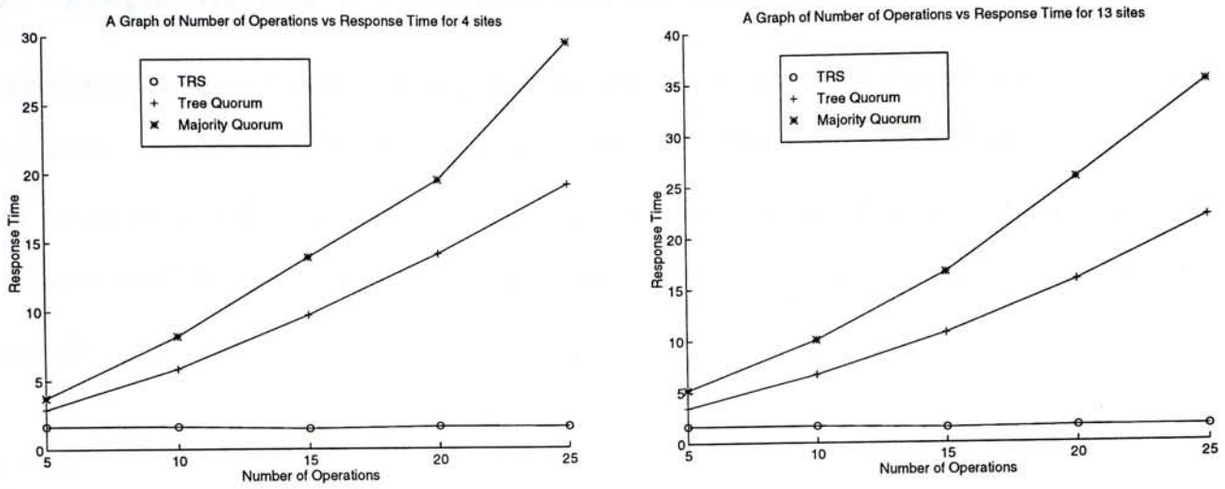
Performance Index	Value
Number of Sites	4, 13
Ratio of Local to Public transaction	1:1
TRS	2
Number of Operations	5,10,15,20,25

Table 5.10: Experiment 4 Varying Number of Operations in a transaction

As we want to compare the performance of TRS with Tree Quorum and Majority Quorum, to be fair for the traditional protocols, we generate only the public transactions in TRS. This is the same for the Tree Quorum and Majority Quorum, they also generate transactions which access the same set of public data. Figure 5.7(a) shows the comparison.

While the number of operations per transaction increases, the response time for TRS, Tree Quorum and Majority Quorum increases. The overall response time of TRS is the best compared with Tree Quorum and Majority Quorum.

The high rate of increase in the response time for Majority Quorum and Tree Quorum is due to the communication cost among the number of operations within a transaction for Quorum Consensus Protocols, this communication cost can be eliminated for TRS.



(a) 4 sites

(b) 13 sites

Figure 5.8: Experiment 4 The diagram of Number of Operations Vs Response time

Conclusion

TRS performs better than Tree Quorum and Majority Quorum when the number of operations of a transaction is high.

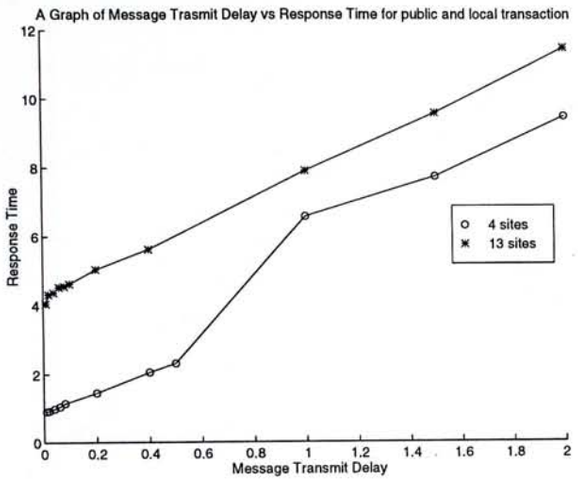
5.1.5 Experiment 5 : Variation of Message Transmit Delay

We vary the message transmit delay time to verify whether TRS performs better for high message cost. In Figure 5.9a, b and c, we have the TRS simulation experiments with the ratio of local to public transaction 1:1. The response time of total transactions, public transactions and local transactions are measured respectively. The range of message transmit delay is from 0.01s to 2s.

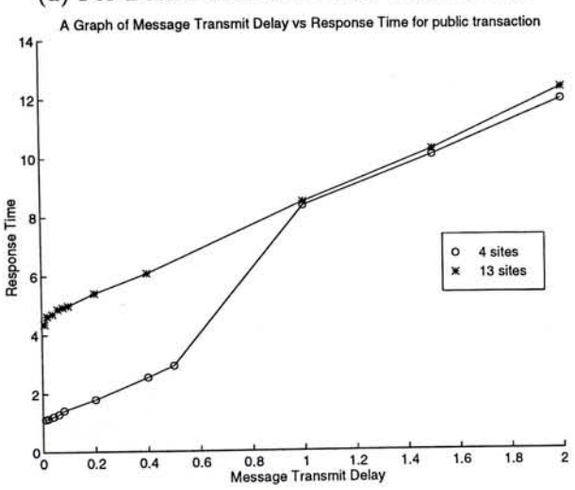
Performance Index	Value
Number of Operations	20
Simulation Time	5000s
Interarrival Time of Transaction	80s
Time Out	100s
TRS period	1s (message transmit delay $<$ 0.5s)
	2s (message transmit delay=0.5s),
	10s (message transmit delay $>$ 0.5s)
Message Transmit Delay	0.01, 0.02, 0.04, 0.06, 0.08, 0.2, 0.4, 0.5, 1, 1.5, 2 (second)

Table 5.11: Experiment 5 Varying Message Transmit Delay

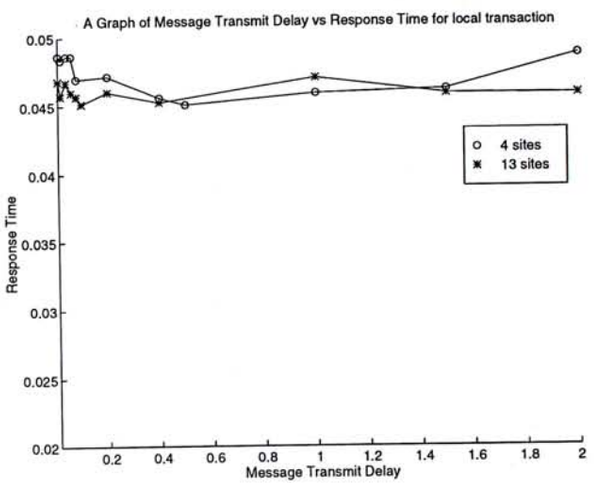
The TRS period has to be increased so as to ensure that the batch of public transactions can be received within one TRS period. For the public transactions (Figure 5.9b), the increase in the message transmit delay causes the increase in the response time of public transactions. This is because the batch of public transactions are broadcast at the end of each TRS period. Thus, if the time for broadcasting message is longer, the response time is also longer. However, for local transactions (Figure 5.9c), as the local transactions are executed immediately under node autonomy, it is not affected by the message time. As we want to compare the performance of TRS with Tree Quorum and Majority Quorum protocols, TRS is executed with public transactions only which access the same set of public data as Quorum Protocols. When the message cost increases, the response time increases correspondingly. The response time of TRS is much lower than those of Majority Quorum and Tree Quorum.



(a) For Both Public and Local Transactions



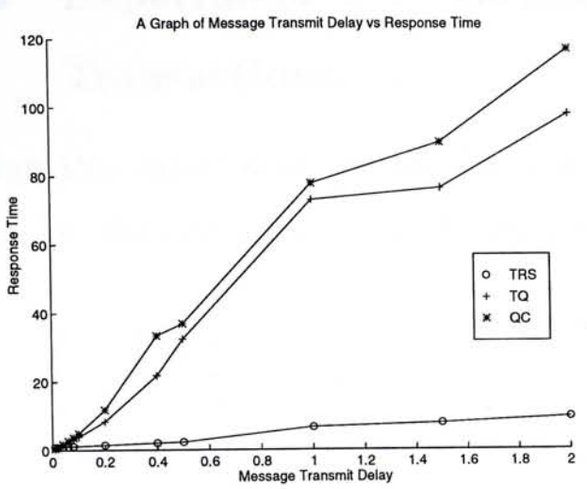
(b) Public Transaction



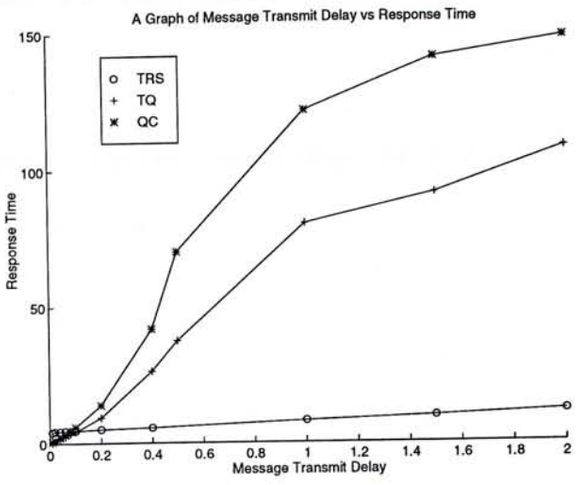
(c) Local Transaction

Figure 5.9: Experiment 5 The diagram of Message Time Vs Response Time for TRS

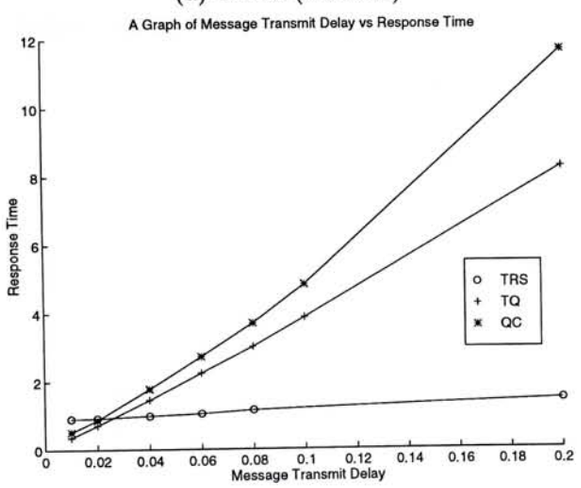
Figures 5.10 a & b shows that rate of increase in response time in TRS is much lower than those of Majority Quorum and Tree Quorum protocols.



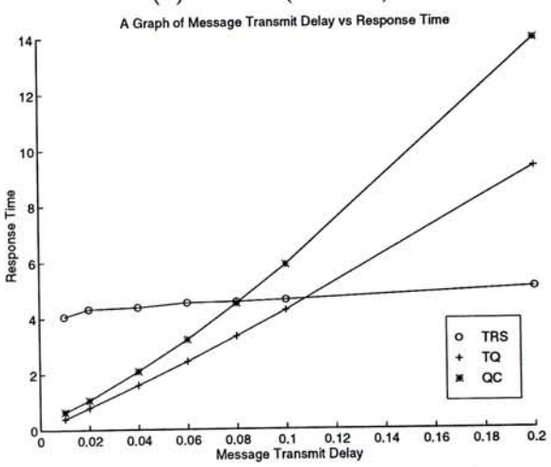
(a) 4 sites (0.01s-2s)



(b) 13 sites (0.01s-2s)



(a) 4 sites (0.01s-0.2s)



(b) 13 sites (0.01s-0.2s)

Figure 5.10: Experiment 5 The diagram of Message Time Vs Response Time for TRS, Tree Quorum and Majority Quorum

Conclusion

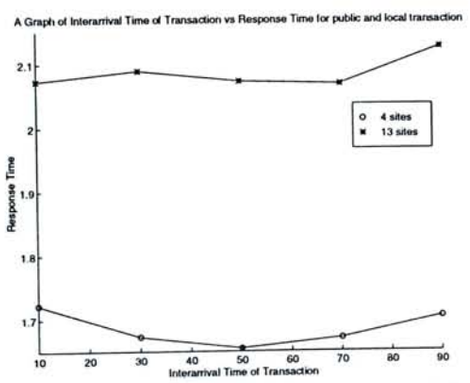
TRS performs the best among the tested protocols when the message transmit delay is reasonably large.

5.1.6 Experiment 6 : Variation of the Interarrival Time of Transactions

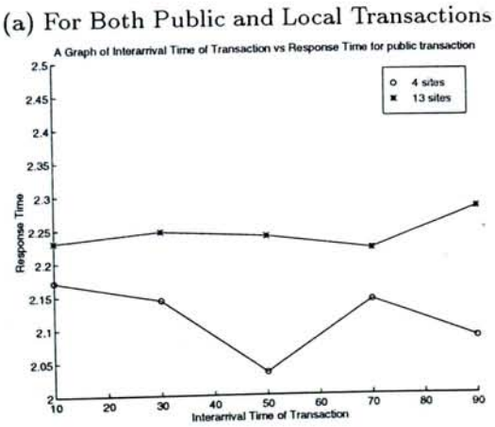
Through this experiment, we can find out which system performs the best in a busy system, i.e. the interarrival time of transactions is small.

Performance Index	Value
TRS period	2s
Interarrival Rate of Transaction	10,30,50,70,90 (second)
Number of Sites	4, 13

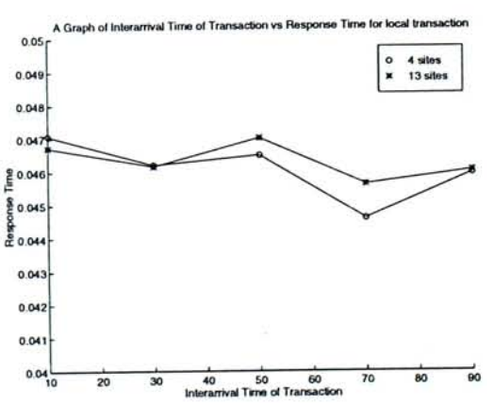
Table 5.12: Experiment 6 TRS Model: Vary Interarrival time of transaction



(a) For Both Public and Local Transactions



(b) Public Transaction

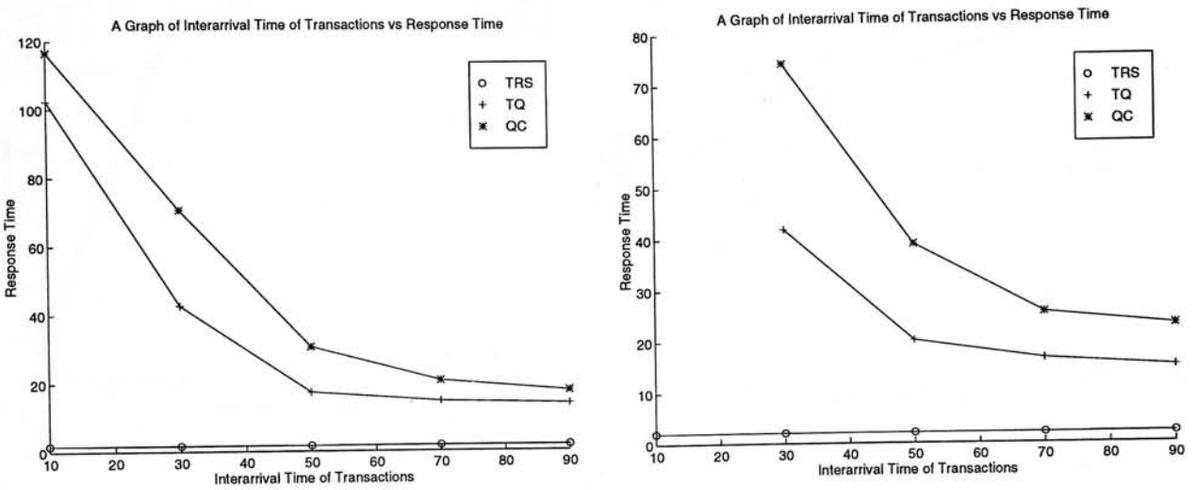


(c) Local Transaction

Figure 5.11: Experiment 6: The diagram of Interarrival time of Transaction Vs Response time for TRS

We set up the experiment with 4 and 13 sites and with the interarrival time of transaction from 10 second to 90 second.

The response time decreases with the increase in the interarrival time of transactions.

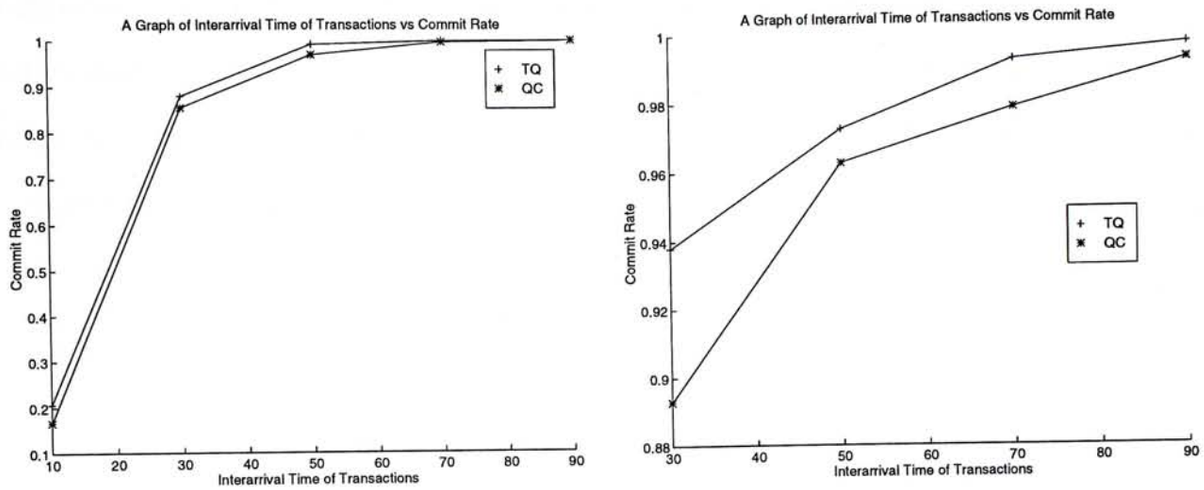


(a) 4 sites

(b) 13 sites

Figure 5.12: Experiment 6: The diagram of Interarrival time of Transaction Vs Response time

For Majority and Tree Quorum protocols, the commit rate of transactions is decreasing, they are overloaded at about the 30s interarrival time point.



(a) 4 sites

(b) 13 sites

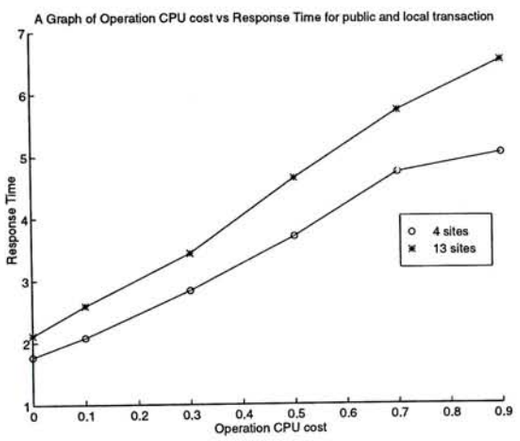
Figure 5.13: Experiment 6: The diagram of Interarrival rate of Transaction Vs Completion Rate

Conclusion

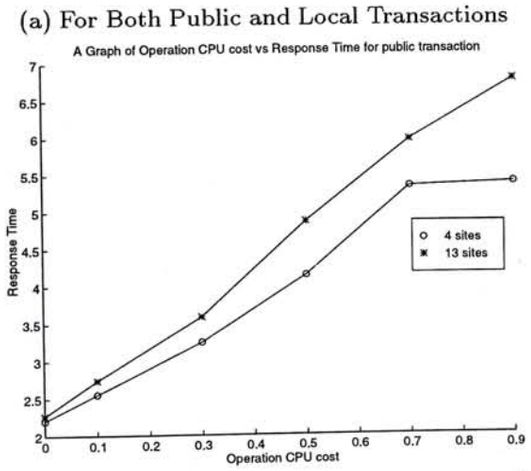
As the interarrival time of transactions decreases, TRS performs the best as it has the lowest response time compared with Majority and Tree Quorum Protocols.

5.1.7 Experiment 7 : Variation of Operation CPU cost

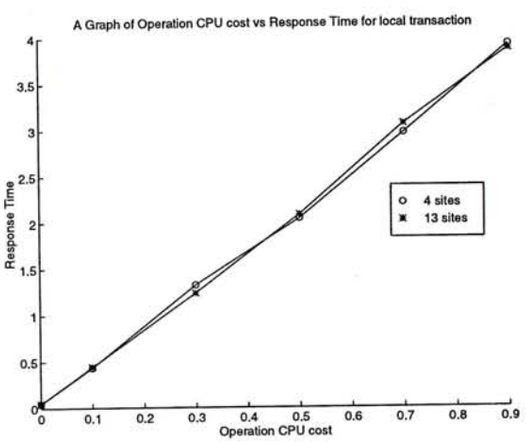
By varying the CPU cost, we can verify the impact of computational cost on TRS when the computational cost is an overhead. However, in business systems, it is rare that the computational cost is high. Instead, the computational cost is usually negligible. The computational cost we vary is the cpu time used for each transaction. The computational cost is varied from 0 second to 0.9 second.



(a) For Both Public and Local Transactions



(b) For Public Transaction



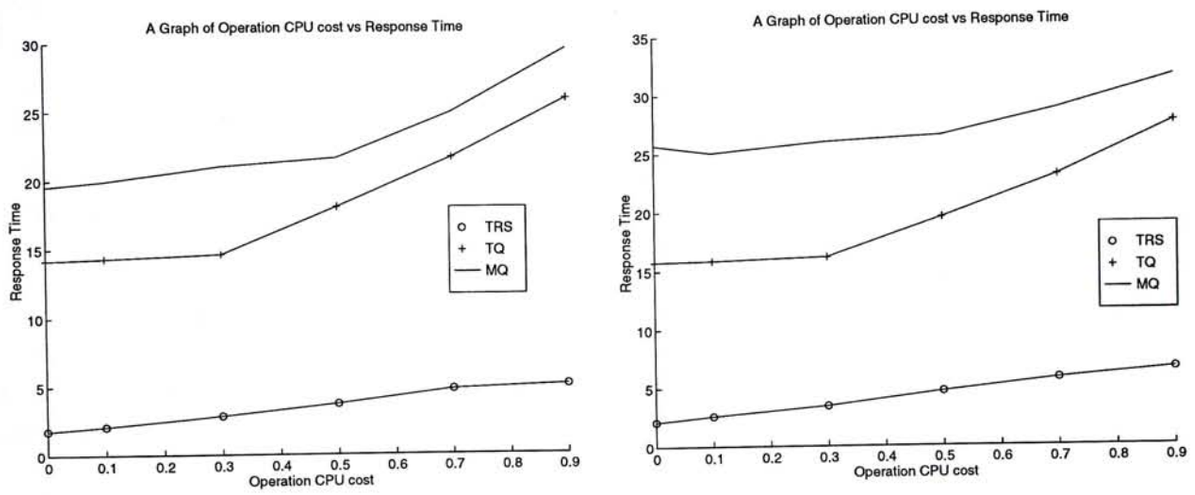
(c) For Local Transaction

Figure 5.14: Experiment 7: The diagram of Operation Computational cost Vs Response time

As the increase in computational cost, there is increase in the response time for both public and local transactions.

Performance Index	Value
TRS	2s
Simulation Time	5000s
Computational cost	0.0, 0.1, 0.3, 0.5, 0.7, 0.9(second)

Table 5.13: Experiment 7 TRS Model: Vary Computational cost

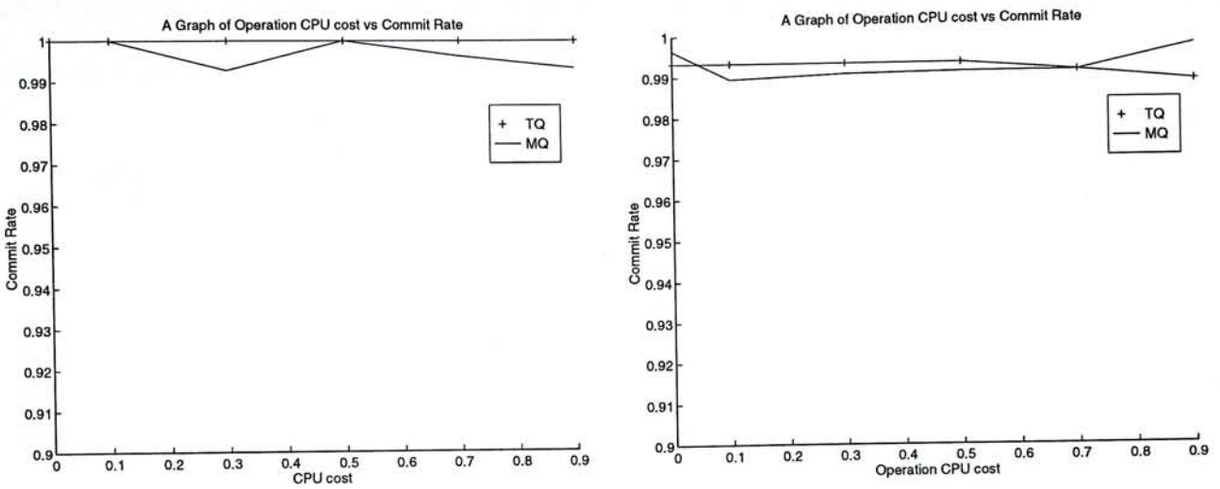


(a) 4 sites

(b) 13 sites

Figure 5.15: Experiment 7: The diagram of Operation Computational cost Vs Response time

The commit rate of transactions is not much affected according to the increase in computational cost.



(a) 4 sites

(b) 13 sites

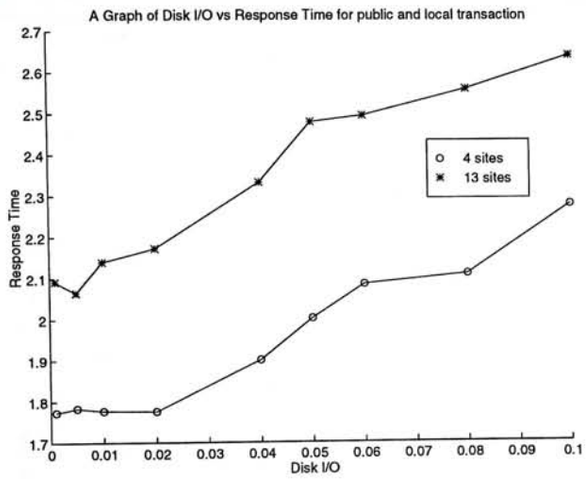
Figure 5.16: Experiment 7: The diagram of Computational cost Vs Completion Rate of Transactions

Conclusion

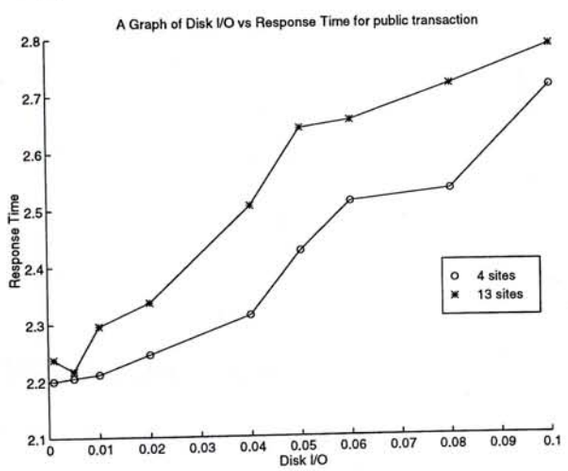
TRS performs the best compared with Tree and Majority Quorum protocols with the increase in computational cost.

5.1.8 Experiment 8 : Variation of Disk I/O time

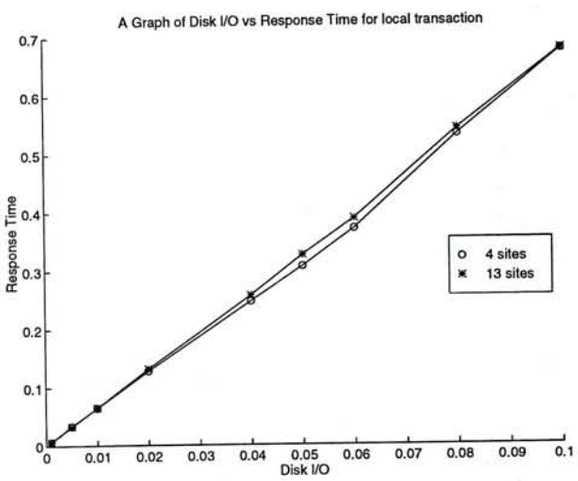
By varying the disk I/O time, we can verify the impact of disk I/O time on the performance of TRS, Quorum Consensus and Tree Quorum. The disk I/O time we vary is from 0.001 second to 0.1 second.



(a) For Both Public and Local Transactions



(b) Public Transaction



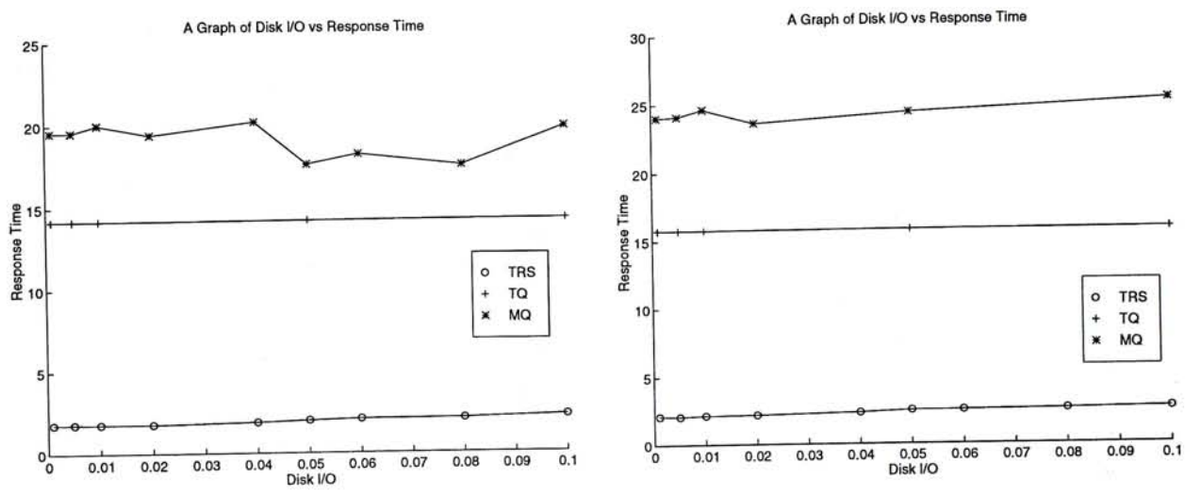
(c) Local Transaction

Figure 5.17: Experiment 8 The diagram of Disk I/O time Vs Response time

As the increase in disk I/O time, the response time for both public and local transactions also increases.

Performance Index	Value
Message Transmit Delay	0.3s
Interarrival Time	80s
Simulation Time	5000s
TRS	2s
Number of Operations	20
Disk I/O	0.001, 0.005, 0.01, 0.02, 0.04, 0.05, 0.06, 0.08, 0.1(second)

Table 5.14: Experiment 8 Varying Disk I/O time



(a) 4 sites

(b) 13 sites

Figure 5.18: Experiment 8 The diagram of Disk I/O time Vs Response time

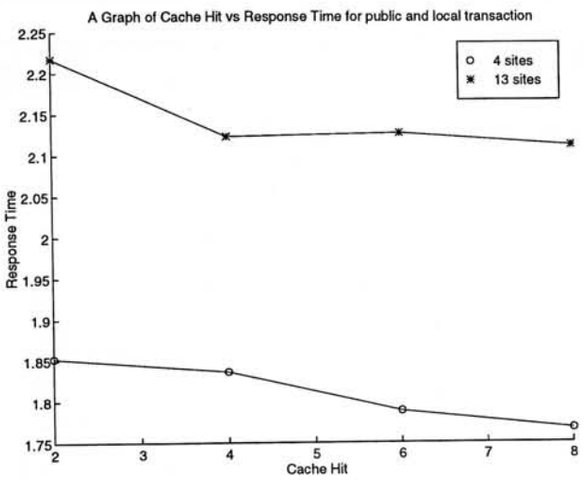
For the Majority Quorum, Tree Quorum protocols and TRS, the response time does not change with the increase of the disk I/O time.

Conclusion

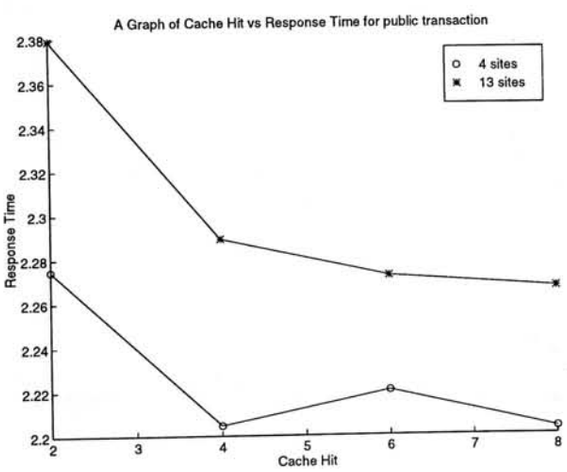
The performance of TRS are not affected by the increase of disk I/O time.

5.1.9 Experiment 9 : Variation of Cache Hit Ratio

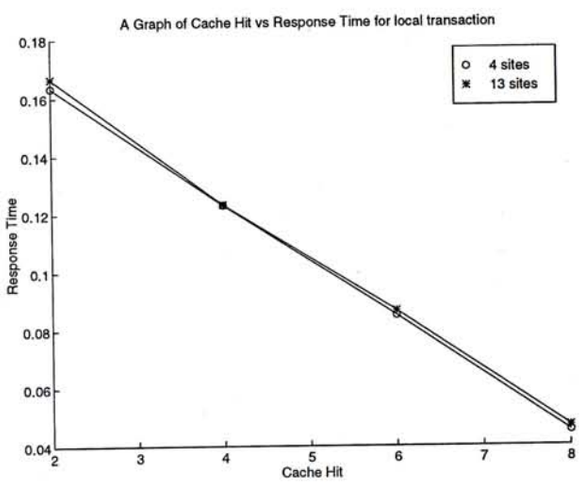
The percentage of cache hit affects the execution time of the transaction. The cache hit ratio we vary in this experiment is from 20% to 80%.



(a) For Both Public and Local Transaction



(b) Public Transaction



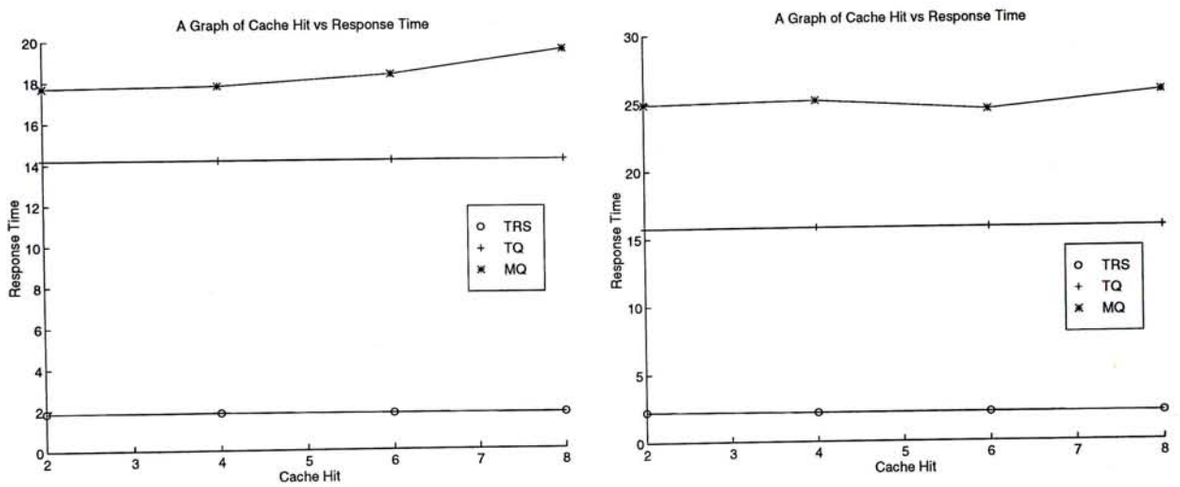
(c) Local Transaction

Figure 5.19: Experiment 9 The diagram of Cache Hit Vs Response time

As the cache hit ratio increases, the response time of both public and local transactions decrease.

Performance Index	Value
Transmit Delay	0.3s
Interarrival Time of transaction	80s
Simulation Time	5000s
Disk I/O	0.007s
Number of Operations	20
TRS	2s
Cache Hit	20%, 40%, 60%, 80%

Table 5.15: Experiment 9 Varying Cache Hit



(a) 4 sites

(b) 13 sites

Figure 5.20: Experiment 9 The diagram of Cache Hit Vs Response time

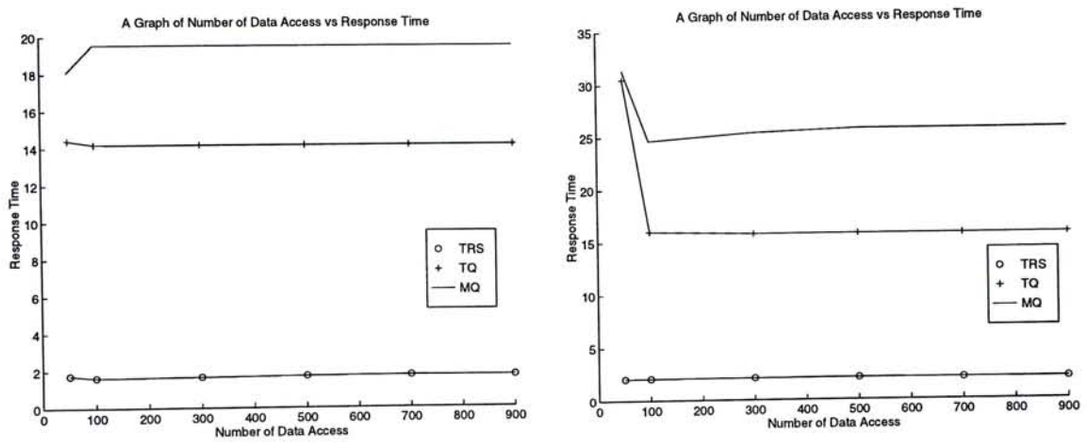
There is nearly no change for the response time in TRS, Majority Quorum and Tree Quorum protocols with the change of the cache hit ratio.

Conclusion

The size of cache hit ratio is not significant enough to affect the performanace of TRS, Tree Quorum and Majority Quorum.

5.1.10 Experiment 10 : Variation of Number of Data Access

Through this experiment, the variation of the number of data access is used to adjust the data conflict such that we can verify the performance of TRS, Tree Quorum and Majority Quorum under the situation of data conflict. As the total number of public data for each site is 1000. We vary the number of data access for each site from 50 to 900. The set of varied data is the same.



(a) 4 sites

(b) 13 sites

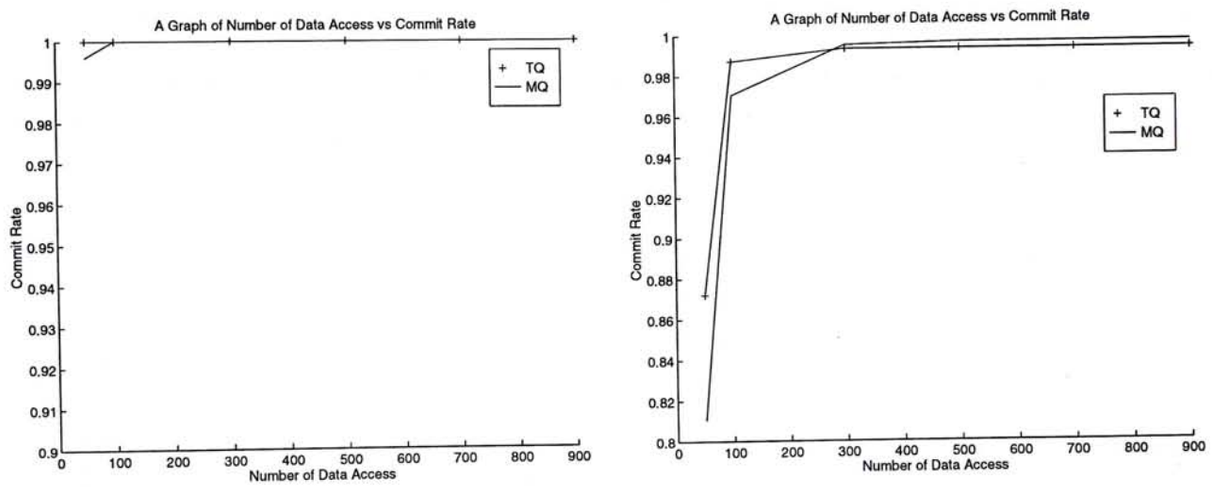
Figure 5.21: Experiment 10 Varying Number of Data Access Vs Response Time

In Figure 5.23, as the number of data access is decreased, the percentage of data conflict increases. The performance of TRS is not affected much because there is no locking required to secure the data. However, for Majority Quorum and Tree Quorum Protocols, data locks are required during data access and that is why the performance of Majority and Tree Quorum protocols is poorer.

Performance Index	Value
Interarrival Time of transaction	80s
TRS	2s
Number of Data Access	50, 100, 300, 500, 700, 900

Table 5.16: Experiment 10 Varying Number of Data Access

From figure 5.25(b), For Majority Quorum and Tree Quorum, their commit rate are decreasing due to data conflict access.



(a) 4 sites

(b) 13 sites

Figure 5.22: Experiment 10 Varying Number of Data Access Vs Commit Rate

Conclusion

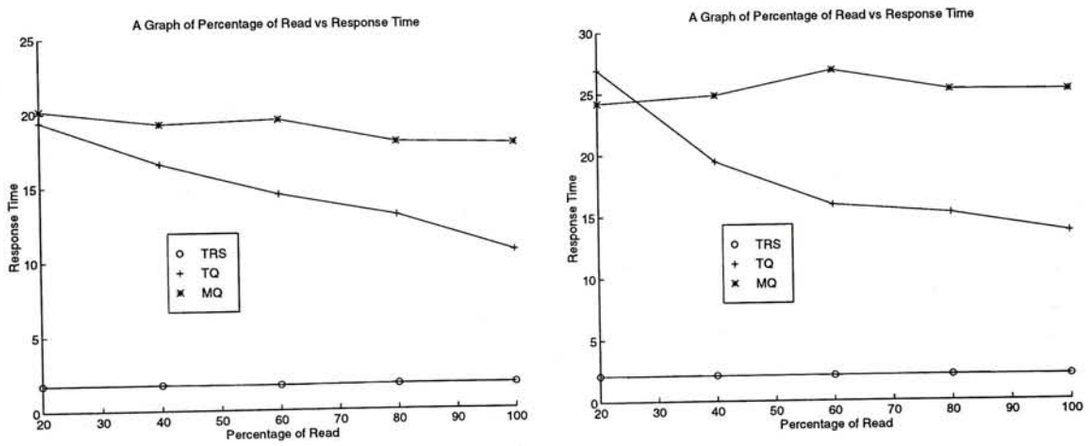
The performance of TRS is not affected by the amount of data conflict. However, for Majority Quorum and Tree Quorum Protocol, its performance decrease by the increase in data conflict.

5.1.11 Experiment 11 : Variation of Read Operation Ratio

By varying the ratio of the read operation to write operation, we find out whether the performance of TRS, Tree Quorum and Majority Quorum are affected. The ratio of read operation to write operation is varied from 20 to 100%.

Performance Index	Value
Interarrival Time of transaction	80s
TRS	2s
Read Operation Ratio	20%, 40%, 60%, 80%, 100%

Table 5.17: Experiment 11 Varying Read Operation Ratio



(a) 4 sites

(b) 13 sites

Figure 5.23: Experiment 11 Varying Read Operation Ratio Vs Response Time

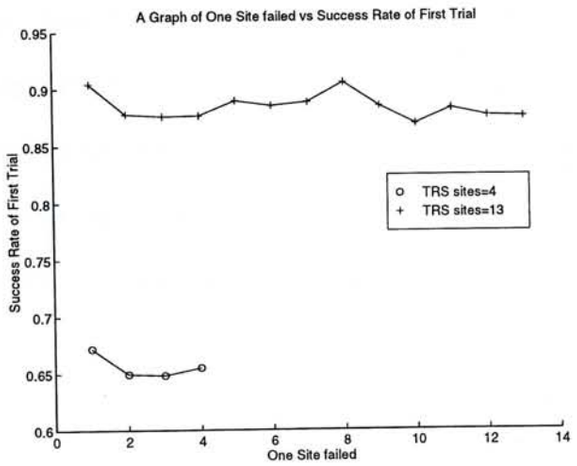
From figure 5.26, as the ratio of read operation increases, the response time decreases for Tree Quorum Protocol. This is because the tree quorum protocol have a read quorum which consists of just one site, i.e. the root of the logical tree. Unlike Majority Quorum, which have the same weight for both read and write quorum. There is nearly no change on the response time for Majority Quorum Protocol. For TRS, there is no problem of acquiring quorums, that is why there is no change on the response time.

Conclusion

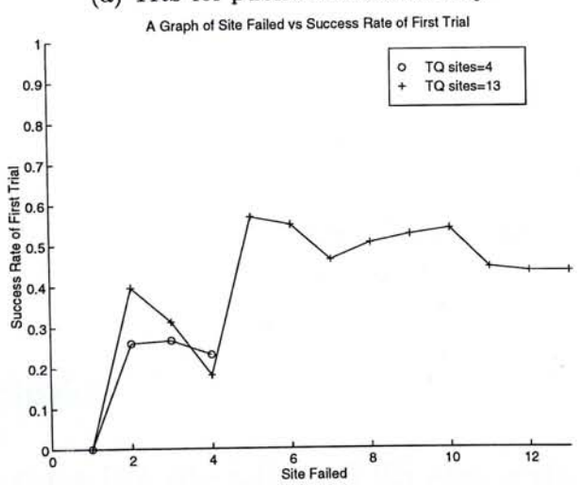
Both the performance of TRS and Majority Quorum is not affected by the ratio of read operation within a transaction. The response time of Tree Quorum is decreased by the increase in the ratio of the read operation.

5.1.12 Experiment 12 : Variation of One Site Failed

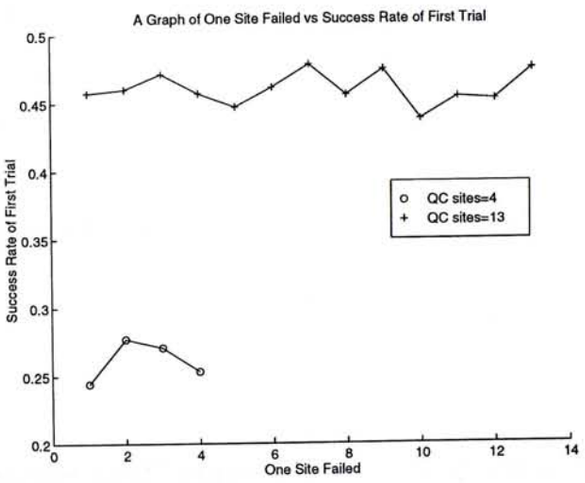
As each site has a unique site identity number in this system, we vary different site failed each time. That is varying different site ID failed. Therefore, we can find out whether there is any effect on the performance of the system by the failure of different site. In this experiment, the success rate of first trial is measured. That means, the success rate is measured due to the site requests for the quorum at first round only. As described in the Section 3.1.11, Virtual Partition Protocol is used in TRS.



(a) TRS for public transaction only



(b) Tree Quorum



(c) Majority Quorum

Figure 5.24: Experiment 12 Varying of Sites Failed Vs Success Rate of First Trial

Performance Index	Value
Transmit Delay	0.3s
Interarrival Time of transaction	80s
Simulation Time	5000s
TRS	2s
Site Failed	1-4, 1-13

Table 5.18: Experiment 12 Varying of Sites Failed

The success rate of first trial is the highest for TRS. It implies that even under the site failure condition, TRS also performs better compared with Tree Quorum and Majority Quorum Protcols. As described in Section 2.2.2, the tree quorum protocol employs the dimension of read quorum <1,2> and write quorum <3,2>. Thus, the failure of the site 1 (i.e. the root) makes the success rate of first trial zero because the root is the critical site of the logical tree at first trial.

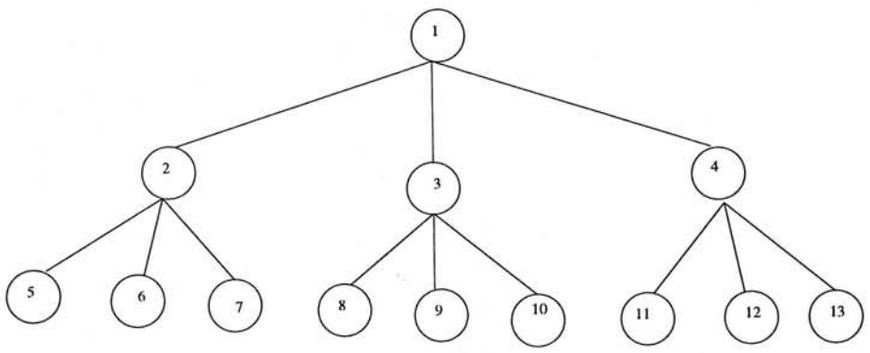


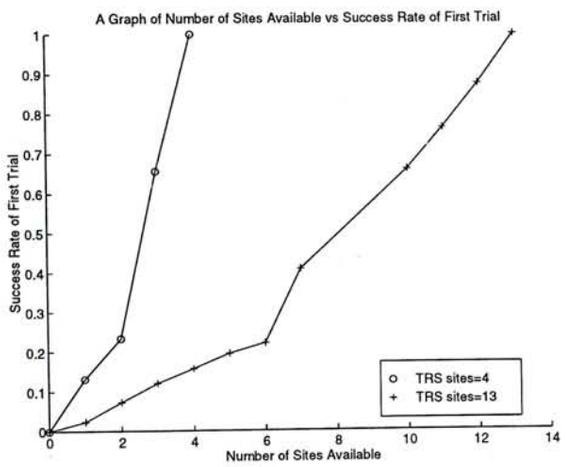
Figure 5.25: The diagram of Ternary Tree

Conclusion

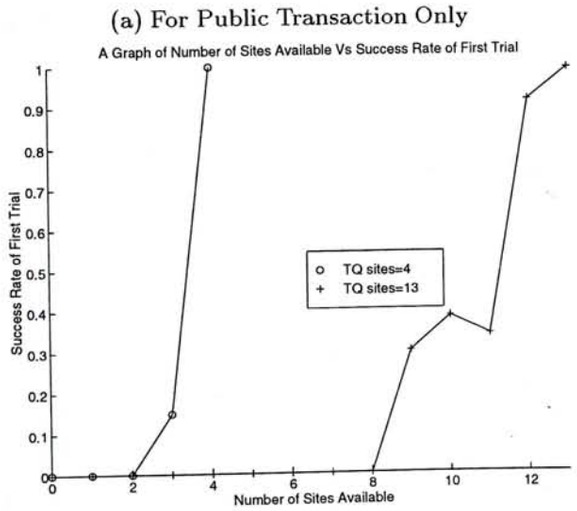
Even there is a site failure, TRS also performs the best compared with Tree Quorum and Majority Quorum Protocols.

5.1.13 Experiment 13 : Variation of Sites Available

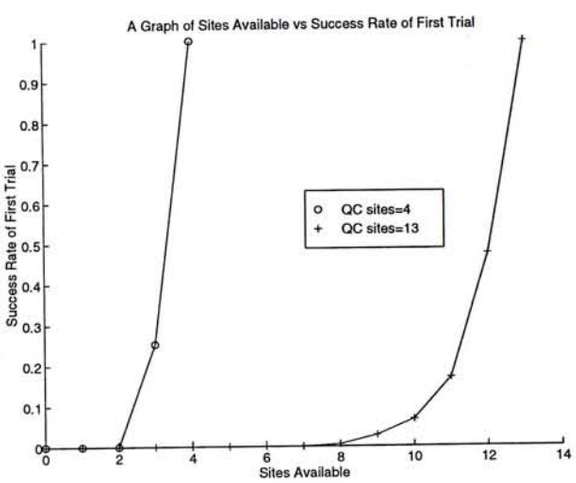
In this experiment, we vary the number of sites available, that means the number of sites which function normally. We vary the number of sites available from 1 to 4 (where the total number of sites is 4) and from 1 to 13 (where the total number of sites is 13) respectively. Through this experiement, we want to find out how TRS performs under failure condition.



(a) For Public Transaction Only



(b) Tree Quorum



(c) Majority Quorum

Figure 5.26: Number of Sites available Vs Success Rate of First Trial

Performance Index	Value
Transmit Delay	0.3s
Interarrival Time of transaction	80s
Simulation Time	5000s
Disk I/O	0.007s
Number of Operations	20
TRS	2s
Sites Available	1-4, 1-13

Table 5.19: Experiment 13 Varying of Sites Failed

As the number of sites failed increases, the commit rate of transaction decreases. For Tree and Majority Quorum, the commit rate decreases to zero when the number of sites failed decrease to a certain number. TRS performs better, this is because even the number of sites failed is more than half of the total number of sites, there are still local transactions can be executed under node autonomy.

Conclusion

Even when more than half of the total number of sites failed, there is still success rate of first trial for TRS, which is better than those of Tree Quorum and Majority Quorum.

Chapter 6

Conclusion

From the above Simulation Results, we can conclude that TRS performs much better than Tree Quorum Protocol and Majority Quorum Consensus Protocol under some conditions. Also, Tree Quorum Protocol performs better than Majority Quorum Consensus Protocol. For the TRS model, selecting TRS period is rather significant because it would affect the overall performance, especially the response time. If TRS period is too large compared to the time required to execute a batch of public transactions, the response time will be greater. However, if the TRS period is too short, execution of a batch of public transaction cannot be finished within one TRS period. Moreover, the frequency of message broadcasting increases. So, the response time increases if the TRS period is too short.

TRS performs better when the ratio of local transactions is larger than that of public transactions. As the local transactions can be executed immediately, its response time is rather short. The overall response time is deteriorated by the high response time of public transactions.

Compared among TRS, Tree Quorum and Majority Quorum Consensus Protocol, TRS performs much better than the others when the number of transaction operation is reasonably large, for example, 25.

TRS performs better if the message cost is much greater than the disk I/O and computational cost. The performance of TRS is not much affected by the variation of disk I/O and computatoinal cost.

From [Mil88] the clock synchronization is within a few milliseconds, compared with TRS period which is in the order of second, the accuracy of the clock synchronization does not have much effect on the performance of the TRS.

Also, data conflict does not affect the performance of TRS because no locking is required due to data access. Unlike TRS, the performance of Tree and Majority Quorum Protocols is affected by the data conflict.

To conclude, TRS is suitable for the nowadays business applications which involve high message cost (e.g. in WAN), and which are more I/O-oriented than computation-oriented.

Bibliography

[AA91] Divyakant Agrawal and Amr El Abbadi. An Efficient and Fault-Tolerant Solution for Distributed Mutual Exclusion. *ACM Transaction on Computer Systems*, February 1991.

[AA92] D. Agrawal and A. El Abbadi. The Generalized Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data. *ACM Transactions on Database Systems*, 17(4):689–717, December 1992.

[Agr85] Rakesh Agrawal. Models for Studying Concurrency Control Performance Alternatives and Implications. *Proceedings of the 1985 Sigmod Conference, Austin*, pages 108–121, May 1985.

[AS89] A. El. Abbadi and S.Toueg. Maintaining Availability in Partitioned Replicated Database. *ACM Transactions on Database Systems*, 14(2):264–290, June 1989.

[BC] Jerry Banks and John S. Carson. Discrete-Event System Simulation.

[BG92] D. Bell and J. Grimson. *Distributed Database Systems.* Addison-Wesley, 1992.

[CDY90] Bruno Ciciani, Daniel M. Dias, and Philip S. Yu. Analysis of Replication in Distributed Database Systems. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):247–261, June 1990.

[cFlC94] Ada Wai chee Fu and David Wai lok Cheung. A Transaction Replication Scheme for a Replicated Database with Node Autonomy. In *Proceedings of the 20th VLDB Conference*, pages 214–225, 1994.

[CL91] Michael J. Carey and Miron Livny. Conflict Detection Tradeoffs for Replicated Data. *ACM Transactions on Database Systems*, 1991.

[Fai] Ted Faison. Borland C++ Object-Oriented Programming.

[FJCS95] Paul J. Fortier and Jr John C. Sieg. Simulation Analysis of Early Commit Concurrency Control Protocols. *Proceedings of the 28th Annual Simulation Symposium*, pages 322–331, April 1995.

[Gra88] Jim Gray. The Cost of Messages. *ACM Transactions on Database System*, 1988.

[Gra91] Jim Gray. *The Benchmark Handbook for Database and Transaction Processing Systems.* Morgan Kanfmann, 1991.

[HM84] J. Y. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In *Proceedings of the Third Symposium on Principles of Distributed Computing*, pages 50–61, August 1984.

[Hug88] J. G. Hughes. *Database Technology.* Prentice Hall, 1988.

[KC91] Akhil Kumar and Shun Yan Cheung. A high availability $\sqrt{N}$ hierarchical grid algorithm for replicated data. *Information Processing Letters, North-Holland*, 40(6):311–316, December 1991.

[KS91] Henry F. Korth and Abraham Silberschatz. *Database System Concepts.* McGraw-Hill, 1991.

[KS93] Akhil Kumar and Arie Segev. Cost and Availability Tradeoffs in Replicated Data Concurrency Control. *ACM Transactions on Database Systems*, 1993.

[Lam78] Leslie Lamport. Time, Clocks and the Ordering of Events. *Communications of the ACM*, 21(7):558–565, July 1978.

[Mae85] Ma Moru Maekawa. a $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, May 1985.

[Mil88] D. L. Mills. Network Time Protocol (Version 1) Specification and Implementation. *DARPA-Internet Report RFC-1059*, July 1988.

[ON92] M.Tamer Ozsu and Youping Niu. Effects of Network Protocols on Distributed Concurrency Control Algorithm Performance. *Proceedings, ICCI'92, Forth International Conference on Computing and Information*, pages 301–306, May 1992.

[PGM89] F. Pittelli and H. Garcia-Molina. Reliable Scheduling in a TMR Database System. *ACM Transactions on Computer Systems*, 7(1):25–60, February 1989.

[PN81] P.A.Bernstein and N.Goodman. Concurrency Control in Distributed Database Systems. *ACM Computing Surveys*, 13(2):185–221, June 1981.

[PVN87] P.A.Bernstein, V.Hadzilacos, and N.Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, ReadingMass., 1987.

[Rah93] Erhard Rahm. Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems. *ACM Transactions on Database System*, 1993.

[Sch] Herb Schwetman. Getting Started with CSIM17 (C++ Version).

[SG84] S.Ceri and G.Pelagatti. *Distributed Databases Principles and Systems.* McGraw-Hill, 1984.

[SHKS95] S.Ceri, M.A.W. Houtsma, A.M. Keller, and P. Sammarati. Independent Updates and Incremental Agreement in Replicated Database. *Distributed and Parallel Databases An International Journal*, 3(3):225–245, July 1995.

[SLSV95] Dennis Shasha, Francois Llirbat, Eric Simon, and Patrick Valduriez. Transaction Chopping:Algorithms and Performance Studies. *ACM Transactions on Database Systems*, 20(3):325–363, September 1995.

[TOV91] M. Tamer-Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991.

[VP] Stewart V.Hoover and Ronald F. Perry. Simulation - A Problem Solving Approach.

[WN90] Kevin Wilkinson and Marie-Anne Neimat. Maintaining Consistency of Client-Cached Data. *Proceedings of the 16th VLDB Conference Brisbane, Australia,* 1990.

Appendix A

Implementation

A.1 Assumptions of System Model

The network is assumed to be completely connected. That is, when a site broadcast messages to all the other sites, it sends message to the other site point-to-point directly.

Each site holds a different number of local data. In our design, it is assumed to be a maximum number of 1000 local data for each site.

And, each site holds the same number of public data. This number can be set to a value <= 100.

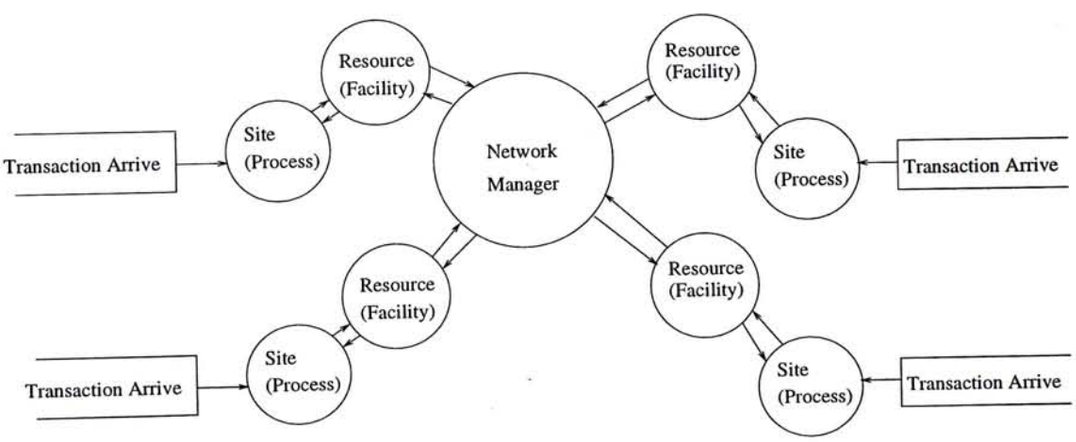


Figure A.1: Simulation Model

A.1.1 Program Description

First of all, we have to simulate a number of sites in the system. In our design, we have implemented 4, 13 and 40 sites and a site is programmed as a class called "serve".

As each site can process the same function, there are member functions for the class "serve". Each site is distinguished by its unique identity number and in our design is "serve_no". For example, site 1 has serve.serve_no=1.

At the beginning of the program, there are two member functions for this class "serve" invoked to initialize and start the site's execution. They are "serveinit" and "start" respectively.

Common Functional Modules for TRS, Tree Quorum and Majority Quorum

- "serveinit" is to initialize the version and timestamp of each site's data.
- "start" is a process created to simulate that the site is now starting to operate throughout the simulation time. In our design, all the "start" processes created at the same time and executed independently.
- "sending" is a process created and get the site's cpu time (simulated by site's facility) for sending message.
- "usechannel" is a process invoked after completion on the holding of site's cpu time for sending message. It is used to reserve the facility "channel" and hold the message transmit time to simulate sending the message.
- "receivetime" is to simulate the receipt sites holding message cpu time to receive the commit message.
- "result" is to generate each sites performance on the execution of transaction.
- "report" and "mdlstat" are CSIM function to report on the utilization of each CSIM facility used.

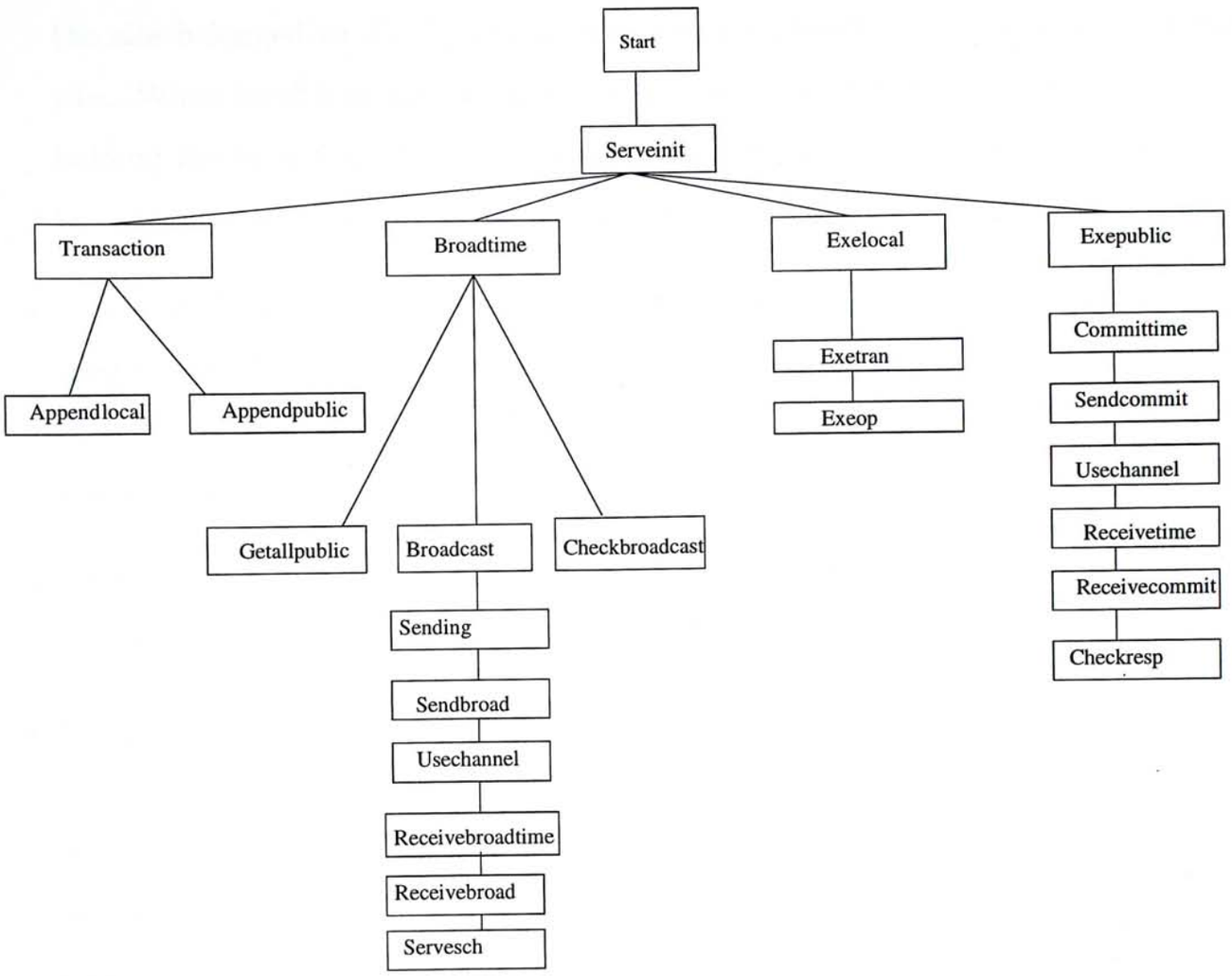


Figure A.2: Program module of TRS

A.1.2 TRS System

The START process is created at each site, the START process invokes four processes simultaneously. They are transaction, broadtime, exelocal and exepublic.

TRS which include the following functions:

- "transaction" is a process which is executed throughout the simulation time and used to generate the local and public transactions to its own site. We can make variation on the interarrival rate of transaction by using "exponential rate", variation on the ratio of local to public transaction. And each transaction consists of 1 to 20 (uniform)read/write operations. The cpu time needed to execute an operation is varied by input of the "cpu time". The data access consists of local or public types,

the site belonged to if it is a local data and the identity number of data for that site. When local transaction arrives at a site, it will be scheduled to a queue for holding the local transactions. Similarly, for the public transaction arrival, it will be scheduled to a queue for holding the public transactions.

- "exelocal" is a process created and executed throughout the simulation time. It is used for handling the local transaction received and then pass to "exetran". When there is no local transaction, this "exelocal" process will wait until another local transaction arrive.

- "exetran" is a process to execute the local transaction independently which is used for passing the transaction operation to "exeop".

- "exeop" is a process to find out whether this site's cpu(simulated by CSIM class "facility") is available or not. If available, it will execute the local transaction immediately and execute the transaction according to the executed time it required. Else if the site's cpu is not available, this "exeop" process will be appended to the queue and wait until the site's cpu is available.

- "broadtime" is a process including both two process of "broadcast" and "check-broadcast".

- "broadcast" is a process created and executed throughout the simulation time. It is used to broadcast all the public transactions and the latest version of its local data that are received within a TRS period. That is, when TRS period arrives, this "broadcast" process will send all the public transactions and the latest version of local data received to all the other sites. During broadcasting, it calls a process "sending".

- "checkbroadcast" is a process to check whether the site can receive all the broadcast message from all the sites within a "time-out" period. Else, after "time-out", the network is assumed to be partitioned.

- "receivebroadtime" is a process created for getting the receiver site's cpu time (simulated by site's facility) for receiving message. After that, it calls "receivebroad".

- "receivebroad" is a process created to store up the public transactions and the latest version of local data received from the sender's site. After storing, the site will check out whether the messages are received from all sites. If so, the site will call another process "servesch" and set the event "done". After so, the process "receivebroad" will terminate.

- "servesch" is a process created to schedule the public transactions received according to the timestamp order of the transaction.

- "exepublic" is a process created to execute the public transactions which are already scheduled. In our design, it will wait until the event "done" set. So, the process "exepublic" starts to execute the scheduled public transaction. It also has to hold the site's cpu time (by reserving the site's facility). If the site's cpu is unavailable, it has to schedule to a queue waiting until the site's cpu is available. When a batch of public transactions are committed in the site, it will invoke a process "committime".

- "committime" is a process created to send commit message to all the other sites to acknowledge them a batch of public transactions are committed. When a site receive commit message from all sites, this batch of public transactions are confirmed to be committed also.

- append is used for queuing up the incoming local transaction.

- appendpublic is used for queuing up the incoming public transaction.

- getallpublic means that the site receives the batch of public transactions within a TRS period from all sites.

- sendbroad is used to invoke the receipt site to receive the broadcast message.

- sendcommit is used to invoke the receipt sites to receive the commit message.

- receivecommit is used to check whether the commit message of the same batch of public transaction within a TRS period are received from all sites.
- checkresp is responsible for checking the response time of a batch of public transaction.

Data Structures

Fields of Transaction_type

- arrive time of the transaction
- number of operation in a transaction
- read operation or write operation
- access which site's data
- access which data for that particular site
- access local or public data for public transaction
- cpu time for processing the operation

Fields of Commit_Batch_Public_Transaction

- timestamp of the batch
- number of public transactions per batch

A.1.3 Common Functional Modules for Majority Quorum and Tree Quorum Protocol

At the beginning of the program, there are two member functions for this class "serve" invoked to initialize and start the site's execution. They are 1)"serveinit" and 2)"start" respectively.

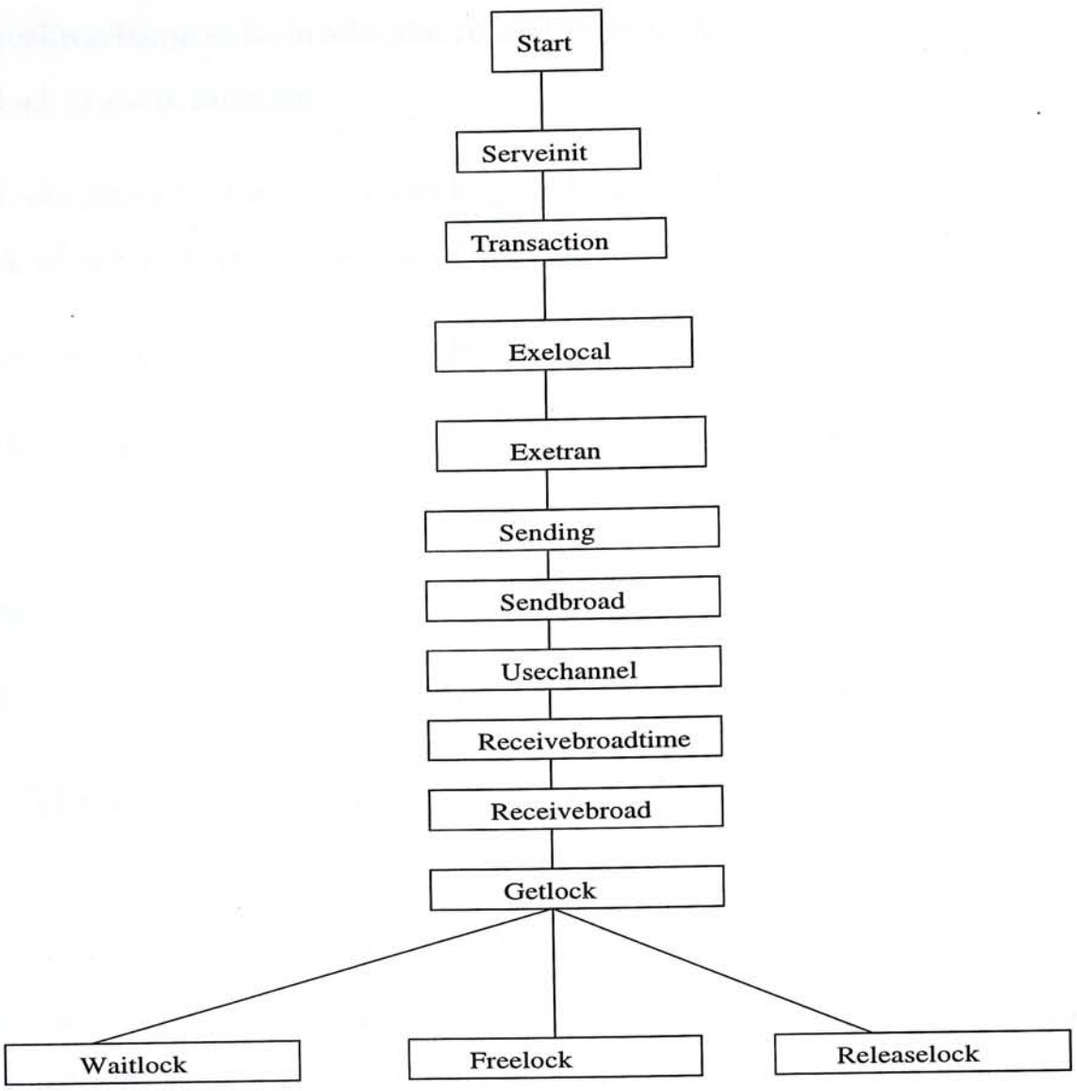


Figure A.3: The Program Module of Majority Quorum Consensus and Tree Quorum

- "transaction" is a process which is executed throughout the simulation time and used to generate the transactions to its own site. We can vary the interarrival rate of transaction by using "exponential rate". And each transaction consists of 1 to 20 (uniform) read/write operations. The cpu time needed to execute an operation is varied by input of the "cpu time". Each data object has its own identity number. When transaction arrives to a site, it will be scheduled to a queue for holding the transactions.
- sendbroad is to ask for "lock request" according to Majority Quorum Consensus.

- receivebroadtime is to invoke the receipt sites to hold message cpu time to receive the lock request message.
- receivebroad is that each site receives its lock message which it requests and has to check whether all the request lock message are received.
- getlock is to check the lock available or not.
- waitlock is a process such that if the lock requested is not available, wait it for a "time-out" period.
- releaselock is for checking whether all the locks hold are released.
- freelock is used for releasing the lock.

A.1.4 Majority Quorum Consensus Protocol

- "exelocal" is a process created and executed throughout the simulation time. It is used for executing the transaction (arrived to its own site only). That is, when there is transaction arriving to the queue, firstly, it will request its own lock for this data accessed. Then, it will invoke another process "sending" which sends the request message for this data lock to the Majority of the sites. That is, in this case, as there are 13 sites total, and one of its own site's lock is already hold, so, it has to ask another 6 sites to get the locks respectively. When the other sites receive the request message, it will send the message back to the request site in case the lock is free. If the lock is hold already, then it will queue up to wait the free lock for a time-out period. If after the time-out period, the site still cannot get the lock, it will abort the transaction. If the site can get the lock on time, it will wait until the majority request locks' message are received. Next, the site will execute the transaction in case its cpu is available. If not, it will schedule to the queue for the cpu. When the operation is executed, it will release its own lock. Then, it will send the release lock message to all the other majority sites.

A.1.5 Tree Quorum Protocol

If there are 13 sites total, they are arranged as a ternary tree. There are read and write quorum. The read quorum has length and width as $\langle 1, 2 \rangle$. i.e. site 1. The Write Quorum has length and width as $\langle 3, 2 \rangle$. i.e. either $\langle 1, 2, 3, 5, 6, 8, 9 \rangle$ or $\langle 1, 2, 3, 6, 7, 8, 9 \rangle$.

- "exelocal" is a process created and executed throughout the simulation time. It is used for executing the transaction (arrived to its own site only). That is, when there is transaction arriving to the queue, it will request the lock according to the read or write operation by its read or write quorum. When the other sites receive the request message, it will send the message back to the request site in case the lock is free. If the lock is already hold, it will wait for a time-out period. If the site still cannot wait the lock after time-out period, it will abort the transaction. Else, if the site can get the lock on time, it will wait until all the request lock's message are received. After that, the site will execute the operation in case its cpu is available. If not, it will schedule to the queue for the cpu. When the operation is executed, it will release its own lock. Then, it will send the release lock message to all the other sites.