

ABSTRACT

Analysis of Transaction Throughput in P2P Environments

Arun Chokkalingam

Mentor: Gregory D. Speegle, Ph.D.

In recent years P2P systems have gained tremendous popularity. Support of a transaction processing facility in P2P systems would provide databases at a low cost. Extending distributed database algorithms such as 2PC and ROWA to P2P environments might not provide the best performance because the P2P systems are characterized by high site failure rates and an unpredictable network topology. The choice of algorithms in building P2PDB is difficult because of the lack of information about the performance of database algorithms in P2P environments. This thesis analyzes the performance of one such algorithm, the epidemic algorithm against the performance of traditional database algorithms in simulated P2P environments.

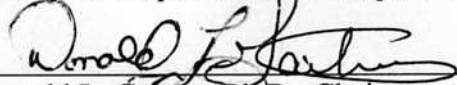
Analysis of Transaction Throughput in P2P Environments

by

Arun Chokkalingam, B.E.

A Thesis

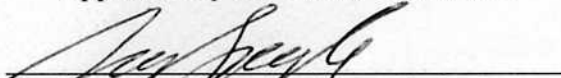
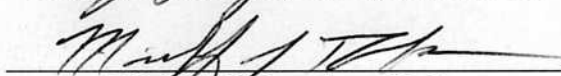

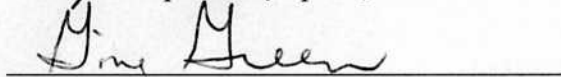
Approved by the Department of Computer Science



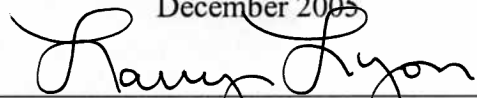
Donald L. Galtros, Ph.D., Chairperson

Submitted to the Graduate Faculty of
Baylor University in Partial Fulfillment of the
Requirements for the Degree
of
Master of Science

Approved by the Thesis Committee


Gregory D. Speegle, Ph.D., Chairperson
Michael J. Donahoo, Ph.D.
Stephen L. Gipson, Ph.D.
Gina C. Green, Ph.D.

Accepted by the Graduate School
December 2005


J. Larry Lyon, Ph.D., Dean

Copyright © 2005 by Arun Chokkalingam

All rights reserved

TABLE OF CONTENTS

LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF EQUATIONS	x
LIST OF ABBREVIATIONS	xi
ACKNOWLEDGMENTS	xii
DEDICATION	xiii
Chapters	
1 Introduction	1
1.1 The Problem.....	4
1.2 Background.....	4
1.2.1 DBMS and Transactions.....	4
1.2.2 Distributed Databases.....	6
1.2.3 Replication.....	7
1.3 Overview.....	7
2 Related Work.	8
2.1 Database Techniques.....	8
2.2 P2P Techniques.....	10
3 Two Phase Locking Scheduler Simulation	13
3.1 Two Phase Locking Scheduler.....	13
3.2 Two Phase Locking Scheduler Simulation Design and Implementation ...	14

3.2.1	The Lock-Wait Table.....	14
3.2.2	The Process Method Operation.....	15
3.3	The Centralized Database System	15
3.3.1	Local Database	16
3.3.2	Transaction Manager.....	17
3.3.3	Scheduler.....	17
3.3.4	Transaction.....	17
3.3.5	Operations.....	18
3.4	Experiments for the verification of the correctness of the Scheduler Simulation	20
3.4.1	Description of Two Phase Locking and Thrashing Behavior	20
3.4.2	Parameters.....	21
3.4.3	Results.....	23
4	Simulation of a Traditional Distributed Database Techniques	32
4.1	Traditional Database Techniques used in the Distributed Database Systems.....	27
4.1.1	Two Phase Locking Scheduler	27
4.1.2	Read One Write All Mechanism.....	27
4.1.2	The Two Phase Commit Protocol.....	28
4.2	Distributed Database Simulation.....	32
4.2.1	Global Manager.....	33
4.2.2	Transaction Manager.....	35
4.2.3	Communication Manager.....	36
4.2.4	Operations.....	37
4.3	Verification of the correctness of our distributed database simulation.....	37

4.3.1	Description of the Simulation in (Carrey and Livny 1991).....	37
4.3.2	Modeling our simulation to match the parameters of the experiments of Carrey and Livny (Carrey and Livny 1991).....	40
4.3.3	Differences between our simulation and (Carrey and Livny 1991).....	41
4.3.3	Results of the Experiments.....	42
5	Simulation of Epidemic Algorithms	43
5.1	Epidemic Algorithms.....	43
5.1.1	Optimistic Protocol.....	43
5.1.2	Pessimistic Protocol.....	46
5.2	Simulation of Epidemic Algorithms.....	46
5.2.1	Global Manager.....	46
5.2.2	Transaction Manager.....	47
5.2.3	Communication Manager.....	49
5.2.4	Time Table.....	49
5.2.5	Transaction Record.....	49
5.2.5	Other Components.....	49
5.3	Verification of our Simulation of Epidemic Algorithms.....	51
5.3.1	Verification Experiments.....	51
6	Experiments and Results	59
6.1	Parameter Settings for the Experiments.....	59
6.1.1	Communication Time.....	59
6.1.2	I/O Time.....	59
6.1.3	Duration of Read Write and Log Write.....	59
6.1.4	Bandwidth.....	59

6.1.5	Time to Failure and Time to Recover.....	63
6.1.6	Other Parameters.....	63
6.2	Experiments and Results.....	65
6.2.1	Experiment 1: Time To Failure Fixed at 60 seconds and Time to Recover Fixed at 2 seconds	66
6.2.2	Experiment 2: Time To Failure Fixed at 60 seconds and Time to Recover Fixed at 10 seconds	67
6.2.3	Experiment 3: Time To Failure Fixed at 60 seconds and Time to Recover Fixed at 60 seconds	67
6.2.4	Experiment 4: Time To Failure Fixed at 60 seconds and Time to Recover Fixed Varied between 2 to 60 seconds... ..	68
6.2.5	Experiment 5: Time To Failure Varied from 30 to 90 seconds and Time to Recover Fixed at 2 seconds	68
6.2.6	Experiment 6: Time To Failure Varied from 30 to 90 seconds and Time to Recover Fixed at 10 seconds	69
6.2.7	Experiment 7: Time To Failure Varied from 30 to 90 seconds and Time to Recover Fixed at 60 seconds	69
6.2.8	Experiment 8: Time To Failure Varied from 30 to 90 seconds and Time to Recover Fixed Varied between 2 to 60 seconds	70
6.2.9	Experiment 9: Low Conflict Scenario with Time To Failure Fixed Varied from 30 to 90 seconds and Time to Recover Varied from 2 to 60 seconds and $MPL = 5$	70
7	Conclusion and Future Work	74
7.1	Conclusion.....	74
7.2	Future Work and Recommendations.....	75
	BIBLIOGRAPHY	78

LIST OF FIGURES

1.1	Example of a distributed database	3
1.2	Inventory table of a database	5
3.1	A transaction's phases when it goes through a two phase locking scheduler	14
3.2	Algorithm used by ProcessOperation of 2PL Scheduler	16
3.3	The system model for the centralized database simulation experiments	17
3.4	Algorithm used by the process method of transaction manager	18
3.5	Graph that shows the variation of throughput with multi programming level at the simulated 32 second interval	23
3.6	Results of analysis of Alexander Thomassian (Thomassian 1993)	24
3.7	Results from simulation at time = 32 seconds	25
3.8	Comparison of median experimental and estimated response times at 32 seconds	25
3.9	Error between the median experimental time and estimated response	26
4.1	Two Phase Commit protocol	30
4.2	The system model of the distributed database simulation	34
4.3	The algorithm of the process method of the global manager	34
4.4	receiveOperation method of the global manager, parameter: operation	35
4.5	processOperationOrMessage method of transaction manager	36
5.1	Transaction Record as perceived during the epidemic algorithms proposal	44
5.2	Process method of global manager for epidemic algorithms	47
5.3	ProcessTransactionRecord of transaction manager	62
6.1	Results for TTF = 60 seconds and TTR = 2 seconds	66
6.2	Results for TTF = 60 seconds and TTR = 10 seconds	67
6.3	Results for TTF = 60 seconds and TTR = 60 seconds	67

6.4	Results for TTF = 60 seconds and TTR varied from 2 to 60 seconds	68
6.5	Results for TTF varied from 30 to 90 seconds and TTR = 2 seconds	68
6.6	Results for TTF varied from 30 to 90 seconds and TTR = 10 seconds.....	69
6.7	Results for TTF varied from 30 to 90 seconds and TTR = 60 seconds.....	69
6.8	Results for TTF varied from 30 to 90 seconds and TTR varied from 2 to 60 seconds.....	70
6.9	Results for TTF varied from 30 to 90 seconds and TTR varied from 2 to 60 seconds and MPL=5	70

LIST OF TABLES

3.1	Structure of an operation	19
3.2	Status of an operation	10
3.3	The parameter settings for the centralized database simulation	21
4.1	The parameter settings for the experiments of (Carrey and Livny 1991)	40
4.2	The parameter settings for our distributed database simulation	41
4.3	The results of our distributed database simulation	42
5.1	The parameter settings for the epidemic algorithms simulation testing	51
5.2	Test case 1 for epidemic algorithms simulation.....	52
5.3	Test case 2 for epidemic algorithms simulation.....	53
5.4	Test case 3 for epidemic algorithms simulation.....	54
5.5	Test case 4 for epidemic algorithms simulation.....	56
5.6	Test case 5 for epidemic algorithms simulation.....	57
6.1	The parameter settings for p2p environment simulation.....	65

LIST OF EQUATIONS

3.1	Thomassian's formula for Mean Response Time	21
3.2	Duration of Read Operation derived by Ulusoy and Belford	22
5.2	Time Table Property for Epidemic Algorithms	45

LIST OF ABBREVIATIONS

P2P – Peer to Peer

P2PDB – Peer to Peer Database

2PL – Two Phase Locking

2PC – Two Phase Commit

ROWA – Read One Write All

SETI – Search for Extra Terrestrial Intelligence

ACKNOWLEDGMENTS

I would like to thank my mentor, Dr. Speegle, for his guidance and time which helped me achieve this scholarly endeavor. His expertise and insights were invaluable. I thank Dr. Donahoo, Dr. Gipson and Dr.Green for providing valuable comments and suggestions on this work.

To Mom for her love and support

CHAPTER ONE

Introduction

In recent years P2P systems have gained tremendous popularity and consumed a tremendous amount of the global network traffic. Estimates are that 60 to 80 percent of the capacity on consumer ISP networks is consumed by P2P applications (Peer to Peer Estimation Website 2005). P2P file sharing applications like Gnutella (Gnutella 2005) and Freenet (Clarke *et al.* 2000) have brought enormous attention to the P2P systems. However there are many P2P applications. Chinook Online (Chinook 2005) dynamically discovers new bioinformatics services across a heterogeneous network and allows the users to run distributed bioinformatics programs. SETI (SETI Homepage 2005) uses the power of peer-to-peer computing to search for extraterrestrial intelligence by analyzing radio telescope data.

Another possibility for P2P systems is the idea of incorporating database functionality. Let us consider a database application distributed among these peers. Consider the scenario of a student group conducting an event and needing a database server to handle the registration and management of the event. If they could achieve this by connecting to an existing P2PDB network or using a network of student computers, they would save a lot of time and cost. In this application, a user would register via a website. The web server would then use the student computers in the P2PDB to store the data. This is better than a single database because the availability is high (in our example, even if some of the student computers experience failure, the P2PDB will still continue working). Once a user registers, the database functionality would ensure that the data at sites contains this information.

Thus, incorporating database functionality on the peer-to-peer systems would equip organizations with powerful database servers for low cost.

Various problems like security, performance, storage, etc. emerge when we try to incorporate database functionality into P2P environments. Storage management is a concern because these peers are usually desktop computers with very little memory for sharing. When data is stored in numerous unknown peers, security of data becomes an important issue. This work is primarily concerned with performance. The peers in the P2P systems are characterized by high failure rates and the composition of the system is not known beforehand, which is not the case in typical distributed database applications. Thus the performance of standard database techniques in P2P is unknown

While trying to choose algorithms for P2PDB, the natural solution is to extend the already existing and stable algorithms used in distributed databases. Distributed databases are those where the data is dispersed among multiple sites (which are stable). An example of a distributed database is that of a company which has its department records (eg: HR, Finance and Sales) in different cities (eg: Austin, Waco, Dallas) as shown in Figure 1.1. Replication of data is done to increase availability. In this example, HR information could be replicated at all three sites to increase its availability. For instance, failure of individual nodes would not prevent retrieval. Likewise, the communication overhead of accessing the data from a remote site is eliminated. When replicated data has to be handled by a distributed database, DDMSs like PostgreSQL (Johnson, 2002) and Oracle (DBASupport's Oracle Replication Site 2005) implement a protocol called ROWA (Read One Write All) to ensure consistency of data across all the sites. As the name implies, when a transaction wants to read the data it does so from one site (usually the nearest) and when a transaction wants to write data, it performs a write at all the database sites that contain a copy of the

data. To ensure global atomicity (the effects of the committed transactions take place at all sites), distributed database systems implement the 2PC (Two-Phase Commit Protocol) (Bernstein *et al.* 1987). Detail descriptions of ROWA and 2PC can be found in Chapter Four.

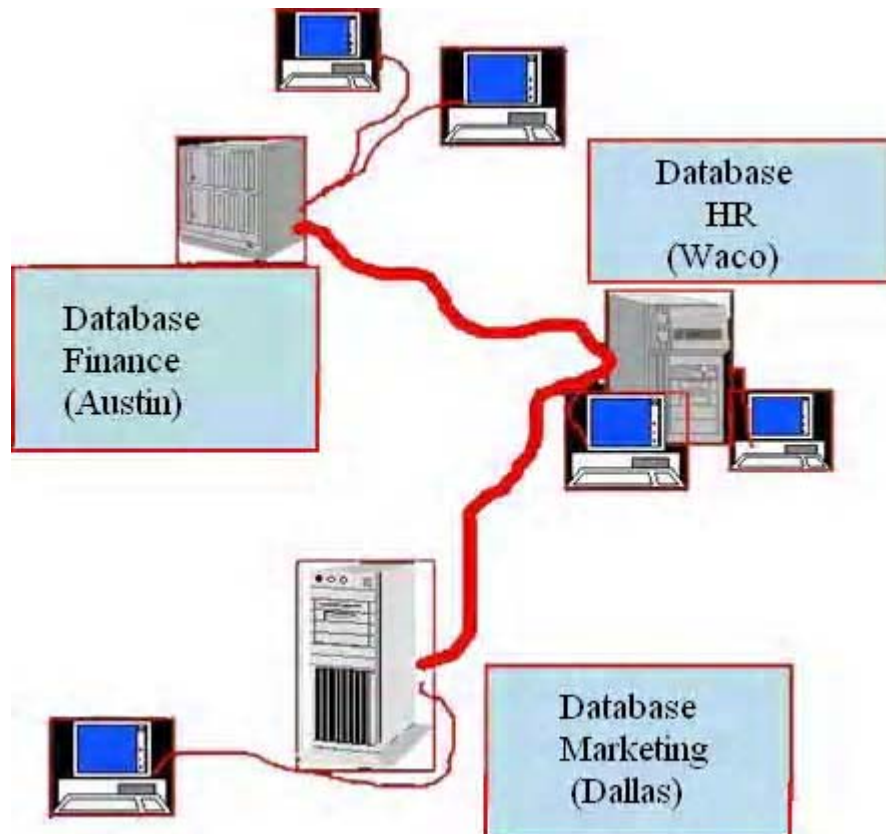


Fig. 1.1. Example of a distributed database

Since P2P systems are characterized by high failure rates of the peers, extension of these techniques to the P2P systems might not scale well as many peers may end up waiting for the response of peers which have already left the system. The lack of instantaneous notification of peer failures also poses a threat to the performance of these algorithms since the transactions would wait until all the peers update their local data without knowing that some of them have failed. Deciding a timeout interval for these transactions is difficult as

the nature of the network cannot be predicted. Hence investigation of other protocols is necessary in order to decide the optimal choice of the algorithms for P2PDBs.

Epidemic Algorithms for replicated databases were proposed before the advent of peer-to-peer systems (Agrawal *et al.* 1997). Here the transactions execute all their operations locally at one site and the effects of these transactions are propagated through the system later like a disease, hence the name “epidemic”. Since the effects of the transactions are propagated later, transactions at peers do not get blocked due to the failure of other peers. Hence epidemic algorithms should prove to be a better choice than the ROWA Protocol coupled with Two Phase Commit for P2PDB. Conflicts detected during the propagation are resolved by application specific rules and the epidemic algorithms work better in environments with low probability of conflicts. A detailed description of epidemic algorithms is presented in Chapter Five.

1.1 The Problem

This thesis compares the performance of 2PC under ROWA semantics against the performance of the Epidemic Algorithms under simulated peer-to-peer environments so as to determine the best approach for P2PDBs.

1.2 Background

1.2.1 DBMS and Transactions

This section presents the background information on databases and transaction processing necessary to have a complete understanding of the thesis. Readers familiar with these concepts can skip this section. A database is an organized collection of information typically represented as tables (Bernstein *et al.* 1987). Each of these tables is a collection of

records of related items. Figure 1.2 shows an inventory table containing inventory records in a database.

ID	Object	Quantity	Price
1001	Table	100	20.00
1002	Chair	50	20.00
1003	Lamp	75	10.00
1004	Bed	100	30.00

Fig. 1.2 Inventory table of a database

Each of the rows of the table represents a record. A DBMS (Database Management System) is a set of programs responsible for modification, extraction, security and storage for the database system. Examples of commercially available DBMS are Oracle, DB2 etc.

A transaction is a unit of interaction with a DBMS. A user who wants to modify or extract information from the database does so by means of a transaction. Modification and extraction of information are achieved by operations, typically SQL statements. A transaction is composed of one or more operations.

The DBMS ensures that the transactions acting in the system abide by the ACID (Atomicity, Consistency, Isolation and Durability) properties (Bernstein *et al.* 1987). Atomicity implies that whenever transactions take place, the database reflects the effect of the transaction completely or not at all. In other words, no partial execution is allowed. Consistency ensures that the database remains in a state consistent with respect to its constraints after the execution of a transaction. Isolation refers to the property that each of the concurrent transactions may execute as though they were the only one in the system.

Durability denotes the property which ensures that the effects of successful transactions will persist in the database.

Transactions that have completed successfully are said to have committed and the others are said to have aborted. In a typical database environment, we have multiple transactions active at the same time. Two such concurrent transactions are said to be conflicting if the write set of one intersects with the read or write sets of another. Transactions are said to be serializable if the effect of their concurrent executions is equivalent to the effect of their serial execution in some order. Thus serialized transactions provide isolation. Databases implement concurrency control algorithms to ensure serializability. The most common of these is called *Two-Phase Locking* (2PL- explained in Chapter Three) which ensures the serializability of transactions by the use of locks.

The DBMS is also responsible for recovery from a site failure. Whenever a site fails the DBMS ensures that the effects of aborted transactions are removed and that the results of committed transactions are not lost.

1.2.2 Distributed Databases

Distributed Databases do not store the data in a single physical location, but disperse data over a network. These locations are known as *sites*. Transactions can start from any of these sites and can span any number of sites depending on the availability of data.

Returning to the example in Figure 1.1, it shows the database setup of a company whose financial records, HR records and marketing records are stored in three different cities (Austin, Dallas and Waco). Transactions can start in any of these three sites and can span one or more of these sites.

1.2.3 Replication

To increase availability of data in the presence of site failures and network failures and to avoid the communication overhead of reading the data from a remote site every time, data can be replicated in more than one site of a distributed database. Consider the scenario shown in Figure 1.1. If the financial department records (currently present at Austin only) are accessed from all three sites very often, replicating the data at all sites eliminates the delay due to the communication time to read the data from the Austin site. Replication is more beneficial when the transaction mix has more reads than updates.

1.3 Overview

Chapter Two presents related research projects in the arenas of databases and P2P systems. Since 2PL is used for concurrency control for both Epidemic Algorithms and the traditional Distributed Database techniques, we developed a simulation of a 2PL scheduler. Chapter Three describes our 2PL simulation and its verification in a centralized database environment. The simulation is then extended to a distributed database environment implementing 2PC and ROWA. Chapter Four presents a detail description of this extension to a distributed database and its verification. The simulation was then modified to implement Epidemic Algorithms, the explanation and performance of which are discussed in Chapter Five. Chapter Six presents the results of the experiments conducted on these two algorithms in simulated P2P environments. Chapter Seven presents the conclusion and future work.

CHAPTER TWO

Related Work

A substantial amount of research has been done in areas of transaction processing and P2P systems. This chapter discusses the work in these areas related to our research. The first half of this Chapter deals with the work in database transaction processing and the second half deals with P2P systems followed by a discussion of the current P2PDBs.

2.1 Database Techniques

To handle replicated environments, an algorithm called Quorum Consensus (Bernstein *et al.* 1987) was proposed where non-negative weights are assigned to each copy of the data item. A read threshold (RT) and a write threshold (WT) for the data items are defined such that both $2 \cdot WT$ and $(RT + WT)$ are greater than the total weight of all copies of the data item. This ensures consistency between read and write operations. Weights are used to indicate the importance of the copies. The Quorum Consensus algorithm has three problems. First, transactions in most applications read more data items than they write. The performance of Quorum Consensus is not good for such applications as the transactions will end up reading multiple copies of the data items from different sites. This can be overcome by making the read quorum of the data item contain only one copy of the data item in which case the write quorum would consist of all the data items. This is exactly the ROWA protocol. A second problem with Quorum Consensus is that it needs a large number of copies to tolerate a given number of site failures. Considering that the quorums are all majority sets, it needs three copies to tolerate one failure, five copies to tolerate two failures, and so forth. A third problem with it is that all the copies must be known in advance.

An alternative to Quorum Consensus, called the Available Copies algorithm (Bernstein *et al.* 1984), was proposed to handle site failures in distributed databases. Here every read is translated into “read any copy” and every write is translated into “write all available” copies. This however has two significant requirements. First, site failures should be clean and detectable. Second network partitions should not occur. Both of these are not always satisfied by P2P networks as peers fail without any notification and the networks span the entire globe.

Another option is to add semantics to transactions. Such approaches have been proposed with long lived transactions (LLT). An LLT holds on to resources for a long time, thereby delaying the other common transactions. SAGAS (Molina and Salem, 1987) alleviate problems that arise due to LLTs. These LLTs were written as sequence of transactions that can be interleaved with other transactions and either all the transactions that constitute the LLT are completed or compensating transactions are run to amend the partial executions. ACTA (Chrysanthis and Ramamritham, 1990) allows for specification of structure and behavior of transactions and reasoning for their concurrency and recovery properties. The semantics of interactions are expressed in terms of transactions’ effects on the commit and abort of other transactions and on objects’ state and concurrency status. NT/PV (Korth and Speegle, 1994) enhanced the standard transaction model with nested transactions, explicit predicates and multiple versions. These are extensions to transactions, except ACTA which unifies existing models, and are not intended to attack the problems of P2P environments like site failures, etc. Moreover sufficient information is not available about their performance on P2P environments.

Carrey and Livny (Carrey *et al.* 1991) have analyzed the performance of 2PC and ROWA in replicated database environments. But this work does not deal with site failures of

any sort. This work has been our reference model in our experiments to verify our distributed database simulation.

The performance of 2PC in the presence of site failures for a distributed database environment is analyzed in (Agrawal *et al.* 1998). This cannot be extended to the peer-to-peer environment since the failure rate of the peers is more frequent and the communication network is larger. They also have not analyzed the performance of 2PC in replicated environments.

Optimistic and Pessimistic Epidemic Algorithms (Agrawal *et al.* 1997) were proposed in 1997 to be used in environments with low probability of conflicts. This work, however, does not deal with site failures of any kind but was used as the reference for our Epidemic Algorithm simulation.

2.2 P2P Techniques

P2P systems entered the arena of distributed systems with file sharing applications like Gnutella (Gnutella), Napster (Napster) and FreeNet (Clarke *et al.* 2000). In these applications, a rendezvous point of the P2P network is known and people can join these networks by first connecting to the rendezvous point. Once part of the P2P network, the person can query the system for the desired files. These queries are transferred across the network until they reach the peer which has the file. This peer then responds back with the results of the query. In Freenet (Clarke *et al.* 2000) the files, which are the results of the query, are replicated along the return path to increase the availability for subsequent queries from same or nearby peers until it reaches the peer from which the query originated. These systems however support only file upload and download facility and do not support any transaction processing facility.

A survey of the current data management techniques in the current P2P systems (Sung *et al.*, 2005) shows the various methodologies being used in the current P2P systems. P2P overlay networks like CAN (Ratnasamy *et al.*, 2001) and Chord (Stoica *et al.*, 2003) provide key-based routing. The introduction of JXTA (JXTA 2005), an open protocol which facilitates any connected devices to communicate in a P2P manner by creating a virtual network, increased the ease of building P2P systems. A Local Relational Model was proposed (Giunchiglia *et al.*, 2002) for the coordination of P2P systems. Piazza (Haley *et al.*, 2003) proposes a method of sharing semantically heterogeneous data in a scalable way where it maintains storage mappings to associate queries with suitable relations and description mappings to associate query results between peers. Multi-Attribute Queries were supported by Multi-Attribute Addressable Network (Cai *et al.*, 2003) and the PIER system (Huebsh *et al.*, 2003) supports join queries. Multi-Attribute Range based searches are supported by Mercury (Bharambe *et al.*, 2004). But none of these handle the issue of incorporating transaction processing facility and ensuring ACID properties in the P2P systems.

Maintaining replica consistency in P2P systems is a challenging problem as there is a lack of global knowledge and low online probability. In the PAST system (Rowstron and Druschel 2001), nodes and files are assigned identifiers and replicas of a file are stored at nodes with identifiers matching closely to the file's identifier. OceanStore (Kubiatowicz *et al.* 2000) and Ivy (Muthitacharoen *et al.* 2002) rely on the underlying DHT to provide the necessary replication. Again, these systems do not provide transaction processing facility.

[Franconi *et al.* 2004] have proposed a P2PDB called DALET which uses a local update technique, where the peers update their data from the data of their neighbors. Nodes are interconnected by coordination rules which allow them to fetch data from their neighbors. Coordination rules are called incoming links if they are used by other nodes for

importing data and outgoing links if that node uses them to import data from its acquaintances. When a node gets a query request it answers it immediately using local data and forwards the query through all the outgoing links. A query request contains the sequence of IDs of the nodes it passes through and a node does not propagate a query request if its ID is present in the sequence already. This work however does not deal with transaction processing and does not present results regarding the performance of this algorithm in a transaction processing environment.

[Vecchio *et al.*, 2005] have proposed a P2PDB which uses an adapted version of the Quorum Consensus Protocol (in which each individual peer is responsible for finding a set of accessible copies of a data item) coupled with the Two-Phase Commit Protocol to take care of update. If consistency cannot be relaxed, then the updates can proceed as long as a write quorum is present. If the write quorum is lost then the transaction should abort. This algorithm thus is threatened by the recurring site failures of a P2P system. Assuming that the probability of reading old data is low when we do not enforce consistency, they propose that the individual peers choose the quorum thresholds according to the tradeoff they can sustain. They have given the performance of the network for an update and they have also presented statistics about the probability of stale data access for different quorum and data replication levels. However they have not given the performance with respect to transaction processing and the effect of this algorithm on transactions in the presence of site failures.

CHAPTER THREE

Two Phase Locking Scheduler Simulation

The first half of this chapter explains the 2PL simulation and the centralized database simulation. The experiments performed to verify the accuracy of our simulation compared to the previous research are explained in the latter half of this chapter.

3.1 Two Phase Locking Scheduler

The scheduler of a database controls the concurrent executions of transactions by ordering the reads, writes, commits and aborts of the different transactions such that the resulting execution is recoverable and serializable. Databases usually implement a well known scheduler called *Two Phase Locking* (denoted 2PL) (Bernstein *et al.* 1987).

Transactions submit read and write operations on data items to the 2PL scheduler. The 2PL scheduler grants a lock on that data item if there are no conflicting locks held on that data item. A read lock conflicts with a write lock held on that data item by another transaction. A write lock conflicts with both read and write locks held on that data item by another transaction. Once a transaction releases a lock, it cannot obtain further locks. This gives rise to the two phases of the transaction: The Growing Phase – where the transaction obtains locks and The Shrinking Phase – where the transaction releases locks. Typical databases implement a version of 2PL called *Strict Two Phase Locking*, in which all the locks of the transaction are released when the transaction commits.

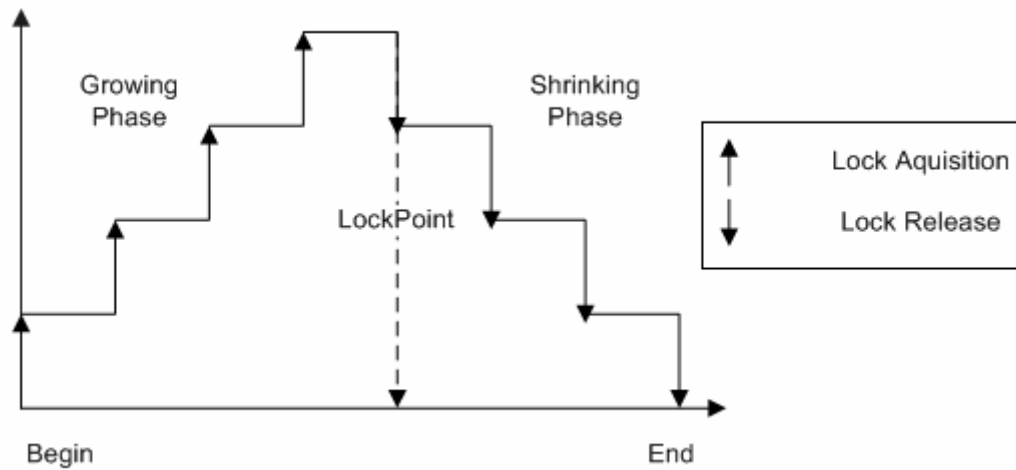


Fig. 3.1. A transaction's phases when it goes through a two phase locking scheduler

3.2 Two Phase Locking Scheduler Simulation Design and Implementation

A simulation of 2PL scheduler was built and tested on a centralized database environment against the analytical results of Alexander Thomassian(Thomassian 1993) to verify its correctness. The detailed high-level design and implementation of the simulation are discussed in this section. The database is initialized with a starting data item and the number of data items it will store, where the data items are continuous non-negative integers starting with 0. There are two major components of the scheduler – the Lock-Wait Table and the ProcessOperation Method.

3.2.1 The Lock-Wait Table

The Lock-Wait Table holds operations of different transactions that have either requested or are holding a lock on the data item. The Lock-Wait Table is implemented as an array of vectors, with a vector for each data item of the scheduler. The vectors for each data item hold all the operations (simulated as JAVA objects and explained in Section 3.3.5) that

have either obtained or are waiting for a lock on that data item. The operations that hold the lock precede the operations that are waiting for a lock in the vector in the FIFO order.

3.2.2 The ProcessOperation Method

The ProcessOperation method simulates the 2PL Protocol and ensures that the transactions abide by it. A transaction (simulated as a JAVA object and explained in Section 3.3.4) that has to perform an operation, submits the operation to the scheduler and the scheduler invokes the ProcessOperation Method on that operation. Figure 3.2 presents the algorithm followed by the ProcessOperation method.

3.3 The Centralized Database System

To verify the correctness of our Two Phase locking simulation it was integrated in a test bed of a centralized database simulation and analyzed against the results of Alexander Thomassian's analytical model (Thomassian 1993). The system model of the centralized database simulation used in our experiments is described in this section. The centralized database simulation experiments consist of the following main components.

1. Transaction Manager
2. Local Database
3. Scheduler
4. Transaction
5. Operation

Each of these is implemented as an object using the JAVA programming language. The transaction is the unit of interaction with the database, comprising of a number of operations. There is an event clock in the system. Every tick of the event clock executes the

process method of the TransactionManager. The system model of the experiment is shown in Figure 3.3.

3.3.1 Local Database

The Local Database consists of the Two-Phase Scheduler simulation. When a Local Database is initialized by the GlobalManager, it initializes the scheduler simulation with the number of data items and the starting data item of the database.

```

Case 1: If the operation received is a read / write operation
    a. Check if the operation conflicts with an operation that has
        either obtained a lock or is waiting on the data item
    b. If there is a conflicting operation, then the lock cannot be
        assigned to the operation.
        i. The status of the operation is set to blocked.
        ii. The operation is added to the vector corresponding to
            the data item
        iii. The operation is made to wait until the conflicting
            operation releases the lock.
    c. If there is no conflicting operation, then lock can be
        assigned to this operation.
        i. The status of the operation is set to locked.
Case 2: If the operation received is a commit operation
    a. All the locks of the transaction are released from the lock
        table with the status completed.
    b. For operations that were waiting on this transaction,
        i. Locks are assigned to the operations that were waiting
            on this transaction in the FCFS (First Come First
            Served) basis.
        ii. The duration of the commit operation is added to their
            waitOnCommitTime field for these operations to
            simulate the start of their execution at the end of this
            transaction's commit.
Case 3: If the operation received is an abort operation
    a. All the locks of the transaction are released from the lock
        table with the status aborted.
    b. All the operations of the transaction that are waiting for
        locks are released with the status aborted.
    c. Same as step b of case 2

```

Fig 3.2 Algorithm used by Process Operation of 2PL Scheduler

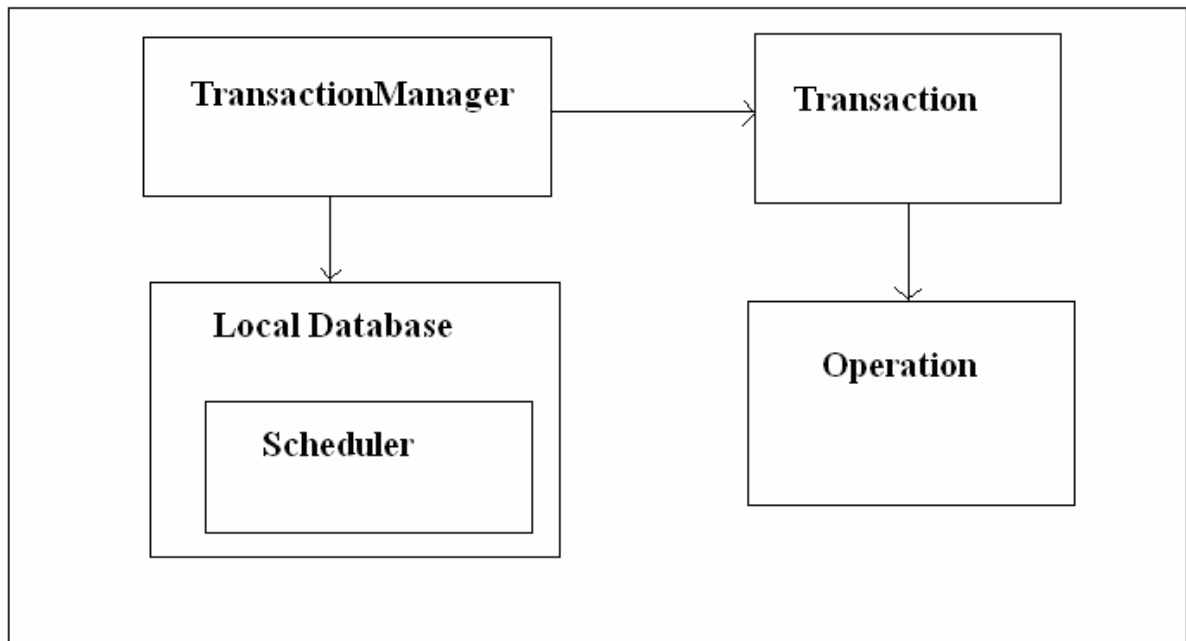


Fig. 3.3. The system model for the centralized database simulation experiments

3.3.2 Transaction Manager

The Transaction Manager is initialized with the multi programming level (number of concurrent transactions) for the experiment. It then creates transaction objects (explained in Section 3.3.4) equal to the multi-programming level and a local database object. The single most important method for the transaction manger is called *process*. The process method is called by the event clock for every clock tick. The algorithm of the process method is shown in Figure 3.4.

3.3.3 Scheduler

The Scheduler used here is the Two-Phase Scheduler simulation described earlier in Section 3.2.

3.3.4 Transaction

The Transaction object represents a transaction and is initialized with the number of operations it should generate, the probability of generating a read operation, the

1. Get an operation from the transaction for all transactions that are not blocked.
2. Submit the operation to the scheduler.
3. Get the results from the scheduler.
4. If the operation was blocked by the scheduler set the status of the transaction and the operation to **blocked**.
5. If the operation has been granted a lock by the scheduler, the operation is considered **active** until it gets completed.
6. If a commit or an abort operation was submitted,
 - a) The scheduler returns a vector of operations consisting of
 - i. the operations of other transactions which obtained locks as a result of the commit or abort of this transaction
 - ii. and the operations of this transaction that are aborted as a result of the abort.
 - b) The transaction manager goes through its list of **blocked** operations and sets the status of the commit or abort operation that was submitted and the operations that were in step 6 a) to decrementing.
 - c) If there were operations returned in the **aborted** status, then the transaction manager sets these operations to the **aborted** status.
7. The deadlock detection mechanism is now invoked.

Fig. 3.4. Algorithm used by process method of transaction manager

Transaction Manager to which it is associated and the Transaction ID. The Transaction is equipped with a `generateNextOperation` method which generates the next operation as specified by the parameters during its initialization and a commit operation after the given number of operations is generated. Alternatively the Transaction also has a `generateAbortOperation` method which is used to generate an abort operation for the transaction and reset it. After the invocation of a `generateAbortOperation`, the `generateNextOperation` is designed such that it generates the same operation sequence as in its previous instance. This simulates the restart of a transaction as the transaction will generate the same operations that it generated before.

3.3.5 Operations

Operations are the basic actions of the system. Each transaction generates a specified number of operations. The structure of the Operation is given in Table 3.1

Table 3.1: Structure of an operation

Trans ID	Opn	Data Item	Submitted Time	Done Time	Time Cntr	Status	OpnNo	Blocked Time	Wait On Commit Time	isBlocked Flag
-------------	-----	--------------	-------------------	--------------	--------------	--------	-------	-----------------	------------------------------	-------------------

TransID represents the transaction to which the operation belongs. *Opn* indicates the type of operation[r - read, w - write, c - commit, a - abort] set during the initialization of the operation. *DataItem* indicates the data item for the read and write operations. It is ignored for commit and abort operations. *SubmittedTime* indicates the time at which the operation was submitted. *DoneTime* indicates the time at which the operation was completed. *TimeCntr* is set to the time required to complete the operation. This is preset depending on the type of the operation. Status indicates the status of the operation. The various statuses are explained in Table 3.2. *OpnNo* indicates the position of this operation in the transaction's sequence of operations. *Blocked Time* gives the duration the operation was blocked, if any. *WaitOnCommitTime* represents the time for which the operation has to wait on the commit of another transaction, if any. This is assigned when the operation gets a lock based on the commit of another transaction. *isBlockedFlag* indicates whether the operation was blocked or currently is blocked.

3.4 Experiments for the Verification of the Correctness of the Scheduler Simulation

This section contains a description of the results of Alexander Thomassian analytical model (Thomassian 1993), followed by our parameter setting of the Centralized Database simulation to match with those of Alexander Thomassian's (Thomassian 1993) and the results of the experiments.

Table 3.2: Status of an operation

Status	Meaning
Before Submission	The operation has been created by the transaction and has not yet been submitted.
After Submission	The operation has been submitted to the scheduler by the TransactionManager.
Decrementing	The operation has been granted lock and is decrementing its counter which is the duration of the operation.
Completed	The operation has been completed
Lock Assigned	The operation has been granted the lock.
Aborted	The operation has been aborted
Wait on Commit	The operation is waiting on a transaction to complete its commit operation.

3.4.1 Description of Two Phase Locking and Thrashing Behavior

Alexander Thomassian(Thomassian 1993) analyzes a transaction-processing system with 2PL considering transaction steps with identical processing time distributions and determines the system performance by the fraction of blocked transactions (β). His results state that regardless of the distribution of transaction size, the system reaches its peak throughput at $\beta = 0.3$, which also holds for transactions with different per step processing times. Also, the mean transaction response time equals the ratio of the sum of the processing times for its steps (executed once) to $1 - \beta$ – the fraction of blocked transactions.

$$\text{Mean Response Time (MRT)} = r(\text{Ma})/(1 - \beta) \quad (3.1)$$

$$\text{and } r(\text{Ma}) = (k+1) s(\text{Ma})$$

Where

$s(\text{Ma})$ is the average per step processing time of a transaction

k is the number of operations

β is the fraction of Blocked Transactions.

3.4.2 Parameters

The parameters of the simulation were set so that they correspond to the parameters used in the analysis conducted by Thomassian (Thomassian 1993). They are shown in Table 3.3.

Table 3.3: The parameter settings for the centralized database simulation

Parameter	Value
No of data items	16384
Probability of read operations generated	0 (all are operations are writes)
No of operations per transaction	16
Duration of Read	1
Duration of Write	2
Duration of Commit	4
Duration of Abort	0
Multi programming level	10-160 (in steps of 10)

The duration of a read operation was derived by Ulusoy and Belford to be (Ulusoy, Belford 1992)

$$T_r = (1 - \text{mem_size}/\text{db_size}) * \text{io_time} \quad (3.2)$$

Where

mem_size is the memory size

db_size is the database size

io_time is the time taken for an i/o operation.

For the experiments performed by Ulusoy and Belford (Ulusoy, Belford 1992) the memory size is 500 and the `io_time` is 18 msec. There are 16384 data items in analysis of (Thomassian 1993) which we are comparing against. Assigning these values, the duration of read operation becomes $((1-(500/16384))*18)=17.45$ msec.

From Ulusoy's and Belford's analysis (Ulusoy, Belford 1992), the time for a write is equal to the `io_time` (18 msec). During a write operation, the transaction first reads the data item and then does a write. Thus the duration of write operation is 35.5 msec (17.5 + 18). Thus we obtain the ratio of the duration of read : write operations as 1 : 2. The duration for a commit operation is determined by a single forced write operation to the log, forced-written to indicate that the transaction has been committed (Jim Gray and Andreas Reuter 1992). The duration of the abort is ignored as there is only one log write involved and that is not a forced log write operation. Thus the ratio of the duration of read: write: commit: abort operations is 1:2:1:0. The size of the database was set to 16384. Transactions in (Thomassian 1993) generate 16 write operations each. Hence the number of operations per transaction was set to 16 and the probability of read was set to 0. Deadlocks are detected for every cycle and the youngest transaction involved in the deadlock is aborted. The multi-programming level (the number of transactions in the system at any point of time) was varied from 10 to 160 in steps of 10. 14 experiments were performed with this setting to avoid statistical anomalies.

3.4.3 Results

The results of 14 experiments are presented in this section. Throughput measurements were taken at 2, 4, 8, 16 and 32 seconds respectively. Figure 3.5 shows the variation of throughput with the Multi-Programming Level at a simulated 32 second interval. It also contains the plot of the Mean β where β is the fraction of blocked

transactions. The throughput increases steadily with the Multi-Programming Level until it reaches a maximum). Beyond that the throughput starts decreasing owing to the occurrence of thrashing. Thrashing occurs because the degree of contention is high. Transactions request locks on data items that are being held by another which in turn are waiting on other transactions.

The results of Alexander Thomassian (Thomassian 1993) say that we would get maximum throughput at $\beta = 0.3$. In the graph, we get maximum throughput at 90 where the β value is 0.35. The throughput obtained at 2, 4, 8 and 16 seconds produce similar results with the maximum throughput occurring near $\beta = 0.3$ in each case.

Figure 3.6 is taken from Alexander Thomassian (Thomassian 1993). It shows the results of their analysis.

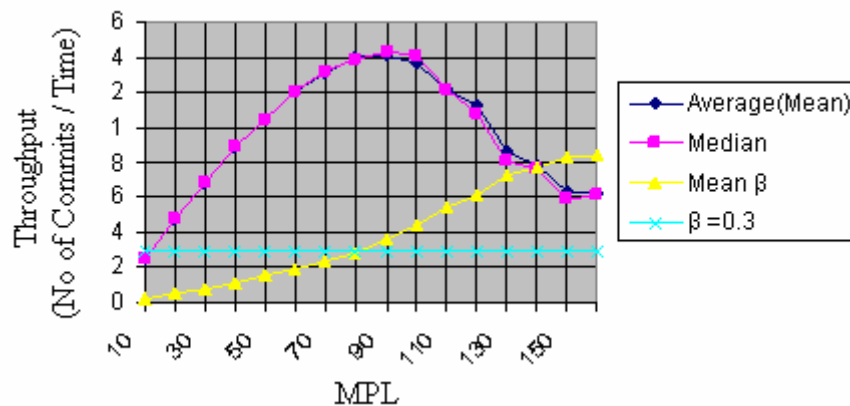


Fig. 3.5. Graph that shows the variation of throughput with multi programming level at the simulated 32 second interval

Figure 3.7 shows the results of our experiments obtained at a simulated time interval of 32 seconds. A comparison of Figures 3.6 and 3.7 shows that the results of our experiments match with that of Alexander Thomassian's (Thomassian 1993) to a great extent.

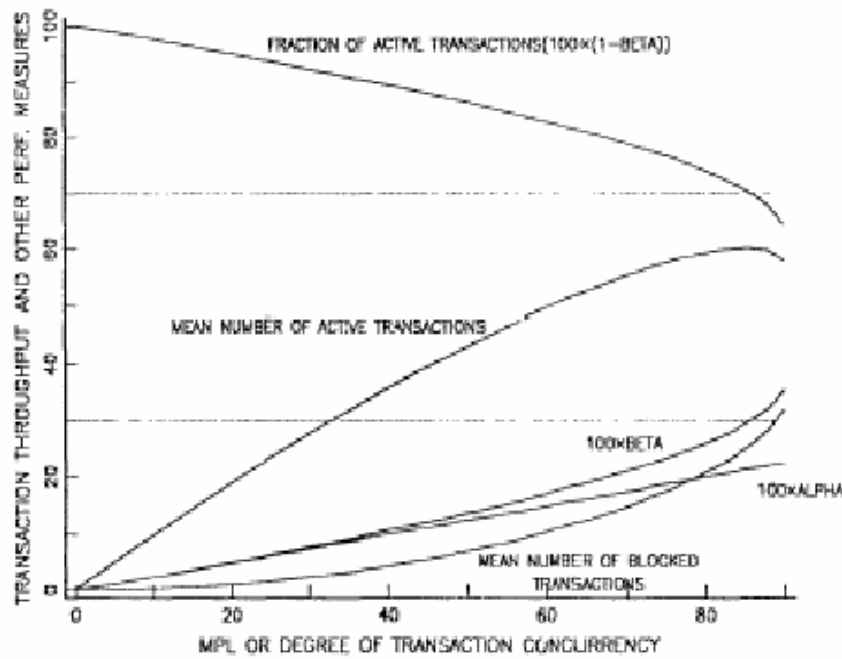


Fig. 3.6. Results of analysis of Alexander Thomassian (Thomassian 1993)

We now show the comparison between the Mean Response Time from our simulation and that obtained from the Response Time Equation from Alexander Thomassian (Thomassian 1993) for each of the experiments. For the latter, we obtain the fraction of blocked transactions from our simulation results. The formula to estimate the response time is obtained from [1] as $r(Ma)/(1-\beta)$ and $r(Ma)=(k+1) s(Ma)$ (Eqn 3.1) where k is the number of operations per transaction (16 in our case), $s(Ma)$ is the mean processing time for a transaction step (2 in our case, as all the operations are writes), β is obtained from the simulation. Figure 3.8 shows the comparison between the median estimated response time and the median experimental response time at simulated time of 32 seconds. The response times match each other closely until thrashing occurs. After thrashing, the median experimental response time starts lagging behind the median estimated response time as the formula to compute the estimated response time does not take into account the occurrence of aborts due to deadlocks.

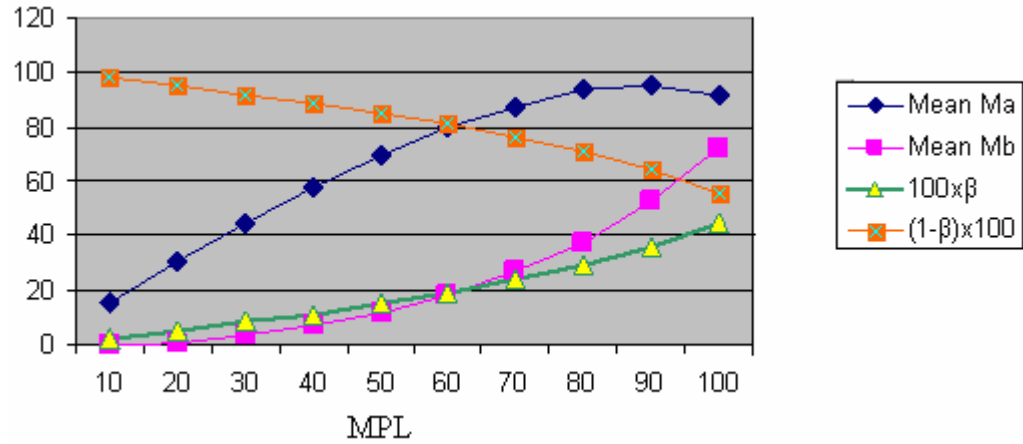


Fig. 3.7. Results from simulation at time = 32 seconds

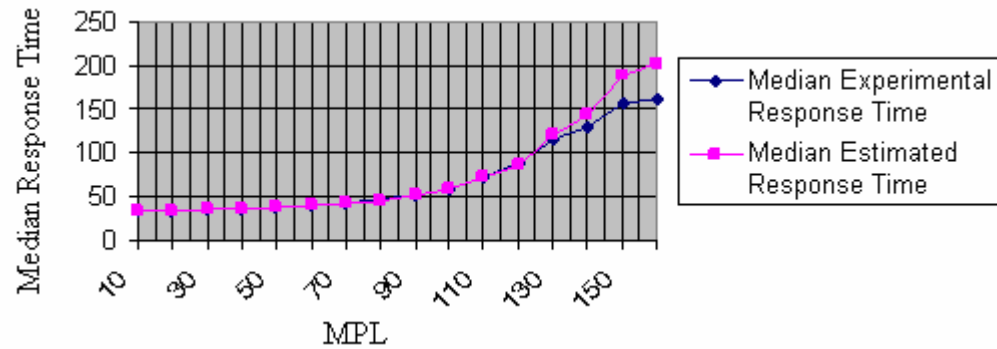


Fig. 3.8. Comparison of median experimental and estimated response times at 32 seconds

The error between the median experimental time and median estimated response time remains around 2% until they reach the point of thrashing after which they increase and almost get to 20%. Figure 3.9 captures the error between the median experimental and median estimated response times.

Thus our two-phase locking scheduler simulation matches the results of the analytical model given by Alexander Thomassian.s

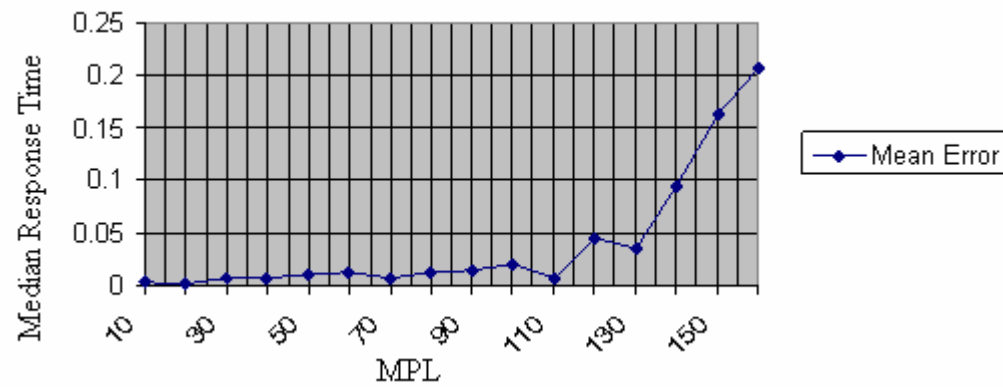


Fig. 3.9. Error between the median experimental and estimated response time at 32 secs

CHAPTER FOUR

Simulation of Traditional Distributed Database Techniques

This chapter begins with a discussion of the traditional techniques that are employed in distributed database systems (DDBMS). This is followed by a description of the simulation of our distributed database and the experiments that were performed to verify the correctness of our distributed database simulation.

4.1 Traditional Database Techniques Used in the Distributed Database Systems

In a distributed database system, the data is distributed among various sites. Current distributed database systems like PostgreSQL(Johnson, 2002) and Oracle (UMBC's Oracle Site 2005) use the traditional database techniques for maintaining concurrency control and for ensuring the ACID properties.

4.1.1 The Two Phase Locking Scheduler

The current DDBMS implement the 2PL Scheduler explained in the previous chapter, at each of their sites, for maintaining concurrency control.

4.1.2 Read One Write All Mechanism

In an environment when the data is replicated in several sites of the system, DDBMS like PostgreSQL (Johnson, 2002) and Oracle simultaneously (DBASupport's Oracle Replication Site 2005) employ the read one write all mechanism (ROWA) to ensure consistency of data across all the sites. As the name implies, when a transaction wants to read the data it does so from one site, preferably the one closest to the transaction and when

a transaction wants to write the data, it performs a write at all the database sites that contain the data. This ensures that data at all sites are consistent.

The ROWA mechanism coupled with the 2PL ensures the consistency of the transactions in the system. When a transaction does a “Write All“ on a data item, any transaction performing a read on the data item will wait on the writing transaction, as the schedulers at all the sites now have the write lock on the data item. The write lock will be released only after the transaction commits or aborts. Thus the concurrency control of all the transactions in the system is ensured. A transaction, while performing a write operation, has to wait until the write locks on all the replicas have been obtained.

There are two different ways of implementing the ROWA mechanism. Typical distributed databases implement a version where a transaction obtains locks on the data item at every site. Then the transaction performs the write in all of them. The commercial database Oracle implements a similar mechanism where in it uses triggers at all the sites containing the replicated data and whenever one is updated, the triggers update every other replicated version simultaneously (DBASupport’s Oracle Replication Site 2005). Alternatively, databases like PostgreSQL (Johnson 2002), implement a variation where the write operations of a transaction are transferred to the sites other than the coordinator during the first phase of two-phase commit. In our simulation we model the first version where the write locks at all the sites containing a replica are obtained at the time of performing the operation.

4.1.3 The Two Phase Commit Protocol

To ensure global atomicity the distributed database systems use a well known protocol called 2PC (Two Phase Commit Protocol) (Bernstein *et al.*, 1987).

Every transaction has a site called the Coordinator, which acts as its coordinator for the Two Phase Commit protocol. All other sites at which the transaction has done either a read or a write operation are known as the “Participant” sites. As the name indicates the protocol has two phases – a voting phase and a decision phase. The voting phases and decision phases of the coordinator and the participant are alternately presented below.

The coordinator and the participant both start in an Initial State. During the first phase the coordinator sends a “Prepare” message to all the participants, writes a “Prepared” record to the log and goes into the Prepared State for that transaction.

When a participant receives a “Prepare” message, it checks if the transaction can be committed at its site. If so the participant responds to the coordinator with a “Vote Yes” message, force writes the transaction logs to stable storage, writes a “Vote Yes” record in its log and goes into the Voted Yes State for that transaction. If the participant cannot commit the transaction, it responds to the coordinator with a “Vote No”, writes a “Vote No” record in its log and goes into the Voted No State for that transaction.

Whenever a coordinator receives a “Vote Yes” message it updates the list of transactions that have replied with a “Vote Yes” message. It checks if all the participants have replied with a “Vote Yes” message. If so, the coordinator decides to commit the transaction, writes a “Commit” record to the log, force-writes it to stable storage, sends a “Commit” message to all the participants and goes to Committed State for that transaction. It then waits for the acknowledgement from the participants within a time out period. On the contrary if the coordinator receives a “Vote No” message, it decides to abort the transaction, writes a “Abort” record to the log, aborts the transaction, sends an “Abort” message to all the participants and goes to Aborted State for that transaction. It then waits for the acknowledgement from the participants within a time out period. Once all the acknowledgements have been received, it writes an “End Transaction” record in the log. If it

does not receive the acknowledgement from a site within the time out period, it resends the global decision and resets the time out period.

When a participant receives a “Commit” message from the coordinator, it writes a “Commit” record to the log and force-writes it to stable storage. It then commits the transaction and sends an acknowledgement to the coordinator. On the contrary if it receives an “Abort” message from the coordinator, it writes a “Abort” record to the log and force-writes it to stable storage. It aborts the transaction and then sends an acknowledgement to the coordinator. Figure 4.1 shows the two phase commit protocol.

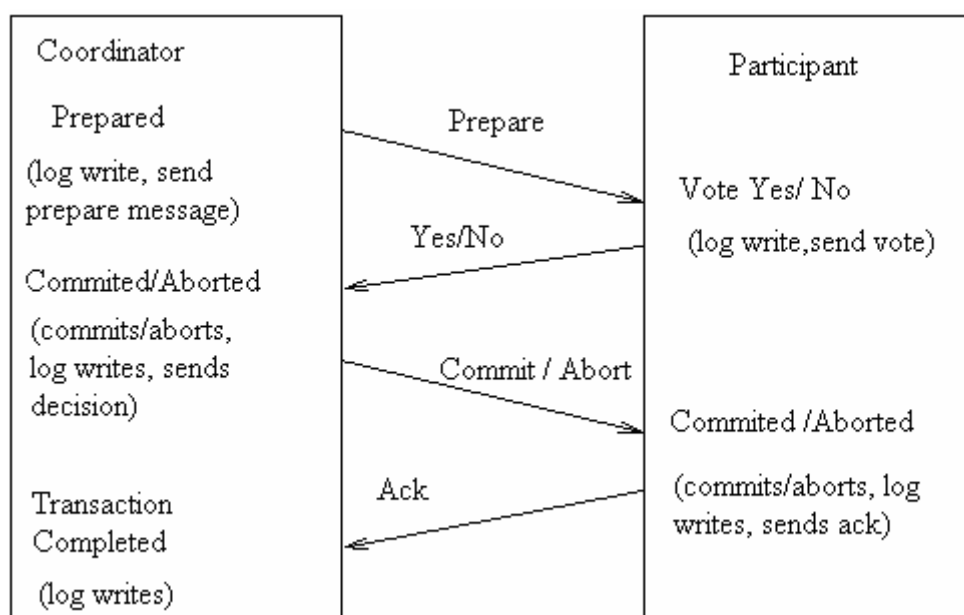


Fig. 4.1. Two Phase Commit protocol

The coordinator or the participants can timeout in any of the states, due to a site failure of the other or a delay in communication between the two. When a time out occurs the following termination protocol is invoked.

The coordinators can time out in either the Prepared State or in the Committed/Aborted State. If the coordinator times out in the Prepared State while waiting for the vote from one or more of the participants, it decides to abort the transaction. If the

coordinator times out in the Committed/Aborted state while waiting for the acknowledgement of a transaction, it resends the decision to the sites which have not acknowledged.

The participants can timeout in the Initial State before receiving a Prepare from the coordinator or in the Voted Yes/No State. If a participant times out in the Initial State before receiving the Prepare from the coordinator, the participant believes that the coordinator must have failed and hence unilaterally aborts the transaction. If it receives a Prepare message later from the coordinator (if the Prepare message was delayed by the traffic in the network, and the participant timed out and aborted) then the participant responds with a Vote No message. If a participant times out in the Vote Yes/No state then participant cannot make a decision on its own as it has already sent the vote. The participant is blocked. Under a variant of 2PC (Connolly and Begg, 2002) it could contact each of the other participants and find any decision that may have been sent to them. If the coordinator failed before sending the decision to any of the participants, the participant remains blocked until the coordinator awakens and sends the decision. This is very critical in the peer-to-peer database environment because peers may not recover.

We now have to consider the action taken by a failed site when it recovers. The coordinator can fail in the Initial State when it has not sent a “Prepare” message, in the Prepared State and in the Committed/Aborted State. If the coordinator fails in the Initial State before sending the “Prepare” message, recovery of the coordinator just restarts the commit procedure. If the coordinator fails in the Prepared State and its log indicates that it has not received “Abort” messages from any of the participants, then the recovery of the coordinator restarts the commit procedure. If the coordinator fails in the Committed/Aborted State and the logs indicate that not all the participants have responded with an

acknowledgement, recovery of the coordinator resends the decision to the sites which have not sent the decision prompting them to send it again.

A participant can fail in the Initial State before receiving a “Prepare” message from the coordinator, or in the Voted Yes/No state or in the Committed/Aborted State. Recovery of the participant in the Initial State before receiving the “Prepare” message from the coordinator prompts it to abort the transaction. Recovery of the participant in the Voted Yes/No state prompts it to resend the decision to the coordinator. Recovery of the participant in the Committed/Aborted state does not necessitate any action on its side, as it has completed the transaction.

4.2 Distributed Database Simulation

The detailed high-level design and implementation of the distributed database simulation are discussed in this section. The simulation contains the following components:

1. Global Manager
2. TransactionManager
3. Communication Manager
4. LocalDatabase
5. Scheduler
6. Transaction
7. Operations

Each of these is implemented as an object using the JAVA programming language. Figure 4.2 shows the system model of the distributed database simulation. A site of a distributed database system consists of a Transaction Manager and a Local Database which contains a Scheduler. It differs from the site of the centralized database system simulation explained in the Chapter Two only in that the Transaction Manager is modified to

accommodate 2PC and ROWA. This system is also driven by an event clock which calls the GlobalManager's process method for every clock tick.

4.2.1 Global Manager

As the name indicates the Global Manager is responsible for the execution of all transactions at the appropriate sites. The GlobalManager interacts with the transactions in the system, receives their operations and ensures that they take place at the right sites by sending them to the appropriate Transaction Manager via the Communication Manager. Furthermore it decides the coordinator site for a transaction manager and implements the ROWA protocol. It is also equipped with a distributed deadlock detection mechanism, which is called at fixed intervals. If a deadlock is detected then the youngest deadlocked transaction is aborted. At every clock tick, the GlobalManager has a method called *Process*, which is executed by the event clock. Figure 4.3 gives a description of the *Process* method.

Apart from the Process Method, the GlobalManager has a receiveOperation method which is called by the CommunicationManager whenever it wants to send an operation to the GlobalManger for reporting the results of an operation. Algorithm 4.2 shown in Figure 4.4 describes the receiveOperation method of the GlobalManager.

The GlobalManager with the help of these two methods implements the ROWA protocol and also ensures that the operations of various transactions take place at appropriate sites.

4.2.2 Transaction Manager

The Transaction Manager is responsible for the execution of the operations at the site, ensuring atomicity of the transactions by implementing the 2PC protocol. It also maintains a TTF (Time To Failure) and a TTR (Time To Recover) for the particular site

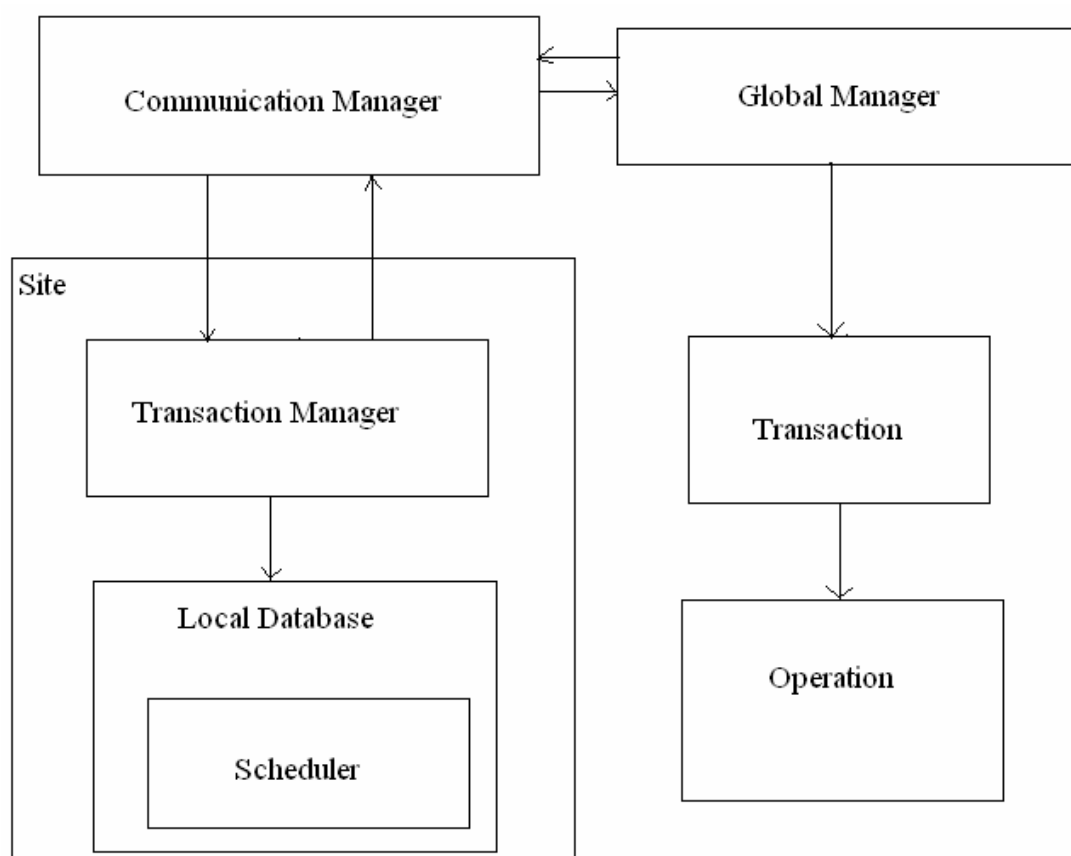


Fig. 4.2. The System Model of the distributed database Simulation

For all transactions that are not blocked

1. Get an operation.
2. If the operation is a read, commit or abort the GlobalManager
 - a. Sets the communication time.
 - b. Sets the TransactionManagerID to the Transaction Manager ID of the Coordinator site of the transaction.
 - c. Enqueues the operation in the CommunicationManager's queue.
 - d. Wait for this operation to complete or abort before obtaining another operation from this transaction.
3. If the operation is a write
 - a. Submit an operation to each of the sites containing a copy of the data item in the manner described in step 2.
 - b. Wait for all the operations at all the sites to be completed before obtaining another operation from this transaction.

At a preset deadlock detection time interval, construct a waits-for-graph and abort the youngest transaction by sending an abort operation to all the sites where the transaction took place.

Fig. 4.3. The algorithm of the process method of global manager

assigned during its instantiation. The event clock decrements the failure time counter of each of the Transaction Managers for each of its ticks and as soon as the counter hits the TTF the Transaction Manager makes the site unavailable for a period of time equal to the TTR thus simulating site failure. After TTR has passed it executes the `recoverFromFailure` method and then resets its TTF to the preset value and the cycle continues. The TransactionManager has a `processOperationOrMessage` method which is called by the CommunicationManager every time an operation has to be sent to this transaction manager. Figure 4.5 shows Algorithm 4.3 which describes the `processOperationOrMessage` method.

1. If the parameter is a read/write operation.
 - a. In the GlobalManager's list of submitted operations find the matching operation and assign the status of the passed parameter operation (aborted or completed) to it.
 - b. The transaction will be considered ***blocked*** if a response operation is not received for all of its submitted operations.
2. If the parameter operation is a commit or abort, the transaction is considered committed or aborted appropriately and the transaction is no longer considered to ***blocked***.

Fig. 4.4. `receiveOperation` method of the global manager, parameter: operation

The log writes of 2PC are simulated with log write operations which are assigned a duration equal to the I/O time. 2PC is implemented as described in the Section 4.1.3. Commits, aborts, reads and writes are performed as in Chapter Three by submitting these operations to the scheduler and processing the resultant locks. The Transaction Manager is also equipped with functions to take care of the recovery from failure by implementing the Recovery protocol of the 2PC. It is also equipped with functions that take care of the time outs of the transactions by implementing the Timeout protocol of the 2PC.

1. If the site is not available due to failure, ignore the operation.
2. If the site is available
 - a. If the parameter is a read or a write operation.
 - i. Submits the operation to the scheduler.
 - ii. Process the results from the scheduler.
 - iii. If the operation is not blocked simulate its execution by decrementing it for the duration equal to the duration of the operation
 - iv. If simulated execution of the operation is completed send the operation in **completed** status to the GlobalManager via the CommunicationManager.
 - v. If the operation did not obtain a lock, it sets the operation and thereby the transaction to the **blocked** status.
 - vi. If it is a commit operation, Transaction Manager becomes the coordinator of the transaction and starts the 2PC Protocol.
 - b. If the parameter is an abort operation
 - i. Submits it to the scheduler and processes all the newly unblocked operations of other transactions as a result of the abort.
 - ii. Send the abort operation in a **completed** status to the GlobalManager via the CommunicationManager.
 - c. If the parameter is a message of the Two Phase Commit Protocol then the TransactionManger responds as discussed in Section 4.13.

Fig. 4.5. processOperationOrMessge method of transaction manager

4.2.3 Communication Manager

The Communication Manager mimics the network existing between the sites of the distributed database. When the CommunicationManager receives an operation/message from either the GlobalManager or the TransactionManger, it puts them in a queue. The event clock calls a method of the CommunicationManager called handleOperations which goes through the operations in the queue and decrements their communication counter. If the communication counter becomes zero it delivers the operation to the respective TransactionManager or the GlobalManager by calling their processOperationOrMessage function or the receiveOperation function respectively. This simulates the delivery of the operation or message through the network.

The Local Database, Scheduler and Transaction are identical to that used in the Centralized Database Simulation detailed in Chapter Three.

4.2.4 Operations

The Operations class used here is identical to the one used in the Centralized Database detailed in Chapter Three, Section 3.3.5 except that it also serves as a message in the 2PC and has a communication time parameter called `commTime` which is set before the operation is placed in the `CommunicationManager`'s queue for the purpose of communication.

There is a flag called *isMessage* which when set indicates that this is a message of the 2PC. If it is a message, then the text of the message can be obtained from the parameter *message*.

4.3 Verification of the Correctness of our Distributed Database Simulation

The correctness of the distributed database simulation was verified running the simulation with the parameters of Carrey and Livny's experiments (Carrey and Livny 1991). This section contains a brief description of their simulation and experiments, followed by a description of the changes to our system to coincide with their experiment settings and the results we obtained from their simulation.

4.3.1 Description of the Simulation in (Carrey and Livny 1991)

4.3.1.1 Transaction. A transaction has a coordinator process that runs at the site where the transaction originated. The coordinator in turn starts a collection of cohort processes to perform the transaction processing. There is at least one cohort for every site the transaction accesses data from. The average length of the transactions is 22.5 operations

(18 reads + 4.5 writes). They restrict the write operations to the data items that have been already read.

In the case of replication, a cohort updating a remote data item has one or more remote update processes associated with it at the sites containing the data item. The cohort communicates with these remote update processes for concurrency control.

4.3.1.2 Two-phase commit protocol. The 2PC protocol resembles in every detail the 2PC protocol described in the first half of this chapter, except in the way the log writes are done. The coordinator does not perform a log write for the prepared state. Usually the log writes are done by appending the log record to the tail of the log. The log records of the site are flushed only during the log writes of commit or abort.

4.3.1.3 Database model. The distributed database is modeled as a collection of files at page level.

4.3.1.4 Site. Each site in the model has the following components:

- Source
- Transaction manager
- Concurrency Control Manager
- Resource Manager
- Network Manager

4.3.1.5 Source. The source is responsible for generating the workload for a site. It generates the transaction according to the parameters that are preset.

4.3.1.6 Transaction manager. A transaction manager is initiated by the source with the set of files that it will handle. The coordinator is created at the originating site, which

starts the cohorts. Each cohort executes the operations assigned to it. A read operation consists of a concurrency request, a disk I/O and a period of CPU processing time. Write requests are almost similar involving a log record append to the tail of the log. Whenever a transaction commits, the writes it performed are read and a separate disk I/O is done for each. If the transaction has to be aborted, the transaction manager aborts the transaction, and delays the transaction for a period of time (one average transaction response time) before restarting it.

4.3.1.7 Resource manager. The resource manager manages the physical resources of the site like CPU and the disks. Disk access times are uniformly taken from [MinDisk Time, MaxDisk Time]. Disk writes are given priority over disk reads.

4.3.1.8 Network manager. The network manager models the communication network. The network model acts as a switch which routes messages between sites as the experiments assume a local-area network, where the actual time on the wire for messages is negligible. The main cost of sending a message thus is the CPU processing cost at the sender and receiver.

4.3.1.9 Concurrency control manager. The concurrency control manager ensures that the transactions abide by 2PL.

The parameters and their values are as presented in Table 4.1

Table 4.1 The parameter settings for the experiments of (Carrey and Livny 1991)

Parameter	Description	Value
NumSites	Total No of Sites	8
NumFiles	Total No of Files	8
FileSize	Size of each file	2400
Num Terminals	Each site has a set of terminals. Each transaction emanates from a terminal.	50 per site
ThinkTime	Time between two transactions	0-10s
Write Probability	Probability of Write	$\frac{1}{4}$
Page CPU	Average CPU Time for Processing a Page	8ms
Num Disks	Total Number of Disks Per Site	2 per site
Min Disk Time	Minimum Disk Access Time	10
Max Disk Time	Maximum Disk Access Time	30
InitWriteCPU	Time to initiate a disk write	2ms
MsgCPUTime	Message send/receive time	1 ms
LogDiskTime	Sequential log write time	10 ms
LogPageSize	Number of log records per page before being flushed	100
Hit Rate	(They have modeled 0% and 80 %). We have taken only the experiments with 0	0
Deadlock Detection Interval	Interval at which the distributed deadlock detection is done	1s

4.3.2 Modeling our Simulation to match the Parameters of the Experiments of Carrey and Livny (Carrey and Livny 1991)

The multiprogramming level was made 400 and the number of data items was made 19200. In Carrey and Livny's experiments (Carrey and Livny 1991) the transactions read 18 data items and write 4 or 5 of those 18 read data items. Hence the transaction class is modified so that half the transactions generate 18 operations out of which four are write and the other half generate 19 operations out of which 5 are write (since a write operation reads and writes the data item). The total time of a read operation was computed to be 28 ms (20

Avg CPU Time + 8 Page CPU). The write operation was computed to be 60 ms (Avg Disk time (for read) 20 ms + Page CPU (8 ms)+ Init Write CPU (2ms)+ Avg Disk time for write (20 ms)+ Log Write Time as the writes are performed during the commit(10 ms)). Also local deadlock detection at individual sites was done every time an operation is blocked and distributed deadlock detection is done every 1000 ms as in (Carrey and Livny 1991). The communication time for remote node is set to 1ms. When transactions are aborted they are restarted after a period equal to the mean transaction response time at the site. The parameters and their values are presented in Table 4.2

Table 4.2 The parameter settings for our distributed database simulation

Parameter	Value
Communication time	1ms
Read Time	28ms
Write Time	60ms
Log Write	10ms
DeadLock Detection Interval	1000ms
MultiProgrammingLevel	400
Reads&Writes per transaction	200 transactions generate 14r+4w and 200 transactions generate 14r+5w

4.3.3 Differences Between our Simulation and (Carrey and Livny 1991)

- In Carrey and Livny's experiments, the resource manager always append the log writes to the log tail and flush the log (which takes 10ms) when the page size is 100 or during the commit log writes. In our simulation we do not simulate the paging of log writes and every log write takes 10ms.
- In Carrey and Livny's experiments they do not write the log record for prepare log during the coordinator's first phase of the two phase commit.

- In Carrey and Livny's experiments the writes of a transaction take place during the commit, whereas in our case they take place when the operation is submitted. Thus we perform the writes for the transactions that are aborted and in Carrey and Livny's experiments (Carrey and Livny 1991) they do not.
- In Carrey and Livny's experiments (Carrey and Livny 1991) there are two disks per site whereas we have only one.

4.3.4 Results of the Experiments

The experiments of Carrey and Livny (Carrey and Livny 1991) have a throughput of about 13. Eight experiments were conducted using our distributed database simulation and the average throughput is as shown below in Table 4.3. The percent error with the experiments of Carrey and Livny's is also shown.

Table 4.3 Results of our distributed database simulation

Time (secs)	Median Throughput	Percents Error
2	11.0	15.4
4	12.8	1.9
8	11.4	12.0
16	11.4	12.0
32	14.2	8.9
64	13.7	5.1
128	13.5	3.5
256	13.6	4.6

The slight variation in the results is because of the differences in the way the simulations work.

CHAPTER FIVE

Simulation of Epidemic Algorithms

The first half of this chapter gives a brief description of the optimistic and the pessimistic protocols of Epidemic Algorithms followed by the changes incorporated in our distributed database simulation to implement it. The last half of this chapter presents the test cases that serve as verification of our simulation.

5.1 Epidemic Algorithms

The Epidemic Algorithms under consideration were proposed in 1997 for environments with low probability of conflict (Agrawal *et al.*, 1997). Two algorithms were proposed – a pessimistic protocol and an optimistic protocol. Detailed explanations of the optimistic protocol and the pessimistic protocol are found in this section.

5.1.1 Optimistic Protocol

The philosophy behind the epidemic algorithms is to execute the update operations of a transaction at a single site. The sites communicate at a later point of time to exchange the up-to-date information. The user does not have to wait for this later communication. Since the updates pass from system to system like an infectious disease it is called an “epidemic” algorithm. Epidemic Algorithms satisfy a level of consistency weaker than serializability. In the optimistic protocol the transactions commit as soon as they terminate locally and inconsistencies are detected and resolved as the transactions pass through the system.

Each site S_j of the distributed database system maintains an event log. Each record of the event log represents a transaction with its read and write sets, the time it occurred and also has a flag to indicate whether it is in conflict with another transaction. Figure 5.1 gives a description of the transaction record as obtained by (Agrawal *et al.*, 1997). For the optimistic algorithm the aborted flag is replaced by inconflit flag and there is a set called readFrom which holds the set of transactions that this transaction has read from.

```

type Transaction =
  record
    RS      :   setof DataObjectType;
    WS      :   setof DataObjectType;
    values   :   setof DataType;
    site     :   SiteId;
    time     :   array [1..n] of TimeType;
    aborted  :   flag;
  end

```

Fig. 5.1. Transaction Record as perceived during the epidemic algorithms proposal

The parameter *RS* defines the readset of the transaction (the dataitems that were read by the transaction) and the parameter *WS* defines the writeSet of the transaction (the dataitems that were written by the transaction). The *values* parameter represents the new values written by the transaction. The parameter *site* represents the site where the transaction originated. The parameter *time* defines the timestamp of the transaction. The *aborted* flag indicates whether the transaction was aborted or not. It is replaced by an *inconflit* flag in the case of Optimistic Epidemic Algorithm to indicate whether the transaction is marked inconflit or not.

Each site S_i keeps a two dimensional timetable T_i which corresponds to the logical clocks of all sites such that if $T_i[k,j]=v$ then S_i knows that S_k has received records of all events at S_j up to time v . Thus, the time-table can be used to define the HasRecvd predicate

corresponding to some event. This is called the time table property. If t is a transaction record,

$$\text{HasRecvd}(T_i, t, S_k) = T_i[k, \text{Site}(t)] > \text{Time}(t) \quad (5.1)$$

According to this, the k_{th} row of T_i is the knowledge of S_i about S_k 's knowledge of events in the system. When S_i does an epidemic communication to S_k it includes all records t such that $\text{HasRecvd}(T_i, t, S_k)$ is false, and it also includes its timetable T_i . When S_i receives an epidemic communication from S_k it applies the updates of all received log records and updates its time-table in an atomic step to reflect the new information received from S_k .

When a transaction takes place at Site S_i , the optimistic epidemic algorithm commits a transaction locally *with the assumption* that no conflict will arise as the commit record of this transaction disseminates through the network. Epidemic communication takes place at a prefixed interval, and the Site S_i checks its timetable and sends to the other sites, all transaction records have not been received by these sites. It also sends its time table to these sites so that they can update their respective time tables with the knowledge of site S_i . Upon receiving a transaction record, a site checks if the transaction conflicts with another transaction in the system. If so, it sets the inconflict flag for the transaction record and updates its timestamp. If the transaction does not conflict, the write operations of this transaction are performed in this site as forced writes. Any other transaction holding locks of data items that are required by these forced writes are aborted unless it is performing a forced write too, in which case this transaction is made to wait. Also when a transaction is marked as inconflict, all the transactions which read from this transaction are also marked as inconflict. These conflicts are resolved by application specific rules.

5.1.2 Pessimistic Protocol

In the case of pessimistic protocol, a transaction is considered committed only if it commits at all sites without any conflict. When a transaction commits on a local site, the site inserts a *pre-commit* record in the log and this is sent to the other sites during epidemic communication. On receiving this pre-commit record, a site checks its knowledge about the transactions in a system, to see if there is a conflicting transaction. If there is no conflicting transaction then a pre-commit record is placed in its log. If there is a conflicting transaction an abort record is placed in the log. The transaction is committed only if all the sites have placed a pre-commit record in their logs for this transaction. It is proven that only the initial precommit (Agrawal et al, 1997) record is necessary because the pre-commits at other sites can be detected when their timetables are propagated and the aborts are detected when the conflicting transactions are propagated to this site via epidemic communication

5.2 Simulation of Epidemic Algorithms

The detailed high-level design and implementation of the epidemic algorithms are discussed in this section. The simulation has the same components as the distributed database simulation implementing 2PC and ROWA discussed in the previous chapter but modified to accommodate the optimistic epidemic protocol. Additionally the simulation contains a TransactionRecord class which is used to represent a transaction record in the event log and a TimeTable class representing the timetable maintained by each site. A description of each of the components is found below.

5.2.1 Global Manager

The GlobalManager interacts with the transactions in the system, receives their operations and ensures that they take place at the right sites by sending them to the

appropriate Transaction Manager via the Communication Manager. The process method of the GlobalManager is modified to accommodate the implementation of epidemic algorithms.

Algorithm 5.1 shown in Figure 5.2 describes the process method.

1. For each unblocked transaction
 - a. Get an operation
 - b. Submit the operation to the site the transaction is associated by,
 - i. Setting the communication time to zero
 - ii. Setting the receiving Transaction Manager ID to the respective site's
 - iii. Queuing it in the Communication Manager
 - c. The next operation from this is submitted only after this operation is completed.
2. If the preset deadlock detection interval has been reached computing a waits-for-graph and if a deadlock is detected the youngest transaction is aborted.

Fig. 5.2. Process Method of globalManager for Epidemic Algorithms

Apart from the Process Method, the GlobalManager has a receiveOperation method which is identical to the one shown in Figure 4.2 in the previous chapter. A transaction is said to be *partially committed* if it hasn't been committed at all sites. A transaction is said to be *fully committed* if it is committed at all the sites. A transaction is considered to be *active* until it has been processed at all sites. The GlobalManager also maintains the statistics of the partially and fully completed transactions of the system.

5.2.2 Transaction Manager

The Transaction Manager is responsible for the execution of the operations at the site and also maintains a TTF (Time To Failure) and a TTR (Time To Recover) for the particular site assigned during its instantiation similar to that in the Chapter Four. Apart from this, the Transaction Manager is responsible for performing the optimistic protocol of the epidemic algorithms, which it does by maintaining a timetable and an event log.

The *process* method of the Transaction Manager is responsible for performing the operations at the site. It is identical to that present in the algorithm shown in Figure 3.3.

Every time a transaction is committed or aborted, a log record for that transaction (TransactionRecord object) is inserted in the event log of the transaction manager and the entry corresponding to its own knowledge about its time table is updated. The method *doEpidemicCommunication* is called at a interval which is fixed for the simulation. The *doEpidemicCommunication* checks the current site's knowledge of other sites in the system about the events of the current site and sends the transaction records that it believes were not received by the other sites. It also sends its timetable to each of the sites.

To aid in the reception of messages during the epidemic communication each transaction manager is equipped with two methods – *processTransactionRecord* and *processTimeTable*. The *processTransactionRecord* is called every time a TransactionRecord is received by the transaction manager. Figure 5.3 shows a description of the *processTransactionRecord* Algorithm.

1. Check if the TransactionRecord is already marked "inconflict".
 - a. If so recursively mark all the transactions that read from it or are in conflict with it as "inconflict".
 - b. If it is not already marked "inconflict", check if it conflicts with other transactions
 - i. If it conflicts with another transaction mark both of them as "inconflict". Then recursively mark the transaction that read from these as "inconflict".
 - ii. If it does not conflict with any of the transactions, perform force writes for all the transaction's write operations and then add a log record to the event log.

Fig. 5.3. *processTransactionRecord* of Transaction Manager

When a force write to a data item is done, if there are transactions which are currently holding locks on data-items that are being forced written, then those transactions

are aborted and the force write is given more priority. If there are other forced writes on the same data item, then this forced write is put in a queue. The processTimeTable method is called whenever a timetable is received by this site. This updates the sender site's knowledge in the time table of this transaction manager from the received timetable.

5.2.3 *Communication Manager*

The Communication Manager is similar to that used in the Distributed Database with 2PC and ROWA simulation explained in the previous chapter, except that it can also transfer timetables and transaction records.

5.2.4 *TimeTable*

Every TransactionManager instantiates an object of this type to maintain the timetable at that site. The timetable has a two dimensional array to help store each of the sites' knowledge about the other sites in the system. It also has a fromSite, toSite and communicationCounter to help in the epidemic communication. The value at (i,j) in the timetable represents Site S_i 's knowledge of the events that have taken place in Site S_j . It is a time value t which indicates that the Site S_i knows the events that took place at Site S_j before time t .

5.2.5 *Transaction Record*

This represents a transaction and is the central object of an event log. When a transaction is committed, aborted or found to be in conflict, a TransactionRecord object is created for that transaction and is inserted in the event log. A TransactionRecord contains the following information: TransactionID, startTime, endTime, siteID (where the transaction began), readSet, writeSet and readFromSet (set of transactions from which this transaction

read), inConflictFlag, fromSite, toSite and communicationTime (the last three parameters are used only for epidemic communication).

An event clock drives the system. For every tick of the event clock the process method of the GlobalManager is called and operations of transactions are submitted to the various sites. The corresponding transaction managers and the schedulers perform the operations. The communication manager transfers the timetables, operations and transaction records that are queued in it. If the epidemic communication interval is reached at any of the sites, epidemic communication is performed by that site. Whenever a TransactionManager receives a TransactionRecord it performs the process outlined in Figure 5.3. Whenever it receives a TimeTable it updates its own.

5.2.6 Other Components

The Operations class is identical to the one in the Distributed Database detailed in Chapter Four, except that the transaction Id includes the start time of the transaction. The Local Database, Scheduler and Transaction are similar to that used in the Distributed Database Simulation detailed in Chapter Four, except that the Transaction object has a starttime in order to uniquely identify the transaction, and an endtime to help detect conflicts.

The communication model is changed to reflect network behavior for heavy traffic. The transaction managers are equipped with a communication queue. Whenever an Operation or TransactionRecord has to be sent, it is queued in the communication queue. The simulation allows the network bandwidth to be set as a parameter and the communication queue is processed depending on the bandwidth.

This simulation counts both the partial commits and full commits. The full commits represent the transactions that are committed at all sites. The partial commits represent the

transactions that are not committed at all sites and these transactions are resolved by application specific rules. Thus the fully committed transactions alone represent the performance of the Pessimistic Epidemic Algorithm. The fully committed and the partially committed transactions together represent the performance of the Optimistic Epidemic Algorithm.

5.3 Verification of our simulation of Epidemic Algorithms

The verification of our Epidemic Algorithms simulation was done by testing as no published performance measurements exist. The test cases used for the verification of epidemic algorithms are provided in this section. The parameters used for testing are shown in the Table 5.1

Table 5.1 The parameter settings for the epidemic algorithms s imulation testing

Parameter	Value
Number of Transactions	2 or 3
Number of Sites	2 or 3
Operations per Transaction	2
Read Time	28ms
Write Time	60ms
Commit Time	10ms
Abort Time	0ms
LogWriteTime	10ms
Epidemic Communication Time	350ms
Network Bandwidth	1Mbps
CommunicationTime	75ms

The duration for read, write, commit and abort are fixed to be the same as 2PC.

5.3.1 Verification Experiments

The test cases used for verification are given below. Every Transaction is identified by the tuple (a,b) where a denotes the transaction number and b denotes the start time. Every operation is identified by tuple (c,d) where c denotes the data item and d denotes the type of

operation (r-read, w-write, c-commit, a-abort). For commit and abort operations c can be anything and is represented as *. The time is measured in milliseconds.

Case 1: Transactions executing at different sites are not found to conflict during the process of epidemic communication and are committed everywhere

This case depicts the execution of non-conflicting transactions being committed at the sites they are propagated to during the process of epidemic communication. The table 5.2 represents the execution of two such transactions in our system.

Table 5.2 Test case 1 of epidemic algorithms simulation

At Site 0	At Site 1	At Site 2
Tran(0,0) submits operation (0,w) at time 0	Tran(1,0) submits operation (1,w) at time 0	
Tran(0,0) submits operation (2,w) at time 60	Tran(1,0) submits operation (3,w) at time 60	
Tran(0,0) submits operation (*,c) at time 121	Tran(1,0) submits operation (*,c) at time 121	
Epidemic Communication Take place at time 350		
Tran(1,0) is received and since it does not conflict the write operations are done as forced writes. The operations are done at 426 and 486 respectively	Tran(0,0) is received and since it does not conflict the write operations are done as forced writes. The operations are done at 426 and 486 respectively	Tran(0,0) is received and since it does not conflict, the write operations are done as forced writes. Tran(1,0) is received and since it does not conflict, the write operations are done as forced writes. The operations are done at 426 and 486 respectively

Transaction(0,0) takes place at site 0 and submits write operations on data items 0 and 2 at times 0 and 60ms after the simulation is started and a commit operation at 121ms. Transaction(1,0) takes place at site 1 and submits write operations on data items 1 and 3 at times 0 and 60ms after the simulation is started and a commit operation at 121ms. Both the transactions get committed at 131ms, but new transactions are not started in their place until

they have been processed at all the sites. At time 350ms, epidemic communication takes place and transactions(1,0) and (0,0) are propagated to all the sites where they are not found to conflict and their write operations are done as forced writes at 426 and 486 seconds respectively.

Case 2: Transactions executing at different sites are found to conflict during the process of epidemic communication

In this case transactions occur at different sites and are propagated to other sites during the process of epidemic communication where they are found to conflict. The table 5.3 represents the execution of two such transactions in our system.

Table 5.3 Test case 2 of epidemic algorithms simulation

At Site 0	At Site 1	At Site 2
Tran(0,0) submits operation (0,w) at time 0	Tran(1,0) submits operation (1,w) at time 0	
Tran(0,0) submits operation (1,w) at time 60	Tran(1,0) submits operation (0,r) at time 60	
	Tran(1,0) submits operation (*,c) at time 89	
Tran(0,0) submits operation (*,c) at time 121		
Epidemic Communication Take place at time 350		
Trans(1,0) is received and found to conflict with Tran(0,0) and are both marked in conflict at time 425.	Trans(0,0) is received and found to conflict with Tran(1,0) and both are marked in conflict at time 425.	Trans(0,0) is received and since it does not conflict, the write operations are queued to be executed. The operations are queued for execution at times 426 and 428.
		Trans(1,0) is received and since it found to conflict with Trans(0,0) both are marked in conflict at time 425.

Transaction(0,0) takes place at Site 0 and submits write operations on data items 0 and 1 at times 0 and 60ms after the simulation is started and a commit operation at 121ms.

Transaction(1,0) takes place at site 1 and submits a write operation on data items 1 and a read operation on 0 at times 0 and 60ms after the simulation is started and a commit operation at 89ms. At time 350ms, epidemic communication takes place. Site 0 receives Transaction(1,0) at 425ms and finding that it conflicts with Transaction(0,0) it marks them both inconflict. Similarly Site 1 marks them both inconflict when it receives Transaction(0,0) at time 425ms. Site 2 receives Transaction(0,0) at time 425ms initially and since it does not conflict with any other transaction, its operations are queued for execution. Transaction(1,0) is received at 425ms but since Site 2 knows about Transaction(0,0) both these transactions are found to conflict and are marked inconflict

Case 3: Transactions marked because the one it read from has been marked

This case tests the scenario of a transaction that read from a transaction that has been marked. The table 5.4 represents this execution.

Table 5.4 Test case 3 of epidemic algorithms simulation

At Site 0	At Site 1
Tran(0,0) submits operation (0,w) at time 0	Tran(1,0) submits operation (1,w) at time 0
Tran(2,0) submits operation (2,w) at time 0	
Tran(0,0) submits operation (1,w) at time 60	Tran(1,0) submits operation (0,w) at time 60
Tran(2,0) submits operation (0,w) at time 60 (Waits on Tran (0,0) to complete and release lock on data item 0)	
Tran(0,0) submits operation (*,c) at time 121	Tran(1,0) submits operation (*,c) at time 121
Tran(2,0) submits operation (*,c) at time 192	

Epidemic Communication Take place at time 350

Continued on next page

Table 5.4. Continued

At Site 0	Site 1
Trans(1,0) is received and found to conflict with Tran(0,0) and are both marked in conflict at time 425.	Trans(0,0) is received and found to conflict with Tran(1,0) and both are marked in conflict at time 425
Tran(2,0) is marked in conflict because it read from Tran(0,0) at time 425.	Trans(2,0) is received and since it read from Tran(0,0) which is marked in conflict it is also marked in conflict at time 425.

In this case Transaction(0,0) takes place at Site 0 and submits write operations on data items 0 and 1 at times 0 and 60ms and a commit operation at time 121ms. Similarly Transaction(2,0) submits write operations on data items 2 and 0 at times 0 and 60ms and a commit operation at time 192ms. The write operation of Transaction(2,0) on data item 0 waits until the Transaction(0,0) has completed because Transaction(0,0) is holding a write lock on data item 0. Transaction(1,0) takes place at Site 1 and submits write operations on data items 1 and 0 at times 0 and 60ms and a commit operation at time 121. During the process of epidemic communication Transaction(1,0) is received at time 425ms by Site 0 and is found to conflict with Transaction(0,0) and both are marked in conflict. Additionally Transaction(2,0) is also marked in conflict as it contains Transaction(1,0) in its readFrom set. At Site1 Transaction(0,0) is received first at 425ms since it conflicts with Transaction(2,0) both are marked in conflict. Transaction(2,0) is received after that and since it contains Transaction(0,0) in its readFrom set it is also marked as in conflict.

Case 4: Transaction received via epidemic communication conflicts with current operations

This case tests the scenario of a transaction taking place at a site conflicting with another transaction which arrived at the site by the process of epidemic communication. In this case epidemic communication of Site 0 was done at time 100ms and the communication time is set to 10ms to capture this scenario.

Table 5.5 Test case 4 of epidemic algorithms simulation

At Site 0	At Site 1
Tran(0,0) submits operation (0,w) at time 0	Tran(1,0) submits operation (0,w) at time 0
Tran(0,0) submits operation (2,r) at time 60	Tran(1,0) submits operation (3,w) at time 60
Tran(0,0) submits operation (*,c) at time 89	
Epidemic Communication was done at time 100	Tran(0,0) was received at 110 and Tran(1,0) is aborted and all its locks are released at the same. The write operations of Tran(0,0) are queued for execution and done at time 111
	Tran(1,113) submits operation (w,113) at time 113

Transaction (0,0) takes place at site 0 and submits a write operation on data item 0 at time 0ms, a read operation on data item 2 at time 60ms and a commit operation at time 89ms. Epidemic communication for Site 0 is done at time 100ms. Transaction(1,0) submits write operations on data items 0 and 3 at times 0 and 60ms. But Site 1 receives Transaction(0,0) via epidemic communication at time 110ms (since communication time is set to 10ms) and it is found to conflict with Transaction(1,0). Transaction(1,0) is aborted whereas the write operations of Transaction(0,0) are queued for execution and are executed at 111ms. This is because the epidemic algorithms abort transactions that are taking place currently and allows transactions that were committed at other sites to take place.

Case 5: Transaction current operations waiting on a forced write lock

This captures the scenario of a transaction's operations waiting on a forced write lock of a transaction which came in via epidemic communication. Table 5.6 represents this execution.

Table 5.6 Test case 5 of epidemic algorithms simulation

At Site 0	At Site 1	At Site 2
Tran(0,0) submits operation (0,w) at time 0	Tran(1,0) submits operation (1,w) at time 0	
Tran(0,0) submits operation (2,w) at time 60	Tran(1,0) submits operation (3,r) at time 60	
Tran(0,0) submits operation (*,c) at time 121	Tran(1,0) submits operation (*,c) at time 89	
Epidemic Communication Take place at time 350		
Trans (1,0) is received and since it does not conflict, the write operations are queued to be executed at times 426 and 486 respectively	Trans(0,0) is received and since it does not conflict, the write operations are queued to be executed at times 426 and 486 respectively	Trans(0,0) is received and since it does not conflict, the write operations are queued to be executed at times 426 and 486 respectively
Trans(0,426) submits operation (1,w) at 426---waits on forced write	Trans(1,426) submits operation (0,r)—waits on forced write	Trans(1,0) is received and since it does not conflict, the write operations are queued to be executed at times 426 and 486 respectively
	Trans(1,426) submits operation(8,w) at time 523	
Trans(0,426) submits operation(9,r) at time 556	Trans(1,426) submits operation(*,c) at time 585	
Trans(0,426) submits operation(*,c) at time 585		

Transaction(0,0) takes place at site 0 and submits write operations on data items 0 and 2 at time 0ms and 60ms respectively. It then submits a commit operation at time 121ms. Transaction(1,0) takes place at site 1 and submits a write operation on data item 1 and a read operation on data item 3 and a commit operation at times 0,60 and 121ms respectively. Epidemic Communication takes place at time 350. At time 425ms, Transaction(1,0) is received by Site 0 and since it does not conflict, its forced write operations take place at time 426 and 486ms respectively. Similarly Transaction(0,0) is received by Site 1 at time 425ms

and its operations are executed as forced writes at time 426 and 486ms respectively. The simulation was made so that the Transaction(0,426) starts at time 426ms and submits a write operation on data item 1 at 426ms which is made to wait on the forced write of Transaction(1,0) and it submits its read operation on data item 9 at time 556ms only. It then submits a commit at time 585ms. Similarly at Site 1 Transaction(1,426) is made to submit a read operation on data item 0 at time 426ms and it waits on the forced write of Trans(0,0). Hence its write operation is submitted only at time 523ms and then it submits a commit operation at time 585ms. At Site 2, Transactions(0,0) and(1,0) are received at 425ms and since they do not conflict with any transactions their write operations are performed as forced writes at time 426ms and 486ms.

Thus five different scenarios of transaction execution were identified and the simulation was tested against these scenarios. This strengthens the confidence on our simulation.

CHAPTER SIX

Experiments and Results

The first section of this chapter describes the parameter settings for the experiments that were conducted on the 2PC and ROWA simulation (explained in Chapter Four) and Epidemic Algorithms simulation (explained in Chapter Five) to simulate P2P environments. The second section of this chapter describes results of these experiments. The last section presents the overall observations from our experiments.

6.1 Parameter Settings for the Experiments

The parameters of the experiments have to be set to reflect P2P environments. These parameters have been modeled based on the available literature. This section describes the parameters of our simulation and the literature that served as guides to assign their values.

6.1.1 Communication Time

Both these simulations necessitate the modeling of the communication time parameter. This is the duration to send a message from one site to another. The simulations assume a uniform communication model for all sites in the network (i.e. the duration to send a message from site A to site B is the same for any A and B)

(Pei et al, 1998) estimates the number of hops and round trip from a UCLA host computer to a randomly selected set of 3,219 hosts in four continents around the world. They use a measurement methodology similar to that of traceroute to measure hop count. Traceroute transmits packets with small TTL values which are decremented for every hop. If the TTL reaches zero, an ICMP Time Exceeded message is sent to the sender. Sending

packets with incremental TTL values will make all the hosts in the path of the packet to send back ICMP Time Exceeded messages, when the TTL expires on reaching them. This helps to identify the route of the packet. The packet also contains a port id that is not normally used by the destination host. When this reaches the destination host, it sends back a ICMP Port Unreachable error. Thus the entire route to the destination is traced. Then the program sends 20 - 48 byte UDP packets to the destination with a time interval of 5 seconds between each. The time from sending a probe packet and receiving it back is calculated as the *Round Trip Time* and the average is calculated as the *Round Trip Delay*. Their results show that the round trip delays for 90% of the hosts are less than 153ms. The average delay for international hosts vary from 116.74ms (Canada) to 1537.52ms (China). This is because there were only two links between China and the US during the time of measurement (1997-98) – one 2Kbps and the other 2Mbps and also the links within China were slow.

In (Graham *et al.*, 1998) they present a novel and low-cost technique for accurately measuring delay, delay variation and packet loss on the intercontinental internet. According to their results the unidirectional delay between Waikato(NZ) and Cambridge(England) varies from 0.15 secs to 0.5 secs at different times of the day with the majority values around 0.2 to 0.25 secs.

(Kaladindi *et al.*, 1998) is a measurement infrastructure that measures end-to-end unidirectional delays, packet loss and route information along Internet paths deployed at about 50 higher education and research sites around the world. The test packets were 12 bytes and sent using UDP (totaling to 40 bytes). The delay for US to Netherlands monitored at different times varied from 60 to 200ms.

(Huitema *et al.*, 2000) measures the quality of web service by selecting and polling a random set of 100 web servers every day. The test was started at 3 pm EST, and the

homepages of 100 random servers were downloaded. According to their results, the median round trip delay is around 130 milliseconds.

A report of the global response time (round trip time) can be obtained for each day and for the past 30 day period from the Internet Traffic Report Website (Internet Traffic Report website). Their reports show that the global response time varies from 30 to 145ms during the time period 01/15/2005 to 02/14/2005.

Assuming symmetric paths (the packets follow the same path to and fro) then the one way delay is equal to half the round trip time. In our experiments the sites have to send operations, messages that are part of the 2PC, transaction records and timetables to other sites. These are small messages (less than 128 bytes in our experiments) and will not usually be segmented. Hence the delay is comparable to any of our references. To model the network we have assumed the diameter of our P2P network to be similar to that of (Pei et al, 1998) spanning the entire U.S.A. Their report shows that the round trip delays for 90% of the hosts are less than 153ms. Assuming a symmetric path the communication time parameter for our experiments has been rounded and fixed at 75ms.

6.1.2 I/O Time

The durations of read and write operations have to be modeled for our experiments. This necessitates the modeling of I/O time. (Chen *et al.*, 2003) calculates the disk access per data block for a disk with 73-GB capacity, 5ms average seek time and a rotational speed of 10,000 RPM or an equivalent of 3ms. The average rotational latency and sustained data transfer rate is 29.8 – 58.0 MB/sec. Assuming 40 MB/sec average transfer rate and each data block to be 40 KB the data transfer time is 1ms per data block. The average disk access time per block obtained by summing up seek time (5ms), rotational latency (3ms) and transfer time (1ms), is 9ms.

The PC Technology Guide (PC Technology Guide Disk Performance) says that by late 2001 the fastest high-performance drives were capable of an average latency of less than 3ms, an average seek time between 4 and 7ms and maximum data transfer rates in the region of 50 and 60 Mbps for EIDE and SCSI-based drives respectively which is similar to the values of (Chen *et al.*, 2003).

IBM (IBM 1998) reports that most of the desktop then had a seek time of 9.5ms and a data transfer time of 0.3ms for 4KB of data. Using the equation that I/O time = command overhead time + seek time + latency + time to transfer ($0.5+9.5+4.2+0.3$) they have computed I/O time to be 14.5ms. This was calculated for a 7200 RPM harddrive.

73.4 GB Cheetah 15 K.3 has claimed read and write seek times of 3.6 and 4.0ms, which is a littler slower than 3.4 and 3.8ms read and write seek times of Maxtor's 73.4 GB (Tech Report, 2003). Seagate also claims transfer rates as high as 86MB/sec for the Cheetah 15K but Maxtor's Atlas 15K tops out at only 75MB/sec.

(Zhu *et al.*, 2004) have measured the average disk access time on an IBM Ultrastar 36Z15 to be 10ms and have used this value in their simulation.

Based on this survey we model the average disk access time as 10ms.

6.1.3 Duration of Read, Write and Log-Write

Equation 3.2 shows the derivation for the duration of read operation (Ulusoy, Belford 1992). Assuming a main memory equal to half the size of the database (Ulusoy, Belford 1992) we get the duration of read operation to be 5ms. The duration for a write is equal to the average disk access time (10ms). But a write operation consists of a read and a write. Hence the duration of a write operation is 15ms. The duration for a log write operation is equivalent to that of the average disk access time

6.1.4 Bandwidth

The epidemic algorithms simulation has a bandwidth parameter associated with it, which defines the common bandwidth of the network explained in Chapter Five. A bandwidth measurement study in P2P systems (Saroiu *et al.*, 2001) reports that only 20% of Napster users and 30% of Gnutella users have bandwidth greater than 3 Mbps, and 50% of Napster users and 60% of Gnutella users have broadband connection. We assume that only such peers will become part of a P2PDB network, and we have fixed our bandwidth parameter at 1Mbps.

6.1.5 Time to Failure and Time to Recover

(Saroiu *et al.*, 2001) monitors 17,215 Gnutella clients for a period of 60 hours and 7,000 Napster clients for a period of 26 hour and reports that the median session duration for both Gnutella and Napster clients is 60 minutes. (Chu *et al.*, 2002) studies the availability of few thousand Gnutella peers for a five week period and observes that 30% of the peers of Gnutella and Napster are available for 10 minutes. (Gummadi *et al.*, 2003) have reported the average session length of Kazaa users to be 2.4 minutes for a 200 day trace period in University of Washington. (Birrer and Bustamante, 2004) have observed that measurement studies of P2P systems have reported median session times ranging from an hour to a minute. They have used two different failure rates for their experiments – one with high failure rate (MTTF 5 minutes and MTTR 2 minutes) and another with low failure rate (MTTF 60 and MTTR 10 minutes). (Rhea *et al.*, 2004) have reported that median session time in deployed P2P networks varies from an hour to few minutes. Furthermore they have observed that the average session time of more than 50% of the peers in the experiments of (Sen and Wang, 2004) is less than one minute.

In our experiments, we have modeled a severe P2P environment with high failure rates resembling the high failure rate of (Birrer and Bustamantem, 2004) because our simulations ran out of heap space if the simulations run for too long due to accumulation of records in the CommunicationManager and event logs and other information needed to maintain transaction history and details for epidemic algorithms. We classified our experiments in two major categories. In the first series of experiments, we kept the *time to failure* of a site constant at 60 seconds. Each site was assigned a random Δ_{failure} value at the startup in the range between 1 to 10 seconds. The first failure of each site occurs at $60 + \Delta_{\text{failure}}$ seconds, and the successive failures of each of the sites occur 60 seconds after their subsequent recovery. The Δ_{failure} ensures that the sites do not go down at the same instant. For this failure scenario, experiments were carried out keeping the time for recovery of the sites constant at 2 seconds, 10 seconds, 60 seconds and varying the *time for recovery* from 2 to 60 seconds.

In the second series of experiments, the *time to failure* of each site is randomly chosen to be between 30 and 90 seconds. For this scenario similar experiments were carried out keeping the *time for recovery* of the sites constant at 2 seconds, 10 seconds, 60 seconds and varying the *time for recovery* from 2 to 60 seconds.

6.1.6 Other Parameters

The *multi-programming level* is set at 50. Each transaction has 18 operations operating on different data items, out of which 4 are write operations. Writesets might be large, but we model low-conflict environments only in our experiments. There are 19,200 data items in each site/peer, and there are totally 8 sites/peers. The replication is maintained at 8 (1 copy per site). The epidemic communication time is fixed at 200ms since it takes

110ms to perform all the operations of a transaction assuming the transaction does not conflict. The parameters are listed in Table 6.1.

Table 6.1: The parameter settings for P2P environment simulation

Parameter	Value
No of data items	19200
No of Sites	8
Replication	8 (1 copy per site)
No of operations per transaction	18
No of read operations	14
No of write operations	4
Duration of Read	5ms
Duration of Write	15ms
Duration of Log Write	10ms
Multi programming level	50
Bandwidth	1Mbps
Communication Time	75ms
Epidemic Communication Time	200

6.2 Experiment and Results

Three experiments were performed for each setting, and the number of transactions committed for both the 2PC and ROWA simulation and the Epidemic Algorithms are measured at time intervals in powers of 2 starting at 2 seconds (2s, 4s, 8s, 16s, 32s, 64s, 128s, and 256s). This section presents the comparative results of both the simulations for different failure and recovery rates. All the experiments have a steady state time of 10 seconds, meaning the system runs for 10 seconds before any measurements are made.

The results of the experiments are presented below. The graphs present the number of transactions committed in the system under the three different algorithms for the specific parameter settings. The continuous line with the values marked in circles represent the results of 2PC+ROWA simulation. The dotted line with the values marked in squares represents the results of the Pessimistic Epidemic Algorithm for which we count the

transactions that are fully committed only. The dashed line with the values marked in triangles represents the results of the Optimistic Epidemic Algorithm for which we count the transactions that are both fully and partially committed. Eight different combinations of *TTF* and *TTR* were identified, and 3 experiments were conducted for each setting. The settings are explained in Section 6.15. The arithmetic mean of the three experiments was taken and the results are plotted as below.

6.2.1 Experiment 1 : Time To Failure Fixed at 60 seconds and Time to Recover Fixed at 2 seconds

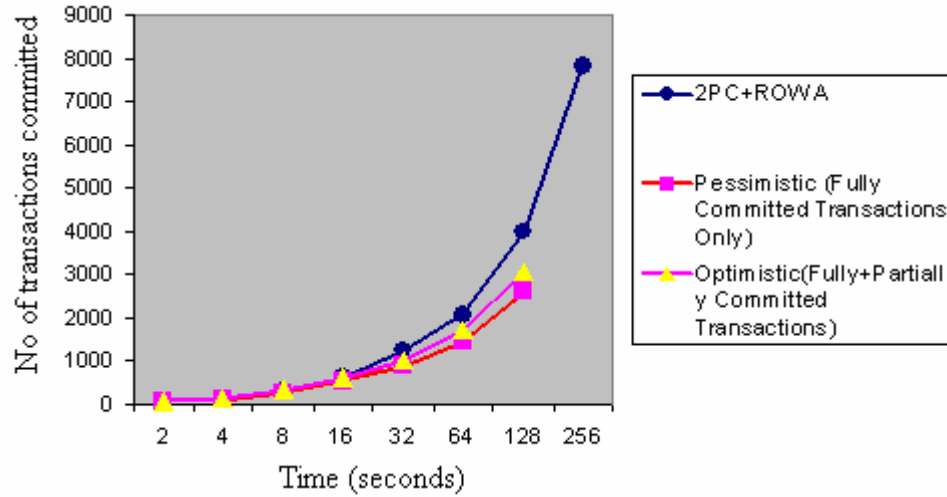


Fig. 6.1 Results for *TTF* = 60 seconds and *TTR* = 2 seconds

In Experiment 1, Epidemic Algorithms simulation did not run to completion since the JVM ran out of heap space. This might be because the *TTR* is very fast and the uptimes of the sites are high. Thus more transactions are committed leading to an increase in the size of the event logs and other information at each site requiring more heap space than the other experiments.

6.2.2 Experiment 2 : Time To Failure Fixed at 60 seconds and Time to Recover Fixed at 10 seconds

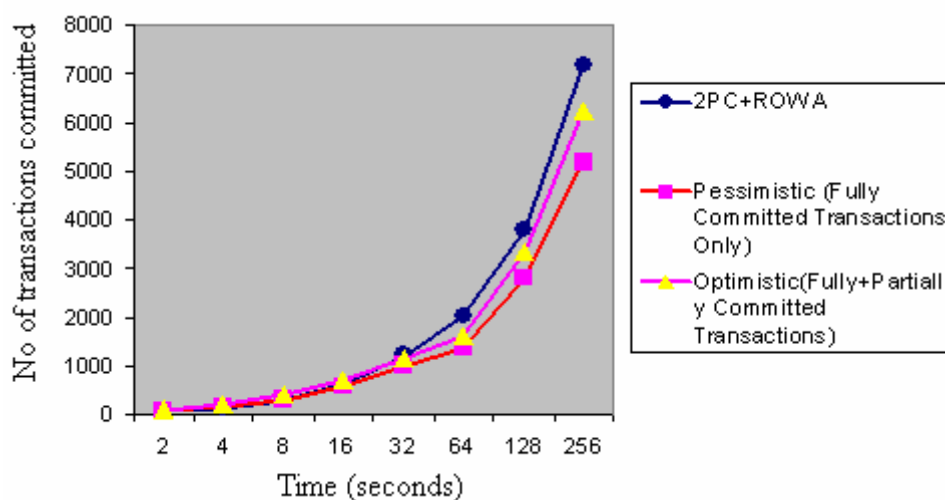


Fig. 6.2. Results for TTF = 60 seconds and TTR = 10 seconds

6.2.3 Experiment 3: Time To Failure Fixed at 60 seconds and Time to Recover Fixed at 60 seconds

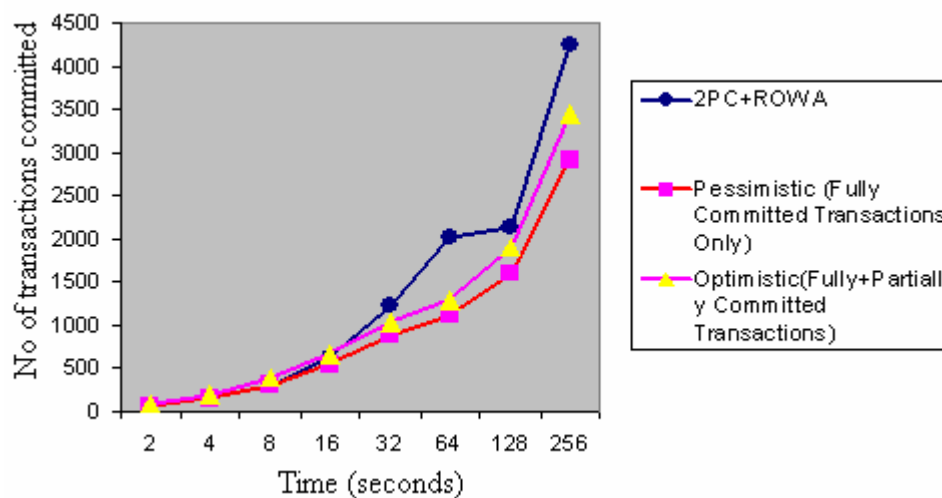


Fig. 6.3. Results for TTF = 60 seconds and TTR = 60 seconds

6.2.4 Experiment 4 : Time To Failure Fixed at 60 seconds and Time to Recover Varied between 2 to 60 seconds

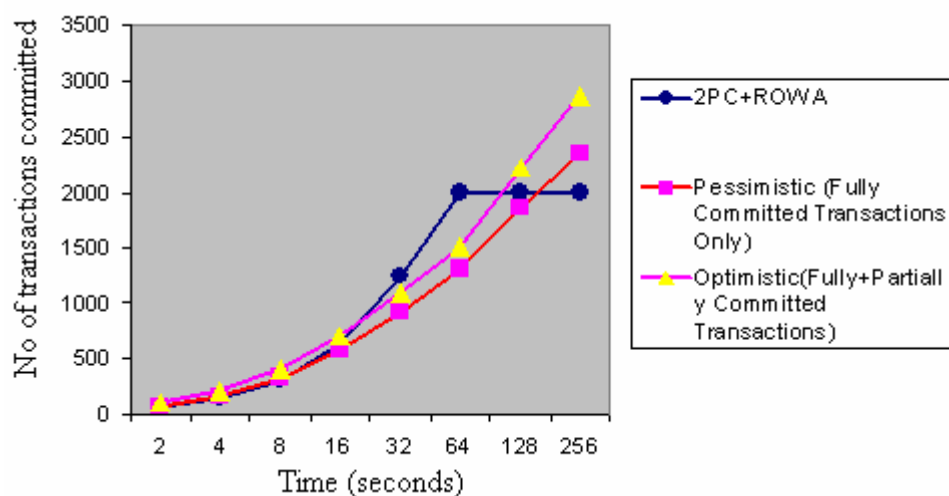


Fig. 6.4. Results for TTF = 60 seconds and TTR varied from 2 to 60 seconds

6.2.5 Experiment 5 : Time To Failure Varied from 30 To 90 seconds and Time to Recover Fixed at 2 seconds

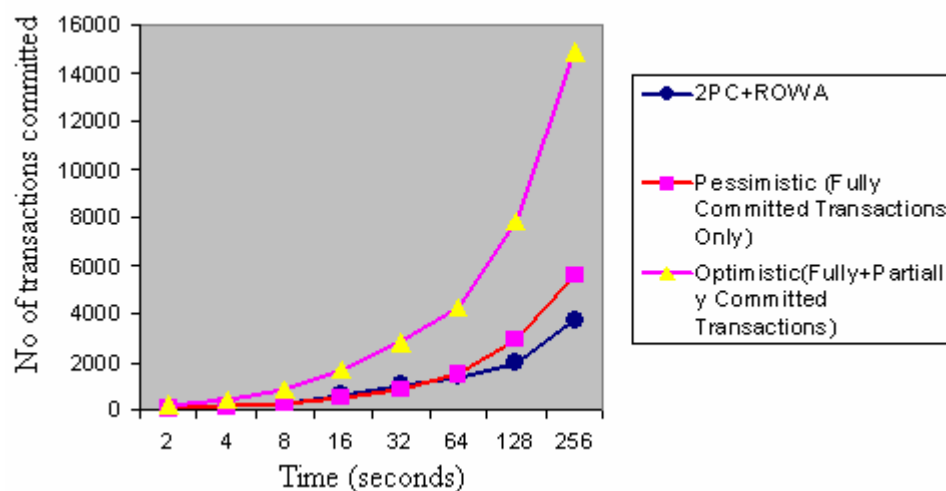


Fig. 6.5. Results for TTF varied from 30 to 90 seconds and TTR = 2 seconds

6.2.6 Experiment 6 : Time To Failure Fixed from 30 To 90 seconds and Time to Recover Fixed at 10 seconds

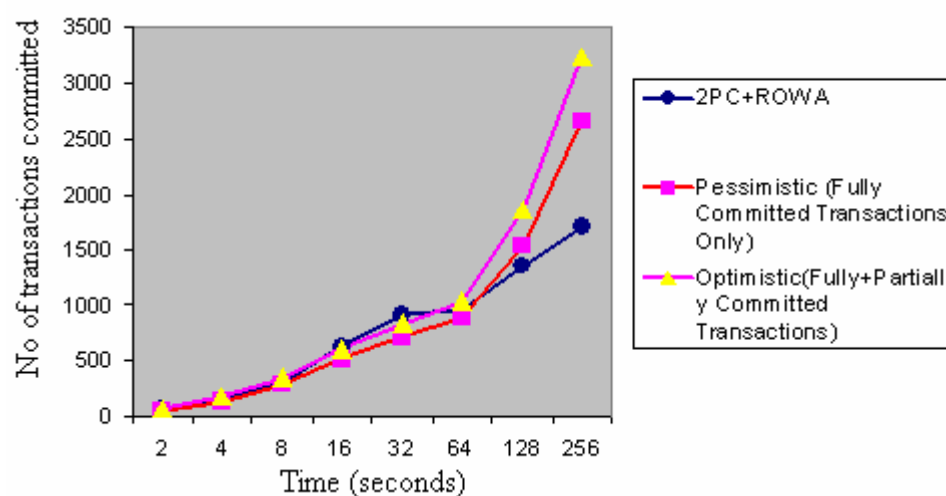


Fig. 6.6. Results for TTF varied from 30 to 90 seconds and TTR = 10 seconds

6.2.7 Experiment 7 : Time To Failure Fixed from 30 To 90 s and Time to Recover Fixed at 60 seconds

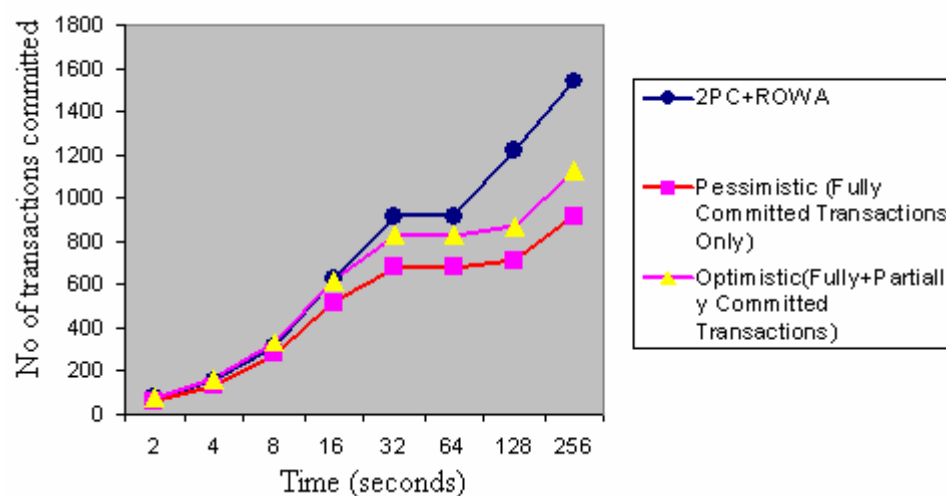


Fig. 6.7. Results for TTF varied from 30 to 90 seconds and TTR = 60 seconds

6.2.8 Experiment 8 : Time To Failure Varied from 30 to 90 seconds and Time to Recover Varied from 2 to 60 seconds

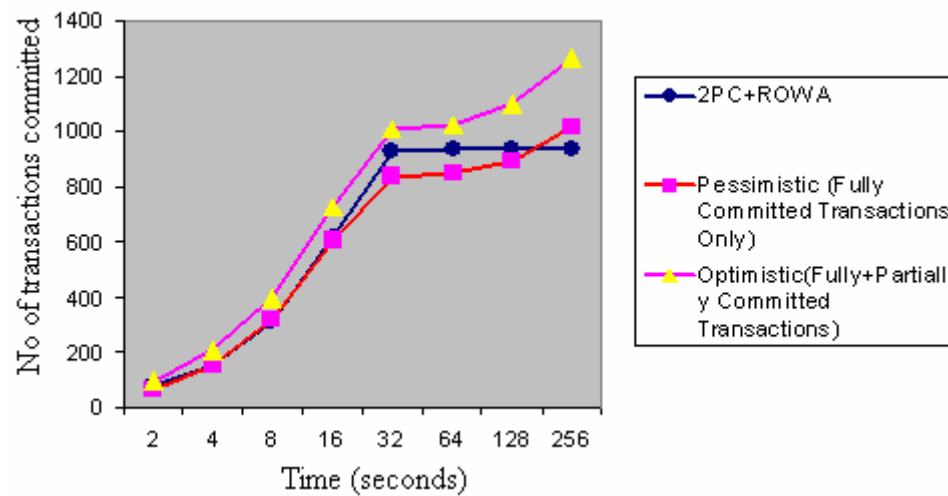


Fig. 6.8. Results for TTF varied from 30 to 90 seconds and TTR varied from 2 to 60 seconds

6.2.9 Experiment 9 : Low Conflict Scenario with Time To Failure Varied from 30 to 90 seconds and Time to Recover Varied from 2 to 60 seconds and MPL = 5

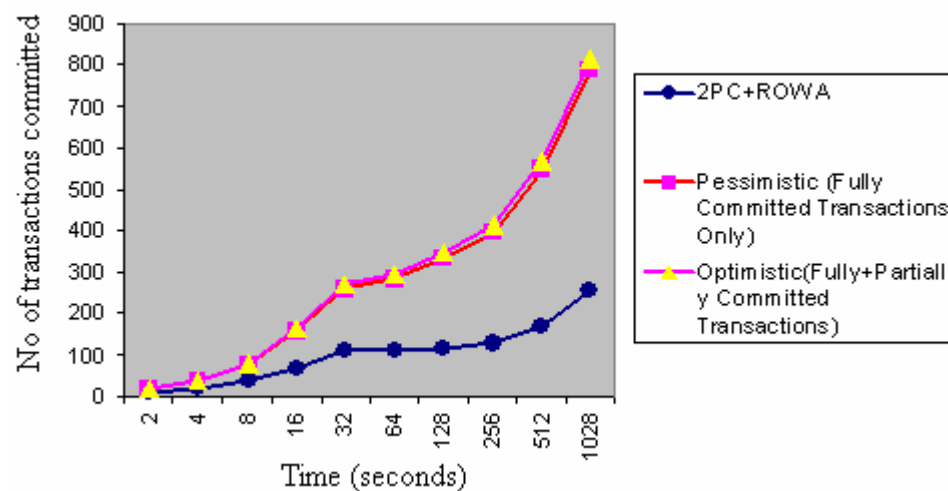


Fig. 6.9. Results for TTF varied from 30 to 90 seconds and TTR varied from 2 to 60 seconds and MPL=5

6.3 Observations and Analysis

For each simulation, except Experiments 3 and 7, the total number of the committed transactions is within 12 % of the reported values. For these two scenarios the values spike up to 28% and 38% respectively. In both these cases, one of the experiments is an outlier [ignoring that experiment brings the variation below 12%]. This could be due to the randomness in the operations of the transactions for each experiment, which gives rise to differences in the number of conflicts. The 2PC+ROWA protocol performs better than the Epidemic Algorithms in the first three settings. The first three cases present P2P environments where the failures and recovery are uniform. As we move towards more non-uniform environments more similar to expected P2P settings, the Epidemic Algorithms commit more number of transactions than the 2PC+ROWA. The reason that the Epidemic Algorithms perform better in Experiments 4 through 8 is because the failures and recoveries happen at different intervals for each of the sites and not all the sites are available at most instants of the experiment. For update, 2PC+ROWA protocol requires all the sites be available to commit the transaction. In the case of Epidemic Algorithms, the transactions are propagated at a later point of time when the sites become available and are performed then.

In all the cases, the Epidemic Algorithms start out better than the 2PC+ROWA protocol because the 2PC+ROWA protocol sends write operations to all sites individually, whereas in the case of Epidemic Algorithms, all the write operations are communicated through one TransactionRecord. The number of transactions committed under the EpidemicAlgorithms then drops due to redundancy in the communication or TransactionRecords. The transaction records from a site will be communicated to other sites repeatedly during the epidemic communication process until the notification of the

transaction being processed at those sites is received. Only after that the next transaction will be started in place of the transaction to maintain the multi-programming level.

In our simulations of Epidemic Algorithms, a transaction is defined as *active* until it has been processed at all sites. The next transaction is started in its place only after the previous one has been processed at all sites. This definition is not fixed as in the case of Optimistic Algorithms where a transaction is committed at a site after it has performed there. Therefore we could start as soon as one has committed at its local site. In this case, the number of transactions committed under of the Optimistic Epidemic Algorithms would be much higher. It is a matter of future study to determine which is a better representation of P2PDB.

On further analysis of the performance of 2PC+ROWA protocol for experiments 4 and 8, the transactions are found to have stopped committing after 64 seconds (indicated from the flatness in the Figures 6.4 and 6.8). This is because in these severe failure scenarios, some of the sites are always down after the failures start happening. Epidemic Algorithms however manage to commit transactions even in these cases owing to the property that the transactions are committed locally and then propagated at other sites and processed there whenever the sites are available.

In a P2PDB environment, if a site leaves permanently then 2PC+ROWA will not commit a transaction under full replication until the permanent unavailability of site is sensed by the system (usually by a huge timeout). Then a replica detail for every site has to be updated with removal of this site's entry. In the case of Optimistic epidemic algorithms permanent failure of sites does not have a drastic effect on the number of commits.

The availability of the database is increased under epidemic algorithms because transactions with write operations can be committed even if one site is available, whereas under 2PC+ROWA write transactions can be committed only when all sites are available.

However epidemic algorithms require more memory space due to the preservation of the transaction in the event logs. Also time for detecting conflicts by searching through the event logs will increase with increase in the event log size. If the size of the transaction becomes large, then communication might get delayed due to segmentation. Similarly if the number of sites is increased, the timetables will expand and similar delay during communication will be experienced.

Experiment 9 represents a low conflict scenario where the multiprogramming level is reduced to 50. As claimed in their proposal (Agrawal *et al.*, 1997), the epidemic algorithms perform extremely well in low conflict environments. This is true even in the case of severe P2P failure environments as simulated in the experiment.

CHAPTER SEVEN

Conclusion and Future Work

The first section of this chapter describes the conclusions drawn from our experiments. The next section of the Chapter presents the reader with the recommended future work

7.1 Conclusion

In typical P2P environments peers or sites arise anywhere and anytime. Knowledge of their location, start, failure and recovery is not available beforehand. Our experiments show that epidemic algorithms commit more number of transactions in the case of fully replicated P2P environments where the sites have varying failure and recovery rates compared to traditional 2PC coupled with ROWA.

Furthermore, since all sites need not be available at the same time for epidemic algorithms to commit transactions and transactions take place at a site and are propagated and performed at a later time, epidemic algorithms handle permanent site failures better than 2PC + ROWA. This increases the chances of committing transactions.

Additionally, this increases the availability of the database because update transactions can be committed even if only one site is available at a particular instant whereas in the case of 2PC+ROWA, the presence of all the sites is necessary. Crucial transactions can be completed even in high failure environments under epidemic algorithms.

Epidemic algorithms have certain drawbacks too. They require more memory as the event log accumulates over time thus imposing a large main memory requirement for participants in the P2PDB network. In the case of our experiments, considering the small

size of the transaction records (less than 100 bytes) and the ephemeral presence of peers, this memory increase is insignificant. But considering a scenario where the transactions have large write sets, this could be more critical.

Transaction-records are redundantly communicated to sites until an acknowledgement of the transaction performed at that site is received. This is indicated in the time table of the other sites. Thus even when the transactions are performed at all sites, there is a duration where the site of origination does not know about this and keeps sending this transaction's transaction-record to the other site assuming that the previously sent ones have not reached the receivers. Apart from the communication overhead, the response time of the transaction is also delayed. Increase in the size of the event logs will increase the time for detecting conflicts. Increase in the size of transactions or the number of sites might lead to the segmentation of the transaction records and timetable during communication increasing the communication delay.

7.2 Future Work and Recommendations

As much as my exploration provides insight into the behavior of epidemic algorithms and 2PC+ROWA in P2P environments, further explorations in this arena would bring to light the more precise behavior. This would provide a better understanding about these algorithms. This section highlights some of our recommendations.

Since our simulation simulates all the sites in one program, our experiments have been restricted to shorter run time due to the program running out of heap space in JAVA. This could be overcome by making each site into a program. Communication between sites can be simulated by using RMI. This could help us model P2P environments where failures and recoveries are not severe.

We have assumed that a transaction remains active until they are processed at all sites even in the case of Optimistic Epidemic Algorithms. The next transaction is started in its place only after it has been processed at all sites. This perception may vary between systems as in Optimistic Epidemic Algorithms transactions are committed at a local site and are resolved when conflicts are discovered at another site during the process of epidemic communication. Simulations can be built allowing transactions to be active only until they commit at their local site and the next transaction could be started immediately. Such simulations would yield better throughput for Optimistic Epidemic Algorithms.

In our experiments we have modeled transactions arising continuously one after the other so that we maintain the multi-programming level at a constant number. In typical database environments, the transactions are not uniform and tend to follow a Poisson distribution. The performance of these algorithms under a Poisson distribution of transactions can be studied.

Our experiments simulate fully replicated environments. Replication is a vast field that requires more study. The performance of these algorithms under various replication environments would be a separate study by itself.

The experiments are performed for a fixed class of transactions. In typical real world environments we will have transactions of various lengths and natures. This can be studied changing our simulation so that transactions of mixed classes exist in the system. Environments with different conflict ratios could be modeled and the effect of larger writesets of transaction on Epidemic algorithms could be analyzed.

We have assumed a uniform communication model, where the time taken for a message to be sent from site A to site B for any site A, B is fixed. In typical P2P networks this is different for every A, B pair. Investigation of typical P2P network topologies could be done and our simulation can be tuned to model a non-uniform communication model.

Thus new studies based on many parameters used in our simulations can provide further insight about the performance of P2PDB algorithms.

BIBLIOGRAPHY

- Agrawal, D., Abbadi, A.E., Steinke, R.C. 1997. Epidemic algorithms in replicated databases (extended abstract). *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp.161-172.
- Ashwin R. Bharambe, Mukesh Agrawal, Srinivasan Seshan. 2004. Mercury: supporting scalable multi-attribute range queries. *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* 34: 353-366.
- Bernstein, P.A. and Goodman, N. 1984. An algorithm for concurrency control and recovery in replicated distributed databases. *ACM Transactions on Database Systems (TODS)* 9: 596-615.
- Bernstein, P.A., Hadzilacos, V. and Goodman, N. 1987. Concurrency control and recovery in Database Systems. *Addison-Wesley*.
- Birrer, S. and Bustamante, F. 2004. Resilient peer-to-peer multicast from the ground up. *Proceedings of Network Computing and Applications* 00: 351-355.
- Cai, M.J., Frank, M., Chen, J. and Szekely, P. 2003. MAAN: A multi-attribute addressable network for grid information services. *Proceedings of the Fourth International Workshop on Grid Computing*, pp.184-206.
- Chu, J., Labonte, K. and Levine, B. 2002. Availability and locality measurements of peer-to-peer file systems. *ITCom: Scalability and Traffic Control in IP Networks, Boston, MA* 4868: 310-321.
- Carrey, M.J. and Livny.M. 1991. Conflict detection tradeoffs for replicated data. *ACM Transactions on Database Systems (TODS)* 16: 703-746.
- Clarke, I., Sandberg, O., Wiley, B. and Hong,T.W. 2000. Freenet: a distributed anonymous information storage and retrieval system. *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability* 2009: 46-66.
- Chen, C. and Cheng, C.T, 2003. Replication and retrieval strategies of multidimensional data on parallel disks. In *Proceedings of CIKM*, pp. 32-39.
- Connolly , T. and Begg, C. (1996). Database systems: a practical approach to design, implementation and management. *Addition Wesley*.
- Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I. 2004. Queries and updates in the coDB peer to peer database system. <http://www.inf.unibz.it/~franconi/papers/vldb-demo-04.pdf>.

- Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I. 2004. A distributed algorithm for robust data sharing and updates in p2p database networks. *Proceedings of the P2P&DB International Workshop, Heraklion - Crete, Greece*.
- Giunchiglia, F. and Zaihrayeu, I. 2002. Making peer databases interact – a vision for an architecture supporting data coordination. *Proceedings of the Conference on Information Agents 6*: 18-35.
- Gray, J. and Reuter, A. 1992. Transaction processing concepts and techniques. *Morgan Kaufmann Publishers Inc.*
- Graham, I.D., Donnelly, S.F, Martin, S., Martens, J. and Cleary, J.G. 1998. Nonintrusive and accurate measurement of unidirectional delay and delay variation on the internet. *ISOC Inet*, pp.21-24.
- Gummadi, K.P, Dunn, R.J., Saroiu, S. and Gribble, S.D. 2003. Measurement, modeling and analysis of a peer-to-peer file-sharing workload. *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 314-329.
- Halevy, A.Y., Ives, Z.G, Mork, P. and Tatarinov, I. 2003. Piazza: data management infrastructure for semantic web applications. *Proceedings of the 12th International Conference on World Wide Web 2003*, pp. 556-557.
- http://www.csee.umbc.edu/help/oracle8/server.815/a67784/ds_ch3.htm#283. Sep 15, 2005. UMBC's Oracle site.
- <http://www.dbasupport.com/oracle/ora9i/ors.shtml>. Sep 15, 2005. DBASupport Oracle replication site.
- <http://internettrafficreport.com>. Oct 12, 2005. Internet traffic report website.
- <http://www.jxta.org>. Sep 15, 2005. JXTA.
- http://www.pctechguide.com/04disks_Performance.htm. Oct 12, 2005. PC technology guide disk performance.
- <http://techreport.com/reviews/2003q4/cheetah-15k/index>. Oct 3, 2003. The Tech Report.
- <http://www.wired.com/news/business/0,1367,67202,00.html>. Sep 15, 2005. Peer-to-Peer estimation site.
- <http://www.seti.org>. Sep 15, 2005. Seti homepage.
- <http://www.gnutella.com>. Sep 15, 2005. Gnutella: The gnutella file-sharing protocol.
- <http://www.napster.com>. Sep 15, 2005. Napster: The napster file sharing system.

- Huebsch, R., Hellerstein, J.M., Lanham, N., Loo, B. T., Shenker, S. and Stoic, I. 2003. Querying the internet with PIER. *Proceedings of 29th International Conference on Very Large Data Bases, Berlin, Germany*, pp. 321-332.
- Huitema, C. and Weerahandi, S. 2000. Internet measurements: The rising tide and the DNS snag. *Proceedings of the 13th ITC Specialist Seminar on Internet Traffic Measurement and Modelling, Monterey, CA*.
- IBM. 1998. Hard drive performance 7200 rpm versus 5400 rpm at interface disk drives. <http://www.cs.ccu.edu.tw/~wcshen/HD.pdf>
- Johnson, D. 2002. The PGReplication project. *PostgreSQL Database Replication Options*.
- Kaladindi, S., and Zekauskas M.J. 1998. Surveyor: an infrastructure for internet performance measurements. *In ISOC inet 99*.
- Kubiatowicz, J., Wells, C., Zhao, B., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S. and Weatherspoon, H. 2000. OceanStore: an architecture for global-scale persistent storage. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems 2000*, 28: 190-201.
- Korth, H.F. and Speegle, G. 1994. Formal aspects of concurrency control in long-duration transaction systems using the NT/PV model. *ACM Transactions on Database Systems (TODS)* 19: 492 – 535.
- Liu, M.L., Agrawal, D. and Abbadi, A.E. 1998. The performance of two phase commit protocols in the presence of site failures. *Distributed and Parallel Databases 1998* 6: 157-182.
- Molina, H.G. and Salem, K 1987. SAGAS. *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, San Francisco, California, United States 1987*. 16: 249–259.
- Muthitacharoen, A., Morris, R., Gil, T.M. and Chen, B. 2002. Ivy: a read/write peer-to-peer file system. *ACM SIGOPS Operating Systems Review 2002*, pp. 31-44.
- Pei, G., Liu, R. and Zhang, L. 1998. Measurement of delay and hop count on the internet. *IEEE GLOBECOM'98 - Internet Mini-Conference, 1998*.
- Ranasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S. 2001. A scalable content-addressable network. *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* 31: 161-172.
- Rhea, S., Geels, G., Roscoe, T. and Kubiatowicz, J. 2004. Handling churn in a DHT. *Proceedings of the USENIX Annual Technical Conference, June 2004*.

- Rowstron, A., Druschel, P. 2001. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, Nov 2001*, pp. 392-350.
- Rowstron, A., Druschel, P. 2001. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles* 35: 188-201.
- Saroiu, S., Gummadi, K.P. and Gribble, S. 2002. A measurement study of peer-to-peer file sharing systems. *Proceedings of Multimedia Computing and Networking Conference, January 2002*.
- Stoica, I., Morris, R., Nowell, D.L., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H. 2003. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking* 2003 1: 17-32.
- Subrata, S. and Wang, J. 2002. Analyzing peer-to-peer traffic across large networks. *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement*. 5: 137 - 150.
- Sung, A.L.G., Ahmed, L., Blanco, R., Li, H., Soliman, M.A. and Hadaller, D. 2005. A survey of data management in peer-to-peer systems. <http://www.cs.uoi.gr/~pitoura/courses/p2p/papers/survey3.pdf>
- Thomasian, A 1993. Two phase locking performance and its thrashing behaviour. *ACM Transactions on Database Systems*, 18: 579-625
- Ulusoy, Ö., Belford, G.G. 1992. A simulation model for distributed real-time database systems. *Proceedings of the 25th Annual Symposium on Simulation*, pp. 232-240
- Vecchio, D.D. and Son, S.H. 2005. Flexible update management in peer-to-peer database systems. <http://www.cs.virginia.edu/~son/publications/p2p.IDEAS05.pdf>.
- Zhu, Q., Shankar, A. and Zhou, Y. 2004. PB-LRU: a self-tuning power aware storage cache replacement algorithm for conserving disk energy. *Proceedings of the 18th annual International Conference on Supercomputing*, pp. 79-88.