

**SOME PERFORMANCE ISSUES
IN
DISTRIBUTED REAL TIME DATABASE SYSTEMS**

A THESIS

*Submitted in fulfillment of the
requirements for the award of the degree
of
DOCTOR OF PHILOSOPHY
in
COMPUTER SCIENCE AND ENGINEERING*

By

UDAI SHANKER



**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE-247 667 (INDIA)**

DECEMBER, 2005

© INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE, INDIA, 2005
ALL RIGHTS RESERVED



INDIAN INSTITUTE OF TECHNOLOGY ROORKEE ROORKEE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled "Some Performance Issues in Distributed Real Time Database Systems" in fulfilment of the requirement for the award of the Degree of Doctor of Philosophy and submitted in the Department of Electronics and Computer Engineering of Indian Institute of Technology Roorkee, Roorkee is an authentic record of my own work carried out during a period from July 2001 to December 2005 under the supervision of **Dr. A. K. Sarje**, Professor & Head and **Dr. Manoj Misra**, Associate Professor, Department of Electronics and Computer Engineering of Indian Institute of Technology Roorkee, Roorkee.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other Institute/University.


(UDAI SHANKER)

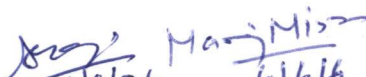
This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Date: 28/12/05


(Dr. Manoj Misra)


(Dr. A. K. Sarje)

The Ph.D. Viva-Voce examination of Mr. Udai Shanker, Research Scholar, has been held on June 05, 2006


Signature of Supervisors


Signature of H.O.D.


Signature of External Examiner

ACKNOWLEDGEMENTS

It is indeed a great pleasure to express my sincere thanks to my august supervisors Dr. Anil K. Sarje, Professor and Head, and Dr. Manoj Misra, Associate Professor, Department of Electronics and Computer Engineering, IIT Roorkee for their invaluable advices and pragmatic views of research. Their keen interest, sincere advice and kind help throughout during the completion of this work had been a regular source of encouragement. I appreciate the hospitality of their families too. Really, it was a pleasant experience for me to be surrounded by these noble and affectionate people.

I am thankful to Prof. (Mrs.) Kumkum Garg, Prof. Kuldip Singh and Dr. M. J. Nigam of Electronics and Computer Engineering Department of the Institute for their invaluable encouragement and support, and above all, the noblest treatment extended by them.

There are not enough words to express my gratitude to Dr. K. Ramamritham, Professor, Dept of Computer Science, IIT Mumbai, Dr. K. Y. Lam and Dr. Victor C. S. Lee, Dept. of Computer Science, City University Hong Kong, Dr. J. Taina, Department of Computer Science, University of Helsinki, Dr. Jan Lindstrom, Senior Software Engineer, Innodb, Finland and Dr. Rashmi Srinivasa, Project Manager/Architect, Qovia, INC, USA for their online help, advices and suggestions on the reported work. They truly embody the best of what the technical education is all about.

Professor Pratap Singh, Principal, M. M. M. Engineering College, Gorakhpur, really deserve my heartiest honour for providing me all the moral and administrative support. I can never forget Dr. Arun Kumar, Professor, Department of Electronics and Communication Engineering, M. M. M. Engineering College, Gorakhpur, for being a torch bearer for this turning period of my life. I am also very much grateful to my colleagues Mr. R. D. Patel, Mr. Y. S. Yadav, Mr. S. P. Singh and Mr. A. K. Daniel for their support and cooperation.

I am extremely thankful to Dr. R. B. Patel and Dr. V. K. Giri and their families for having spent their invaluable time in constructive motivation and encouraging environment. I also take this opportunity to extend my sincere thanks to Mr. Ajey Kumar, Mr. Amit T. Saornerkar and Dr. R. S. Yadav, who have provided me considerable help in implementing the simulator and were always available for discussion and any kind of

help during my work. I am very much thankful to my friends Mr. Rajwinder Singh, Mr. Narottam Chand, Ms. Bhawana Jharia, Ms. Navdeep Kaur, Dr. A. K. Pandey & family and Dr. Govind Pandey & family for their inspiration and encouraging attitude to get through the difficult periods of my stay at Roorkee. I also express my sense of gratitude to Mr. Raj Khati, Mr. Anoop Kumar and Mr. Dhanpat Singh for their co-operation, assistance and technical supports in the software lab of the department. It is hard to single out any one in this list.

Honestly, I have been able to complete my work and present the thesis only due to the constant encouragement and love of my brother Sri Daya Shanker & his family, sister-in-law Ms. Vandana Thakur and brothers-in-law, whose unfailing love, affection, sincere prayers and best wishes had been a constant source of strength and encouragement. My parents-in-law really deserve special thanks because of always being in touch with me during my difficult period.

Words can hardly explain the co-operation and patience of my lovely son Ankit Aakash who has missed me so many days and nights for his studies and homeworks to be done. His enthusiastic welcome, unconditional love and affection kept me renewing my spirits. I have no words to express the appreciation for my beloved wife Mrs. Archana Sharma who stood by me at every moment encouraging me and keeping myself free from almost all the liabilities of home issues during my work. Her love, patience and devotion kept my motivation up and waning away, and completing the task.

I owe a debt of gratitude to my parents who brought me up to be a confident and well adjusted individual, and always believed in my ability to finish the things I set out to do. Last but not the least, I am thankful to the Almighty who gave me the strength and health for completing the work. I dedicate this work to my mother whose soul will feel very much proud of me. She deserves real credit for getting me this far, and no words can ever repay for her.

Udai Shanker

LIST OF ABBREVIATIONS

1PC	One Phase Commit
2PC	Two Phase Commit
2PL	Two Phase Locking
2SC	Double Space Commit
3PC	Three Phase Commit
AAP	Adaptive Access Parameter
ACID	Atomic, Consistent, Isolated, Durable
ACK	Acknowledgement
AD	Abort Dependency
ADS	Abort Dependency Set
AED	Adaptive Earliest Deadline
AEVD	Adaptive Earliest Virtual Deadline
AR	Transaction Arrival Rate
CCST	Communication among the Cohorts of Same Transaction
CD	Commit Dependency
CDS	Commit Dependency Set
CL	Coordinator Log
CPU	Central Processing Unit
CR	Conditional Restart
D2PL	Dynamic Two Phase Locking
DBS	Database System
DC	Data Contention
DDBMS	Distributed Database Management System
DDBS	Distributed Database System
DDCR	Deadline Driven Conflict Resolution
DDCR-S	Deadline Driven Conflict Resolution with Similarity
DRTDBS	Distributed Real Time Database System
ED	Effective Deadline
EDF	Earliest Deadline First
EP	Early Prepare
EQF	Equal Flexibility
EQS	Equal Slack

FCFS	First Come First Serve
FHR	Flexible High Reward
H2PL	High Priority Two Phase Locking
HF	Health Factor
HP	Higher Priority
I/O	Input/Output
IYV	Implicit Yes-Vote
LSF	Least Slack First
MECP	Memory Efficient Commit Protocol
MinHF	Minimum Health Factor
MM	Mixed Method
MP	Miss Percentage
NB-SPAC	Non-Blocking Single-Phase Atomic Commit
NL	Number of Locks
PA	Presumed Abort
PC	Presumed Commit
PCP	Priority Ceiling Protocol
PrN	Presumed Nothing
PROMPT	Permits Reading of Modified Prepared- Data for Timeliness
RC	Resource Contention
RTDBS	Real Time Database System
RTS	Real Time Systems
RT-S2PL	Real Time Static Two Phase Locking
S2PL	Static Two Phase Locking
S2PL-HP	Static Two Phase Locking with High Priority
SE	Serial Execution
SEQS	Static Equal Slack
SF	Slack Factor
SWIFT	Static Two Phase Locking with Higher Priority Based, Write-Update Type, Ideal for Fast and Timeliness Commit Protocol
SWIFT-PV-1	SWIFT- preliminary- Version- One
SWIFT-PV-2	SWIFT- preliminary- Version- Two
UD	Ultimate Deadline

LIST OF FIGURES

Figure No.	Caption	Page No.
Fig. 3.1	Distributed Real-time Database System Model	33
Fig. 3.2	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	44
Fig. 3.3	Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load	45
Fig. 3.4	Break-up of Miss % of Transactions at Communication Delay=100ms & System Load (50% Transactions of Size 3-5 & 50% Transactions of Size 20-25) in 2SC with EDF Priority Assignment Policy	46
Fig. 3.5	Break-up of Miss % of Transactions at Communication Delay=100 ms & System Load (50% Transactions of Size 3-5 & 50% Transactions of Size 20-25) in 2SC with EDF and Temporary Intermediate Priority Assignment Policy	46
Fig. 3.6	Miss % with (RC+DC) at Communication Delay=100ms and System Load (50% Transactions of Size 3-5 & 50% Transactions of Size 20-25)	47
Fig. 3.7	Miss % (RC+DC) at Communication Delay=100 and Transaction Arrival Rate=10 Transactions/Second	48
Fig. 3.8	Miss % (RC+DC) at Communication Delay=0ms & Transaction Arrival Rate=30 Transactions/Second	48
Fig. 4.1	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	68
Fig. 4.2	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	68
Fig. 4.3	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	69
Fig. 4.4	Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load	69
Fig. 4.5	Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load	70

Figure No.	Caption	Page No.
Fig. 4.6	Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load	70
Fig. 4.7	Miss % with (RC+DC) at Communication Delay=0ms Normal Load	71
Fig. 4.8	Miss % with (RC+DC) at Communication Delay=0ms Heavy Load	72
Fig. 4.9	Miss % with (RC+DC) at Communication Delay=0ms Normal Load	72
Fig. 4.10	Miss % with (RC+DC) at Communication Delay=0ms Heavy Load	73
Fig. 4.11	Miss % with (RC+DC) at Communication Delay=0ms Normal Load	73
Fig. 4.12	Miss % with (RC+DC) at Communication Delay=0ms Heavy Load	74
Fig. 4.13	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	74
Fig. 4.14	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	75
Fig. 4.15	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	75
Fig. 4.16	Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load	77
Fig. 4.17	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	78
Fig. 4.18	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	78
Fig. 4.19	Miss % with (RC+DC) at Communication Delay=0ms Normal Load	79
Fig. 4.20	Miss % with (RC+DC) at Communication Delay=0ms Normal Heavy Load	79

Figure No.	Caption	Page No.
Fig. 4.21	Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load	80
Fig. 4.22	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	81
Fig. 4.23	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	81
Fig. 4.24	Miss % with (RC+DC) at Communication Delay=0ms Normal Load	82
Fig. 4.25	Miss % with (RC+DC) at Communication Delay=0ms Heavy Load	82
Fig. 4.26	Break-up of Miss % with (RC+DC) at Communication Delay=100ms	83
Fig. 4.27	Break-up of Miss % with (RC+DC) at Communication Delay=0ms	84
Fig. 4.28	Break-up of Miss % with (RC+DC) at Communication Delay=100	85
Fig. 4.29	Break-up of Miss % with (RC+DC) at Communication Delay=0ms	85
Fig. 5.1	Miss % with (RC+DC) at Communication Delay=0ms Normal Load	98
Fig. 5.2	Miss % with (RC+DC) at Communication Delay=0ms Heavy Load	98
Fig. 5.3	Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load	99
Fig. 5.4	Miss % with (RC+DC) at Communication Delay=0ms & Transaction Arrival Rate=10 Transactions/Second	100
Fig. 5.5	Miss % with (RC+DC) at Communication Delay=100ms Transaction Arrival Rate=10 Transactions/Second	100

LIST OF TABLES

Table No.	Caption	Page No.
Table 3.1	Default Values for the Model Parameters	43
Table 5.1	Memory Requirement of 2SC and MECP	96

ABSTRACT

Distributed real time database systems (DRTDBSs) can be defined as database systems that support real time transactions. They are used for a wide spectrum of applications such as air traffic control, stock market trading, banking, telemedicine etc. In DRTDBS, there are two types of transactions: global and local. The global transactions are distributed real-time transaction executed at more than one site whereas the local transactions are executed at generating site only. A commonly model used for distributed real time transaction consists of a process, called coordinator, which is executed at the site where the transaction is submitted, and a collection of other processes called cohorts executing at various sites where the required data items reside.

Transactions in a real time database are classified into three types, viz. hard, soft and firm. The classification is based on how the application is affected by the violation of transaction time constraints. This thesis reports efficient solutions for some of the issues important to the performance of firm deadline based DRTDBS.

The performance of DRTDBS depends on several factors such as specification of transaction's deadline, priority assignment policy, scheduling transactions with deadlines, time cognizant buffer and locks management, commit procedure etc. One of the primary performance determinants is the policy used to schedule transactions for the system resources. The resources that are typically scheduled are processors, main memory, disks and the data items stored in database.

In order to resolve the contention for these resources, DRTDBSs have to establish a priority ordering among the cohorts. This ordering should minimize the percentage of missed transactions which is the primary performance metric, defined as percentage of input transactions that the system is unable to complete before their deadlines. We proposed a scheme where the priority of each cohort is determined independently on the basis of the locks required by it at its execution site. This is in contrast to earlier schemes where cohort inherits its real time priority from its parent. In our scheme, each cohort is assigned an initial priority based on the number of locks required by the cohort at its execution site. The cohort uses this priority for

central processing unit (CPU) scheduling. However, when there is a data contention between a low priority executing cohort and a high priority newly arrived cohort, temporary intermediate priorities of both are calculated. These intermediate priorities are based on the remaining execution time needed by the lock holding low priority cohort and the slack time available with the newly arrived higher priority cohort. The data contention is resolved on the basis of these priorities. The deadlines of the global and the local transactions are computed based on the formula developed.

DRTDBS implements a transaction commit protocol to ensure transaction atomicity. A commit protocol ensures that all participating sites agree on the final outcome of the transaction (commit or abort). The commit processing can result in a significant increase in transaction execution time due to exchange of multiple messages in multiple phases among the participating sites and maintaining several log records. This may influence the transaction miss percentage. Therefore, the design of commit protocol becomes an important performance issue in the design of DRTDBS. The existing commit protocols try to improve system performance by permitting a lock holding transaction (lender) to lend its data to other transactions (borrower). This creates dependencies between lender and borrower. We first redefine the dependencies created due to read/update type locks, and then propose a static two phase locking with higher priority based, write-update type, ideal for fast and timeliness commit protocol (named as SWIFT). We observe that, when communication delay is large, most of the transactions are aborted during their commit or locking phase; and not during their execution phase, particularly when database is main memory resident. Based on this observation, a cohort sends a WORKSTARTED message in SWIFT *before* the start of its execution in contrast to earlier protocol where cohort sends WORKDONE message *after* the completion of its execution. Our protocol also allows a dependent cohort to send WORKSTARTED message if the dependency is only commit dependency. The simulation results show that substantial gain in performance can be achieved using this protocol. The performance of SWIFT has also been analyzed for partial read-only optimization, which minimizes intersite message traffic, execute-commit conflicts and log writes thus resulting in a better response time. The impact of permitting communication

between cohorts (sibling) of the same transaction has also been examined both for the main memory and the disk resident database with and without communication delay.

Transaction processing generally requires large amount of main memory to store intermediate data. When the memory is running low, a transaction may be blocked or a new transaction may not be admitted in the system. Therefore, access methods and query processing algorithms must optimize memory space as well as processing time. Transaction processing requires concurrency control algorithm and commit protocol to maintain the consistency of data. So, memory requirements must be taken into consideration while, designing the concurrency control algorithms and the commit protocols. These protocols should be designed in a way to create less temporary data items to save the memory space. We propose a memory efficient real time commit protocol (MECP) based on a new locking scheme in which a lock not only shows the lock obtained by the lender but also the lock obtained by the borrower. Our protocol maintains only a single list compared to other commit protocols where each lender requires two lists to be maintained. Two types of write operation are defined: blind write and update. Based on this, dependencies that may arise by allowing a committing cohort to lend its data to an executing cohort have been redefined.

To summarized, this thesis proposes new methods for priorities assignment policies, transaction commitment and memory optimization. The simulation results show that the methods/protocols proposed in this thesis improve the performance of DRTDBS substantially.

CONTENTS

	Page No.
Candidate's Declaration	i
Acknowledgement	iii
List of Abbreviations	v
List of Figures	vii
List of Tables	xi
Abstract	xiii
Contents	xvii
CHAPTER 1: INTRODUCTION	
1.1 Introduction	1
1.2 Performance Issues and Research Challenges	2
1.3 Contributions of the Thesis	4
1.4 Organization of the Thesis	5
CHAPTER 2: BACKGROUND AND LITERATURE REVIEW	
2.1 Introduction	7
2.2 Distributed Database System	7
2.3 Distributed Real Time Database System	8
2.3.1 Distributed Real Time Transaction	9
2.3.2 Types of Distributed Real Time Transaction	10
2.3.3 Distributed Transaction Execution Model	10
2.3.4 Locking Mechanism	11
2.3.5 Issues in Distributed Real Time Database Systems	13
2.4 Priority Assignment Policy	14
2.4.1 Priority Assignment Policies in Centralized Environment	14
2.4.2 Priority Assignment Policies in Distributed Environment	16
2.5 Commit Protocols	18
2.5.1 Two Phase Commit Protocol (2PC)	18

2.5.2 Three-Phase Commit Protocol	22
2.6 Real Time Commit Protocols	22
2.7 Memory Optimization	27
2.8 Conclusions	27
CHAPTER 3: PRIORITY ASSIGNMENT POLICY	
3.1 Introduction	29
3.1.1 Issue of Fairness	29
3.1.2 Background	30
3.2 Distributed Real Time Database System Model	32
3.2.1 System Model	33
3.2.2 Database Model	34
3.2.3 Network Model	34
3.2.4 Cohort Execution Model	34
3.2.5 Locking Mechanism	35
3.2.6 Model Assumption	35
3.3 Deadline Assignment	36
3.4 Double Space Commit Protocol (2SC)	37
3.5 Priority Assignment Heuristic & Temporary Intermediate Priority Assignment Policy	39
3.6 Performance Evaluations	42
3.6.1 Performance Parameters and Measures	42
3.6.2 Simulation Results	43
3.7 Conclusions	49
CHAPTER 4: SWIFT - A DISTRIBUTED REAL TIME COMMIT PROTOCOL	
4.1 Introduction	51
4.2 Background and Related Work	51
4.3 Distributed Real Time Database System Model	53
4.3.1 System Model	54
4.3.2 Model Assumptions	54

4.4	Data Access Conflicts Resolving Strategies	55
4.4.1	Types of Dependencies	57
4.4.2	Type of Dependencies in Different Cases of Data Conflicts	58
4.4.3	Mechanism of Interaction between Lender and Borrower Cohorts	59
4.5	A New Commit Protocol SWIFT	60
4.5.1	Basic Idea	61
4.5.2	Algorithm	63
4.6	Performance Measures and Evaluation	65
4.6.1	Performance Parameters and Measures	65
4.6.2	Simulation Results	66
4.6.2.1	Main Memory Resident Database	67
4.6.2.2	Disk Resident Database	71
4.7	Performance of SWIFT with Partial Read Optimization	76
4.7.1	Possible Cases of Data Conflicts	76
4.7.2	Type of Dependencies in Cases of Data Conflicts	76
4.7.3	Simulation Results	77
4.8	Communication among the Cohorts of Same Transaction	80
4.9	Impact of Early Sending of WORKSTARTED Message	83
4.9.1	Main Memory Database with Communication Delay of 100ms	83
4.9.2	Main Memory Database with Communication Delay of 0ms	84
4.9.3	Disk Resident Database with Communication Delay of 100ms	84
4.9.4	Disk Resident Database with Communication Delay of 0ms	85
4.10	Conclusions	86

CHAPTER 5: MECP-A MEMORY EFFICIENT REAL TIME COMMIT ROTOCOL

5.1	Introduction	87
5.2	New Locking Scheme and Data access Conflict Resolving Strategies	89
5.2.1	New Locking Scheme	89
5.2.2	Data access Conflicts Resolving Strategies	90
5.2.3	Mechanics of Interaction between Lender and Borrower Cohorts	91
5.3	Algorithm	92

5.4	Memory Optimization	94
5.4.1	Case 1: Memory Requirement in 2SC	94
5.4.2	Case 2: Memory Requirement in Proposed Scheme (MECP)	95
5.5	Model Parameters, Simulation Results and Performance Evaluation	97
5.5.1	Impact of Transaction Arrival Rate	97
5.5.2	Impact of Transaction Size	99
5.6	Conclusions	101
CHAPTER 6: CONCLUSIONS AND SCOPE FOR FUTURE RESEARCH		
6.1	Conclusions	103
6.2	Scope for Future Research	106
REFERENCES		109
Author's Research Publication		123

INTRODUCTION

1.1 INTRODUCTION

Many applications such as military tracking, medical monitoring, stock arbitrage system, network management, aircraft control and factory automation etc. that depend heavily on database technology for the proper storage and retrieval of data located at different remote sites have certain timing constraints associated with them [5,25,30,68,114]. Such applications introduce the need for distributed real time database systems (DRTDBSs) [84]. A DRTDBS is a collection of multiple, logically interrelated databases distributed over a computer network [23]. They support transactions that have explicit timing constraints. The timing constraint of a transaction is expressed in the form of a deadline, which indicates that it must complete before some specific time in future [4,5,68,114]. The transactions can be classified as hard, firm or soft type based on the effect of missing their deadlines [5,68]. A hard real time transaction must meet its deadline strictly. A missed deadline may result in a catastrophe [15,114]. A firm real time transaction does not result in a catastrophe, if the deadline is missed [51]. However, the results have no value after the expiry of deadline. A soft real time transaction has some value even after expiry of its deadline, but the value typically diminishes with time [5,68,114].

In contrast to traditional databases, where the primary goal is to minimize the response time of transactions and maximize throughput [120,151], the main objective of DRTDBS is to minimize the percentage of the transactions that miss their deadlines [4,5,53,68,114]. The scheduling of real time transaction is far more complex than traditional real time scheduling as the database management algorithms for accessing and manipulating data in DRTDBS should not only ensure database consistency, but should also satisfy the timing constraints. The goal of this chapter is to introduce various aspects of DRTDBS, the issues and challenges involved and the work carried out in this thesis.

This chapter explores the basic issues and research challenges having key importance to the performance of DRTDBS followed by the contributions and organization of this thesis.

1.2 PERFORMANCE ISSUES AND RESEARCH CHALLENGES

The implementation of DRTDBS is difficult due to the conflicting requirements of maintaining data consistency and meeting transaction's deadlines [97,148]. The difficulty comes from the unpredictability of the transactions' response times [70,71,114]. Each distributed transaction accessing a data item takes a variable amount of time due to concurrency control, I/O and communication delays. While maintaining the consistency of underlying database, scheduling and management of the system resources in DRTDBS should also take into account the timing constraints. Access to CPU, main memory, I/O devices and shared data should be managed to make the best effort to satisfy the transaction deadlines.

One of the most important issues in design of DRTDBS is transaction scheduling [18,68,88]. The transaction scheduling in DRTDBS involves both the CPU scheduling and the data scheduling and is done according to the priorities assigned to the transactions. As a result, the role of the priority assignment policy becomes an important issue in deciding the performance of the system because priorities determine the order of the transactions to access resources which in turn affects their likeliness to meet the deadlines. In traditional databases, when conflicts occur, the preferences tend to be based either on fairness or on resource consumption [151]. However, the transaction scheduling in DRTDBS is done according to the urgency of the transactions that decides their priorities. The priority assignment problem has been addressed by very few researches [86]. Generally, the priority of a transaction is determined on the basis of its deadline such as in earliest deadline first (EDF) [85,93] priority assignment policy; both fairness and maximum resource utilization become secondary goal [130]. This can cause two problems. First, more CPU resource is wasted if closer to completion transactions are aborted in favor of higher priority transactions [68]. Second, longer transactions may be harder to finish creating a starvation problem [63]. Execution of a global transaction in a distributed system requires the execution of cohorts on different sites. Most heuristics [66,67,85] for priority assignment in DRTDBS consider that subtasks (cohorts) of a transaction are executed sequentially. Except ultimate deadline (UD), other heuristics are not suitable when the subtasks (cohorts) of a transaction are executed in parallel. The UD also becomes ineffective when data contention is non-trivial [85].

The atomic commit protocols play a key role in supporting global atomicity for the distributed real time transactions [9,13,82,83,101]. These protocols are used to ensure that all cohorts agree on the final outcome of the transaction. They typically require exchange of multiple messages [127] in multiple phases among the participating sites, and also require to maintain logs of data to recover from failures [59,80,117]. This significantly increases the execution time of the transactions and can adversely affect the system's ability to meet transaction deadlines. Due to distributed nature of the transactions and in presence of other sources of unpredictability such as data access conflicts, uneven distribution of transactions over the sites, variable local CPU scheduling time, communication delay, failure of coordinator and cohort's sites etc., it is not easy to meet the deadline of all transactions in DRTDBS. The unpredictability in the commitment phase makes it more serious because the blocking time of the waiting cohorts due to execute-commit conflict may become longer. Hence, due to unique characteristics of the committing transactions [81] and unpredictability in the commitment process, design of an efficient commit protocol is another important issue that affects the performance of DRTDBS [49].

Important database system resources are main memory, CPU, disk and data items [20,104,105]. Before the start of execution of a transaction, buffer space in main memory is allocated for the transaction [62]. When the main memory is running low, a transaction may be blocked from execution. The amount of memory available in the system thus limits the number of concurrently executable transactions. In large scale real time database systems, the execution of transactions will be significantly slowed down if available memory is low. So, the effective use of available main memory space in data intensive applications is another challenging issue. During the execution of a transaction, temporary records are created to maintain the status of the transaction's execution. These temporary records are kept in the main memory until the transaction commits. This consumes a substantial amount of main memory. Since, unpredictability in the commitment phase may make the transaction to stay for a long period in the system; memory will be held up for a long period and will be not available for other transactions. So, this necessitates the design of commit protocols that save memory by creating less temporary objects.

The design and implementation of DRTDBS introduce several other interesting problems. Among these problems, predictability and consistency are fundamental to real time transaction processing, but sometimes these require conflicting actions [70,84,124]. To ensure consistency, we may have to block certain transactions. Blocking of these transactions, however, may cause unpredictable transaction execution and may lead to the violation of timing constraints. There are a number of other sources of unpredictability such as communication delays, site failures [22] and transaction's interaction with the underlying operating system and I/O subsystems [16]. Other design issues of DRTDBS are data access mechanism and invariance, new metrics for database correctness and performance, maintaining global system information, security, fault tolerance, failure recovery etc. Again, there is also no adequately designed technique for scheduling the CPU as being the primary resource in the DRTDBS [6].

Although, a lot of research has been done on these issues, there still exist many challenging and unresolved issues. Due to the heterogeneity of these issues, we have confined our work to only some of these issues. Our work involves design of new priority assignment policies and commit protocols and the comparison of their performance with existing policies/protocols. We assumed that the transactions are firm real time and data items accessed by the transactions are known before the start of execution of the transactions. Two locking approaches are used by the transactions to obtain a lock on data items viz., static two phase locking (S2PL) and dynamic two phase locking (D2PL). The deadlock freedom and lower communication overhead of locking by using S2PL makes it attractive for DRTDBS [73,75,132]. So, we used S2PL with higher priority concurrency control algorithm to access data in mutually exclusive way.

1.3 CONTRIBUTIONS OF THE THESIS

The work reported in this thesis provides better solutions for some of the issues mentioned above. Major contributions of this thesis may be described as follows:

- (i) Development of a simulator for both the main memory resident and the disk resident DRTDBS.

- (ii) A new scheme to determine the priorities of cohorts executing in parallel along with the method to compute the deadlines of the global and the local transactions have been proposed. In our scheme, each cohort is assigned an initial priority which is inversely proportional to the number of locks required by the cohort at its execution site. A temporary intermediate priority of the cohort is calculated when a data contention occurs and initial priority of newly arrived cohort (T_A) is higher than the priority of lock holding cohort (T_L). The intermediate priorities are based on the remaining execution time needed by T_L and the slack time available with T_A . This minimizes the abort of near completion low priority lock holding cohorts. The proposed scheme has been compared with EDF priority assignment policy
- (iii) The dependencies created due to read/update type locks have been redefined, and then a static two phase locking with high priority (S2PL-HP) based commit protocol named as SWIFT has been proposed. The performance of SWIFT has been compared with 2SC and PROMPT for both main memory resident and disk resident databases with and without communication delay. The performance of SWIFT protocol has also been analyzed (i) for partial read-only optimization and (ii) for the case when cohorts of the same transaction (siblings) are allowed to send messages to each other.
- (iv) A new locking scheme has been developed for the database model that permits two types of write operations: blind write and update. All types of dependencies that may arise by allowing a committing cohort to lend its data to an executing cohort have been redefined. A memory efficient commit protocol (MECP) has been proposed on the basis of the new locking scheme, and its performance has been compared with 2SC and PROMPT.

A list of the author's research publication is given at the end of the thesis.

1.4 ORGANIZATION OF THE THESIS

Chapter 2 of the thesis reviews the past and ongoing work in the areas of priority assignment policies, commit protocols and effective use of available main

memory space in data intensive applications. The design of new priority assignment policies for the allocation of CPU and data to cohorts has been discussed in chapter 3. Chapter 4 deals with the design and performance evaluation of SWIFT, a new commit protocol. Chapter 5 of the thesis describes the design of a memory efficient commit protocol (MECP) to reduce the storage space requirement for intermediate (temporary) records and compares its performance with other commit protocols. Finally, chapter 6 concludes the thesis and proposes some research directions.

BACKGROUND AND LITERATURE REVIEW

2.1 INTRODUCTION

Databases and database systems have become an essential component of everyday life in modern society. In the course of a day, most of us encounter several activities that involve some interaction with databases. Nowadays, because of the information technology revolution, fast access to information and its efficient management are key to the success of any activity such as business and other similar ones [50,60,99]. Today's business applications are not the old-styled batch applications; rather they do their data processing activities on-line [123]. Modern electronic services and electronic commerce applications, characterized by high volume of transactions, cannot survive without on-line support of computer systems and database technology [7]. Therefore, database systems (DBSs) play an important role in managing the information of fast growing current businesses environment.

In this chapter, we review the current status of research in the area of DRTDBS and identify important performance issues which are unique to their study.

2.2 DISTRIBUTED DATABASE SYSTEM

DBS can be viewed as a collection of the data items which are shared by many users [109,140]. They are designed to manage huge amount of the data. The management of data basically involves the definition of structures for its storage and provision of mechanisms for manipulation of this stored information. Thus, a DBS is a collection of objects, which satisfy the need of users besides a set of integrity constraints. Database Systems can be broadly classified as centralized or distributed. The centralized database systems are those that run on a single computer system. Such systems may range from single-user database systems running on personal computers to high-performance database systems running on mainframe systems. The distributed database systems (DDBS) consist of a collection of sites, connected together via some means of communication networks, in which, each site is a database system site in its own right but the

sites have agreed to work together, so that a user at any site can access data from anywhere in the network, exactly as if, the data are all stored at the user's own site [34]. We can define DDBS as a collection of multiple logically interrelated databases distributed over a computer network, and a distributed database management system (DDBMS) which manages distributed databases while making the distribution transparent to the user [147]. DDBS bring the advantages of distributed computing to the database management domain and fit more naturally in the decentralized structure of many business organizations.

2.3 DISTRIBUTED REAL TIME DATABASE SYSTEM

Real Time systems (RTS) are those for which correctness depends not only on the logical properties of the produced results, but also on the temporal properties of these results [7,118,149]. Typically, real time systems are associated with critical applications, in which human lives or expensive machineries may be at stake [79,87]. Hence, in such systems, an action performed too late (or too early) or a computation which uses temporally invalid data may be useless and sometimes harmful even if such an action or computation is functionally correct. As RTS continue to evolve, their applications become more and more complex, and often require timely access and predictable processing of massive amounts of real time data [144]. The database systems, which are especially designed for the efficient processing of these types of real time data, are referred to as distributed real-time database systems (DRTDBS). DRTDBS can be viewed as an amalgamation of the conventional DDBS and RTS, and like a conventional DDBS, it has to process distributed transactions and guarantee their basic correctness criteria [64,16,128]. Thus, DRTDBS are collection of multiple, logically interrelated databases distributed over a computer network where transactions have explicit timing constraints, usually in the form of deadlines [90,91]. In such a system, data items shared among transactions are spread over remote locations. Accessing the shared data items must be controlled in order to maintain database's logical consistency by applying a concurrency control algorithm. At the same time, transactions have to be scheduled according to their timeliness to finish within their timing constraints, i.e., transaction processing in DRTDBS must satisfy both database logical consistency and timing constraints. What makes DRTDBS different from a conventional real-time system is the requirement of preserving the

consistency of data besides considering the real-time requirements of the transaction [14]. Satisfying the timing constraints of various real time activities in distributed systems may be difficult due to the distributed nature of the transactions.

2.3.1 Distributed Real Time Transaction

When the user programs for interaction with database, partially ordered sets of read and write operations are generated [36]. This sequence of operations on the database is called a transaction and it transforms the current consistent state of the database system into a new consistent state. In DRTDBS, there are two types of transactions: global and local. Global transactions are distributed real-time transaction executed at more than one site whereas the local transactions are executed at the generating site only. A common model of a distributed transaction is that there is one process called coordinator which is executed at the site where the transaction is submitted and a set of other processes called cohorts that execute on behalf of the transaction at other sites that are accessed by the transaction.

In fact, the distributed real time transaction processing is a form of transaction processing that supports transactions whose operations are distributed among different computers or among different databases from different vendors. So, in a distributed real time transaction, the operations are executed at the site where the required data item resides and is associated with time constraints. Transfer of money from one account to another, reservation of train tickets, filing of tax returns, entering marks on a student's grade sheet etc. are some of the examples of distributed real time transactions. The transaction is an atomic unit of work, which is either completed in it's entirely or not at all. Hence, a distributed commit protocol is needed to guarantee the uniform commitment of distributed transaction execution [115]. The commit operation implies that the transaction is successful, and, hence all of its updates should be incorporated into the database permanently. An abort operation indicates that the transaction has failed, and hence, requires the database management system to cancel or abolish all of its effects in the database system. In short, a transaction is an "all or nothing" unit of the execution.

2.3.2 Types of Distributed Real Time Transaction

In general, Real time transactions are classified into three types namely hard, soft and firm. No hard real time transaction should have its deadline missed, and its deadline must be guaranteed by the system. On the other hand, any deadline violations of the soft real time transactions may only result in the performance degradation of the system. The major performance metric for the soft real time transaction is the number or percentage of deadline violations or their average or worst case response time. The firm real time transactions are a special kind of soft real time transactions except that firm real time transactions will be killed when their deadline expire. The performance metrics is the number or percentage of deadline violations. The completion of a real time transaction might contribute a value to the system. The relationship between the value imparted by a real time transaction and its completion time can be considered as a value function of the time. After the soft real time transaction misses its deadline, its value might decrease with time. A firm real time transaction loses its value after its deadline expires. When a hard real time transaction misses its deadline, its value becomes negative. It means that a catastrophe might occur.

2.3.3 Distributed Transaction Execution Model

There are two types of distributed transaction execution model, i.e., sequential and parallel [4,14]. In the sequential execution model, there can be at most one cohort of a transaction at each execution site, and only one cohort can be active at a time. After the successful completion of one operation, next operation in the sequence is executed by the appropriate cohort. At the end of execution of the last operation, the transaction can be committed. In the parallel execution model, the coordinator of the transaction spawns all cohorts together and sends them for execution on respective sites [14]. All cohorts then execute in parallel. The assumption here is that the operations performed by one cohort during its execution at one site are independent of the results of the operations performed by some other cohort at some other site. In other words, the sibling cohorts do not require any information from each other to share [144].

2.3.4 Locking Mechanism

One of the fundamental properties of a transaction is isolation. When several transactions execute concurrently in the database, the isolation property must be preserved. To ensure this, the system must control the interaction among the concurrent transactions; this control is achieved through concurrency control schemes [109,140]. Some of the main concurrency control techniques such as two phase locking (2PL) are based on the concept of locking of data items. Locks are used to ensure the noninterference property of concurrently executing transactions and to guarantee serializability of the schedules. A transaction is said to follow the two-phase locking protocol, if all locking operations precedes the first unlock operation in the transaction. Such a transaction can be divided into two phases: an expanding or growing (first) phase, during which new locks on data items can be acquired but none can be released; and a shrinking (second) phase, during which existing locks can be released but no new locks can be acquired [95]. It ensures serializability, but not deadlock freedom. The two phase locking can be static or dynamic. The working principle of static two phase locking (S2PL) is similar to dynamic two phase locking (D2PL) except for the procedure of setting locks [73]. In D2PL, transactions acquire locks to access data items on demand and release locks upon termination or commit [133]. In S2PL, the required locks of a transaction are assumed to be known before its execution [136]. The prior knowledge of the required data items by a transaction is easy to address in DRTDBS as it is generally agreed that the behavior and the data items to be accessed by real-time transactions, especially hard real-time transactions, are much more well-defined and predictable. So, as a result of the better defined nature of real time transactions, it is not uncommon to assume that the locking information of a transaction is known before its processing. For example, in priority ceiling protocol (PCP), the locks required by the transactions must be known before their arrivals with predefined priorities [119]. A transaction has to obtain all its required locks before the start of execution. If any one of its locks is being used by another transaction, it is blocked and releases all seized locks. The locks to be accessed by a transaction at each site can be packed into a single message for transmission. In DRTDBS, the number of messages for setting the locks is generally smaller for distributed S2PL than for D2PL. So, the number of messages and the time delay for remote locking can be significantly reduced.

There is no local deadlock and a distributed deadlock is much easier to resolve with S2PL than with D2PL. S2PL protocol is deadlock free [121] because blocked transactions cannot hold locks. In the last two decades, a lot of work has been done to compare D2PL with S2PL [86]. Most researchers agree that D2PL is a better choice for conventional non-real-time database systems than S2PL because of the followings reasons [65].

- (i) Smaller probability of lock conflicts due to the shorter average lock holding time in D2PL; and
- (ii) Difficulty in determining the required locks before the processing of transaction

However, the meaning of “better” performance in DRTDBS is quite different from that in conventional non-real-time database systems. In the conventional database systems, the major performance measures are mean system throughput and mean system response time. On the contrary, minimizing the number of missed deadlines is the main concern in DRTDBS.

In non-real-time S2PL, a transaction is blocked if any of its required locks are seized in conflicting mode by any other transactions. While it is being blocked, some of its required locks which may be free initially, can be seized by other transactions. Thus, even when the original conflicting locks are released, the transaction may be blocked by other transactions which arrive after it. So, the blocking time of higher priority transaction can be arbitrarily long due to prolonged blocking as a result of waiting for multiple locks [75]. An alternative for concurrency control in DRTDBS is to use real time S2PL (RT-S2PL). In RT-S2PL, each lock in the database is defined with a priority equal to the priority of the highest priority transaction which is waiting for that lock. All the locks of the data items to be accessed by a transaction have to be set in appropriate modes before processing of the transaction. If any of the required locks is in a conflicting mode or has a priority higher than that of the requesting transaction; none of the required locks will be set and the transaction will be blocked instead. However, for the locks with lower priorities, their priorities will be updated to that of the requesting transaction. These features of RT-S2PL make it attractive for DRTDBS [73,75,132]. In RT-S2PL protocols, the problem of locking-induced thrashing can

be prevented because lock requesting transactions can be blocked due to a lock conflict [137,138,139].

Based on S2PL and H2PL, static two phase locking with high priority (S2PL-HP) is used in the present work. Here, the lock requesting cohort waits for the data item to be released when the requested data item is held by one or more higher priority cohorts in a conflicting mode. On the other hand, if the data item is held by a lower priority cohort in a conflicting way, the lower priority cohort is aborted and requesting cohort is granted the desired locks.

2.3.5 Issues in Distributed Real Time Database Systems

The time expressed in the form of a deadline is a critical factor to be considered in distributed real time transaction [102]. The completion of transactions on or before its deadline is one of the most important performance objectives of DRTDBS. There are several important factors that contribute to the difficulty in meeting the deadlines of a distributed transaction [38]. One of the most significant factors is the data conflict among transactions [26]. The data conflict that occurs among executing transactions is referred to as executing-executing conflict. The conflict involving executing-committing transactions is termed as executing-committing conflict. The executing-executing conflicts are resolved by distributed concurrency control protocols. A number of real time concurrency control protocols have been proposed in the past. When a data conflict occurs between an executing and a committing transaction, a commit protocol has to work with concurrency control protocol to handle it and to ensure the transaction atomicity. The traditional commit protocols block the lock requesting transactions until the lock holding transaction releases the lock. The blocked transactions may seriously affect the performance of DRTDBS, especially when failures occur during the commitment phase.

Some of the other issues in DRTDBS are scheduling of distributed transactions, optimizing the use of memory, management of distributed transactions, deadline assignment strategies, difficulty in maintaining global system information, possibility of distributed deadlocks etc. [17]. Among these issues, priority assignment policy for transaction scheduling, commit protocol and memory optimization are the only issues considered in this thesis. In the following sections, we will review the literature which addresses these issues.

2.4 PRIORITY ASSIGNMENT POLICY

Usually, a real time database system is a part of a large and complex real time system. The tasks in RTS and transactions in DRTDBS are similar in the sense that both are units of work as well as units of scheduling [61,70,71]. However, tasks and transactions are different computational concepts and their differences affect how they should be scheduled and processed. Unlike transactions, tasks in real time systems do not consider consistency of the data items used. Though many real time task scheduling techniques are still used for scheduling real time transactions, the transaction scheduling in real time database systems needs a different approach than that of which is used in scheduling tasks in the real time systems. The following sub sections next to it review the literature on task/transaction scheduling in centralized and distributed environments respectively.

2.4.1 Priority Assignment Policies in Centralized Environment

The priority assignment techniques proposed for centralized real time systems can be broadly classified into three categories: static, dynamic or hybrid of both. A scheduling algorithm is said to be static, if priorities are assigned to tasks once and for all. A scheduling algorithm is said to be dynamic, if the priority of a task changes from request to request. Liu C. L. and Layland J. W. have developed a rate monotonic static assignment scheme to determine the schedulability of a set of periodic tasks [93]. One of the most used algorithms belonging to this class is Earliest Deadline First (EDF), according to which, priorities assigned to tasks are inversely proportional to the absolute deadlines of active jobs where deadline of a job depends on the arrival time of its next occurrence. A scheduling algorithm is said to be a mixed scheduling algorithm if the priorities of some of the tasks are fixed and priorities of the remaining tasks vary from request to request.

Most concurrency controllers block or restart transactions when data conflicts are detected. The victim selection policy is based on the rules of the specific concurrency controller. The traditional no priority based concurrency control algorithm penalizes the transaction that requests the lock last [142]. The cohort remains blocked until the conflicting lock is released. The real time priority of the cohort is not considered in processing the lock request. Recent studies show that the system performance can be significantly improved by using priority based

scheduling. The transaction with closest deadline is assigned highest priority. It is called EDF, initially designed for real time tasks scheduling. If the two cohorts have same deadline, one with earlier arrival time is assigned a higher priority on the basis of first come first serve (FCFS). Other priority assignment policies are Random and Least Slack First (LSF) [2,141]. The random priority assignment policy assigns priority to each transaction on a random basis and the priority assigned to transaction is independent of transaction deadline. In the LSF policy, the transaction with less slack time will have higher priority. The slack time can be defined as the amount of time the cohort can afford to wait in order to complete before its deadline [145].

The performance of different scheduling policies for the soft deadline based transactions was first addressed by Abbot R. and Garcia-Monila H. [2]. They studied the performance of three priority assignment techniques; FCFS, EDF and LSF, with different concurrency control methods namely serial execution (SE), high priority (HP), and conditional restart (CR) through simulation. The pioneering work in RTDBS performance evaluation of various scheduling options for a real time database system with disk and shared locks is reported again by Abbot R. and Garcia-Monila H. [1]. The scheduling algorithms used for this study are FCFS, EDF and LSF along with the concurrency control algorithms such as wait, wait-promote, high priority & conditional restart.

Pang H. et al. investigated the problem of "bias" against longer transactions under "earliest-deadline-based" scheduling policies in a centralized RTDBS [104,105]. Their approach to solve the problem of bias assigns virtual deadlines to all transactions. A transaction with an earlier virtual deadline is served before one with a later virtual deadline. The virtual deadline of a transaction is adjusted dynamically as the transaction progresses and is computed as a function of the size of the transaction.

In a real-time database system, an application may assign a value to a transaction to reflect the return it expects to receive if the transaction commits before its deadline [51,52]. Haritsa J. R. et al. addressed the problem of establishing a priority ordering among transactions characterized by both values and deadlines that results in maximizing the realized value. They proposed the Adaptive Earliest Deadline (AED) protocol for priority assignment as well as for load control of the transactions [56]. AED was later improved to Adaptive Earliest

Virtual Deadline (AEVD) policy using virtual deadline based on both arrival time and deadline. Datta et al. addressed some of the weaknesses in AEVD, and proposed the Adaptive Access Parameter (AAP) method for explicit admission control [25].

Dogdu Erdogan and Ozsoyoglu Gultekin proposed new priority assignment and load control policies for repeating real-time transactions [31]. Based on the execution histories of the transactions, they showed that a widely used priority assignment technique EDF is biased towards scheduling short transactions favorably and proposed protocols that attempt to eliminate the discriminatory behavior of EDF by adjusting the priorities using the execution history information of transactions. They introduced the notion of “fair scheduling” of transactions in which, the goal was to have “similar” success ratios for all transaction classes (short to long in size).

2.4.2 Priority Assignment Policies in Distributed Environment

In DRTDBS, a transaction is generally divided into several sub transactions (cohorts). These cohorts execute on different sites. The system performance is heavily dependent on the local scheduling of the cohorts at different sites. It has been shown by Kao B. and Garcia-Monila H. that the distributed real time system performance, in terms of meeting task deadlines, can be improved by assigning appropriate priority to the sub-tasks of a task [66]. They suggested four different heuristics, i.e., ultimate deadline (UD), effective deadline (ED), equal slack (EQS) and Equal flexibility (EQF) for assigning the deadline to sub-tasks [66]. These heuristics consider only real time constraints and may not be suitable for DRTDBS as they do not consider their impact on data contention which can seriously affect the system performance. Lee Victor C. S. et al. examined the performance of these four heuristics and suggested three other alternatives that take into consideration the impact of data conflicts [85]. These alternative priority assignment strategies are Number of Locks held (NL), Static EQS (SEQS) and Mixed Method (MM). The NL strategy assigns the priority to cohorts on the basis of the number of locks being held by its parent transaction while other two heuristics are improved version of the heuristics discussed by Kao B. and Garcia-Monila H. [66]. However, both of the above studies consider sequential executions

of task/cohorts. These heuristics, except UD, are not suitable for cohorts executing in parallel.

Complex distributed tasks often involve parallel execution of the subtasks at different nodes. To meet the deadline of a global task, all of its parallel subtasks have to finish in time. In comparison to a local task (which involves execution at only one node), a global task may find it much harder to meet its deadline because it is fairly likely that at least one of its subtasks run into an overloaded node. Another problem with complex distributed tasks occurs when a global task consists of parallel and serial subtasks. If one parallel subtask is late, then the whole task is late. The problem of assigning deadlines to the parallel and the serial subtasks of the complex distributed tasks is addressed by Kao B. and Garcia-Monila H. [67]. They studied the problem of automatically translating the deadline of a real time activity to deadlines for all its sequential and parallel subtasks constituting the activity. Each sub task deadline is assigned just before the sub task is submitted for execution. The structure of the complex tasks is assumed to be known in advance. To meet the deadline of a global task, the scheduler must estimate the execution times of the subtasks and assign them to processors in such a way that all will finish before the deadline of the global task. A number of strategies for assigning a deadline to each parallel subtask have been proposed. The strategies have also been proposed for assigning deadlines to sequential subtasks. The problem of assigning deadlines to parallel and serial subtasks of complex distributed tasks in a real time system has been studied through simulation. The empirical results are provided for assigning virtual deadlines to parallel subcomponents of a task in order to meet the global task deadline.

Lam K. Y. et al. investigated the effects of different priority assignment heuristics using optimistic concurrency control protocol and high priority two phase locking [77,86]. The results of their performance experiments show that optimistic concurrency control protocols are more affected by the priority assignment policies compared to locking based protocols. It was also shown that considering both transaction deadline and current data contention in assigning transaction priorities provides best performance among a variety of priority assignment techniques.

Traditional real time schedulers do not consider the impact of communication delay for transferring the remote data and results. To reduce the miss percentage of transactions and the wastage of time for remote transaction due to communication delay, a new real time scheduler called Flexible High Reward (FHR) is proposed by Chen Hong-Ren et al. [22]. FHR tries to reduce the miss percentage by giving slightly high priority to remote transactions.

Most of the previous work on priority assignment focuses either on centralized database or on distributed databases where subtasks (cohorts) of transaction are executed sequentially. Very few researches have considered the scheduling of distributed cohorts executing in parallel [59].

2.5 COMMIT PROTOCOLS

A distributed transaction is executed at more than one site. In such an environment, the transaction may decide to commit at some sites while at some other sites it could decide to abort resulting in a violation of transaction atomicity [100,109]. To overcome this problem, distributed database systems use a distributed commit protocol which ensures the uniform commitment of the distributed transaction, i.e. all the participating sites agree on the final outcome (commit/abort) of the transaction [8,83]. Commit protocol ensures that either all the effects of the transaction persist or none of them persist despite of the site or communication link failures and loss of messages.

2.5.1 Two Phase Commit Protocol (2PC)

The two phase commit protocol (2PC) referred to as the Presumed Nothing 2PC protocol (PrN) is the most commonly used protocol in the study of DDBS [39,40]. It guarantees uniform commitment of distributed transactions by maintaining the logs and by exchanging explicit multiple messages among the sites. In the absence of failures, the protocol is straight forward in that a commit decision is reached if all cohorts are ready to commit; otherwise, an abort decision is taken.

Assuming no failures, it works as follows [127].

- (i) The coordinator sends a vote request (VOTE REQ) message to all cohorts.

- (ii) When a cohort receives a VOTE REQ, it responds by sending a yes or no vote message (YES or NO VOTE, YES VOTE also known as PREPARED message) to the coordinator. If the cohort's vote is NO, it decides to abort and stops.
- (iii) The coordinator collects the vote messages from all cohorts. If all of them are YES and the coordinator's vote is also YES, the coordinator decides to commit and sends commit messages (COMMIT) to all cohorts. Otherwise, the coordinator decides to abort and sends abort messages (ABORT) to all cohorts that voted YES.
- (iv) Each cohort that voted YES waits for a COMMIT or ABORT message from the coordinator. When it receives the message, it decides accordingly, sends an acknowledgement message (ACK) to the coordinator and stops.
- (v) Finally, the coordinator, after receiving the ACK messages from all the cohorts, writes an end log record and then forgets the transaction.

2PC satisfies the aforementioned listed rules for atomic commitment as long as failures do not occur. However, if due to some reason the communication between cohorts and the coordinator is distracted, it is possible that the cohort is in uncertain state. It can neither abort nor commit since it does not know what the other cohorts and the coordinator have decided. The cohort is in a blocked state and may wait as long as the failure is corrected. Unfortunately, this blocking may continue for an indefinitely long period of time. To handle this situation, 2PC ensures that sufficient information is force-written on the stable storage to reach a consistent global decision about the transaction [98]. Hence, 2PC imposes a great deal of overhead on the transaction processing. There has been a lot of research to mitigate the overhead of 2PC and a number of 2PC variants have been proposed. These variants can be classified into following four groups [82].

(i) Presumed Abort/Presumed Commit Protocols

A variant of 2PC protocol, called Presumed Abort (PA), attempts to reduce 2PC overheads by requiring all cohorts to follow a "in the no information case, abort" rule [10,96,127,98,82]. If after coming up from a failure, a site queries the coordinator about the final outcome of a transaction and finds no information available with the coordinator, the transaction is assumed to be aborted. The

coordinator of the transactions does not log information nor wait for ACK message regarding aborted transactions. Hence, PA behaves identical to 2PC for committing transactions, but has reduced message and logging overheads for aborted transactions.

In general, the number of committed transactions is much more than the number of aborted transactions [11,89]. Based on this observation, another variant of 2PC i.e. the Presumed Commit (PC) is proposed that attempts to reduce the messages and logging overheads for committing transactions rather than aborting transactions. The coordinator interprets missing information about transactions as commit decision. Unlike PA, the coordinator has to force write starting of the voting phase. This is to ensure that an undecided transaction is not presumed as committed when the coordinator recovers from a failure.

(ii) One Phase Commit Protocols

The one-phase commit (1PC) protocol has been first suggested by Skeen D. [40]. There is no explicit "voting phase" to decide on the outcome of the transaction because cohorts enter in "prepared state" at the time of sending the work done message (WORKDONE) itself, i.e., the commit processing and data processing activities are effectively overlapped. By eliminating an entire phase and thereby reducing commit processing overheads and durations, 1PC protocols seem to be potentially capable of completing more transactions before their deadlines.

Several variant of 1PC have been proposed in the literatures. The Early Prepare (EP)-a representative 1PC protocol forces each cohort to enter in a PREPARED state after the execution of each operation [127]. It makes cohort's vote implicit and exploits the feature of 1PC. The Coordinator Log (CL) protocol avoids the cohort's disk blocking time by centralizing the cohort's log on the coordinator [127]. However, this violates site autonomy and can be practical only when the entire distributed system sits in a single machine room rather than several geographically distant locations connected by communication lines. The Implicit Yes-Vote (IYV) protocol adapts the CL in the case of gigabit-networked database systems. The IYV protocol allows failed participants to perform part of the recovery procedure independently without involving the coordinator, and to

resume the execution of transactions that are still active in the systems instead of aborting them.

Finally, non-blocking single-phase atomic commit (NB-SPAC) protocol is a non-blocking version of CL and preserves site autonomy by logging logical operations instead of physical records on the coordinator [3]. But, these protocols have largely been ignored in the implementation of distributed transactional systems due to the strong assumptions they make about the cohort's behavior identified and formalized by Abdallah M. [3]. These protocols cause longer blocking times since the prepared cohorts cannot be aborted before final consensus is reached. Hence, 1PC and its variants are best suited for distributed transactions with small size cohorts.

(iii) Group Commit Protocols

In group commit protocol, transactions are not committed as soon as they are ready [39,40]. To reduce the number of disk writes, several transactions are grouped together and committed with one disk write. Even though the group commit can enhance the system performance by reducing the number of disk writes, it also increases the lock holding time of committing transactions. Hence, group commit is usually applied in conjunction with pre-commit. When pre-commit is applied to the distributed main memory database system, it is possible for the global transactions to commit in an inconsistent order. To prevent such inconsistency, it is proposed to propagate the transaction dependency information [106]. It is also to note that pre-commit can also result in cascading abort if there is a site failure in the middle of commit process. It can be catastrophic for the system performance.

(iv) Pre Commit/Optimistic Commit Protocols

The optimistic commit protocol concentrates on reducing the lock waiting time by lending the locks the committing transactions hold [45]. Since, the lock lending is done in a controlled manner; there is no possibility of cascading aborts even if the committing transaction is aborted. These protocols have good performance due to the reduction of the blocking arising out of locks held on prepared data.

2.5.2 Three-Phase Commit Protocol

Fault tolerant computer systems prevent the disruption of services provided to users. A fundamental problem with all of the above protocols is that cohorts may become blocked in case of a site failure and remain blocked until the failed site recovers. For example, if the coordinator fails after initiating the protocol but before conveying the decision to its cohorts, these cohorts will become blocked and remain so until the coordinator recovers and informs them about the final decision. During the blocked period, the cohorts may continue to hold system resources such as locks on data items making them unavailable to other transactions, which in turn become blocked waiting for the resources to be relinquished. It is easy to see that, if the blocked period is long, it may result in a major disruption of the transaction processing activity. To address the failure-blocking problem, a three-phase commit (3PC) protocol was proposed by Skeen D [59]. This protocol achieves a non-blocking capability by inserting an extra phase, called the "pre-commit phase", in between the two phases of 2PC protocol. In pre-commit phase, a preliminary decision is reached regarding the fate of the transaction. The information made available to the participating sites as a result of this preliminary decision allows a global decision to be made despite a subsequent failure of the coordinator. However, note that the price of gaining non-blocking functionality is an increase in the communication overheads since there is an extra round of message exchange between the coordinator and the cohorts. In addition, both the coordinator and the cohorts have to force-write additional log records in the pre-commit phase. This overhead can be reduced by having only a few sites to execute the three phase protocol and the remaining can execute the cheaper two phase protocol. The transaction will not block as long as one of the sites executing the three phase protocol remains operational.

2.6 REAL TIME COMMIT PROTOCOLS

Due to series of synchronous messages and logging cost, commit processing can result in a significant increase in the transaction execution time. In a real-time environment, this is clearly undesirable. It may also result in priority inversion, because, once a cohort reaches the prepared state, it has to retain all its data locks until it receives the global decision from the coordinator. This retention is fundamentally necessary to maintain atomicity. Therefore, if a high priority

transaction requests access to a data item that is locked by a "prepared cohort" of lower priority, it is not possible to forcibly obtain access by preempting/aborting the low priority cohort. In this sense, the commit phase in DRTDBS is inherently susceptible to priority inversion. More importantly, the priority inversion interval is not bounded since the time duration, that a cohort is in the prepared state, can be arbitrarily long (e.g. due to blocking or network delays). This is especially more problematic in distributed context. Therefore, in order to meet the transaction deadlines, the choice of a better commit protocol is very important for DRTDBS. For designing the commit protocols for DRTDBS, we need to address two questions.

- (i) How do we adapt the standard commit protocols into the real-time domain?
- (ii) How can we decrease the number of missed transactions in the system?

Researchers have proposed some real-time commit protocols in the literature to address this issue. Soparkar et al. have proposed a protocol that allows individual sites to unilaterally commit [125]. If, later on, it is found that the decision is not consistent globally then compensation transactions are executed to rectify errors. The problem with this approach is that many actions are irreversible in nature. The scheme proposed by Yongik Y. et al. is also based on the theme of allowing individual sites to unilaterally commit - the idea is that unilateral commitment results in greater timeliness of actions [150]. If later on, it is found that the decision is not consistent globally, "compensation" transactions are used to rectify the errors. While the compensation-based approach certainly appears to have the potential to improve timeliness, yet there are quite a few practical difficulties described below.

- (i) The standard notion of transaction atomicity is not supported - instead, a "relaxed" notion of atomicity is provided.
- (ii) The design of a compensating transaction is an application specific task since it is based on the application semantics.

- (iii) The compensation transactions need to be designed in advance so that they can be executed as soon as error is detected. This means that the transaction workload must be fully characterized a priori.
- (iv) Some real actions such as firing a weapon or dispensing cash may not be compensated at all.

From a performance viewpoint also, there are some difficulties.

- (i) The execution of compensation transactions is itself an additional burden on the system.
- (ii) It is not clear as to how the database system should schedule the compensation transactions relative to the normal transactions.

A centralized timed 2PC protocol guarantees that the fate of a transaction (commit or abort) is known to all the cohorts before the expiry of the deadline when there are no processor, communication or clock faults [27,28]. In case of faults, however, it is not possible to provide such guarantees and an exception state is allowed which indicates the violation of the deadline. Further, the protocol assumes that it is possible for DRTDBS to guarantee allocation of resources for a duration of time within a given time interval. Finally, the protocol is predicated upon the knowledge of worst-case communication delays.

Ramesh Gupta et al. did a detailed study of the relative performance of different commit protocols [42,43,44,45,46,47]. Using a detailed simulation model for firm-deadline DRTDBS, the authors have evaluated the deadline miss performance of a variety of standard commit protocols including 2PC, PA, PC and 3PC. Then they have proposed and evaluated the performance of a new commit protocol called OPT designed specifically for the real-time environment [45,46]. It is a 2PC variant that attempts to alleviate priority inversion in 2PC and includes features such as controlled optimistic access to uncommitted data, active abort and silent kill. This protocol allows a high priority transaction to borrow (i.e., access) data items held by low priority transaction that is waiting for the commit message under the assumption that the low priority transaction will most probably commit. This creates dependencies among transactions. If a transaction depends on other transactions, it is not allowed to start commit processing and is blocked

until the transactions, on which it depends, have committed. The blocked committing transaction may include a chain of dependencies as other executing transactions may have data conflicts with it. They have also suggested two variants of OPT, i.e., Shadow-Opt and Healthy-Opt protocols [44]. In Healthy-Opt, a health factor is associated with each transaction and the transaction is allowed to lend its data, only if, its health factor is greater than a minimum value. However, it does not consider the type of dependencies between two transactions. The abort of a lending transaction aborts all transaction that has borrowed the data from it. The performance of the system is dependent on chosen threshold value of health factor (HF), which is defined as ratio of TimeLeft to MinTime, where TimeLeft is the time left until the transaction's deadline and MinTime is the minimum time required for commit processing. In Shadow-Opt, a cohort creates a replica of the cohort called a shadow, whenever, it borrows a data page. The original cohort continues its execution and the shadow is blocked at the point of borrowing. If the lending cohort commits, the borrowing cohort continues and the shadow is discarded, otherwise if the lender aborts, the borrowing cohort is aborted.

A new protocol Permits Reading of Modified Prepared-Data for Timeliness (PROMPT) proposed by Haritsa J. R. et al. is also designed specifically for the real-time environment and includes features such as controlled optimistic access to uncommitted data, active abort, silent kill and healthy lending [59]. The performance results of PROMPT show that it provides significant improvement over the classical commit protocols, and makes extremely efficient use of the lending premise. Finally, the authors have also evaluated the priority inheritance approach to address the priority inversion problem associated with prepared data, but found it to be inherently unsuitable for the distributed environment. Due to sharing of data items, it creates commit or abort dependencies between the conflicting transactions. The dependencies limit the commit order of the transactions and thus may cause a transaction to miss its deadline while it is waiting for its dependent transaction to commit. The impact of buffer space and admission control is also not studied since it is assumed that buffer space is sufficient large to allow the retention of data updates until commit time.

Lam et al. proposed deadline-driven conflict resolution (DDCR) protocol which integrates concurrency control and transaction commitment protocol for firm real time transactions [74,78]. DDCR resolves different transaction conflicts by

maintaining three copies of each modified data item (before, after and further) according to the dependency relationship between the lock-requester and the lock holder. This not only creates additional workload on the systems but also has priority inversion problems. The serializability of the schedule is ensured by checking the before set and the after sets when a transaction wants to enter the decision phase. The protocol aims to reduce the impact of a committing transaction on the executing transaction which depends on it. The conflict resolution in DDCR is divided into two parts (a) resolving conflicts at the conflict time; and (b) reversing the commit dependency when a transaction, which depends on a committing transaction, wants to enter the decision phase and its deadline is approaching.

If data conflict occurs between the executing and committing transactions, system's performance will be affected. Based on the DDCR, Pang Chung-leung and Lam K. Y. proposed an enhancement in DDCR called the DDCR with similarity (DDCR-S) to resolve the executing-committing conflicts in DRTDBS with mixed requirements of criticality and consistency in transactions [103]. In DDCR-S, conflicts involving transactions with looser consistency requirement and the notion of similarity are adopted so that a higher degree of concurrency can be achieved and at the same time the consistency requirements of the transactions can still be met. The simulation results show that the use of DDCR-S can significantly improve the overall system performance as compared with the original DDCR approach.

Based on PROMPT and DDCR protocols, B. Qin and Y. Liu proposed double space commit (2SC) protocol [108]. They analyzed and categorized all kind of dependencies that may occur due to data access conflicts between the transactions into two types commit dependency and abort dependency. The 2SC protocol allows a non-healthy transaction to lend its held data to the transactions in its commit dependency set. When the prepared transaction aborts, only the transactions in its abort dependency set are aborted and the transactions in its commit dependency set execute as normal. These two properties of the 2SC can reduce the data inaccessibility and the priority inversion that is inherent in distributed real-time commit processing. 2SC protocol uses blind write model. Extensive simulation experiments have been performed to compare the performance of 2SC with that of other protocols such as PROMPT and DDCR.

The simulation results show that 2SC has the best performance. Furthermore, it is easy to incorporate it in any current concurrency control protocol.

2.7 MEMORY OPTIMIZATION

The Important data base system resources are the data items that can be viewed as logical resource, and CPU, disks and the main memory which are physical resources [35]. Though the cost of the main memory is dropping rapidly and its size is increasing, the size of database is also increasing very rapidly. In real time applications, where the databases are of limited size or are growing at a slower rate than the memory capacities are growing, they can be kept in the main memory. However, there are many real time applications that handle large amount of data and require support of an intensive transaction processing [110]. The amount of data they store is too large (and too expensive) to be stored in the non volatile main memory. Examples include telephone switching, satellite image data, radar tracking, media servers, computer aided manufacturing etc. In these cases, the database can not be accommodated in the main memory easily. Hence, many of these types of database systems are disk resident. The buffer space in the main memory is used to store the execution code, copies of files & data pages, and any temporary objects produced. With the new functionalities and features of the light weight devices, there is need of new policy/protocols so that memory utilization can be improved [113]. Ramamritham K. and Sen R. utilized a novel storage model, ID based storage, which reduces storage costs considerably. They present an exact algorithm for allocating memory among the database operators. Because of its high complexity, a heuristic solution based on the benefit of an operator per unit memory allocation has been also proposed.

2.8 CONCLUSIONS

This chapter described the basic concepts and definitions of the database system and reviewed the work carried out in the areas of priority assignment policies for transaction scheduling, commit processing and memory optimization. The priority assignment policies for the transaction scheduling both for the centralized and the distributed RTDBS are discussed. Most of the priority assignment policies are suitable only when the cohorts of the transactions execute sequentially. The traditional commit processing protocol 2PC and its variants have

also been described. Different real time commit protocols proposed in the literature are explained and compared. Most of these protocols try to improve the performance by allowing a transaction to use a data item locked by some other transaction. There is complete lack of policies/protocols which can be well suited to efficient use of memory by creating lesser temporary objects.

Some of the highlighting factors about the work done in this thesis are re-evaluation of the static two phase locking mechanism, proposal for a new priority assignment policy, development of new deadline assignment heuristic and commit protocols suitable for handling huge volume of data and large number of transactions.

PRIORITY ASSIGNMENT POLICY

3.1 INTRODUCTION

The processing of transactions in DRTDBS is much more complex than in centralized real time database systems. The transactions in DRTDBS have constraints on their completion time, which are usually expressed as their deadlines. DRTDBS must process the transactions before their deadlines and should also guarantee that the database consistency is not violated [21,14]. The execution of a transaction involves CPU scheduling, data scheduling and scheduling of other resources. In DRTDBS, scheduling of CPU and data is done according to the priorities assigned to the transactions [31]. Since, these priorities determine the execution order of the transactions, which indirectly affects the extent of data conflicts, the policy used for assigning priority becomes highly critical in deciding the performance of DRTDBS. It plays an important role in reducing the percentage of the transactions that miss their deadlines. The resource scheduler of a real time database allocates the system resources to highest priority transaction so as to give it the best chance of completion before its deadline. The success of one transaction may be achieved at the cost of other transactions resulting in unfair treatment of those transactions.

3.1.1 Issue of Fairness

Fairness is the assurance of granting the resources to each request within a predetermined bounded time. In DRTDBS, timely execution of the transactions is considered more important than fairness, freedom from starvation and efficient resource utilization [130]. However, many real time applications such as network management, multimedia systems demand a balance between fairness and priority. The resources allocation policy must not lead to starvation and it should provide some degree of fairness to each competing transaction within its real time constraints [151]. Sometimes, it may be preferable to let the higher priority transactions wait for low priority transaction, if the low priority transaction has already consumed a lot of resources and time. Aborting a near completion low

priority transaction may cost more than blocking a higher priority transaction for a limited period of time. Also, giving a preferential treatment to short transactions does not satisfy properties such as the fairness and freedom from starvation. Long transactions have higher probability of being starved because of their higher probability of access conflicts. This results in a lower deadline guarantee ratio for them than the short transactions. Because of their higher probability of conflict with other transactions, the long transactions are likely to be repeatedly restarted, and thus have less chance to meet their deadlines. The survival of a long transaction is very tough due to the scenarios described below [92].

- (i) **Wasted Restart/Abort:** A wasted restart/abort happens if a higher priority transaction restarts/aborts a lower priority transaction and then higher priority transaction is discarded as it misses its deadline.
- (ii) **Wasted Wait:** A wasted wait happens if a lower priority transaction waits for the commit of the higher priority transaction and later higher priority transaction is aborted as it misses its deadline.
- (iii) **Wasted Execution:** This happens when a lower priority transaction that is near to the completion of commit processing is restarted/aborted due to a conflict with higher priority transaction arrival.
- (iv) **Unnecessary Restart:** An unnecessary restart happens when a transaction is restarted even when the execution was serializable.

3.1.2 Background

In order to maintain database consistency and to ensure failure atomicity, a real time concurrency control with atomic commitment protocol is used in DRTDBS. Incorporating proper transaction scheduling policy with concurrency control methods has a marked impact on the effective level of concurrency, especially, when data contention is high. In DRTDBS, transaction is generally divided into several subtasks (cohorts). Kao B. and Garcia-Molina H. [66,67] addressed the issue of the subtask deadline assignment in a distributed environment. A typical global transaction processed in a distributed system possesses subtasks (i.e., cohort) to be executed on various system sites. A single

value of an end-to-end global deadline might not truly reflect the urgency of each individual subtask. The subtask deadlines should be earlier than the end-to-end global deadline so as to speed up the progress of the global transaction. Kao and Garcia-Molina [67] suggested and evaluated heuristic based scheduling policies for the subtask deadline assignment problem. The problem was reduced into two sub problems: one deals with *serial* subtasks (where a global transaction consists of a number of serially executing subtasks), and the other one with *parallel* subtasks (where a global transaction involves parallel execution of subtasks at different nodes).

The performance of heuristics proposed by Kao B. and Garcia-Molina H. [66] has been examined by Lee Victor C. S. et al. [85] and they suggested three other alternatives that take into consideration the impact of data conflicts. These alternative priority assignment strategies are the Number of Locks held (NL), Static EQS (SEQS) and Mixed method (MM). The NL assigns the priority to cohorts on the basis of the number of locks being held by its parent transaction while other two heuristics are improved version of heuristics discussed by Kao B. and Garcia-Molina H. [66]. However, both of these studies consider the sequential executions of the task or cohorts. These heuristics, except ultimate deadline (UD), are not suitable for cohorts executing in parallel. The heuristics proposed for DRTDBS where the subtasks of a transaction are executed sequentially may not suit well when the subtasks of a transaction are executed in parallel. Moreover, Lee Victor C. S. et al. [85] have not studied the fairness property of these schemes. In NL, the transaction can complete faster and release the locks earlier by giving the highest priority to the transaction which holds the largest number of locks. However, this approach can not be applied to cohorts executing in parallel.

The priority assignment policy proposed in this chapter is based on the following observations:

- (i) If a long transaction is divided into many cohorts executing in parallel and each one is having only a small number of operations then assigning cohorts a priority same as their parents is not justified. This

- may result in giving higher priority to the local transactions even if they have more number of operations.
- (ii) Aborting a near completion low priority transaction may cost more than blocking a higher priority transaction for a limited period of time.

In the proposed priority assignment heuristic, the cohorts are assigned the priority based on the number of locks required by them instead of inheriting the priority of its parents. We also schedule cohorts in a way that favors the one that has finished most of its work (holding locks) over those that have just begun or are waiting for locks. This not only increases the effective level of concurrency but also increases success ratio. To achieve this, temporary intermediate priorities of cohorts are calculated at the time of data contention and are used to schedule them more effectively. In this chapter, we study the effects of the proposed heuristic and the temporary intermediate priority assignment policy on the performance of DRTDBS by incorporating them with S2PL concurrency control algorithm [73] and 2SC commit protocol [108]. However, this is not limited to only S2PL and 2SC and the same gain can be achieved while using any concurrency control algorithm and 2PC or any other commit protocol.

This chapter first presents the transaction model, deadline estimation method for global and local transactions and 2SC distributed real time commit protocol. We, then discuss our heuristic and temporary intermediate priority assignment policy with performance evaluation parameters and simulation results.

3.2 DISTRIBUTED REAL-TIME DATABASE SYSTEM MODEL

The structure of our simulation model including the description of its various components such as system model, database model, network model, cohort execution model, locking mechanism and the model assumptions is given in Fig 3.1. The common model for DRTDBS is given below [129].

At each site, two types of transactions are generated: global transactions and local transactions. Each global transaction consists of m cohorts, where m is less than or equal to the number of database sites N_{site} . We use the same model for local and global transactions. Each local transaction has a coordinator and a

single cohort both executing at the same site. Each transaction consists of N_{oper} number of database operations. Each operation requires locking of data items and then processing.

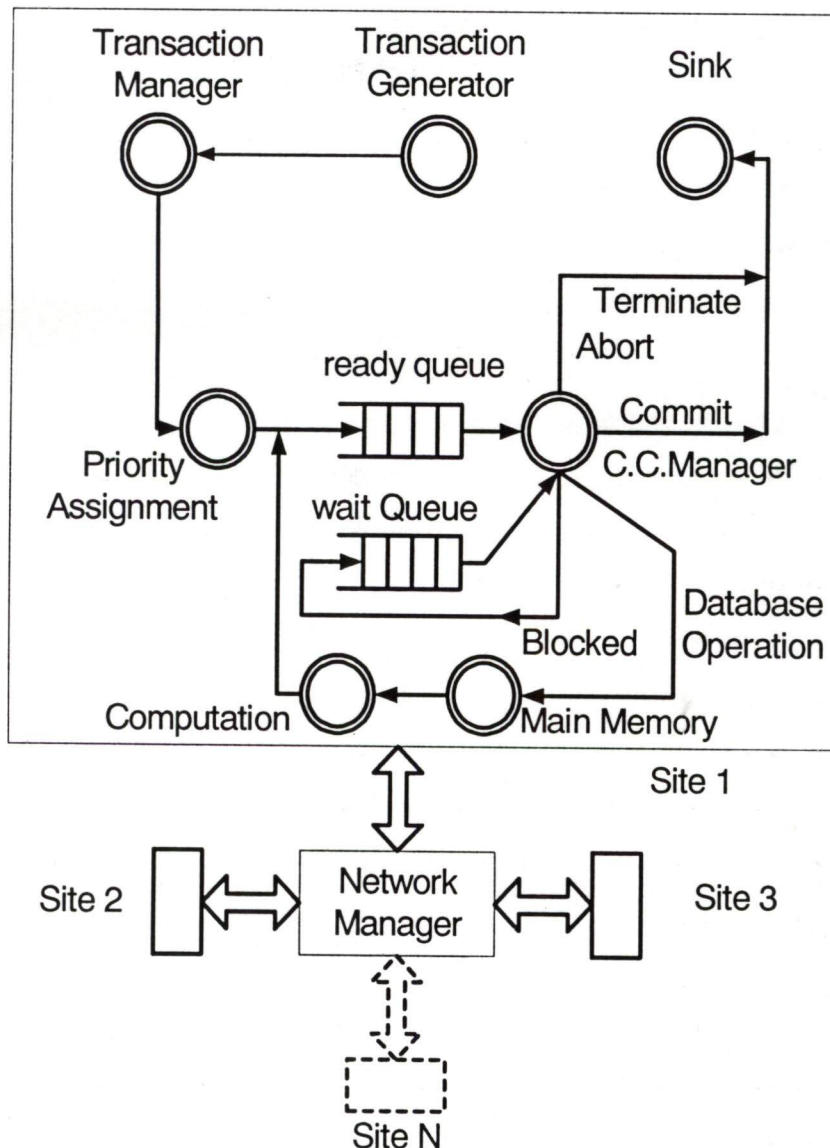


Fig. 3.1: Distributed Real-time Database System Model

3.2.1 System Model

Each site consists of a transaction generator, a transaction manager, a concurrency controller, a CPU, a ready queue, a local database, a communication interface, a sink and a wait queue. The transaction generator is responsible for creating the transactions independent to the other sites using Poisson distribution

with the given inter-arrival time. The transaction manager generates cohorts on remote site on behalf of the coordinator. Before a cohort performs any operation on a data item, it has to go through the concurrency controller to obtain a lock on that data item. If the request is denied, the cohort is placed in the wait queue. The waiting cohort is awakened when the requested lock is released and all other locks are available. After getting all locks, the cohort accesses the memory and performs computation on the data items. Finally, the cohort commits/aborts and releases all the locks that it is holding. The sink component of the model is responsible for gathering the statistics for the committed or terminated transactions.

3.2.2 Database Model

The database is modeled as a collection of data items that are uniformly distributed across all the sites. Transactions make requests for the data items and concurrency control is implemented at the data item level. No replication of data items at various sites is considered here.

3.2.3 Network Model

A communication network interconnects the sites. There is no global shared memory in the system. All sites communicate via messages exchange over the communication network. The network manager models the behavior of the communications network.

3.2.4 Cohort Execution Model

In our work, we have considered the parallel execution of cohorts. The coordinator of the transaction spawns all cohorts together by sending messages to remote sites to activate them, lists all operations to be executed at that site and then cohorts may start execution at the same time in parallel. The assumption here is that a cohort does not have to read from its sibling and operations performed by one cohort during its execution are independent of the results of the operations performed by other cohorts at some other sites. In other words, the sibling cohorts do not require any information from each other to share.

3.2.5 Locking Mechanism

The main technique used to control concurrent execution of transactions is based on the concept of locking data items. A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. Generally there is one lock for each data item in the database. Locks are means for synchronizing the access of concurrent transactions to the database items.

A transaction is said to follow the two phase locking protocol if all locking operations precede the first unlock operation in the transaction. There is a number of variations of the two phase locking (2PL) such as static two phase locking (S2PL) and dynamic two phase locking (D2PL). The static 2PL (S2PL) requires a transaction to lock all needed data items before the transaction begins execution, by predeclaring its read-set and write-set. If any of the predeclared data item can not be locked, the transaction does not lock any items; instead, it waits until all data items are available for locking.

3.2.6 Model Assumptions

We assume that the transactions are firm real time transactions. The model assumptions are listed below.

- Processing of a transaction requires the use of CPU and data items located at local or remote site.
- Arrival of the transactions at a site is independent of the arrivals at other sites and uses Poisson distribution.
- Each transaction pre-declares its read-set (set of data items that the transaction will only read) and update-set (set of data items that the transaction will update).
- The cohorts are executed in parallel.
- A lending transaction cannot lend the same data item in read/update mode to more than one cohort to avoid cascaded abort.

- A cohort already in the dependency set of another cohort cannot permit a third incoming cohort to perform read or update.
- The communication delay considered is either 0ms or 100ms to study the impact of the network delay on the system.
- A distributed real time transaction is said to commit, if the coordinator has reached to commit decision before the expiry of the deadline at its site. This definition applies irrespective of whether cohorts have also received and recorded the commit decision by the deadlines.
- The database is in the main memory at all sites.

3.3 DEADLINE ASSIGNMENT

The deadlines of transactions (both global and local) are calculated based on their expected execution times [73]. The deadline (D_i) of transaction (T_i) is defined as:

$$D_i = A_i + SF * R_i$$

where, A_i is the arrival time of transaction (T_i) at a site.

SF is the slack factor.

R_i is the minimum transaction response time.

As cohorts are executing in parallel, R_i is calculated as:

$$R_i = R_p + R_c$$

where, R_p is the time for execution phase and R_c is the time for commitment phase which are given as below.

For global transactions

$$R_p = \max\{T_{i_p} \times N_{i_{local}}, \max_{\substack{1 \leq j \leq m_i \\ j \neq local}}(T_{i_j} \times N_{i_j}) + 2 \times T_{com}\}$$

$$T_{i_p} = 2 \times T_{lock} + T_{processing}$$

$$R_c = N_{comm} \times T_{com}$$

For local transactions

$$R_p = (2 \times T_{\text{lock}} + T_{\text{process}}) \times N_{\text{oper_local}}$$

$$R_c = 0$$

Where,

m_i is the number of cohorts of i^{th} transaction;

N_{local} is the number of operations in local cohort of T_i ;

N_{i_j} is the number of operations in j^{th} cohort of T_i ;

T_{lock} is the time required to lock/unlock a data item;

T_{process} is the time to process a data item (assuming read operation takes same amount of time as write operation);

N_{comm} is the number of messages exchanged between the coordinator and a cohort;

T_{com} is the communication delay, i.e., the constant time estimated for a message going from one site to another;

$N_{\text{oper_local}}$ is the number of local operations;

$N_{\text{oper_remote}}$ is maximum number of remote operations taken over by all cohorts.

3.4 DOUBLE SPACE COMMIT PROTOCOL (2SC)

In this section, 2SC protocol has been described in brief to ensure completeness. Additional details of 2SC protocol can be found in "High Performance Distributed Real - time Commit Protocol" by Biao Qin and Y. Liu [108]. The 2SC protocol allows two transactions to share the data by allowing a transaction (borrower) to borrow the data from another transaction (lender) in its commit phase. The sharing of data items in conflicting modes creates dependencies among the conflicting transactions and constraint their commit orders. Basically there are two kinds of dependencies; (i) commit dependency, and (ii) abort dependency. If a transaction T_2 writes a data item after another transaction T_1 , a commit dependency is created from T_2 to T_1 denoted by T_2 CD T_1 . T_2 is not allowed to commit until T_1 has ended. If T_2 reads an uncommitted data item written by T_1 , an abort dependency is created from T_2 to T_1 denoted by T_2 AD T_1 . T_2 has to commit after T_1 and the abort of T_1 causes T_2 to abort.

A transaction table at each site maintains the following information for each local active transaction or cohort (T):

Abort dependency set (ADS (T))

Set of transactions that are abort dependent on transaction T.

$$\text{ADS (T)} = \{T_i \mid T_i \text{ AD T}\}$$

Commit dependency set (CDS (T))

Set of transactions that are commit dependent on transaction T.

$$\text{CDS (T)} = \{T_i \mid T_i \text{ CD T}\}$$

When data conflicts occur, there are three possible cases of conflict:

Read - Write Conflict

If T_2 requests a write-lock while T_1 is holding the read lock on the same item, a commit dependency is defined from T_2 to T_1 . After the transaction id of T_2 , id_2 , is added to $\text{CDS}(T_1)$, T_2 acquires the write-lock.

Write - Write Conflict

If both locks are write locks, a commit dependency is defined from T_2 to T_1 . After the transaction id of T_2 , id_2 , is added to $\text{CDS}(T_1)$, T_2 acquires the write-lock.

Write - Read Conflict

If T_2 requests a read lock while T_1 is holding a write-lock, an abort dependency is defined from T_2 to T_1 . If $\text{HF}(T_1) > \text{MinHF}$, the transaction id of T_2 , id_2 , is added to $\text{ADS}(T_1)$ then T_2 acquires the write-lock; Otherwise, if $\text{HF}(T_1) < \text{MinHF}$, T_2 is blocked (see section 2.6 for HF and MinHF).

When T_2 accesses a data item already locked by committing cohort T_1 , any one of the following three scenarios may arise.

(i) **T_1 Receives Decision Before T_2 Has Completed Its Local Data Processing**

If the global decision is to commit,

- a. T_1 commits.
- b. All transactions in ADS (T_1) and CDS (T_1) will execute as usual.
- c. Set of ADS (T_1) and CDS (T_1) will be deleted.

If the global decision is to abort,

- a. T_1 aborts.
- d. The transactions in the dependency set of T_1 will execute as follows:
 - All transactions in ADS (T_1) will be aborted.
 - All transactions in CDS (T_1) will execute as usual.
 - The set of ADS (T_1) and CDS (T_1) will be deleted.

(ii) **T_2 Completes Data Processing Before T_1 Receives Global Decision**

T_2 is not allowed to send WORKDONE message to its coordinator. So the coordinator cannot initiate commit processing. Instead, it waits until either T_1 receives its global decisions or its own deadline expires, whichever occurs earlier. In the former case, the system will execute as the first scenario. In the latter case, T_2 will be killed and will be removed from the dependency set of T_1 .

(iii) **T_2 aborts before T_1 receives decision**

In this situation, T_2 's updates are undone and T_2 will be removed from the dependency set of T_1 .

3.5 PRIORITY ASSIGNMENT HEURISTIC & TEMPORARY INTERMEDIATE PRIORITY ASSIGNMENT POLICY

The proposed priority assignment heuristic is based on the number of locks required by the cohort. The initial priority of cohort (T_i) of transaction (T) is computed as below.

$$P = 1/N$$

where, $N \geq 1$, is the number of locks required by T_i . Higher the value of N lesser is the priority of T_i . As this may favor short cohorts, we also calculate a temporary intermediate priority, whenever a data contention occurs as explained in the next paragraph.

The lifetime of a cohort can be divided into two phases: execution phase and commit phase. During execution phase, the cohort locks the data items, does some necessary computation, and finally, sends WORKDONE message to its coordinator. If there are dependencies then the sending of WORKDONE message is deferred till the removal of the dependencies [78,108]. In our model, arrival of a higher priority cohort may abort the lock holding low priority cohort, if it has not sent YES VOTE message. The abort of lock holding cohort is done on the basis of the temporary intermediate priorities assigned to the arriving cohort and the lock holding cohort. It is a temporary priority assignment and does not affect the initial priorities of the two cohorts. It is based on the remaining execution time of the lock holding low priority cohort (T_L) which is in execution phase and the slack time available of the newly arrived higher priority cohort (T_A). It also solves the problem of starvation of long cohorts that arises due to the high probability of their access conflicts. Slack time is the amount of time a cohort can afford to wait in order to complete before its deadline. The remaining execution time (T_{remain}) of the lock holding cohort (T_L) is given as:

$$T_{remain} = R_i - T_{elapse}$$

T_{elapse} is elapsed execution time of T_L .

If $T_{remain}(T_L)$ is less than the slack time(T_A) then the priority of T_L is greater than T_A ; otherwise, T_A deserves the higher priority.

Complete pseudo code for scheduling algorithm

Let T_{CH} be the cohort holding the CPU, T_A is the cohort requesting the CPU and T_L is the lock holding cohort in conflict with T_A .

Algorithm for CPU scheduling

CPU_scheduling_function ()

{

```

if (New Arrival)
{
    Assign priority to  $T_A$  based on initial priority assignment
    policy;
    if (CPU is not free)
    {
        if (initial_priority ( $T_{CH}$ ) > initial_priority ( $T_A$ ))
            insert  $T_A$  in wait queue;
        else
        {
            abort  $T_{CH}$  ;
            allocate CPU to  $T_A$  ;
            call lock_acquire_function ( );
        }
    }
    else
    {
        allocate CPU to  $T_A$  ;
        call lock_acquire_function ( );
    }
}
}

```

Algorithm to resolve data contention

```

lock_acquire_function ( )
{
    for each data item:→
    if(!lock_conflict)
        allocate data item to  $T_A$ ;
}

```

```

else
{
    check the priority of conflicting cohorts  $T_L$ ;
    if (initial priority ( $T_A$ ) < priority( $T_L$ ))
        insert  $T_A$  in wait queue;
    else
    {
        if(slack time( $T_A$ ) <  $T_{remain}(T_L)$ ) //slack time( $T_A$ )  $\geq 0$ 
        {
            abort  $T_L$ ; allocate data item to  $T_A$ ;
        }
        else
            insert  $T_A$  in wait queue;
    }
}
}

```

3.6 PERFORMANCE EVALUATIONS

This section describes the performance parameters and measures used for the simulation and analyze the results of the simulation.

3.6.1 Performance Parameters and Measures

There are no practical benchmark programs for DRTDBS in the market or in the research community to evaluate the performances of protocols [86]. Therefore, a distributed real time database system consisting of N sites was simulated in the present work on the basis of [12,29,33,72,73,76,126,131,142,143,144,146]. The default values of different parameters used in the simulation experiments are given in Table 3.1. They are chosen to be in accordance with those used in earlier studies [78,73,59]. For each set of experiment, the final results are calculated as an average of 10 independent runs. In each run, 20000 transactions are initiated.

Table 3.1: Default Values for the Model Parameters

Parameters	Meaning	Default setting
N_{site}	Number of Site	4
AR	Transaction Arrival Rate	4 Transactions/ Second
T_{com}	Communication Delay	100 ms (Constant)
SF	Slack Factor	1-4 (Uniform Distribution)
N_{oper}	No. of Operations in a Transaction	3-20 (Uniform Distribution)
PageCPU	CPU Page Processing Time	5 ms
PageDisk	Disk Page Processing Time	20 ms
DBsize	Database Size	200 Data Objects/Site
P_{write}	Write Operation Probability	0.60

Our primary performance measure is the percentage of missed deadlines (or Miss Percentage, MP) which is defined as the percentage of input transactions that system is unable to complete on or before their deadlines.

$$MP = \frac{\text{number of transactions aborted}}{\text{no. of transactions submitted to the system for processing}} \times 100$$

3.6.2 Simulation Results

In this section, we report the results and findings of our simulation experiments. We compared the performance of our scheme with EDF based priority assignment scheme. In EDF, the cohort with closest deadline is assigned highest priority [54]. If any two of the cohorts have same deadline, the one with earliest arrival (FCFS) is assigned a higher priority. To investigate the performance of the proposed heuristic and temporary intermediate priority assignment policy, a wide range of transaction size ($N_{oper} = 3$ to 20) has been used both for global and local transactions.

Fig. 3.2 compares the performance of our heuristics with EDF, when communication delay is 100ms. The performance is slightly better than EDF

rate due to high level of conflicts for the data items which are handled in a more effective way seeing the local condition of site. The heuristic associated with temporary intermediate priority assignment policy outperforms the EDF due to careful aborting of low priority executing cohort at the time of contention with higher priority cohort. There is almost same improvement gain from transaction arrival rate 4 to 14.

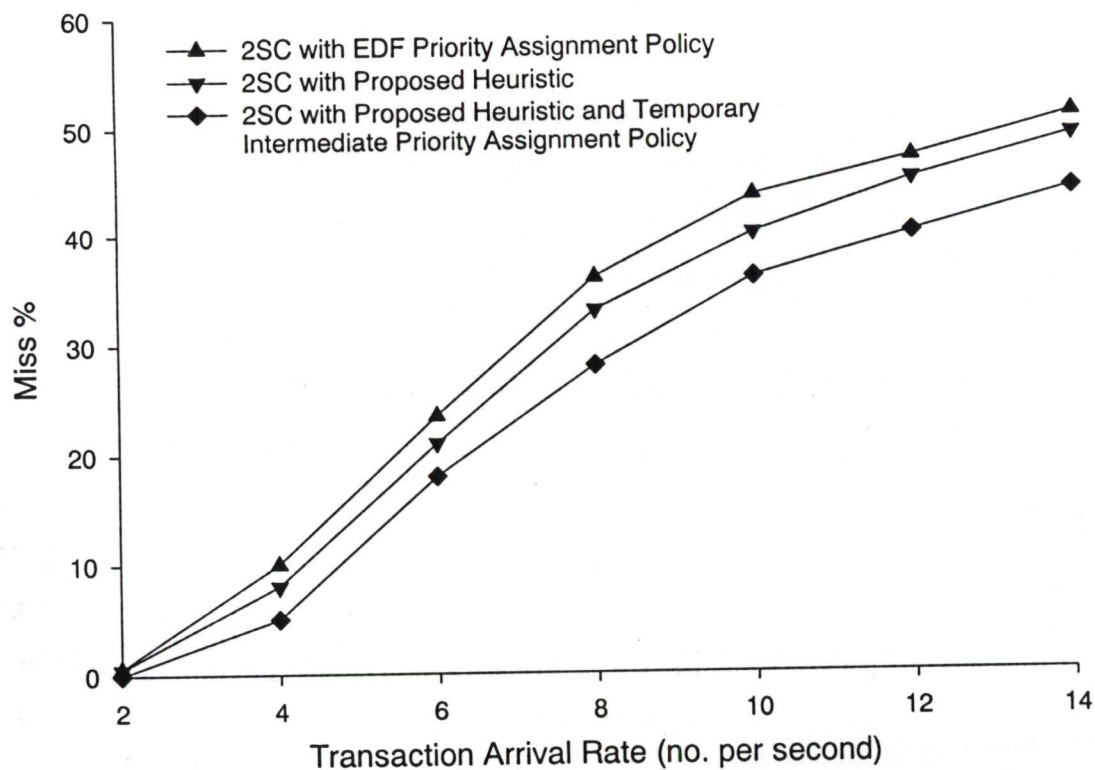


Fig. 3.2: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load

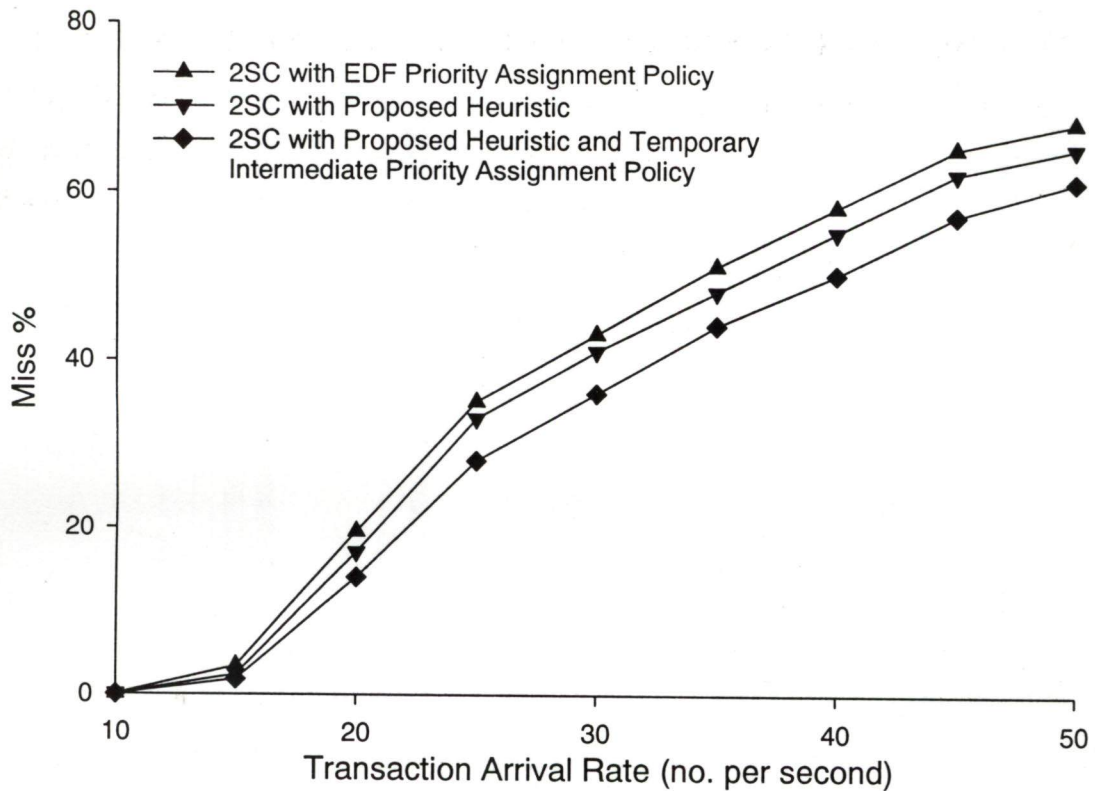


Fig. 3.3: Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load

Fig 3.3 compares the result at communications delay of 0ms. We can observe that the performance gain is almost similar to that shown in Fig. 3.2. The data conflicts are infrequent at very low transaction arrival rate. The Miss Percentage is at par with EDF based priority scheme at low transaction arrival rate since the data contention is low. The performance improves at high arrival rate by effectively resolving the high level of conflicts for the data items.

Fig 3.4 and 3.5 compare the fairness of 2SC+EDF+proposed temporary intermediate priority assignment scheme with 2SC+EDF priority assignment scheme. To investigate the performance of the proposed scheme for above purpose, a wide range of number of operations ($N_{oper} = (50\% \text{ 3-5 and } 50\% \text{ 20-25})$) in global as well as in local transactions have been considered. It can be observed that proposed temporary intermediate priority assignment scheme minimizes overall Miss Percentage of long transactions, and hence exhibits better fairness due to careful abortion of near completion transaction in case of conflicts.

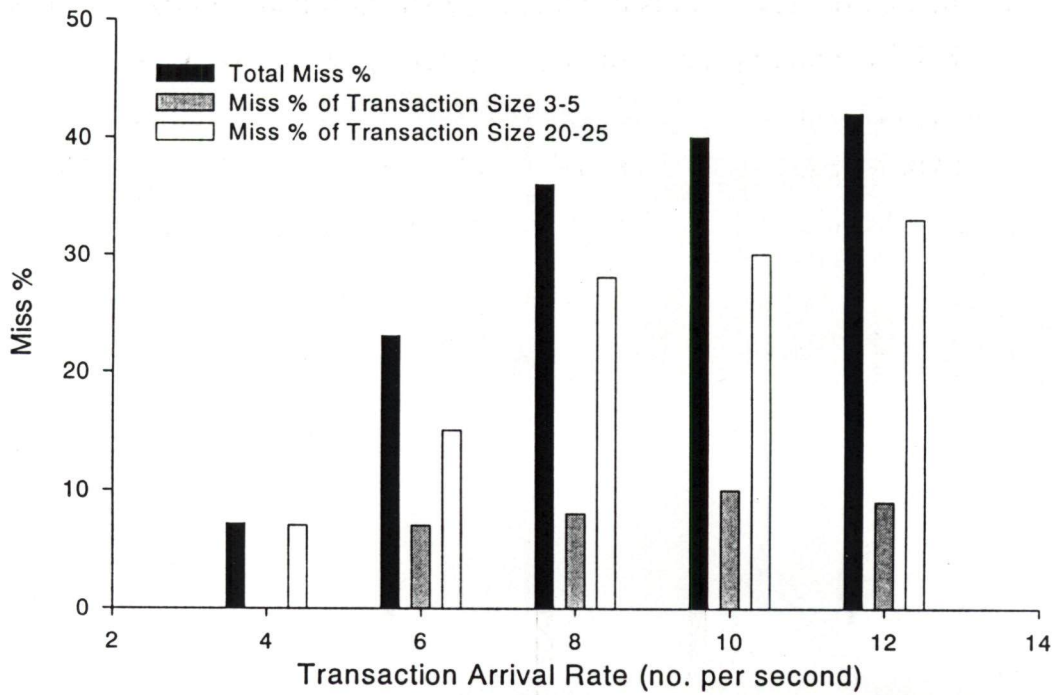


Fig. 3.4: Break-up of Miss % of Transactions at Communication Delay=100ms & System Load (50% Transactions of Size 3-5 & 50% Transactions of Size 20-25) in 2SC with EDF Priority Assignment Policy

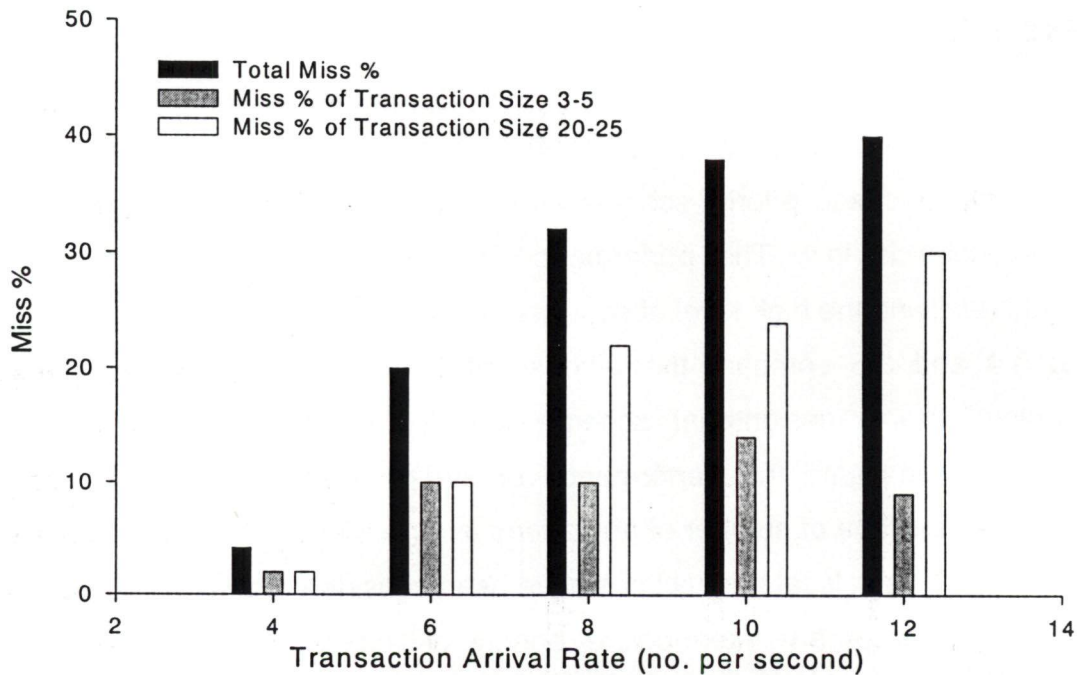


Fig. 3.5: Break-up of Miss % of Transactions at Communication Delay=100 ms & System Load (50% Transactions of Size 3-5 & 50% Transactions of Size 20-25) in 2SC with EDF and Temporary Intermediate Priority Assignment Policy

Fig. 3.6 compares 2SC+heuristic+temporary intermediate priority assignment policy with 2SC+EDF priority assignment policy at communication delay of 100ms and a mixed system load (50% transaction of size 3-5 (uniform distribution) and 50% transaction of Size 20-25 (Uniform Distribution)). It can be seen that the scheme proposed in this chapter outperforms EDF Priority Assignment Policy.

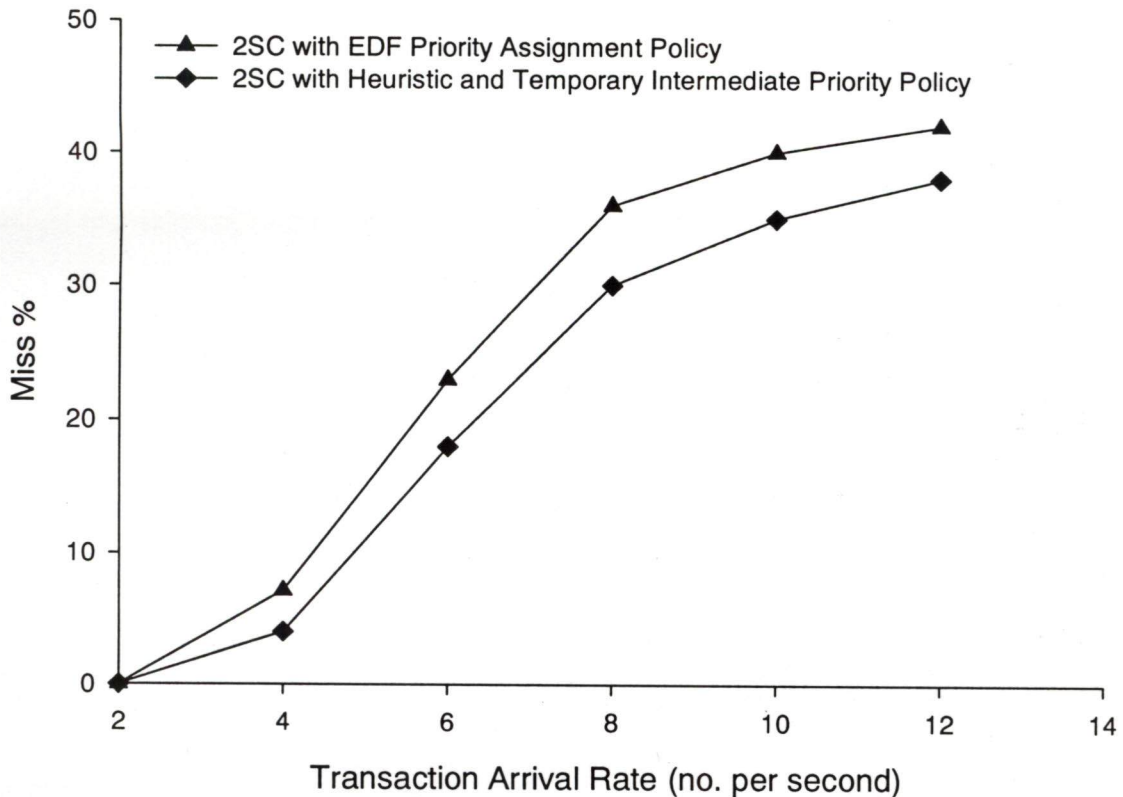


Fig. 3.6: Miss % with (RC+DC) at Communication Delay=100ms and System Load (50% Transactions of Size 3-5 & 50% Transactions of Size 20-25)

Fig. 3.7 and Fig. 3.8 compare 2SC+heuristic+temporary intermediate priority assignment policy with 2SC+EDF priority assignment policy at communication delay of 100ms and 0ms respectively with different transaction sizes. Cohorts are carefully granted a requested lock if it is in a conflicting mode against another cohort. Miss Percentage is almost same for all schemes at low transaction size since the data contention is low. However the performance of proposed scheme improves at high transaction size due to selective abort of long transactions and allocation of data items in a more effective way.

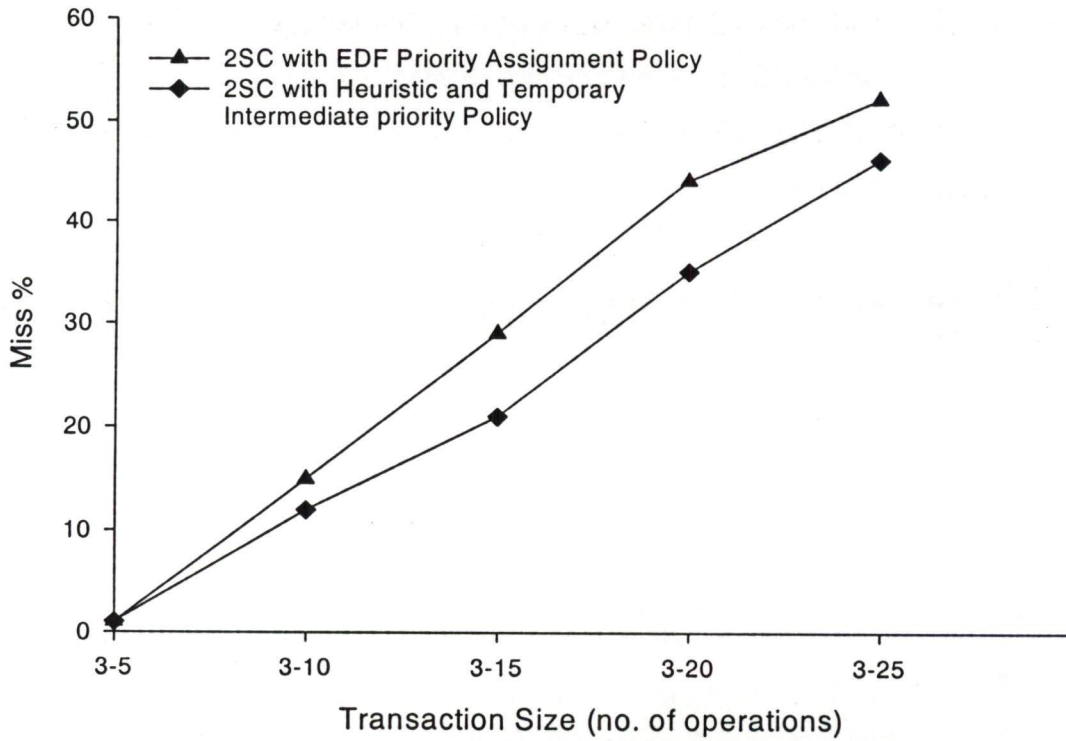


Fig. 3.7: Miss % (RC+DC) at Communication Delay=100ms & Transaction Arrival Rate=10 Transactions/Second

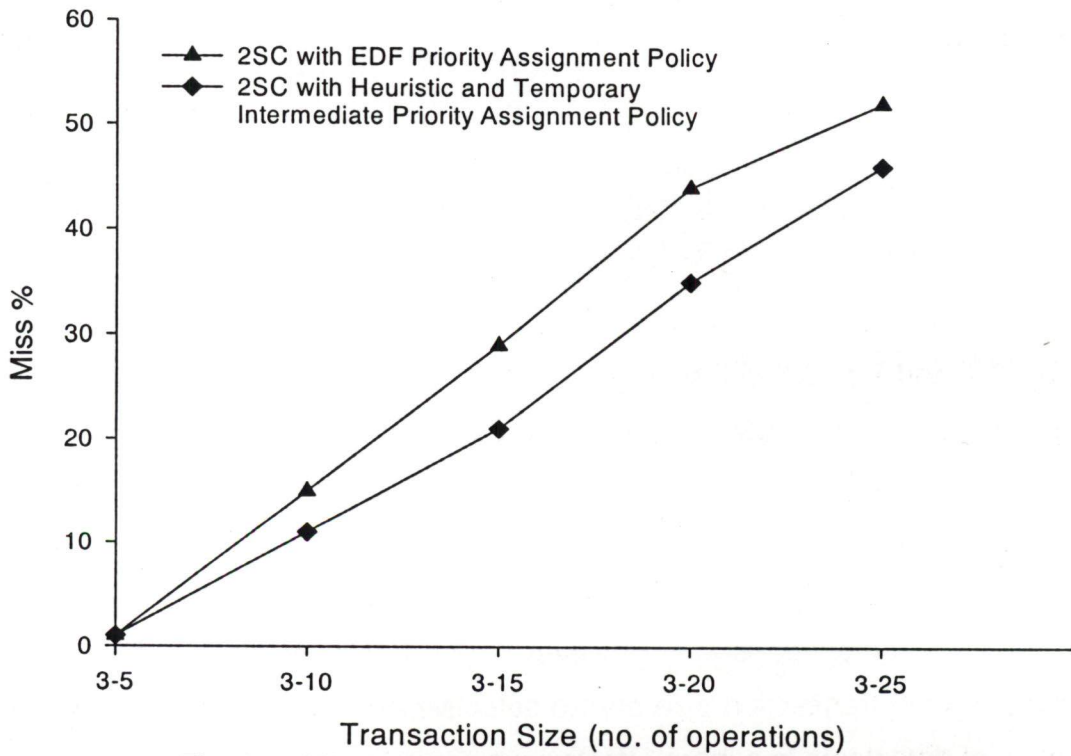


Fig. 3.8: Miss % (RC+DC) at Communication Delay=0ms & Transaction Arrival Rate=30 Transactions/Second

3.7 CONCLUSIONS

This chapter introduces new heuristic and temporary intermediate priority assignment policy to determine the priorities of transactions. At the time of data contention between two cohorts, the temporary intermediate priority assignment policy is used to avoid the wastage of time due to the abort of near completion cohorts. Thus, the proposed scheme improves the system performance by favoring cohorts demanding lesser number of locks and those that are near completion. Although, this requires the calculation of cohort's temporary intermediate priority every time when a data contention occurs but reduces a large amount of wasted work. Our performance results indicate that the heuristic used for initial priority assignment combined with temporary intermediate priority assignment policy considers both transaction real-time constraints and fairness. We compared the proposed scheme with EDF priority assignment policy. Simulation results show that the proposed scheme combined with temporary intermediate priority assignment policy gives better performance than EDF based scheme. We have also shown that this exhibits greater flexibility in coping with starvation problem for longer transactions. The priority assignment policy proposed in this chapter not only reduces the percentage of transactions that miss their deadlines but also gives a fair treatment to all the transactions.



SWIFT - A DISTRIBUTED REAL TIME COMMIT PROTOCOL

4.1 INTRODUCTION

Earlier research on real time database systems primarily focuses on centralized database systems. DRTDBSs have received very little attention, making it difficult for their designer to make informed choices [43,59]. A real time database system operating on distributed data should not only maintain data consistency, but it must also satisfy timing constraints associated with the transactions, typically expressed in terms of their deadlines. The lifetime of a transaction is divided into two stages, viz., execution stage and commitment stage [74,78]. In the execution stage, the operations of a transaction are processed at different sites of the system, while, in the commit stage, a commit protocol is used to ensure transaction atomicity. Although, there are several factors that contribute to the difficulty in meeting distributed real time transaction deadlines, data conflicts among transactions, especially in the commitment phase, are the prime factor resulting in system performance degradation. Data conflicts can occur between two transactions in execution stage (execute-execute conflicts) or between one transaction in execution stage and the other in commit stage (execute-commit conflicts). In the literature, the issue of handling data conflicts between two executing transactions has been addressed up to some extent. However, very little work has been done on the issue of handling data conflicts between the executing-committing transactions. The commit processing in DRTDBS can significantly increase the execution time of a transaction. This may adversely affect the system's ability to meet transaction deadlines. Therefore, designing of a good commit protocol is important for DRTDBS not only for fault resilience and speed of recovery but also for normal processing.

4.2 BACKGROUND AND RELATED WORK

The Two Phase Commit (2PC) is still one of the most commonly used protocols in the study of DRTDBS [40]. Most of the existing protocols proposed in the literatures [39,43,44,45,46,47,57,58,59,74,78,94,98,117,125] are based on it.

The 2PC based optimistic commit protocols proposed by Gupta et al. [44,46] for real-time databases try to improve system concurrency by allowing executing transactions to borrow data from the transactions in their commit stage. This creates dependencies among transactions. If a transaction depends on other transactions, it is not allowed to start commit processing and is blocked until the transactions, on which it depends, have committed. The blocked committing transaction may include a chain of dependencies as other executing transactions may have data conflicts with it. Enhancement has been made in the PROMPT commit protocol, which allows executing transactions to borrow data in a controlled manner only from the healthy transactions in their commit phase [55,59]. However, it does not consider the type of dependencies between two transactions. The abort of a lending transaction aborts all the transactions dependent on it. The technique proposed by Lam et al. maintains three copies of each modified data item (before, after and further) for resolving execute-commit conflicts [74,78]. This not only creates additional workload on the system but also has priority inversion problems. Based on the concepts of above protocols [59,78], Biao Qin and Y. Liu [108] proposed a protocol 2SC which classifies the dependencies between lender and borrower into two types; commit and abort. The abort of a lending transaction only forces transactions in its abort dependency set to abort. The transactions in the commit dependency set of the aborted lending transaction continue as normal. However, 2SC creates inconsistency in case of write-write conflicts.

The protocols [43,44,45,46,47,57,58,59,108], that allow an executing cohort to borrow data from a committing cohort, do not allow the borrower to send WORKDONE/PREPARED message and block it until the lender commits. These protocols either use blind write model or update model. In this chapter, we analyze all kind of dependencies that may arise due to data access conflicts between the executing-committing cohorts considering an update model. Then, we propose a new protocol SWIFT, that is a static locking and higher priority based distributed real time commit protocol. Here, the execution phase of a cohort is divided into two parts locking phase and processing phase, and in place of WORKDONE message, a WORKSTARTED message is sent just before the start of processing

phase of the cohort. A cohort may wait due to data contention only during its locking phase. Once, it acquires all the locks and enters into the processing phase, the transaction either completes or is aborted by a higher priority transaction, the chances of abortion being very low. In case of dependency, borrower is allowed to send WORKSTARTED message instead of being blocked as opposite to the other protocols [43,44,45,46,47,57,58,59,108], if the dependency is commit dependency only. This reduces the time needed for commit processing and is free from cascaded aborts. To ensure non-violation of the ACID [39,40,41,48,109,140] properties, checking of completion of cohort's processing and the removal of its dependency are required before sending YES VOTE message. The performance analysis of SWIFT shows that the proposed protocol considerably improves the success ratio of transactions. The performance of SWIFT has also been analyzed for partial read-only optimization [24], which minimizes intersite message traffic, execute-commit conflicts and log writes, resulting in a better response time. Analysis has also been done for the case where cohorts of the same transaction are permitted to communicate with each other [78].

The remainder of the chapter is organized as follows. We first introduce the DRTDBS model, and then the strategies for resolving data access conflicts among transactions followed by the basic concept and pseudo code of SWIFT. The performance measures, simulation results of SWIFT, impact of partial read optimization and impact of permitting communication among the cohort & its siblings on the performance of the SWIFT have also been discussed. We have also studied the percentage of the aborted transactions during different phases of the transaction's lifetime through simulation.

4.3 DISTRIBUTED REAL TIME DATABASE SYSTEM MODEL

In this section, we describe DRTDBS model that we used to evaluate the performance of the proposed commit protocol. The model consists of a database distributed in a non-replicated manner over several sites connected by a network.

4.3.1 System Model

The performance of SWIFT was evaluated by developing two simulation models for DRTDBS as follows.

- (i) The first one is for the main memory resident distributed real time database system which eliminates the impact of different disk scheduling algorithms on the performance. Whole database is assumed to reside in the main memory.
- (ii) Since, the main memory resident databases are not common in commercial database systems, we also employ another model which is for the disk resident distributed real time database system where whole database resides in the disk. This model allows the database to reside on the disk with a portion residing in the main memory buffer pool.

The structure of our simulation model and the description of its various components such as system model, database model, network model, locking mechanism, cohort execution model etc. are same as in section 3.2. The deadlines of the global and local transactions are computed by the method already described in section 3.3.

4.3.2 Model Assumptions

We assume that the transactions are firm real time. Each transaction in this model exists in the form of a coordinator process that executes at the originating site of the transaction and a collection of the cohorts executing at remote sites, where the required data items reside. If there is any local data in the access list of the transaction, one cohort is executed locally [42]. Before accessing a data item, the cohort needs to obtain locks on the data item. We also assume that:

- Processing of a transaction requires the use of CPU and data items located at local site or remote site.
- Arrival of the transactions at a site is independent of the arrivals at other sites and uses Poisson distribution.

- Each cohort makes read and update accesses.
- Each transaction pre-declares its read-set (set of data items that the transaction will only read) and update-set (set of data items that the transaction will update).
- S2PL-HP protocol is used for locking the data items.
- The cohorts are executed in parallel.
- A lending transaction cannot lend the same data item in read/update mode to more than one cohort to avoid cascaded abort.
- A cohort already in the dependency set of another cohort cannot permit a third incoming cohort to perform read or update.
- A distributed real time transaction is said to commit if the coordinator has arrived at the commit decision before the expiry of the deadline at its site. This definition applies irrespective of whether the cohorts have also received and recorded the commit decision by the deadlines or not [59].
- The database is either in the main memory or in the disk at all sites depending on whether the database is main memory resident or the disk resident. Studies have been made for both the cases.
- The communication delay considered is either 0ms or 100ms.
- In the case of disk resident database, the buffer space is sufficiently large to allow the retention of data updates until the commit time.

4.4 DATA ACCESS CONFLICTS RESOLVING STRATEGIES

The operations performed by distributed real time transactions can be classified as given below [107,114].

- **Blind Write Model (Write Only):** In this operation model, the transactions obtain state of the environment and write into the database.

- Update Model (Read before Write): In this operation model, the transactions derive new data and store in the database.
- Read only: In this operation model, the transactions only read data from the database.

Updating data items is more complex than querying data items. Update means editing (or changing) the database data satisfying some conditions. When a cohort updates a data item, all constraints must be enforced and controlled, so that the action of one cohort does not interfere with those of others and database remains consistent.

Biao Qin and Y. Liu [108] define two different types of dependencies in their paper: commit and abort. They assume that a write-write conflict only creates a commit dependency that allows a borrowing transaction to continue if the lending transaction aborts. In this case, if the borrowing transaction has to update some data item, it will first acquire a read lock (which will create an abort dependency) and will then upgrade it to write lock. Implementing an update in this way may sometimes lead to deadlocks. Let us consider the scenario where two transactions T_1 and T_2 want to update the same data item. They both get read lock. None of them can now upgrade it to write lock leading to a deadlock. In many commercial database systems, transactions request for an update lock for the avoidance of deadlock in a similar situation. If T_1 has an update lock, T_2 cannot get an update lock but T_2 can get a read lock. The protocol 2SC can be easily extended to handle read, write and update locks. If a transaction T_1 has obtained a write lock on the data item (x), then T_1 can read and write x . In other words, we can say that write lock allows the user receiving the lock to read and to modify the data [32].

Failure atomicity is required to ensure that the transactions are atomic in the presence of failure. Consider the case of write-write conflicts in 2SC [108]. If cohort T_2 wants to lock the uncommitted data item (x) in write mode which has been already locked by committing cohort T_1 in write mode. It will get permission to lock and will be commit dependent on T_1 . If T_1 commits, T_2 will commit. Again, if T_1 aborts (abort decision of its coordinator due to abort of one or more of its

siblings), T_2 will still commit. The consistency of the database is only maintained, if both writes are blind write; otherwise it will leave database in an inconsistent state because T_2 has read the uncommitted dirty value of the data item and modified it which has not been made permanent due to abort of T_1 . If both writes are not blind, history would not be equivalent to any serial history and therefore, it is not serializable. If write is update type, then abort of T_1 must lead to abort of T_2 . So, T_1 will be abort dependent on T_2 .

4.4.1 Types of Dependencies

Sharing of data items in conflicting modes creates dependencies [111,112] among the conflicting transactions and constraint their commit order. We assume that a cohort requests an update lock if it wants to update a data item (x). The modified definitions of dependency used in this chapter are given below:

Commit Dependency (CD)

If a transaction T_2 updates a data item read by another transaction T_1 , a commit dependency is created from T_2 to T_1 . Here, T_2 is not allowed to commit until T_1 commits.

Abort Dependency (AD)

If transaction T_2 reads or updates an uncommitted data item written by transaction T_1 , an abort dependency is created from T_2 to T_1 . T_2 aborts, if T_1 aborts and T_2 is not allowed to commit before T_1 .

These dependencies are required to maintain the ACID properties of the transaction. Each transaction T_i , that lends its data while in PREPARED state to an executing transaction, maintains two sets: a commit dependency set (CDS) and an abort dependency set (ADS).

CDS (T_i): the set of transactions T_j , that are commit dependent on transaction T_i .

ADS (T_i): the set of transactions T_j , which are abort dependent on transaction T_i

Associated with each transaction is a health factor defined as follows:

$$HF(\text{health factor}) = \frac{\text{TimeLeft}}{\text{MinTime}}$$

Where, TimeLeft is the time left until the transaction's deadline, and MinTime is the minimum time required for commit processing. The health factor is computed at the point of time when the coordinator is ready to send YES VOTE messages. MinHF is the threshold that allows the data held by committing transaction to be accessed. The variable MinHF is the key factor to influence the performance of the protocol. In our experiments, we have taken MinHF as 1.2, the value of MinHF used in [59,108].

4.4.2 Type of Dependencies in Different Cases of Data Conflicts

When data conflicts occur, there are three possible cases of conflict.

Case 1: Read-Update Conflict.

If transaction T_2 requests an update-lock while transaction T_1 is holding a read-lock, a commit dependency is defined from T_2 to T_1 . First, the transaction identity (id) of T_2 is added to the CDS (T_1). Then T_2 acquires the update-lock.

Case 2: Update-Update Conflict.

If both locks are update-locks and $HF(T_1) \geq \text{MinHF}$, an abort dependency is defined from transaction T_2 to transaction T_1 . The transaction identity (id) of T_2 is added to ADS (T_1), and T_2 acquires the update-lock; otherwise, T_2 is blocked.

Case 3: Update-Read Conflict

If transaction T_2 requests a read-lock while transaction T_1 is holding an update-lock and $HF(T_1) \geq \text{MinHF}$, an abort dependency is defined from T_2 to T_1 . The transaction identity (id) of T_2 is added to ADS (T_1), and T_2 acquires the read-lock; otherwise, T_2 is blocked.

On the basis of the data conflicts discussed above, the access of data items in conflicting mode are processed by lock manager as follows.

```
If (T2 CD T1)
{
    CDS (T1) = CDS (T1) ∪ {T2};
    T2 is granted update lock;
}
else
{
    if ((T2 AD T1) AND (HF(T1) ≥ MinHF))
    {
        ADS (T1) = ADS (T1) ∪ {T2};
        T2 is granted the requested lock (read or update);
    }
    else
        T2 will be blocked;
}
```

4.4.3 Mechanics of Interaction between Lender and Borrower Cohorts

If transaction T₂ has borrowed the data item locked by transaction T₁, following three scenarios are possible:

Scenario 1: T₁ receives decision before T₂ is going to start processing phase after getting all its locks.

If the global decision is to commit, T₁ commits.

- (i) All cohorts in ADS (T₁) and CDS (T₁) will execute as usual and the sets ADS (T₁) and CDS (T₁) are deleted.

(ii) If the global decision is to abort, T_1 aborts. The cohorts in the dependency sets of T_1 will execute as follows:

- All cohorts in ADS (T_1) will be aborted;
- All cohorts in CDS (T_1) will execute as usual;
- Sets ADS (T_1) and CDS (T_1) are deleted.

Scenario 2: T_2 is going to start processing phase after getting all locks before T_1 receives global decision.

T_2 is allowed to send a WORKSTARTED (discussed later) message to its coordinator, if it is commit dependent only; otherwise it is blocked from sending the WORKSTARTED message (So, the coordinator cannot initiate the commit processing operation). It has to wait, until

Alternative 1: either T_1 receives its global decisions, or
Alternative 2: its own deadline expires,
whichever occurs earlier.

In case of alternative 1, the system will execute as in scenario 1, whereas in the case of alternative 2, T_2 will be killed and will be removed from the dependency set of T_1 .

Scenario 3: T_2 aborts before T_1 receives decision

In this situation, T_2 's updates are undone and T_2 will be removed from the dependency set of T_1 .

4.5 A NEW COMMIT PROTOCOL SWIFT

A critical task in the execution of a transaction in DRTDBS is to ensure its consistent termination. This is the atomic commitment problem. To address this issue, we have designed a new real time commit protocol based on the concepts describe below.

4.5.1 Basic Idea

A commit protocol can improve transaction success percentage by

- (i) reducing the commit duration for each transaction,
- (ii) causing locks to be released sooner resulting in reduction of the wait time of other transactions, or
- (iii) allowing ordered sharing of the locks [24].

The ideas discussed below are based on the factors listed above.

(a) The execution of a transaction may be delayed due to resource (CPU and disk) or data contentions. The optimization proposed in this chapter is based on the following observations:

- (i) The data contention is the main cause of delay in a transaction's execution.
- (ii) A cohort may wait due to data contention only during its locking activity, i.e., in locking phase.

We, therefore, propose to divide the execution phase of the transaction into two parts:

- (i) Locking phase, and
- (ii) Processing phase

The execution of a cohort is carried out according to the following steps:

- (i) During the locking phase, the transaction locks the data items.
- (ii) Just before the start of processing phase, the cohort sends a WORKSTARTED message to its coordinator. Then, it is executed.
- (iii) After the receipt of WORKSTARTED messages from all its cohorts, the coordinator sends VOTE REQ message to all its cohorts at time t calculated as follows:

$$t = \max \{t_i + \text{processing_time}_i\} - T_{\text{com}}$$

where,

t_i = Arrival time of WORKSTARTED message from cohort $_i$

processing_time_i = Processing time needed by the cohort $_i$

T_{com} = Communication Delay from one site to another

(b) If cohort T_i utilizes dirty data items already locked by other cohorts, one of the following three types of dependencies may arise.

- (i) T_i may be commit dependent on other cohorts.
- (ii) T_i may be abort dependent on other cohorts.
- (iii) T_i may be both commit dependent as well as abort dependent on other cohorts.

Let us consider the case where transaction T_2 (borrower) acquires all locks and enters its processing phase before its lender transaction T_1 receives the global decision. If T_2 completes the processing before T_1 receives the global decision, existing commit protocols, including 2SC, block the borrower from sending WORKDONE message until the lender commits or aborts. We propose to allow a commit dependent borrower to send WORKSTARTED message as abort of the lender never aborts the borrower. Hence, one of the following two decisions is taken based on the type of dependencies when the cohort is about to enter its processing phase after getting all the required locks.

- (i) T_2 is allowed to send WORKSTARTED message to its coordinator if it is only commit dependent on other cohorts. This is free from cascaded aborts because abort of T_1 (lender) does not cause T_2 (borrower) to abort.
- (ii) T_2 is not allowed to send WORKSTARTED message to its coordinator if it is abort dependent on other cohorts. So, the coordinator cannot initiate

commit processing. Instead, it has to wait until either T_1 receives its global decisions or its own deadline expires, whichever occurs earlier.

(c) The cohort sends a YES VOTE message in response to its coordinator's VOTE REQ message only when its dependencies are removed and it has finished its processing. If it is still dependent on any cohort or has not finished its processing, YES VOTE message is deferred. The borrower sends the deferred YES VOTE message after the completion of processing and removal of the dependencies. This may be either due to abort or commit of the lender.

(d) The CPU scheduling algorithm for the cohorts is described below. Cohort processing is done on the basis of CPU availability and cohort's priority.

- (i) If two cohorts are ready to run on the same processor, the higher priority cohort is scheduled first.
- (ii) While in locking period, if a higher priority cohort (T_h) arrives, the lower priority cohort (T_L) aborts. T_L releases all its locked data items.
- (iii) If a higher priority cohort (T_h) arrives during the processing phase of T_L it aborts the lower priority transaction (T_L). The aborted cohort/transaction releases all its locked data items.

The important point to note here is that all modifications discussed above are local to each site and do not require inter-site communications. Moreover, the proposed optimization can be integrated with any other protocol based on 2PC.

4.5.2 Algorithm

On the basis of above discussion, the complete pseudo code of the proposed protocol is given below.

if (T_1 receives global decision before T_2 is going to start processing phase after getting all locks)

{

```

ONE: if (T1' s global decision is to commit)
{
    T1 commits;
    All cohorts in ADS (T1) and CDS (T1) will execute as usual;
    Delete set of ADS (T1) and CDS (T1);
}
else // T1' s global decision is to abort
{
    T1 aborts;
    All cohorts in CDS (T1) will execute as usual;
    Transaction in ADS (T1) will be aborted;
    Delete sets of ADS (T1) and CDS (T1);
}
}

else if (T2 is going to start processing phase after getting all locks before
T1 receives global decision)
{
    check type of dependencies;
    if (T2' s dependency is commit only)
        T2 sends WORKSTARTED message;
    else
    {
        T2 is blocked for sending WORKSTARTED message;
        while ( !(T1 receive global decision OR T2 misses
        deadline));
        if (T2 misses deadline)
            {
                Undo computation of T2;
            }
    }
}

```

```

        Abort T2;
        Delete T2 from CDS (T1) & ADS (T1);
    }
else
    GoTo ONE;
}
}
else
    // T2 is aborted by higher transaction before T1 receives decision
    {
        Undo computation of T2;
        Abort T2;
        Delete T2 from CDS (T1) & ADS (T1);
    }
}

```

This protocol has been named SWIFT. The name is derived from a static two phase locking with higher priority based, write-update type, ideal for fast and timeliness commit protocol.

4.6 PERFORMANCE MEASURES AND EVALUATION

For performance evaluation of different protocols, a simulator was constructed. The purpose of simulator is to facilitate the performance evaluation of the system under a variety of operational conditions that affect the performance.

4.6.1 Performance Parameters and Measures

A distributed real time database system consisting of N sites was simulated in the present work on the basis of simulator developed in section 3.6.1. The default values of different parameters used in the simulation experiments are same as given in Table 3.1.

The concurrency control scheme used is S2PL-HP. When a cohort requests a lock on a data item that is held by one or more higher priority cohorts in a conflicting mode, the requesting cohort waits for the data item to be released. On the other hand, if the data item is held by a lower priority cohort in a conflicting way, the lower priority cohort is aborted and requesting cohort is granted the desired locks.

EDF [93] policy is used to assign priorities to cohorts in all the experiments. If two cohorts have same deadline, the one with earliest arrival is assigned higher priority. Our primary performance measure is the proportion of missed deadlines (i.e. Miss Percentage, MP) which is defined as the percentage of input transactions that the system is unable to complete on or before their deadlines.

The Miss Percentage values in the range of 0 to 20 percent are taken to represent system performance under normal load, while Miss Percentage in the range of 20 to 100 percent represents system performance under heavy load [37,51].

In our simulation model, a small database (200 data items) was used to create a high data contention environment. This helps us in understanding the interaction among the different policies [73]. A small database means that degree of data contention in the system can easily be controlled by the sizes of the transactions. A small database also allows us to study the effect of hot-spots, in which a small part of the database is accessed very frequently by most of the transactions. However, we have not considered the hot spot issue. An event driven based simulator was written in C language. For each set of experiments, the final results were calculated as an average of 10 independent runs. In each run, 20000 transactions were initiated.

4.6.2 Simulation Results

Studies have been made for both the main memory resident and the disk resident databases at communication delay of 0ms and 100ms. It has been shown by Biao Qin and Y. Liu [108] that 2SC protocol performs better than other distributed real time commit protocols including PROMPT. Therefore, we compare SWIFT with following protocols.

- (i) PROMPT
- (ii) 2SC
- (iii) SWIFT- preliminary- Version- One (SWIFT-PV-1)
- (iv) SWIFT- preliminary- Version- Two (SWIFT-PV-2)

In SWIFT-PV-1, we have considered the basic concept of sending the WORKDONE message only, if the cohort is commit dependent on other cohorts, whereas, in SWIFT-PV-2, sending of WORKSTARTED message is considered before the start of processing phase. SWIFT is a combination of SWIFT-PV-1 and SWIFT-PV-2, where the concepts of the both protocols are used. SWIFT-PV-1 and SWIFT-PV-2 have also been compared with PROMPT and 2SC.

4.6.2.1 Main Memory Resident Database

Fig. 4.1 to Fig. 4.6 show the Miss Percentage for SWIFT-PV-1, SWIFT-PV-2 and SWIFT compared with PROMPT and 2SC at communication delays of 100ms and 0ms respectively as a function of the average transaction inter-arrival rate/site under normal and heavy load conditions. It can be seen that the proposed protocol SWIFT works better than PROMPT, 2SC, SWIFT-PV-1 and SWIFT-PV-2 under all load conditions. The performance improvements are primarily due to permitting the commit dependent cohorts to send their WORKSTARTED messages and by not allowing higher priority transactions to abort a borrower. Early sending of WORKSTARTED messages by the cohorts with commit dependency only minimizes the communication delay by overlapping the transaction processing time and message transmission time, thus reducing the overall time needed for commit processing. The performance is much better as compared to PROMPT and 2SC and is also better than SWIFT-PV-1 and SWIFT-PV-2 individually.

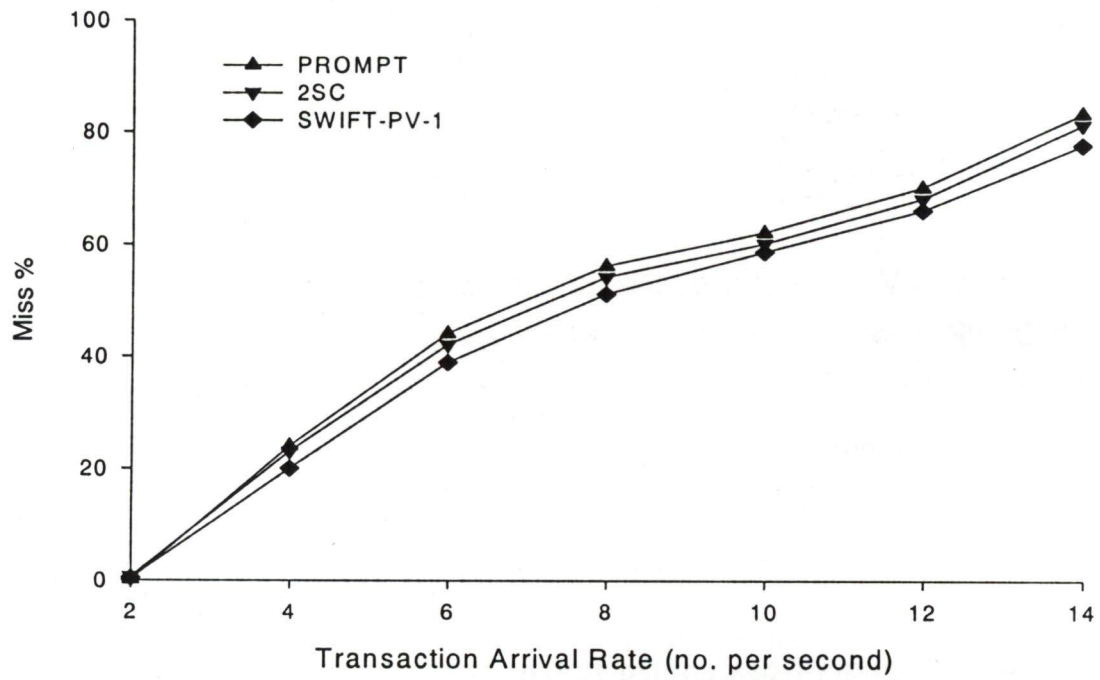


Fig. 4.1: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load

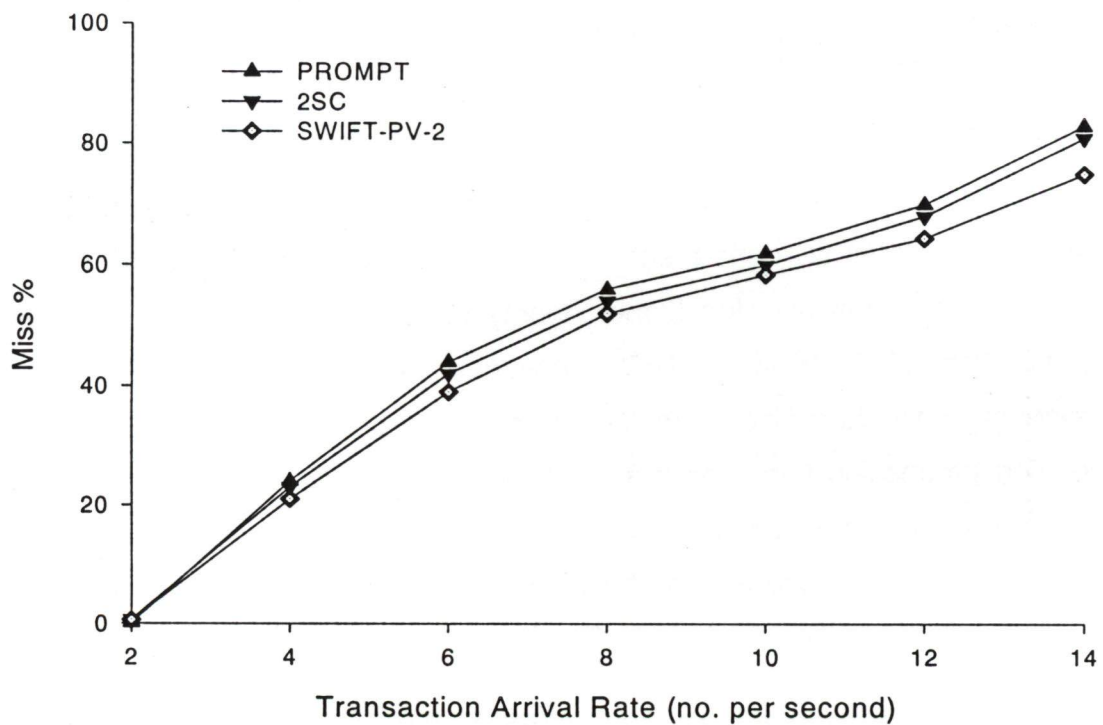


Fig. 4.2: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load

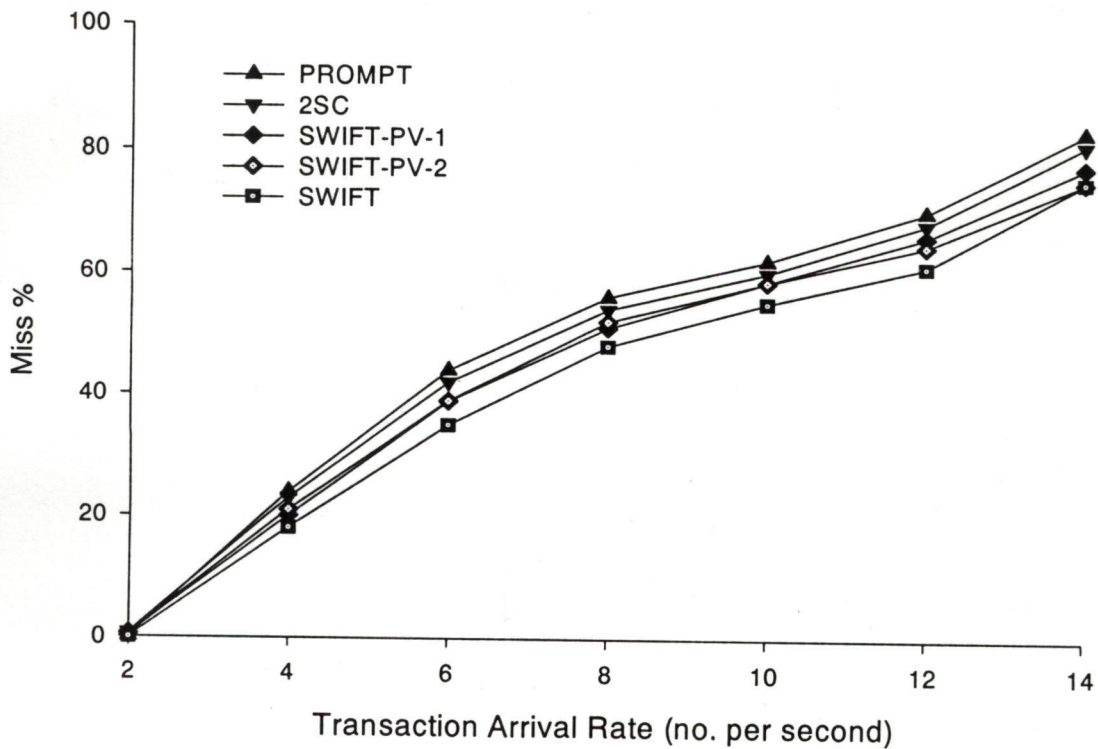


Fig. 4.3: Miss % with (RC+DC) at Communication Delay=100ms
Normal & heavy Load

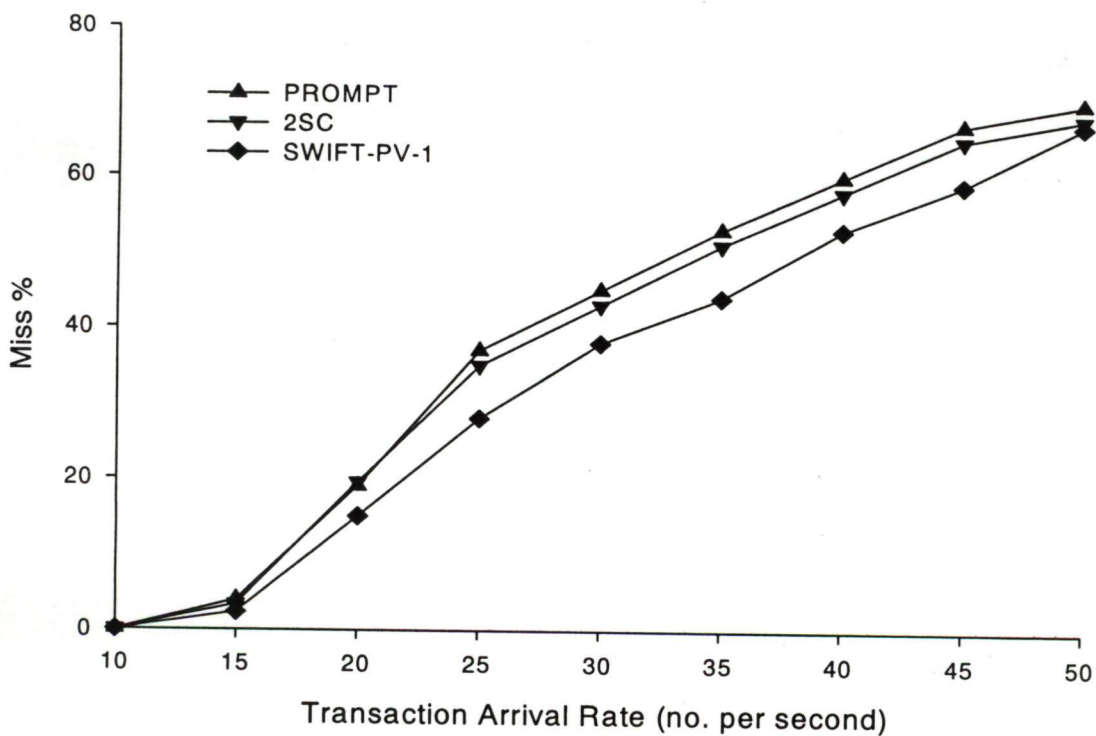


Fig. 4.4: Miss % with (RC+DC) at Communication Delay=0ms
Normal & Heavy Load

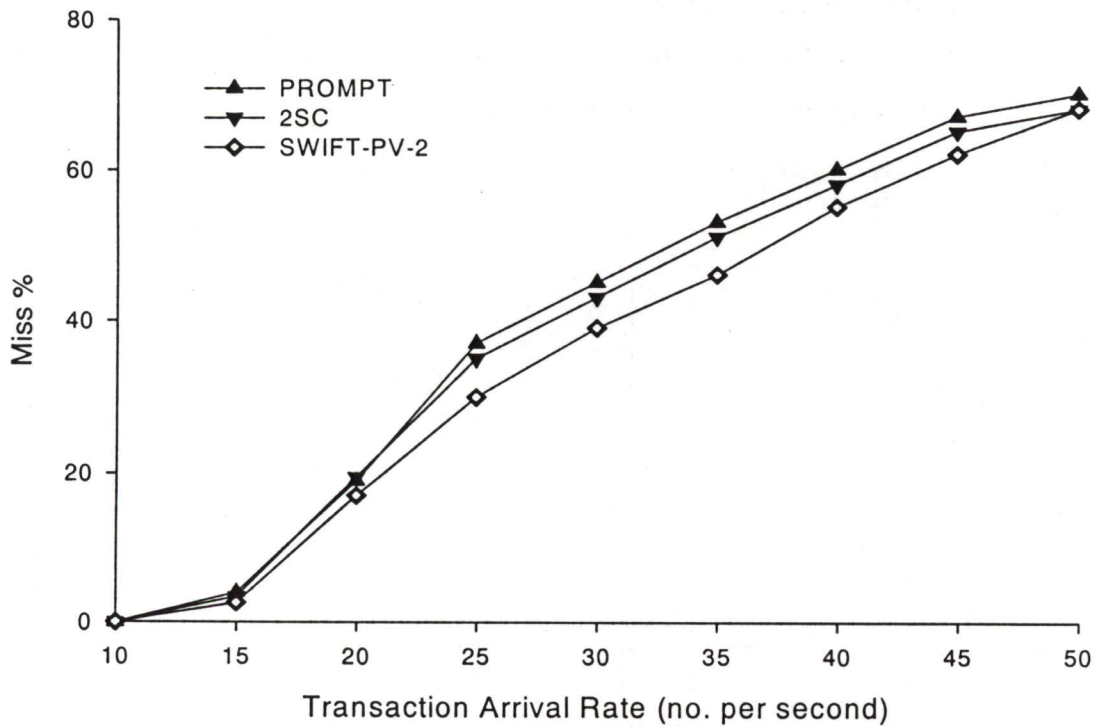


Fig. 4.5: Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load

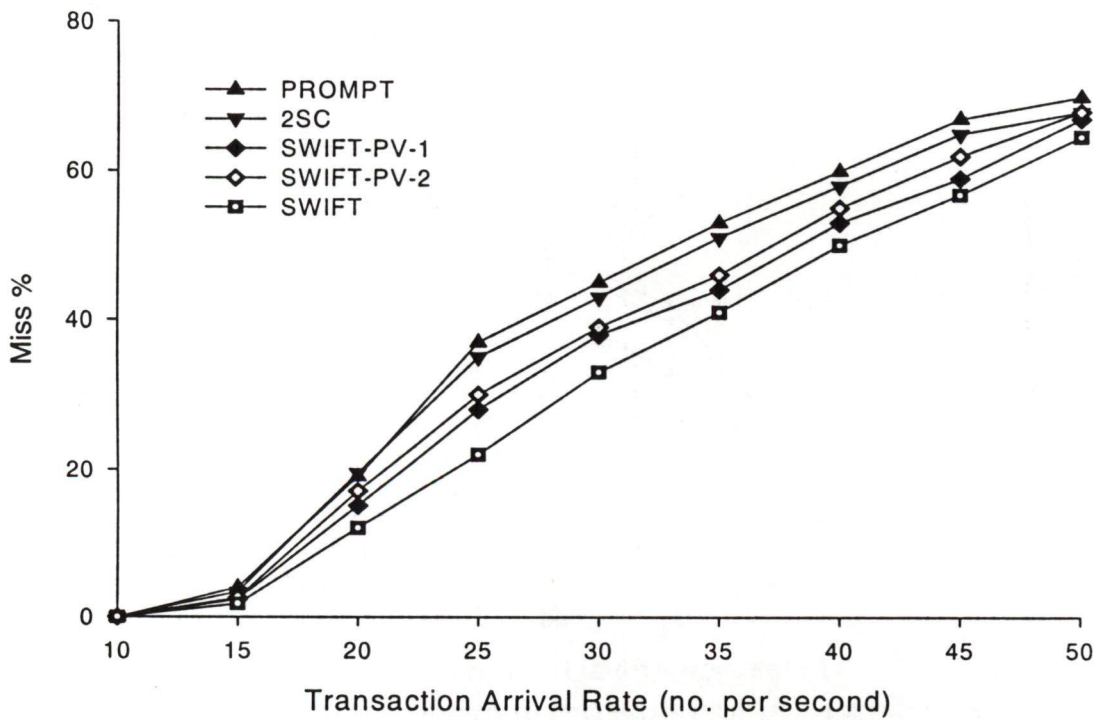


Fig. 4.6: Miss % with (RC+DC) at Communication Delay=0ms Normal & Heavy Load

4.6.2.2 Disk Resident Database

Fig. 4.7 to Fig. 4.15 show the Miss Percentage at communication delay of 100ms as well as 0ms for different transaction arrival rates in a disk resident database. It can be seen that the proposed protocol again works better than 2SC and PROMPT at communication delay of 100ms under all load conditions. The performance improvements are primarily due to permitting the commit dependent cohorts to send their WORKSTARTED messages and minimizing the queuing delay. Early sending of WORKSTARTED message by the cohorts having commit dependency only also minimizes the communication delay.

However, it is not better at communication delay of 0ms at higher transaction arrival rate. Rather, it is almost at par with 2SC and PROMPT due to higher number of aborts, increased number of dependent cohorts and longer queuing delay for the use of resources in the system.

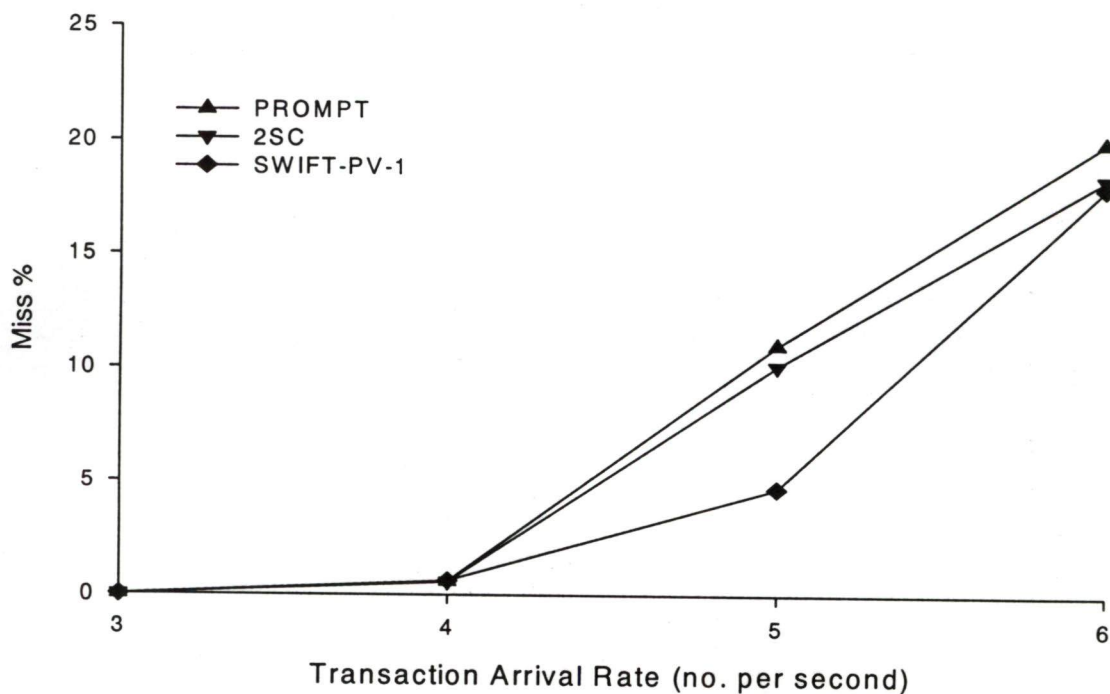


Fig. 4.7: Miss % with (RC+DC) at Communication Delay=0ms Normal Load

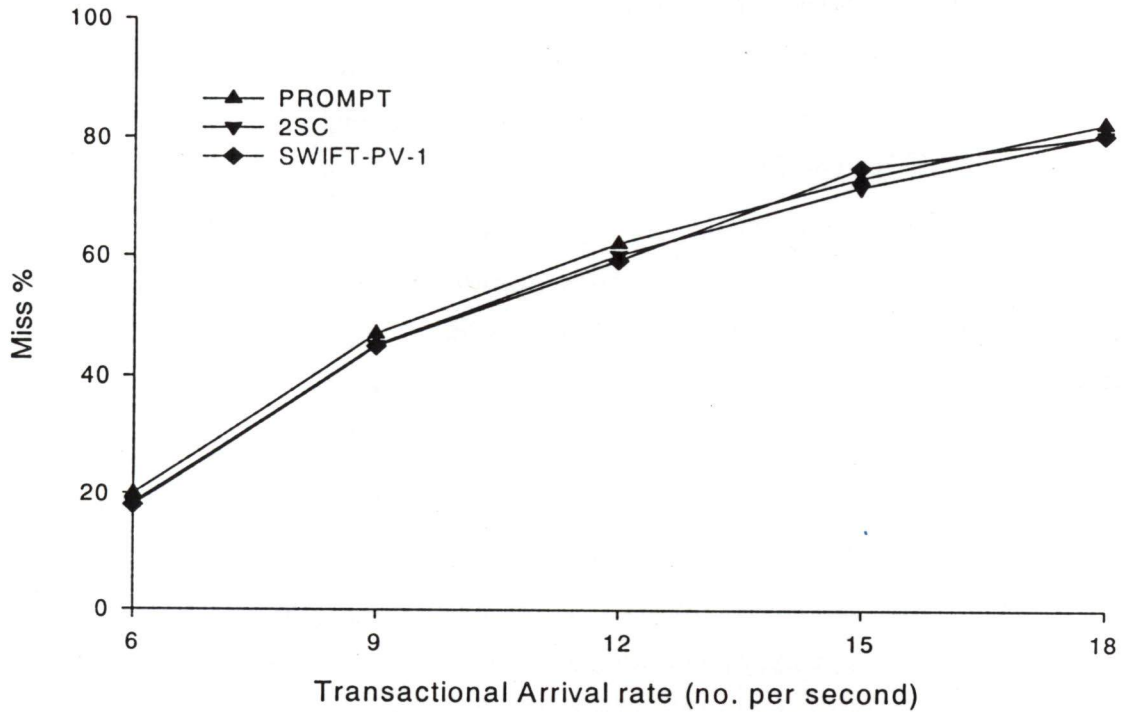


Fig. 4.8: Miss % with (RC+DC) at Communication Delay=0ms Heavy Load

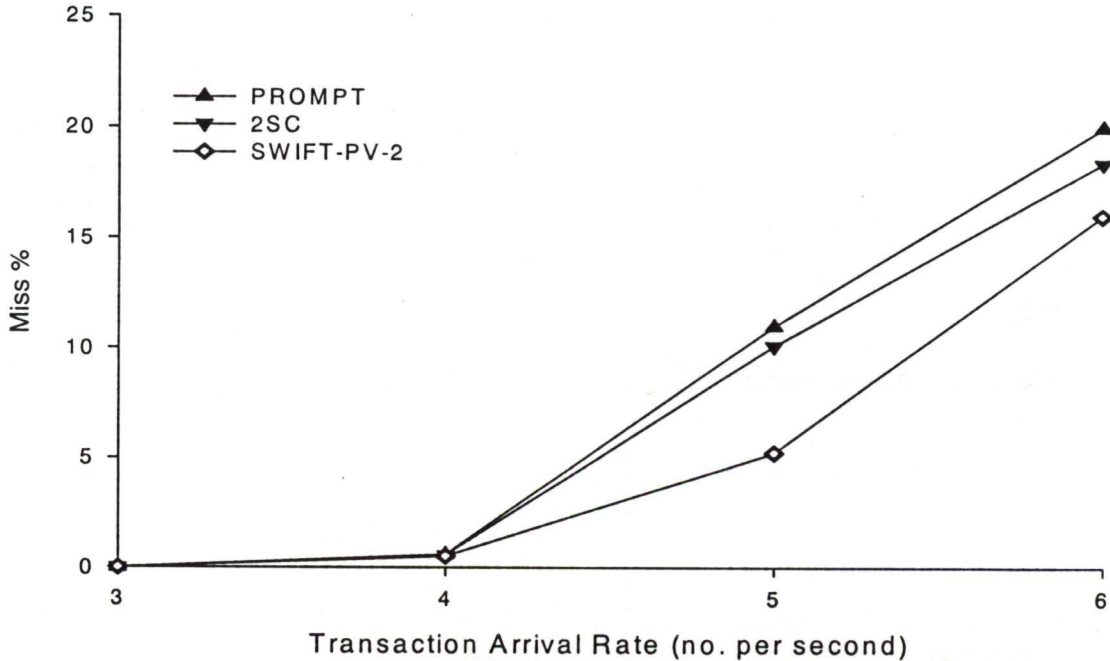


Fig. 4.9: Miss % with (RC+DC) at Communication Delay=0ms Normal Load

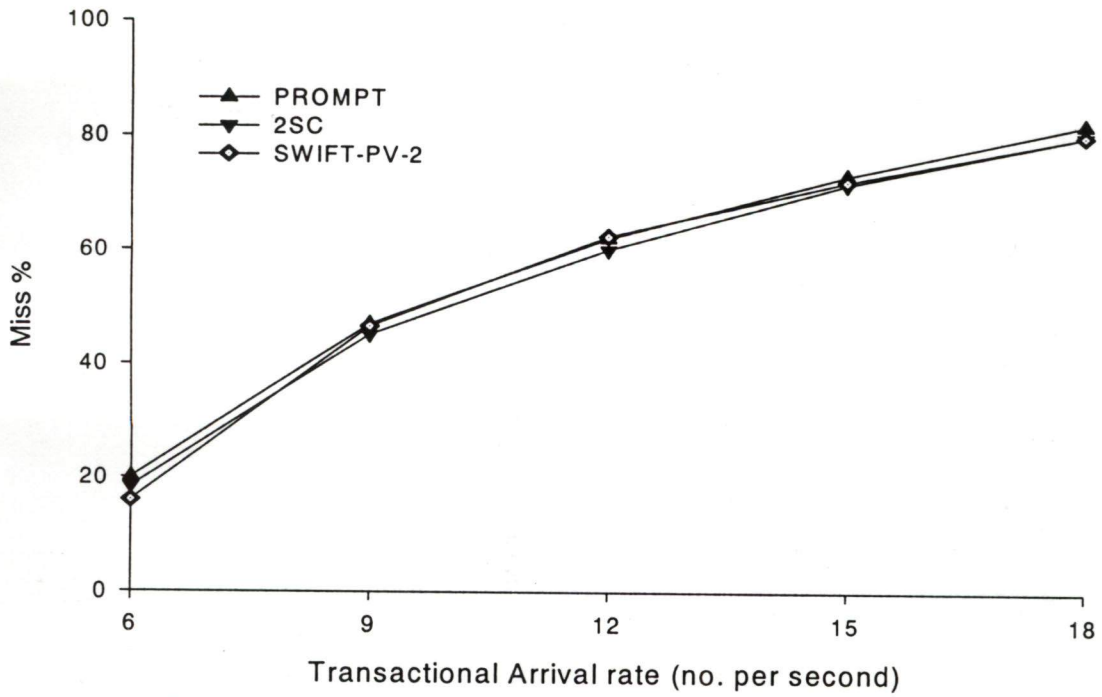


Fig. 4.10: Miss % with (RC+DC) at Communication Delay=0ms Heavy Load

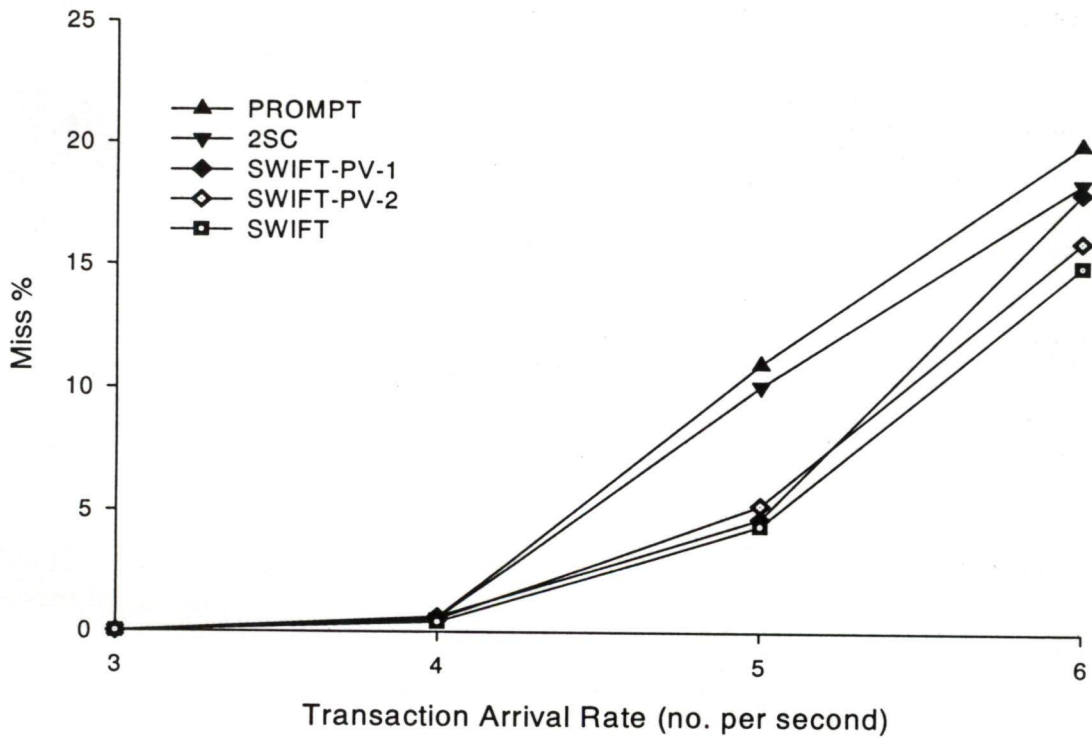


Fig. 4.11: Miss % with (RC+DC) at Communication Delay=0ms Normal Load

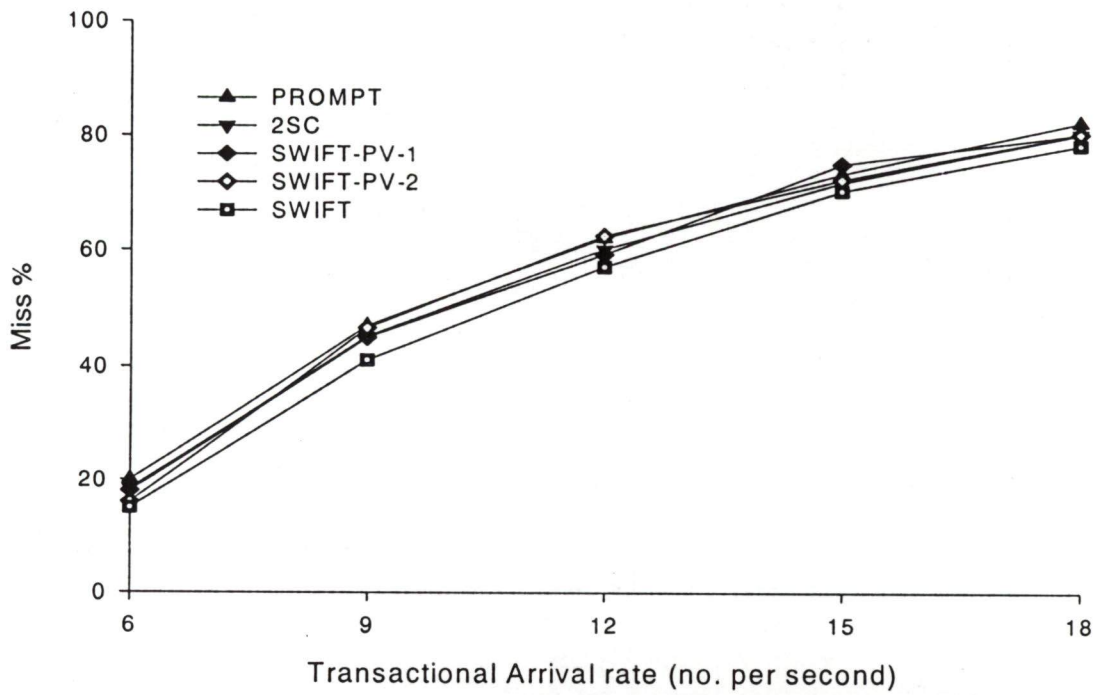


Fig. 4.12: Miss % with (RC+DC) at Communication Delay=0ms Heavy Load

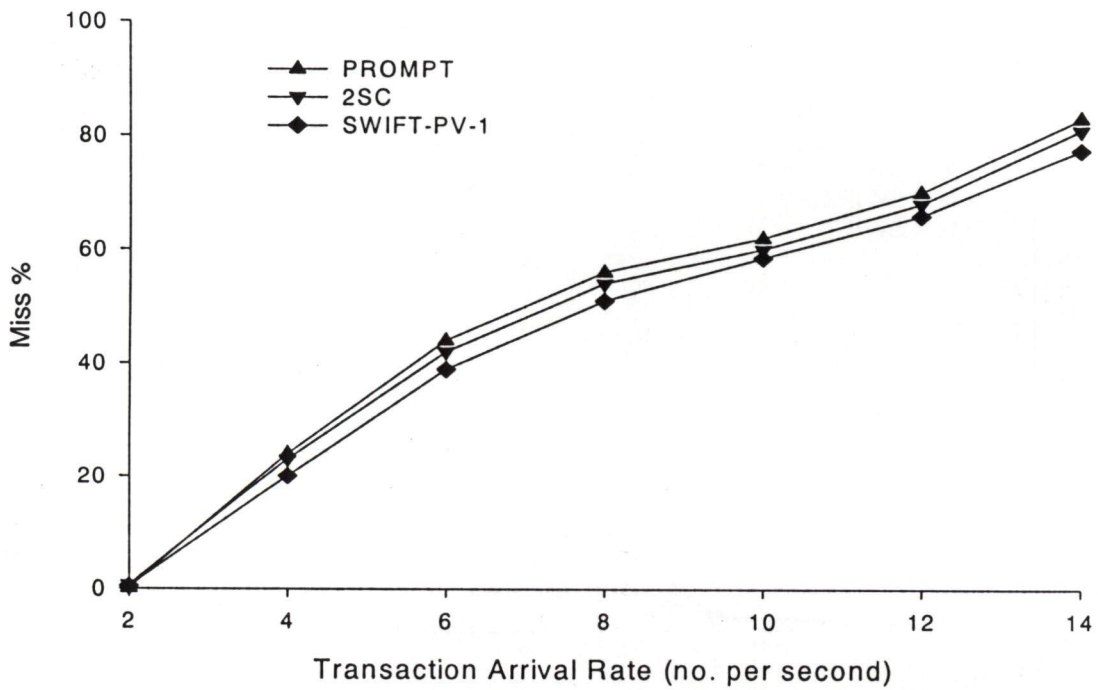


Fig. 4.13: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load

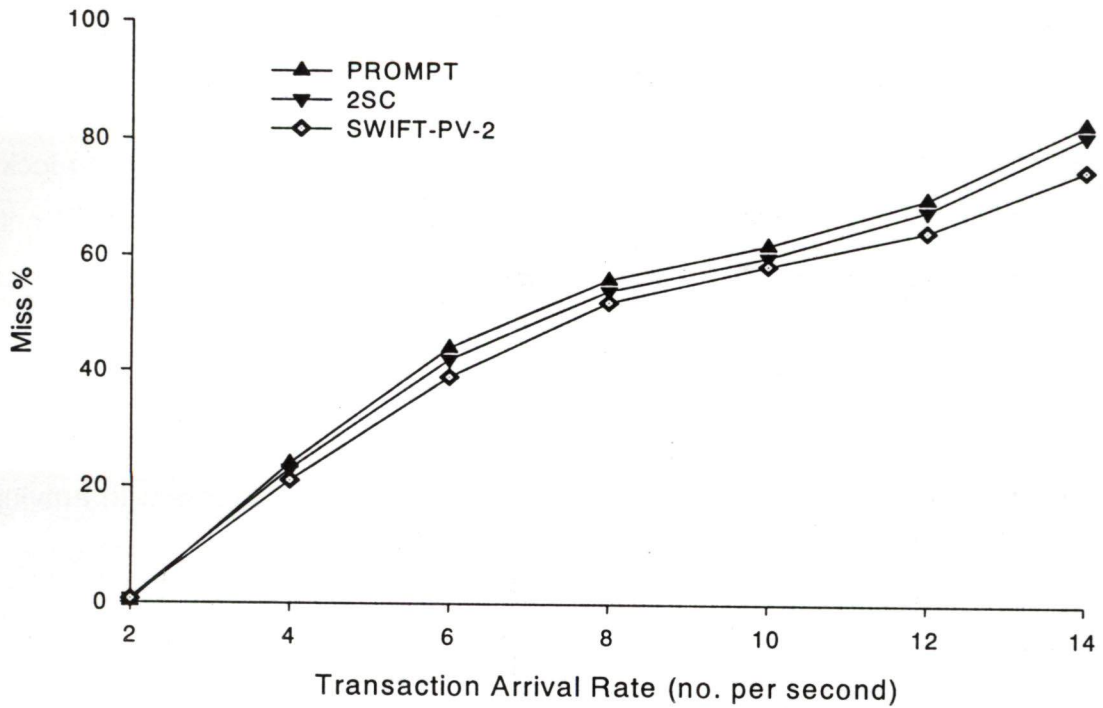


Fig. 4.14: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy Load

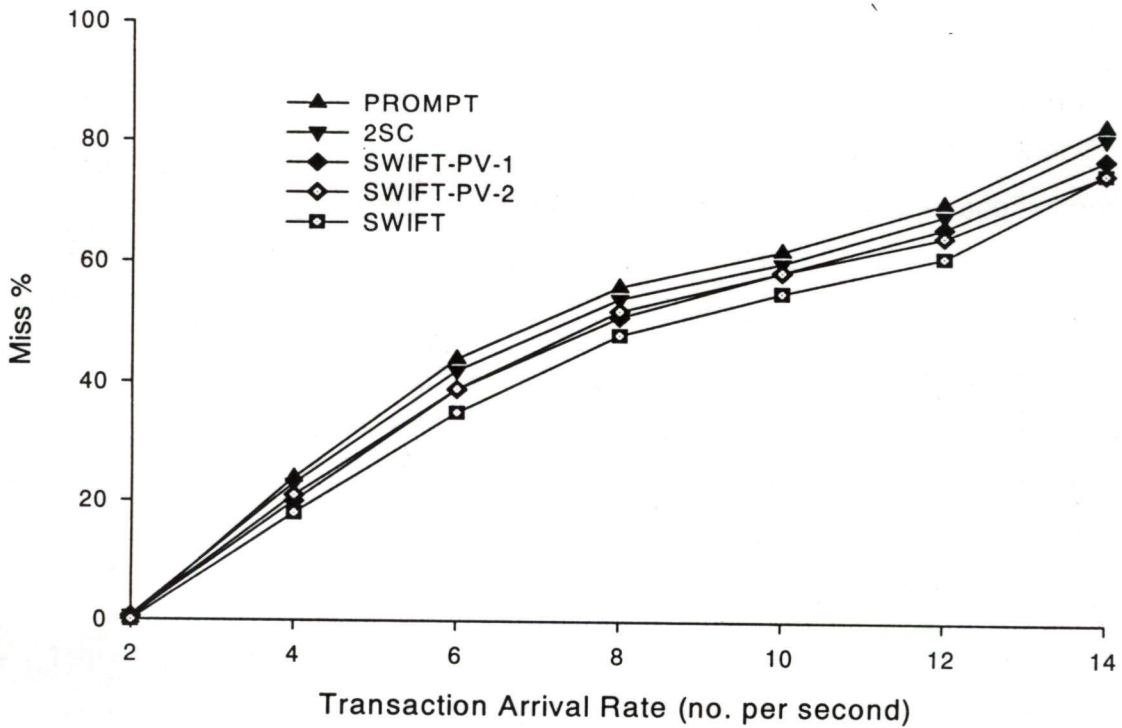


Fig. 4.15: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy load

4.7 PERFORMANCE OF SWIFT WITH PARTIAL READ OPTIMIZATION

Partial read optimization means that a cohort having read only locks has no work to do during commit and so does not need a VOTE REQ, commit or abort message from its coordinator. A cohort having read only locks will have no locks after sending WORKSTARTED message. This cohort may send a read-only WORKSTARTED message to its coordinator indicating that it is no longer needed by the cohort to participate in 2PC.

4.7.1 Possible Cases of Data Conflicts

Update-Update and Update-Read are the only possible conflicts with arriving cohorts in this case. So, the only dependency required in this case is given below:

Abort Dependency (ADS)

If transaction T_2 reads or updates an uncommitted data item updated by transaction T_1 , an abort dependency is created from T_2 to T_1 . T_2 aborts, if T_1 aborts and T_2 is not allowed to commit before T_1 .

4.7.2 Type of Dependencies in Cases of Data Conflicts

There are two possible cases of data conflict [108]. Let T_1 be the transaction in commit phase and T_2 be the transaction in execution phase.

Case 1: Update-Update Conflict

If both locks are update-locks and $HF(T_1) \geq \text{MinHF}$, an abort dependency is defined from T_2 to T_1 . The transaction identity (id) of T_2 is added to ADS (T_1), and T_2 acquires the update-lock; otherwise, T_2 is blocked.

Case 2: Update-Read Conflict

If T_2 requests a read-lock while T_1 is holding an update-lock and $HF(T_1) \geq \text{MinHF}$, an abort dependency is defined from T_2 to T_1 . The transaction identity (id) of T_2 is added to ADS (T_1), and T_2 acquires the read-lock; otherwise, T_2 is blocked.

4.7.3 Simulation Results

The effect of partial read only optimization has also been studied both for the main memory and the disk resident databases at communication delay of 0ms and 100 ms. As expected, the performance gain is better in all the cases. It varies in between 1% to 5% as shown in Fig. 4.16 to Fig. 4.20 for different cases. At low arrival rates, the gain is slight but it improves at higher arrival rates.

Main Memory Resident Database

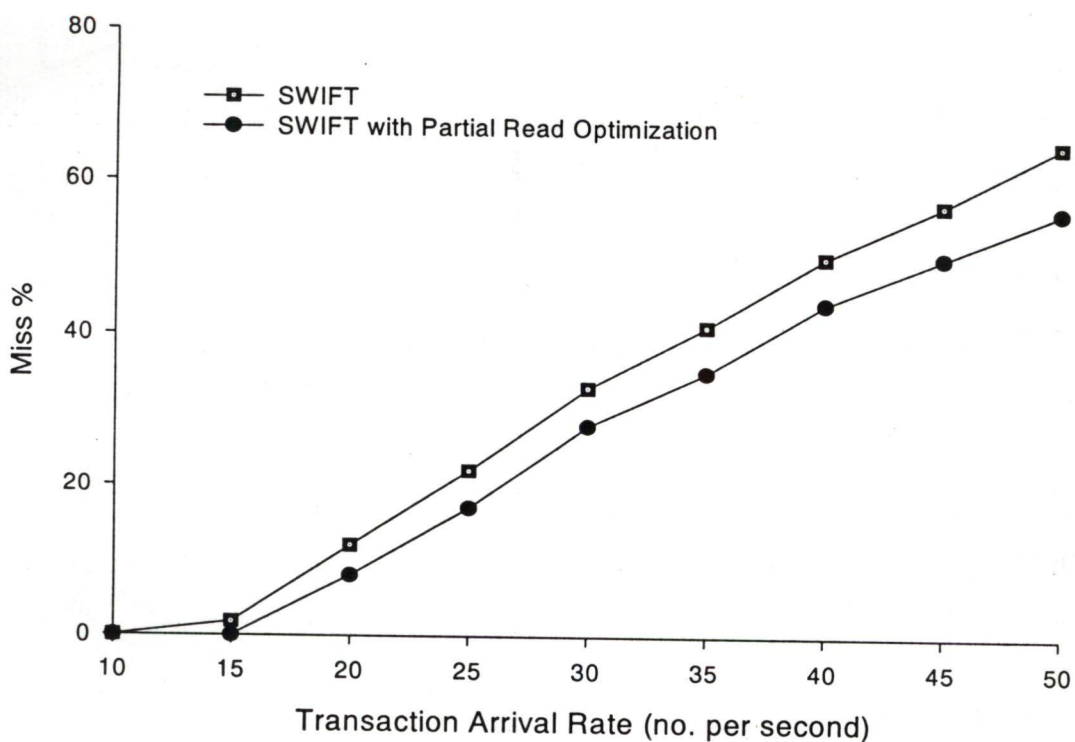


Fig. 4.16: Miss % with (RC+DC) at Communication Delay=0ms
Normal & Heavy Load

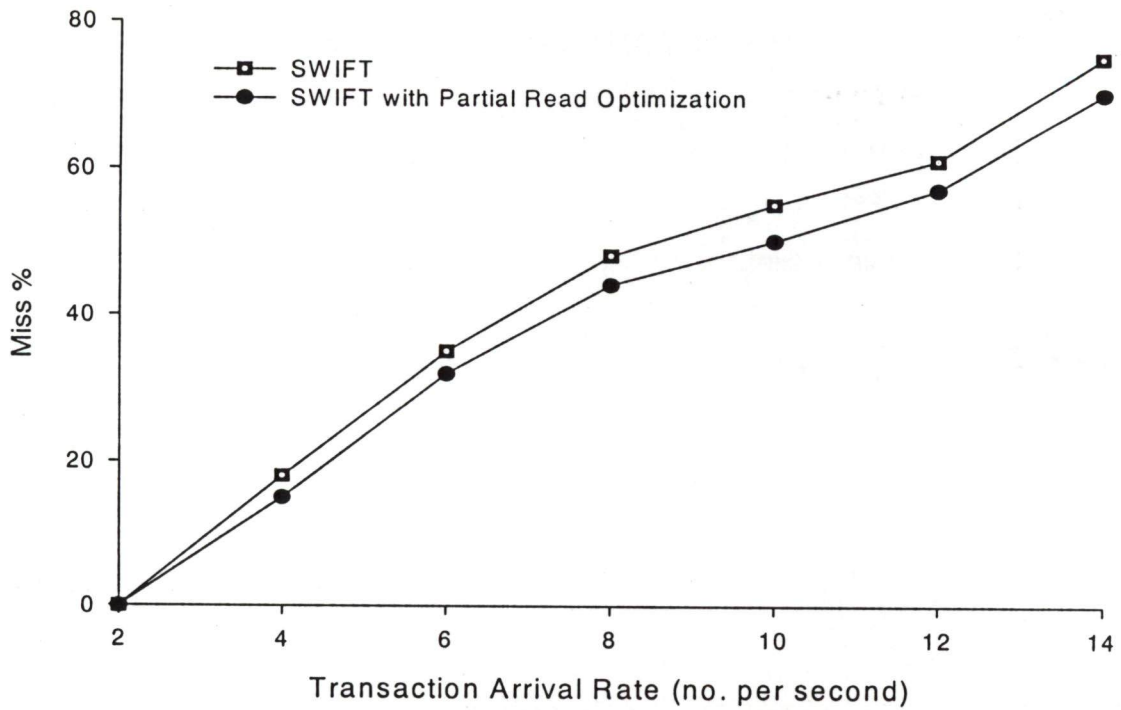


Fig. 4.17: Miss % with (RC+DC) at Communication Delay=100ms Normal & heavy Load

Disk Resident Database

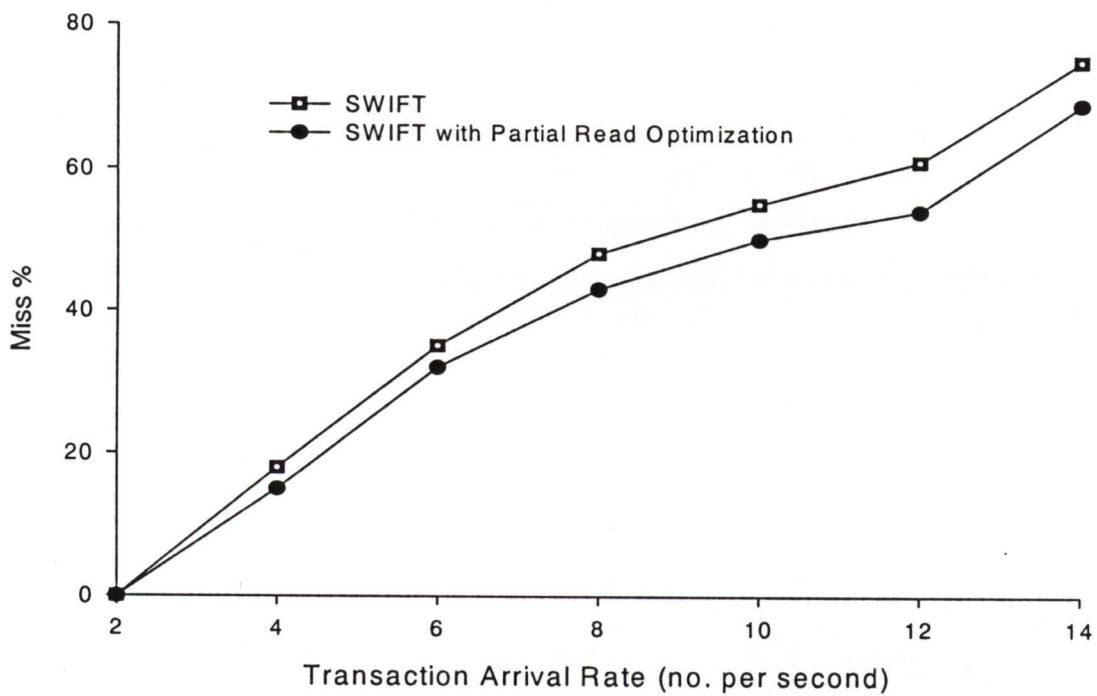


Fig. 4.18: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy load

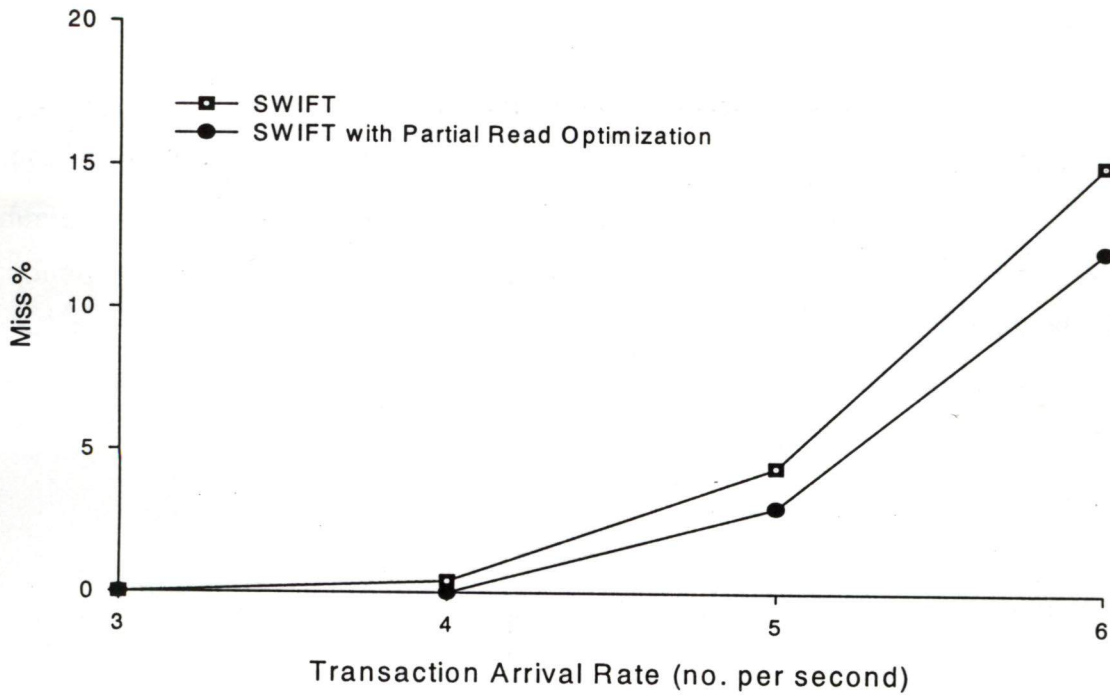


Fig. 4.19: Miss % with (RC+DC) at Communication Delay=0ms Normal Load

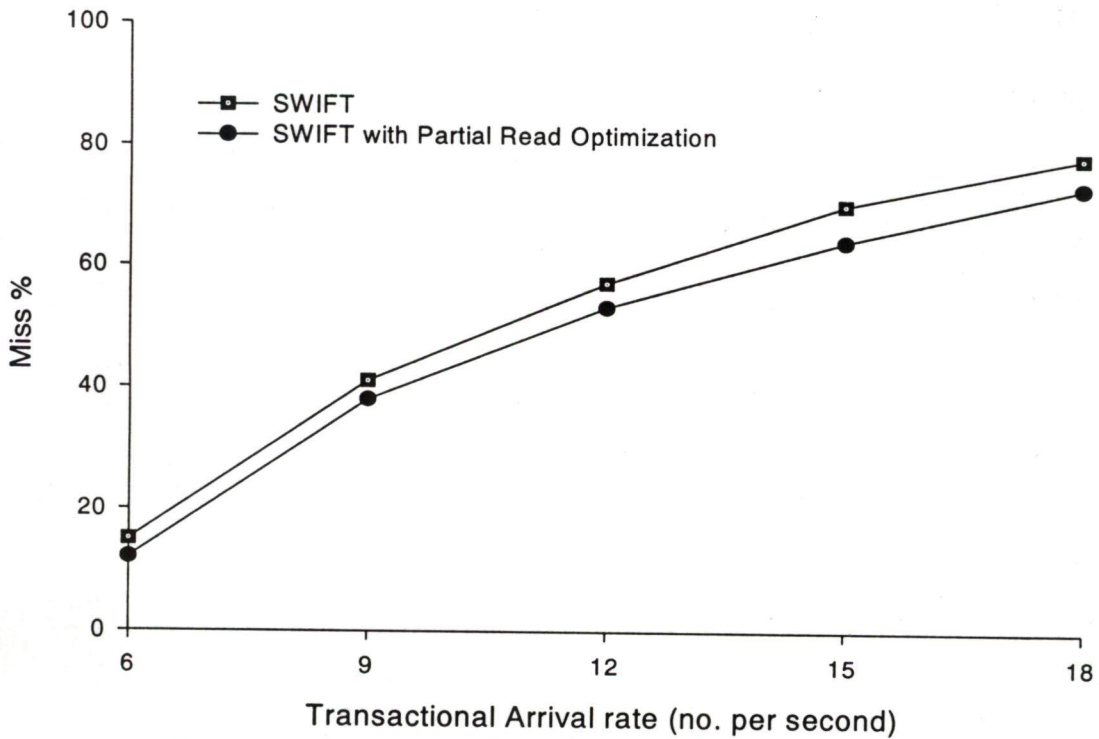


Fig. 4.20: Miss % with (RC+DC) at Communication Delay=0ms Heavy Load

4.8 COMMUNICATION AMONG THE COHORTS OF SAME TRANSACTION (CCST)

The effect of permitting the cohorts of the same transaction to communicate with each other has also been studied both for the main memory and the disk resident databases at communication delays of 0ms and 100 ms. An aborting cohort sends the announcement of its abort directly to its siblings as well as to its coordinator. Therefore, the coordinator does not need to send the abort message to rest of its cohorts. However, the performance gain with this optimization is very low. It is in between 1% to 3%, which has been shown in Fig. 4.21 to Fig. 4.25 for different cases.

Main Memory Resident Database

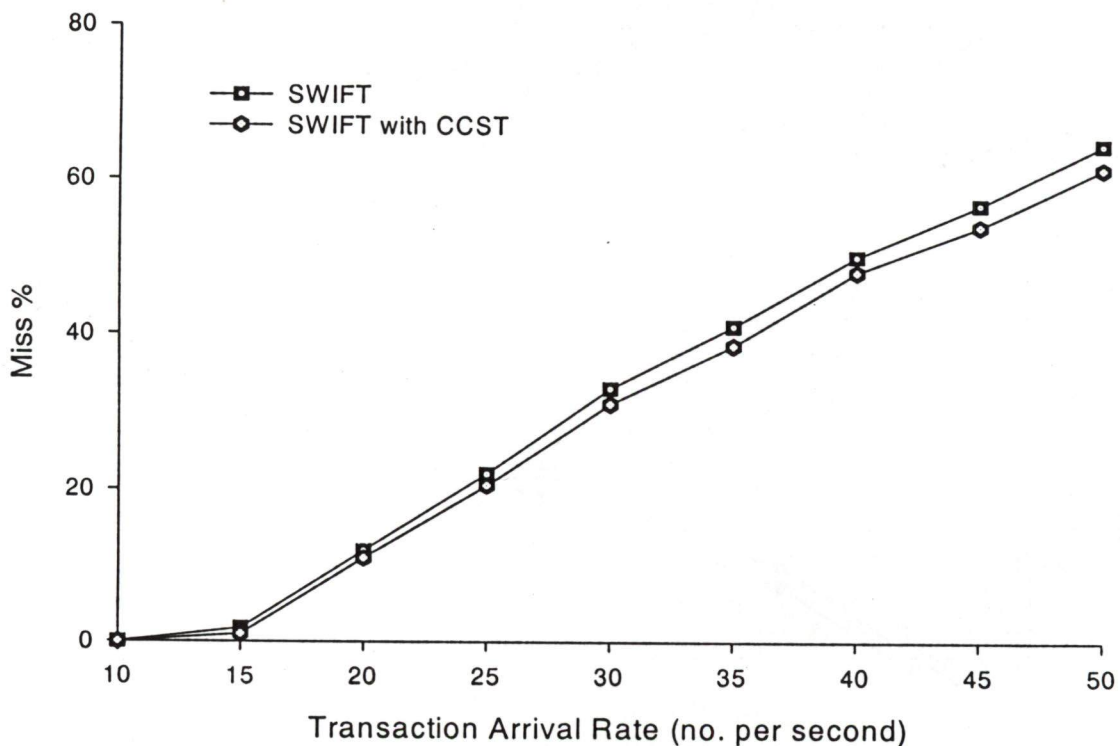


Fig. 4.21: Miss % with (RC+DC) at Communication Delay=0ms
Normal & Heavy Load

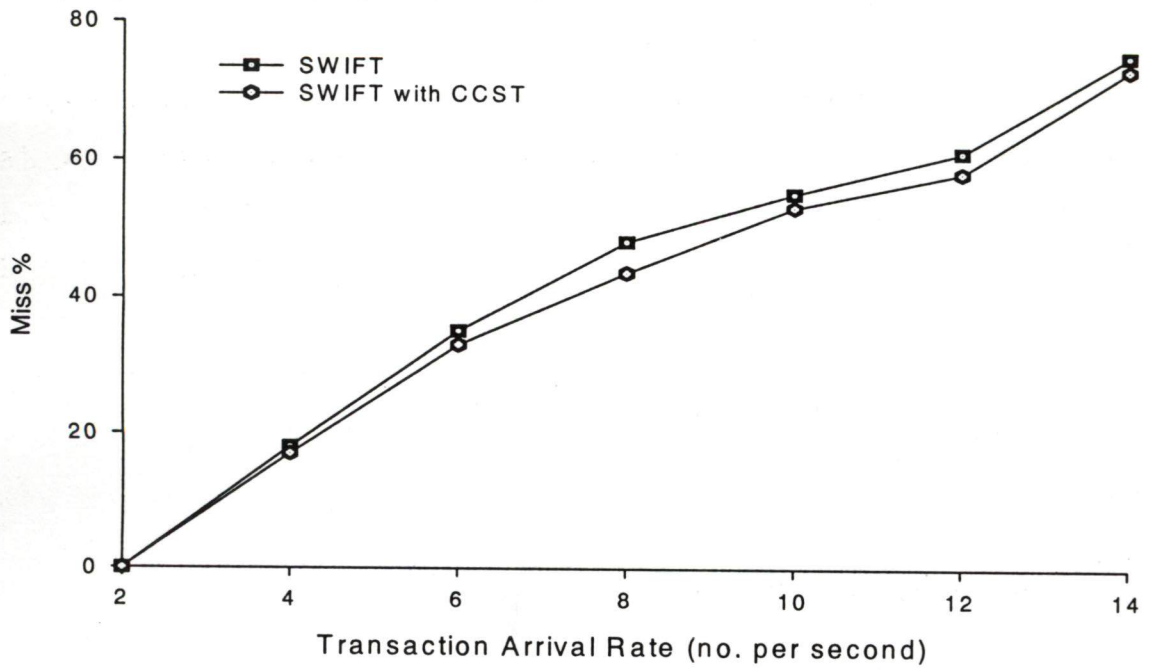


Fig. 4.22: Miss % with (RC+DC) at Communication Delay=100ms Normal & heavy Load

Disk Resident Database

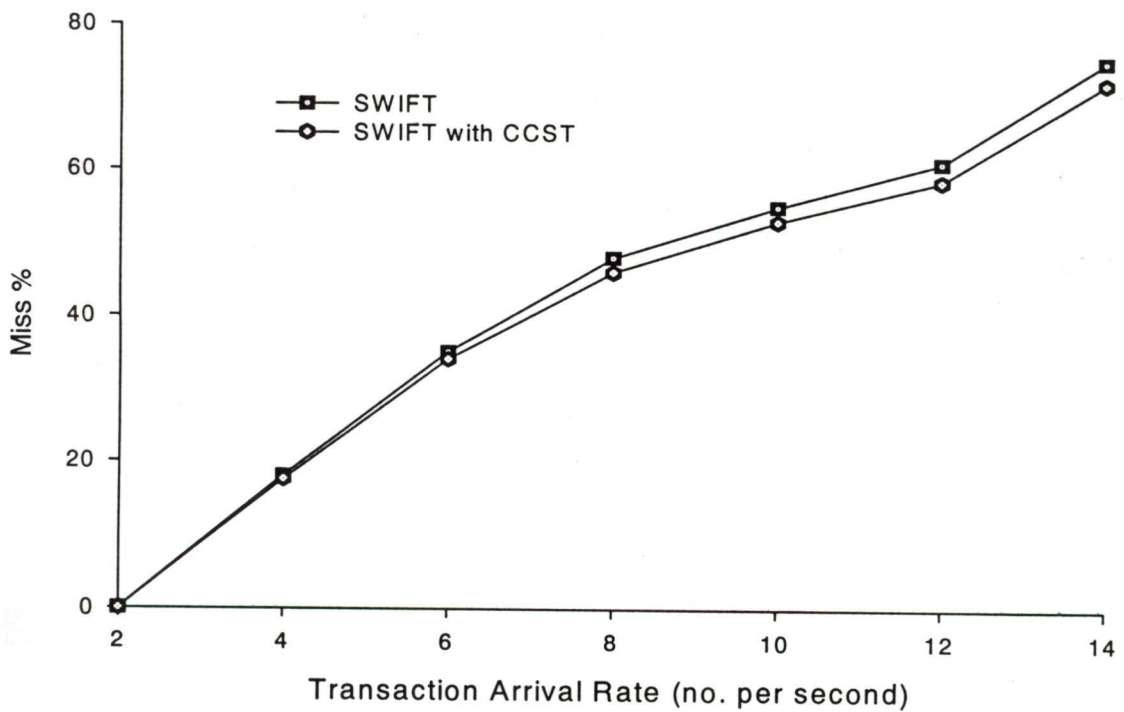


Fig. 4.23: Miss % with (RC+DC) at Communication Delay=100ms Normal & Heavy load

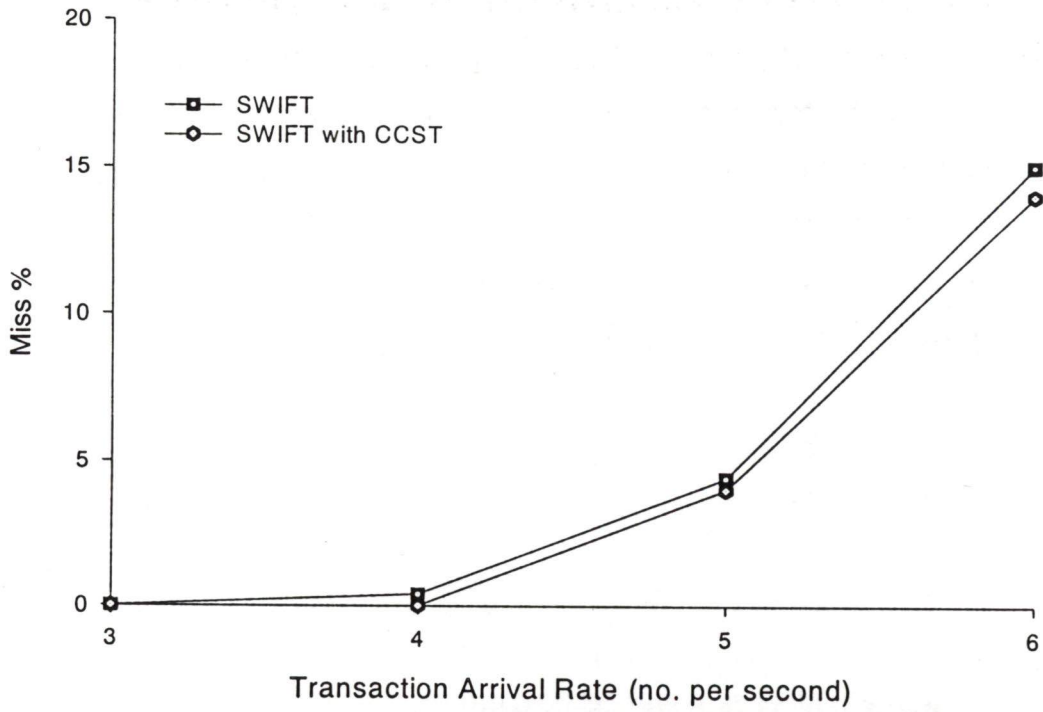


Fig. 4.24 Miss % with (RC+DC) at Communication Delay=0ms Normal Load

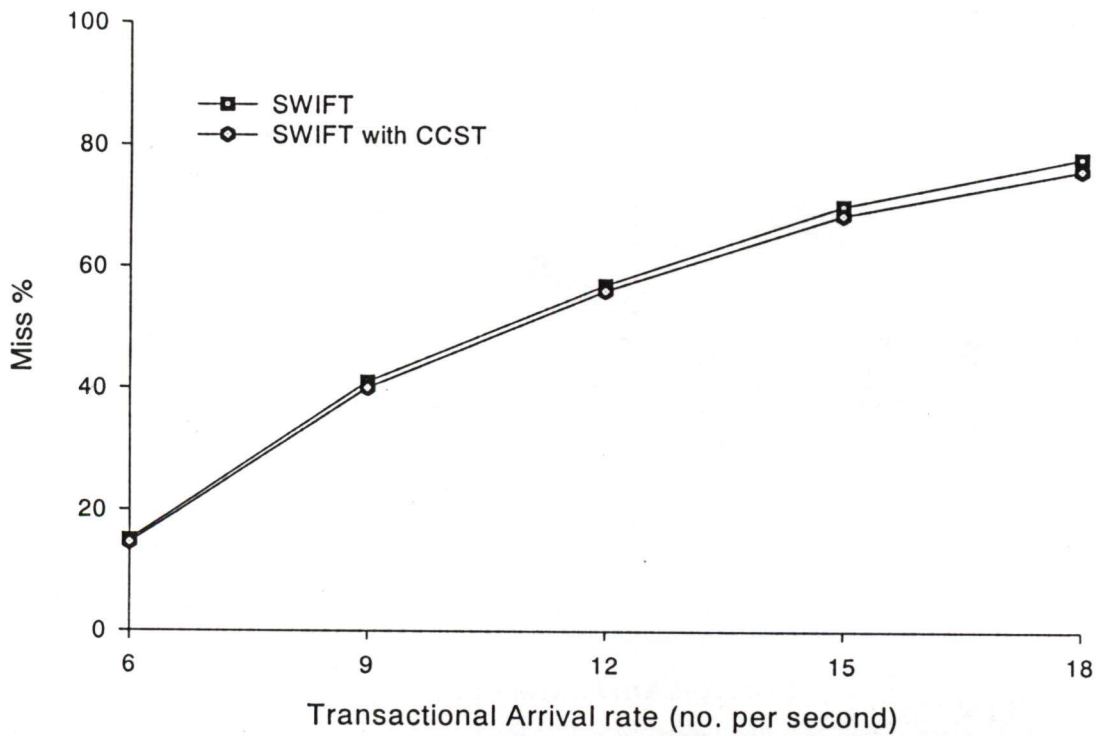


Fig. 4.25: Miss % with (RC+DC) at Communication Delay=0ms Heavy Load

4.9 IMPACT OF EARLY SENDING OF WORKSTARTED MESSAGE

There are several points and phases during the life time of a transaction, when it can be aborted. It is aborted because it misses its deadline or there is a contention with a newly arrived higher priority transaction. In this section, a study has been done to see the percentage of transactions that miss their deadline during processing phase in comparison to the total transaction miss percentage. The different cases considered for the study are given below.

4.9.1 Main Memory Database with Communication Delay of 100ms

The results given in Fig. 4.26 show that no transaction misses its deadline during processing phase except at transaction arrival rates of 8 and 10. Most of the transactions miss their deadline while waiting for the locks or YES REQ message.

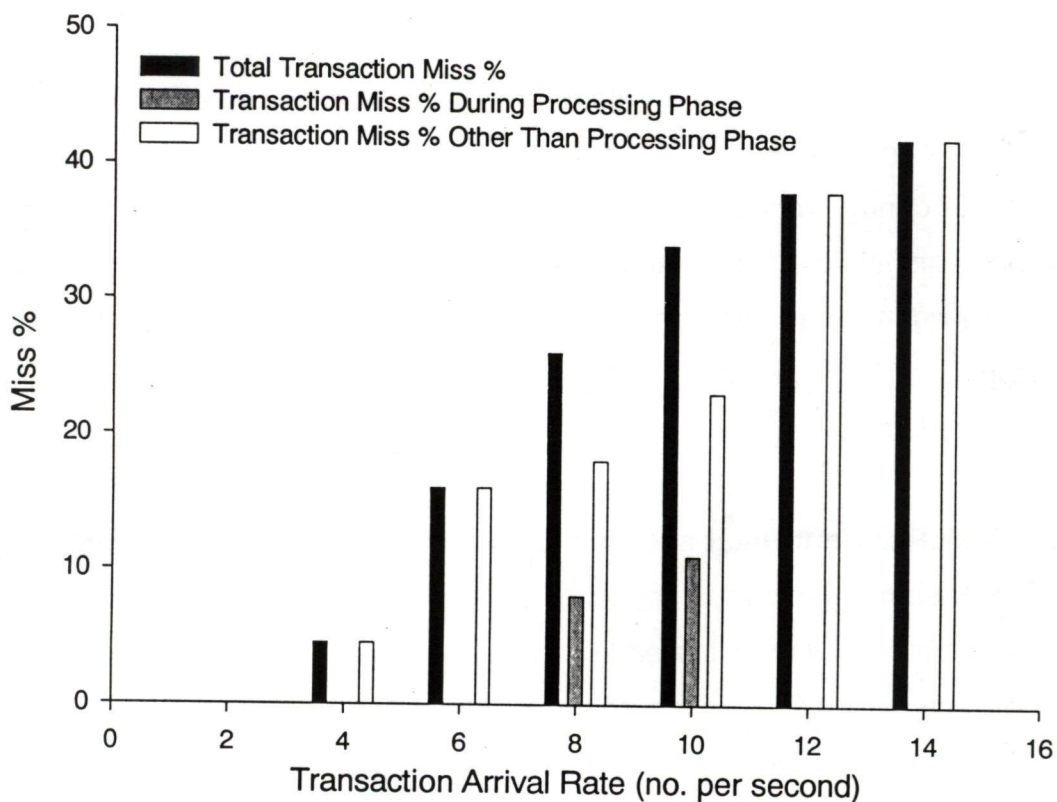


Fig. 4.26: Break-up of Miss % with (RC+DC) at Communication Delay=100

4.9.2 Main Memory Database with Communication Delay of 0ms

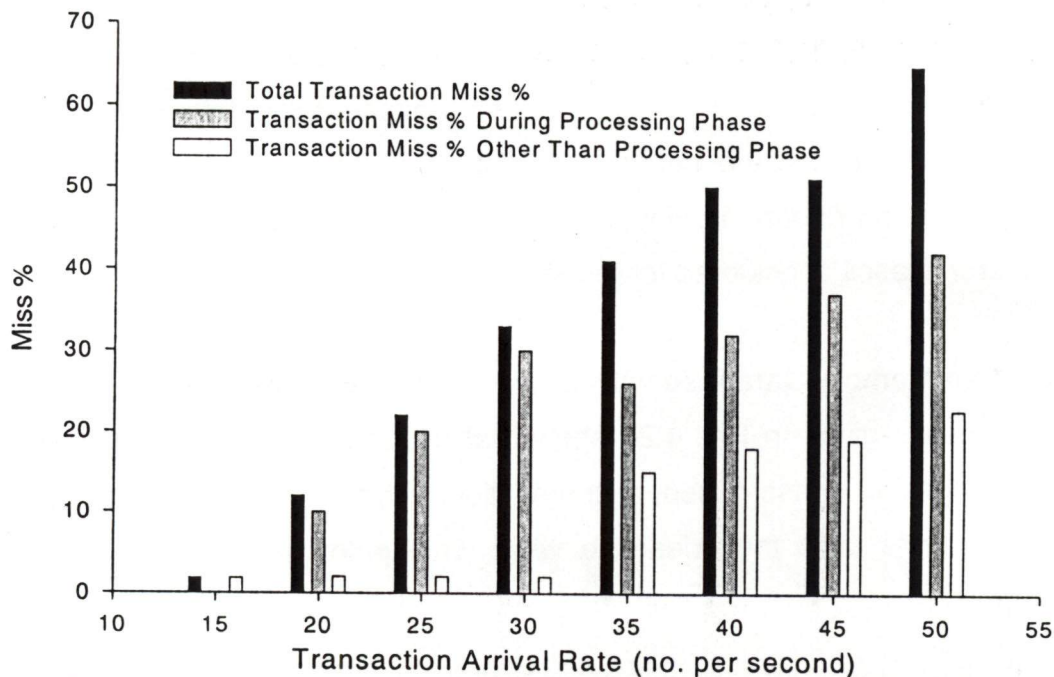


Fig. 4.27: Break-up of Miss % with (RC+DC) at Communication Delay=0ms

The results given in Fig. 4.27 show that, in this case, most of the transactions miss their deadline during the processing phase when transaction arrival rate is 15, 20, 25, and 30. For higher transaction arrival rates, the transaction miss percentage during processing phase is approximately $2/3^{\text{rd}}$ of the total transaction miss percentage. This demonstrates that sending of WORKSTARTED Message is not useful when communication delay is very small. This is due to the fact that the overlapping of communication delay and time taken in processing phase is not becoming effective.

4.9.3 Disk Resident Database with Communication Delay of 100ms

This case represents the impact of communication delay and disk processing time in a collective way on the early sending of WORKSTARTED Message. It can be seen from Fig. 4.28 that the share of transaction miss percentage during the processing phase is very variable in nature for different transaction arrival rates. It increases from zero to $2/3^{\text{rd}}$ and then decreases to $1/3^{\text{rd}}$ of the total transaction miss percentage as arrival rate increases. Here, the more disk accessing time is becoming the reason for poor performance.

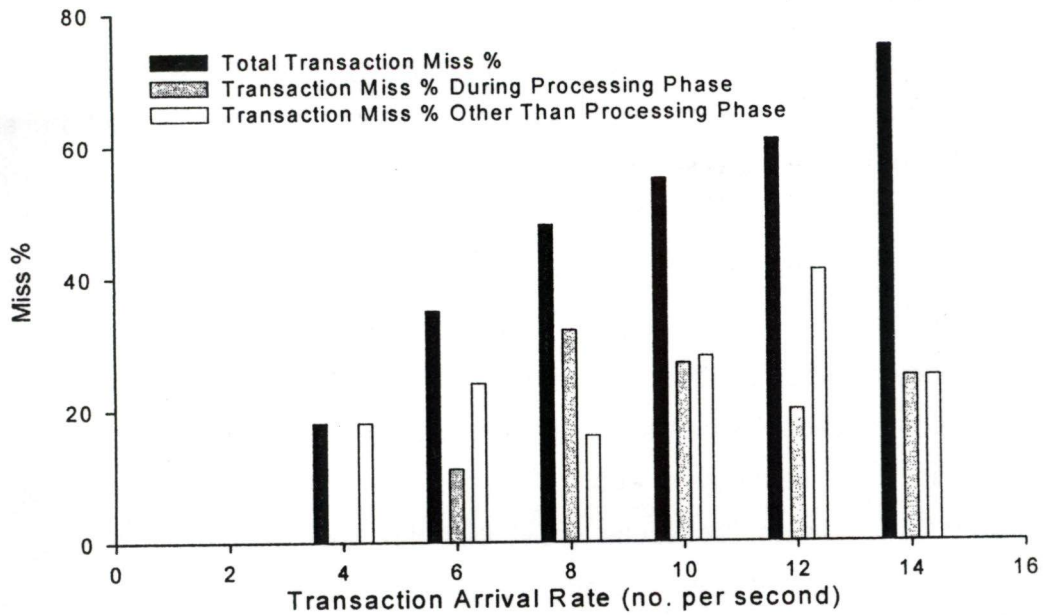


Fig. 4.28: Break-up of Miss % with (RC+DC) at Communication Delay=100

4.9.4 Disk Resident Database with Communication Delay of 0ms

In this case, the disk processing time plays an active role since the communication delay is zero. Except at transaction arrival rate 6, at higher arrival rates most of the transactions miss their deadlines during processing phase due to the disk accessing time (see Fig. 4.29).

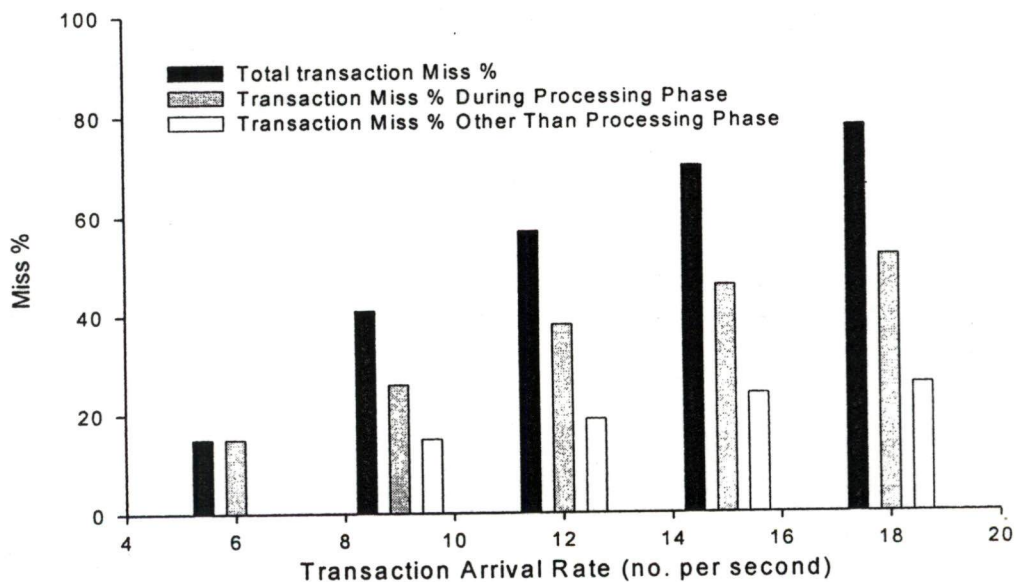


Fig. 4.29: Break-up of Miss % with (RC+DC) at Communication Delay=0ms

4.10 CONCLUSIONS

In a large network, communication and queuing delays become a bottleneck. In this chapter, we propose a commit protocol where the cohort send WORKSTARTED message just before the start of their processing in place of sending WORKDONE message. This overlap the message transmission time with the cohort's processing time and reduces the overall transaction's completion time. The borrower is also allowed to send WORKSTARTED message if the dependency between the borrower and its lenders is commit dependency only. It is free from cascaded aborts since borrower with only commit dependency is not aborted in case its lenders abort. This reduces the blocking period of the borrower. The simulation results show that the gain in performance can be achieved at low and moderate loads. A suitable modification in distributed real time commit protocol at the time of sending YES VOTE message has been made to ensure atomicity. The important point to note here is that the new protocol's features proposed in this chapter are local to each site and do not require inter-site communications. Moreover, the proposed optimization can be integrated with any other protocol based on 2PC.

MECP- A MEMORY EFFICIENT REAL TIME COMMIT PROTOCOL

5.1 INTRODUCTION

Important data base system resources are the data items that can be viewed as logical resource, and CPU, disks and the main memory which are physical resources [35,135]. Though the cost of main memory is dropping rapidly and its size is increasing, the size of database is also increasing very rapidly. In real time applications, where the databases are of limited size or are growing at a slower rate than memory capacities are growing, they can be kept in the main memory. However, there are many real time applications, which handle large amount of data and require support of an intensive transaction processing. The amount of data they store is too large (and too expensive) to be stored in the main memory. Examples include telephone switching, satellite image data, radar tracking, media servers, computer aided manufacturing etc. In these cases, the database can not be accommodated in the main memory easily. Hence, many of these types of database systems are disk resident. The buffer space in the main memory is used to store the execution code, copies of files, data pages and any temporary objects produced. The buffer manager controls the main memory and the availability of main memory space affects transaction's response time [35]. Before starting the execution of a transaction, buffer is allocated for the transaction. When the memory is running low, a transaction may be blocked from execution. The amount of memory available in the system thus limits the number of concurrently executable transactions [35]. In the large-scale real time database systems, the execution of the transaction will be significantly slowed down, if available main memory is low. When the total maximum memory requirement of the admitted transactions exceeds the available memory, DRTDBS must decide how much memory should be given to each transaction. This decision must also take into account the transactions' timing requirements to ensure that the transactions receive their required resources in time to meet their deadlines. In addition, the effectiveness of memory allocation in reducing individual transaction's response time should be considered, so as to make the best use of

the available memory. Therefore, it is important for the database designer to develop memory efficient protocols, so that more number of transactions can be executed concurrently at any time instant. In this chapter, design of a distributed commit protocol which optimizes memory usage has been presented.

The development of commit protocols for the traditional database systems has been an area of extensive research in the past decade. However, in case of distributed real time commit protocols, very little amount of the work has been reported in the literature. The real time commit protocol PROMPT and DDCR were proposed by Gupta et al. and Lam et al. respectively [59,78]. Based on the concepts of PROMPT and DDCR, Biao Qin and Yunsheng Liu proposed a new commit protocol double space commit (2SC) [108]. All the above protocols consume considerable amount of main memory for maintaining the intermediate temporary records created during the execution of transactions. In PROMPT, the lender maintains extra data structures to record the types of dependencies of its borrowers and DDCR uses more than one copy of the data items (i.e. before, after and further). All of this not only requires extra memory but also creates additional workload on the system. Furthermore, the locking scheme used by PROMPT and 2SC protocols specifies the lock held by the lender only and these protocols either use read-before-write model or write only (blind write) model. The effect of using both models collectively has not been investigated in any previous work. So, another significant difference between our work and the works reviewed above is that we have considered both blind write (a read is not performed before the data item is written) as well as update (read - before - write). A blind-write model is not unrealistic [19] fine and it occurs in real life information processing for example, recording and editing new telephone numbers, opening new accounts, changing addresses etc. There are also many applications such as banking, intelligent network services database etc. where we need write without ever read model.

The rest of the chapter is organized as follows. First, we describe distributed real time database system model and model assumptions, and then new locking scheme, data access conflict resolving strategies and mechanics of interaction between lender and borrower cohorts. Thereafter, the complete pseudo code,

memory optimization achieved and simulation results of the proposed protocol MECP is also discussed.

5.2 NEW LOCKING SCHEME AND DATA ACCESS CONFLICT RESOLVING STRATEGIES

A database is considered as being made up of a set of data items associated with lock variable. The proposed memory efficient commit protocol (MECP) uses a new locking scheme which reduces the need for large number of temporary intermediate records and thus relieves the system from additional workload.

5.2.1 New Locking Scheme

A transaction (or a cohort) can lock a data item using a read/write (blind)/update lock depending on the operation that it needs to perform. In MECP, the write operation is categorized into two types: blind write and update. Subsequent occurrences of "write" should be treated as "blind write" in this chapter.

In addition to lock information, a flag is also attached with each data item. The flag is set to any one of the following three modes when a cohort locks a data item at the time of its arrival at a site.

Mode 1

If a cohort wants to use a data item and it is not locked by any other cohort, it sets the flag of data item in Mode 1.

Mode 2

If a cohort T_2 wants to write/update a data item read by another cohort T_1 in its committed phase, it changes the flag of the data item from Mode 1 to Mode 2. T_2 is now not allowed to commit until T_1 commits. However, if T_1 aborts, T_2 does not abort.

Mode 3

If a cohort T_2 wants to read/write/update an uncommitted data item written by another cohort T_1 , it converts the flag of the data item from Mode 1 to Mode 3. Here, T_2 is not allowed to commit until T_1 commits and if T_1 aborts, T_2 also aborts.

These Modes are required to maintain the ACID properties of the transaction. If a data item is already locked and its Mode is either 2 or 3. Then, other lock requesting cohorts are not permitted to lock that data item and they will be blocked. Each site S_i maintains a Borrower_List that contains the following information.

Borrower_List (S_i) : $\{(T_j, D) \mid T_j \text{ is a borrower and has locked the dirty data } D\}$

5.2.2 Data access Conflicts Resolving Strategies

Let T_1 be a cohort in commit phase holding a lock on data item (x) and T_2 be the cohort requesting a lock on the same data item (x). The data item (x) is in mode 1 (as T_1 has locked it). Hence, there are six possible cases of data conflict.

Case 1: Read - Write OR Update Conflict

If cohort T_2 requests a Write OR Update Lock while cohort T_1 is holding a Read Lock, the flag associated with the data item is set in Mode 2 from Mode 1.

Case 2: Write - Write Conflict

If cohort T_2 requests a Write lock while cohort T_1 is holding the Write Lock, the flag associated with data item is set in Mode 3 from Mode 1.

Case3: Update – Update Conflict

If both locks are Update - Locks, then the flag associated with data, item is set in Mode 3 from Mode 1.

Case 4: Update - Write Conflict

If cohort T_2 requests a Write Lock while cohort T_1 is holding an Update Lock, flag associated with data item is set in Mode 3 from Mode 1.

Case 5: Write- Update Conflict

If cohort T_2 requests an Update Lock while cohort T_1 is holding Write Lock, flag associated with data item is set in Mode 3 from Mode 1.

Case 6: Write OR Update - Read Conflict

If cohort T_2 requests a Read Lock while cohort T_1 is holding a Write OR Update Lock, then flag associated with data item is set in Mode 3 from Mode 1.

5.2.3 Mechanics of Interaction between Lender and Borrower Cohorts

When transaction T_2 accesses a data already locked by transaction T_1 , following possible scenarios may arise.

Scenario 1: T_1 receives decision before T_2 has completed its local data processing:

- (i) If global decision is to commit, T_1 commits.
 - All cohorts using the data items locked by T_1 whose flag is either in Mode 2 or in Mode 3 will execute as usual.
 - The flags of the data items will change from Mode 2 or Mode 3 to Mode 1.

- (ii) If the global decision is to abort, T_1 aborts.
 - All cohorts using the data items locked by T_1 whose flag is in Mode 2 will execute as usual. Flags of the data items will change from Mode 2 to Mode 1.
 - All cohorts, using the data items locked by T_1 whose flag is in Mode 3 will abort. Flags of the data items will change from Mode 3 to 0.

The cohorts dependent on data set of T_1 will be deleted from the Borrower_List.

Scenario 2: T_2 completes data processing before, T_1 receives global decision:

- (i) T_2 does not send WORKDONE message.
- (ii) T_2 is blocked and It has to wait, until

Case1: either T_1 receives its global decisions, or
Case2: its own deadline expires,
whichever occurs earlier.

- (iii) In case 1, the system will execute as in the scenario 1, whereas in case 2, T_2 will be killed and will be removed from the Borrower_List.

Scenario 3: T_2 aborts before T_1 receives decision:

In this situation, T_2 's updates are undone and T_2 will be removed from the Borrower_List.

5.3 ALGORITHM

On the basis of the above discussion, the complete pseudo code of the protocol is given as below.

if (T_1 receives global decision before T_2 ends execution) then

{

One: if (T_1 's global decision is to commit) then

{

T_1 commits;

All cohorts using the data items in Mode 2 or 3 locked by T_1 will execute as usual;

Flags, either in Mode 2 or 3, of data items locked by T_1 are set to Mode 1.

The cohorts dependent on T_1 will be deleted from the Borrower_List;

}

else // T_1 's global decision is to abort

```

{
    T1 aborts;
    All cohorts using the data items with Mode 2 flag and locked by T1 will
    execute as usual. Flag will change from Mode 2 to Mode 1 of data items
    locked by T1;
    All cohorts using the data items with Mode 3 flag and locked by T1 will
    abort. Flag in Mode 3 of data items locked by T1 is set to 0.
    The cohorts dependent on data set of T1 will be deleted from the
    Borrower_List;
}
}
else
{
    if ( T2 ends execution before T1 receives global decision )
    {
        T2's WORKDONE message is blocked;
        T2 waits for next event/message;
        Switch (type of next event/message)
        {
            Case 1: if (T2 misses deadline)
            {
                Undo the computation of T2;
                Abort T2;
                Delete T2 from the Borrower_List;
            }
            Case 2: if (T1 commits/aborts)
                GoTo One;
        }
    }
}
else // T2 is aborted by higher transaction before T1 receives decision
{
    Undo the computation of T2;
}

```

```

    Abort T2;
    Delete T2 from the Borrower_List;
}
}

```

5.4 MEMORY OPTIMIZATION

It is assumed that the number of data items in the database at each site is N . The amount of memory required for maintaining the record of data items lent by a single cohort is computed and compared with 2SC below.

5.4.1 Case 1: Memory Requirement in 2SC

At least a flag is required corresponding to every data item to show its locking status. The minimum memory required to keep this information for all data items is $N/8$ bytes (a flag needs at least single bit storage). Further, each site maintains a list of lenders, and each lender maintains two lists: commit dependent cohorts and abort dependent cohorts with dirty data used by them. This can be implemented in two ways.

- (i) Using a linear list, or
- (ii) With a linked list

In the first case, there may be a lot of wastage of memory in maintaining the records of dependency information due to high level of dynamism in the conflicts.

In the second case, a dependency list has to be maintained which contains the identity (id) of committing cohorts (lenders) that have lent their modified data to newly arrived cohorts. Each lender in this dependency list also maintains two lists to record ids of abort and commit dependent cohorts with dirty data items utilized by them. The memory required for keeping the record of data items lent by a single cohort is computed below. Let us assume that, on an average, each lender has p nodes in commit dependency list and q nodes in abort dependency list. The total memory (M) required by each lender is given as:

$$M=M1+ (p+q)*M2, \text{ where}$$

- M1 Memory required for a node of the dependency list corresponding to the lending cohort. It is 14 bytes (2 bytes for id of the lender, 4 bytes for the address of commit dependent cohort list, 4 bytes for the address of abort dependent cohort list and 4 bytes for address of the next node)
- M2 Memory required by one node of the list of commit/abort dependent cohorts. It is 8 bytes. (8 bytes=2 bytes for borrower cohort id + 2 bytes for dirty data + 4 bytes for address of the next node)
- N_d No. of data items that is lent by a cohort= $p+q$

5.4.2 Case 2: Memory Requirement in Proposed Scheme (MECP)

Memory required by MECP is given below.

- Minimum two bits are needed to record the Modes of every data item at a site. So the total memory required for all data items is $2*N/8$ bytes.
- The proposed protocol maintains a single Borrower_List for keeping the information of borrowers and the dirty data used by them. This requires 8 bytes of memory (2 bytes for borrower id + 2 bytes for dirty data + 4 bytes for address of the next node) for one node of the Borrower_List.
- Total number of nodes in this Borrower_List corresponding to one lender will be N_d .

$$\text{Total Memory Requirement (per lender in MECP)} = 2*N/8 + 8*N_d.$$

Compared to case 1, there is an additional need of $N/8$ bytes at each site to keep the information about the Mode of every data item. On the other hand if there are N_L lenders at any instant of time, the additional memory required is $14*N_L$ bytes in case 1 as compared to case 2 (see in table 1). With the increase in the transaction arrival rate and transaction size, there are more chances of conflicts resulting in more number of dependencies and more lenders. Although, it

seems that initially more number of bytes are needed for keeping the Mode information of data items, the proposed protocol competes with 2SC at high transaction arrival rate.

Table 5.1: Memory Requirement of 2SC and MECP

Commit Protocol	flags at each site (Memory in bytes)	Single lender (Memory in bytes)
2SC	$N/8$	$14+8*N_d$
MECP	$2*N/8$	$8*N_d$

The MECP will be beneficial, if the following condition holds.

$$(2*N/8 + 8*N_d* N_L) < (N/8 + N_L*(14+8*N_d)), \text{ or}$$

$$N_L > N/112$$

Hence, the proposed protocol is memory efficient only, if the number of lenders at an instant is more than the number of data items/112 at a site; otherwise, it consumes the same amount of memory and will be at par with 2SC and PROMPT.

The MECP improves the system performance due to the following reasons.

- By reducing the memory requirements needed by the system as compared to 2SC. Thus, this protocol is suited to real time applications where there is a scarcity of the available memory [69].
- The new locking scheme ensures that a borrower can't be a lender simultaneously at the same site. This relieves the system from the burden of checking that a borrower is not trying to lend as compared with PROMPT and 2SC.
- The new locking scheme ensures that the same data can not be used by another borrower simultaneously as compared to PROMPT and 2SC, where there is a need for checking this.

- By not aborting a cohort (T_j) that has completed the execution and is in committing mode when a higher priority cohort requests for a data item locked by T_j .

5.5 MODEL PARAMETERS, SIMULATION RESULTS AND PERFORMANCE EVALUATION

The performance of the protocol is evaluated by developing detailed simulation models for a disk resident DRTDBS consisting of N sites. The structure of our simulation model followed by the description of various components such as system model, database model, network model, cohort execution model, locking mechanism etc. are the same as given in the section 3.2. The model assumptions are also same as in section the 3.2 except that the database is disk resident at all sites. The default values of different parameters used in the simulation experiments are same as given in table 3.1. The concurrency control scheme used is S2PL-HP. Miss Percentage is the primary performance measure used in the experiments. EDF [93] policy is used to assign priorities to cohorts in all the experiments.

In this section, the results and findings of our simulation experiments are reported. We compare the performance of MECP with PROMPT and 2SC.

5.5.1 Impact of Transaction Arrival Rate

Fig. 5.1 to Fig. 5.3 show the impact of transaction arrival rates on the miss percentage of transactions at transaction length 3-20 (uniform distribution). As anticipated, the Miss Percentage for the protocols increases with increase in transaction arrival rate. At higher arrival rate, the probability of lock conflicts for the data items and queueing delay for the use of system resources are more. The performance of the MECP is marginally better in comparison to 2SC and PROMPT due to the better approach used for resources utilization and minimizing the queueing delay.

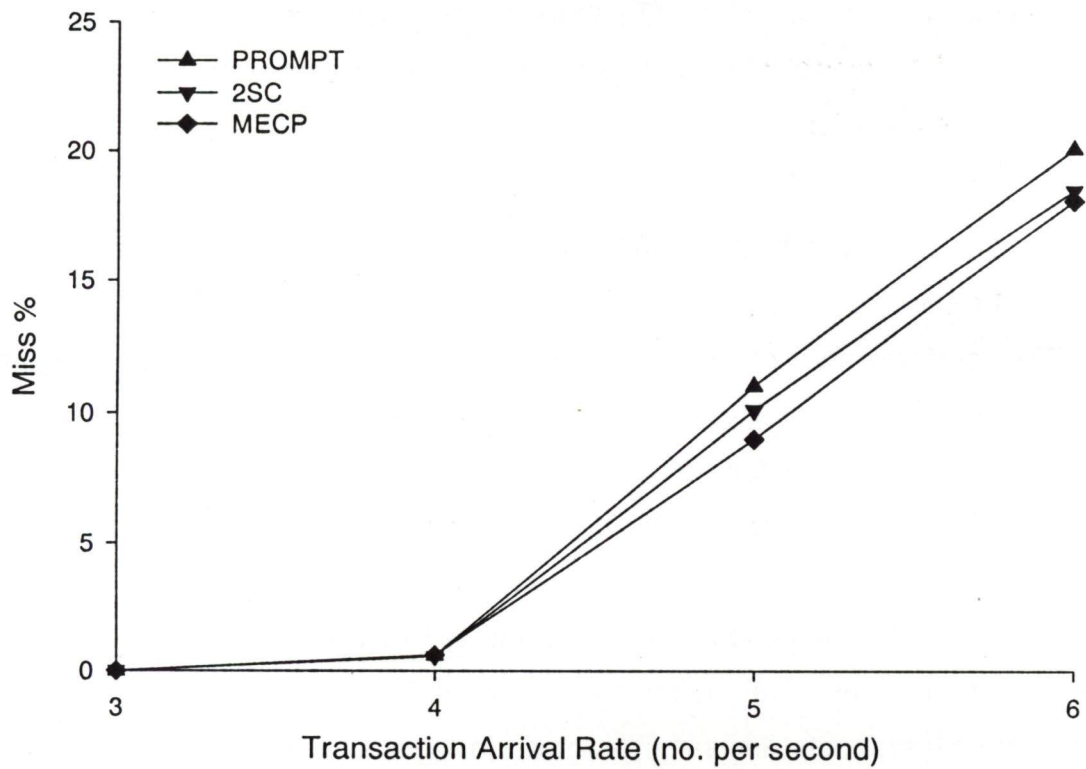


Fig. 5.1: Miss % with (RC+DC) at Communication Delay=0ms Normal Load

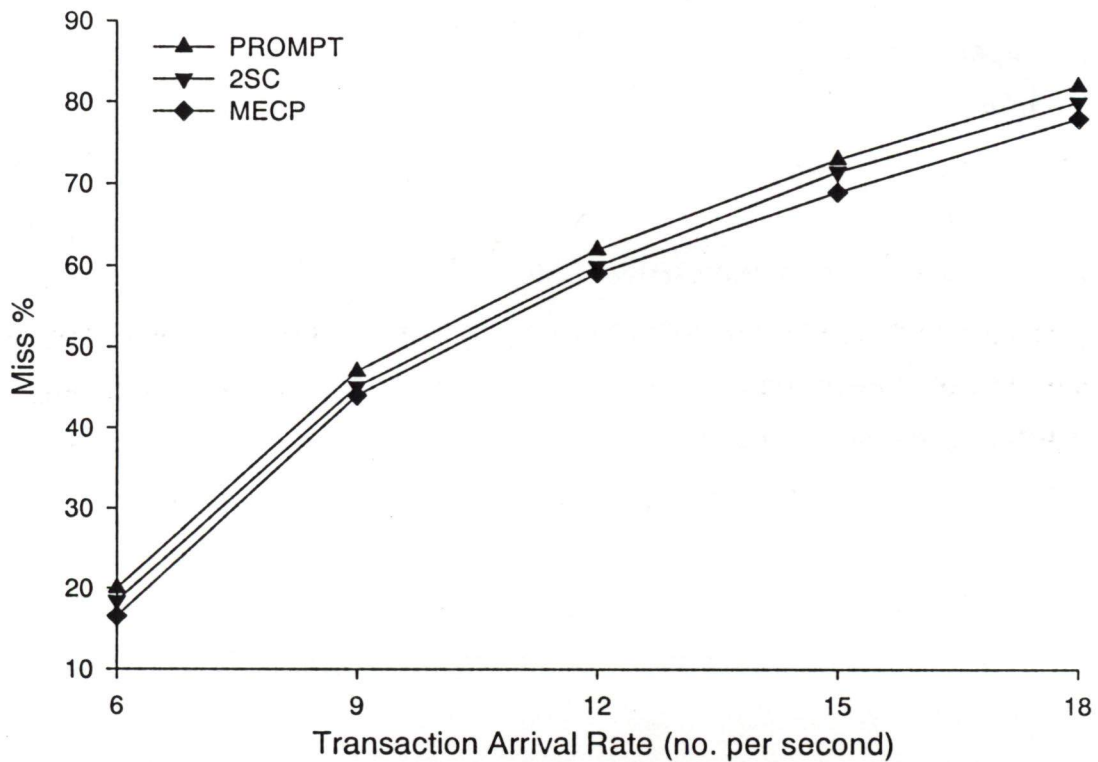


Fig. 5.2: Miss % with (RC+DC) at Communication Delay=0ms Heavy Load

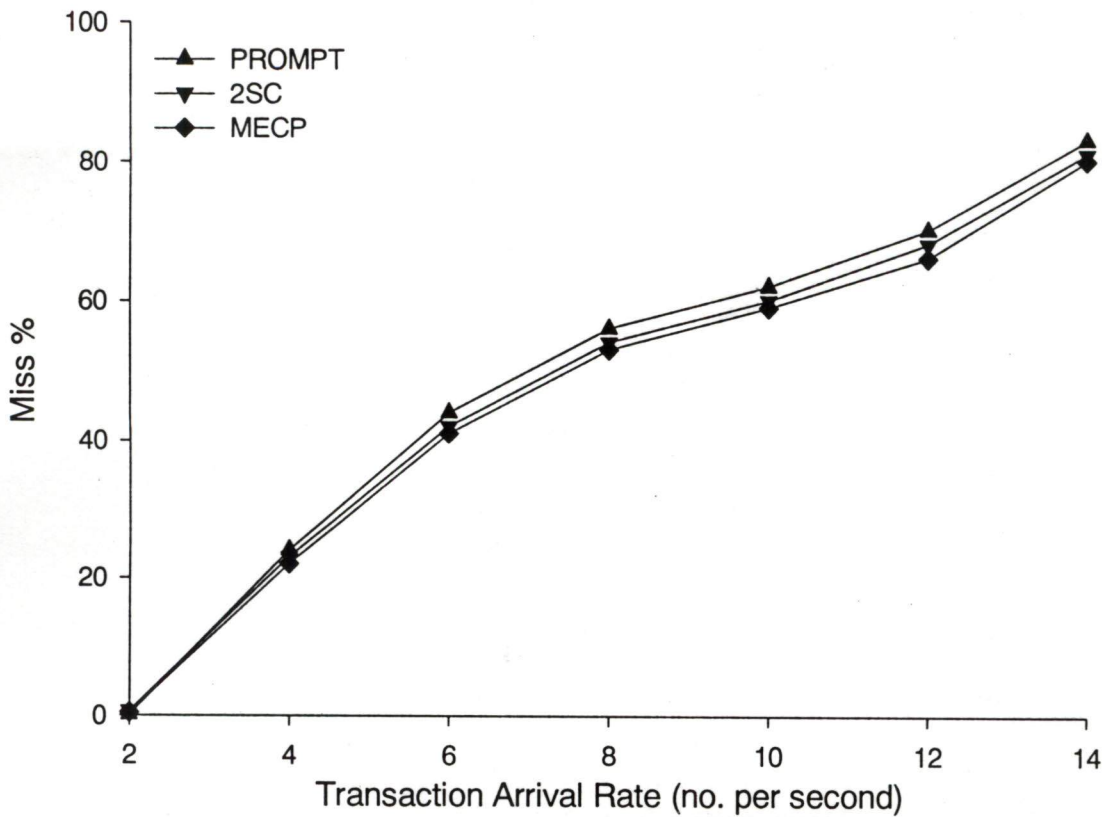


Fig.5.3: Miss % with (RC+DC) at communication Delay=100ms
Normal & heavy load

5.5.2 Impact of Transaction Size

Fig. 5.4 and Fig. 5.5 show the Miss Percentage for the protocols for different transaction size [116] at network communication delay of 100ms & 0ms respectively and transaction arrival rate of 10 transactions/second in both the cases. It is observed that increase in the size of transaction also increases the possibility of more conflicts. The performance of MECP is approximately at par with PROMPT and 2SC irrespective of transaction size due to better buffer management.

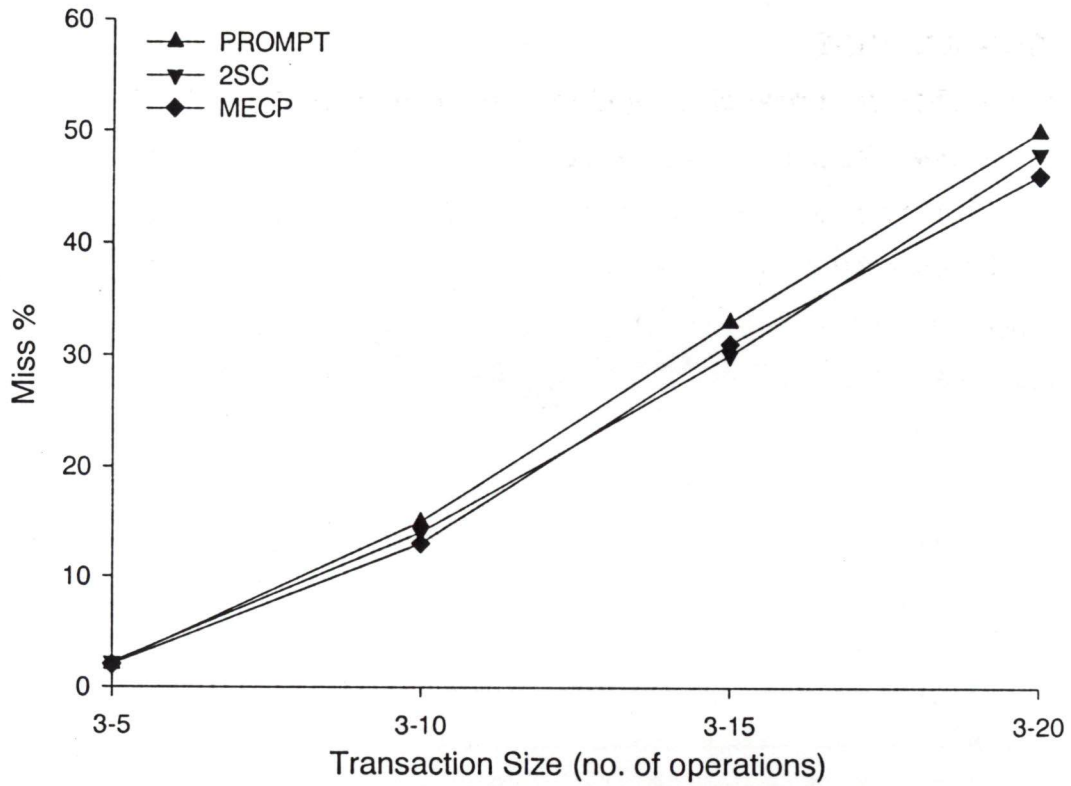


Fig. 5.4: Miss % with (RC+DC) at Communication Delay=0ms & Transaction Arrival Rate=10 (no. per second)

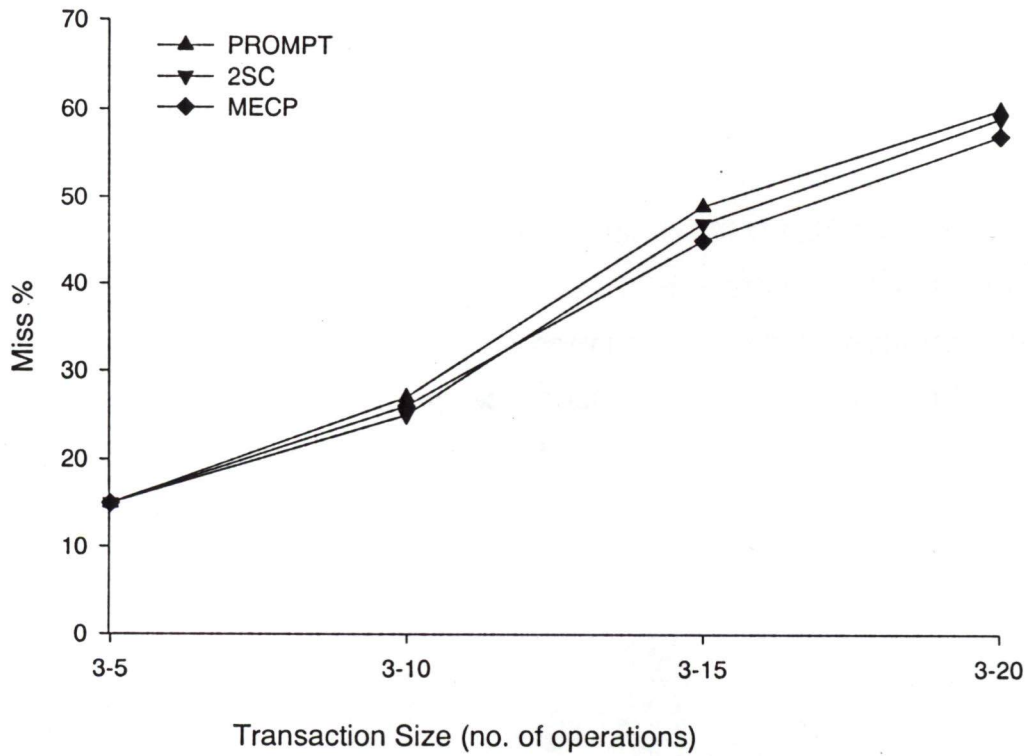


Fig. 5.5: Miss % with (RC+DC) at Communication Delay=100ms & Transaction Arrival Rate=10 (no. per second)

5.6 CONCLUSIONS

In this chapter, a new distributed real time commit protocol (MECP) has been presented that uses a new locking scheme. The new locking scheme ensures that a borrower can't be a lender simultaneously at the same site and the same data can not be used by another borrower simultaneously as compared to PROMPT and 2SC, where there is a need for checking this. It not only optimizes the storage cost but also considers blind and update type writes collectively for DRTDBS. The simulation results show that the protocol performs marginally better than PROMPT and 2SC commit protocols. It is well suited to data intensive applications where transaction arrival rate is high and the sizes of transactions are large.

CONCLUSIONS AND SCOPE FOR FUTURE RESEARCH

Database performance is an important aspect of database usability. The performance of databases depends not only on database architecture and algorithms, but also on the requests served within the given time limit, especially in case of DRTDBS. The database must be designed on all levels of database architecture to support timely execution of requests. The primary performance objective in DRTDBS is to minimize the number of missed deadlines. Due to the demanding nature of this objective, traditional approaches are inadequate for such systems. However, the research in DRTDBSs has been mostly devoted to extending traditional transaction processing techniques to solve the issues important for the design of DRTDBS. In this environment, new policy/protocols must be designed to efficiently handle the transactions execution.

Hence, in this thesis, we have taken some important issues such as priority assignment policy for transaction scheduling, commitment procedure for cohorts executing in the parallel and optimizing the use of memory to study real time transaction processing in a distributed environment.

The conclusions of the work carried out in this thesis and the scope for the future research are given as follows.

6.1 CONCLUSIONS

The main contributions reported in thesis are summarized below.

1. The transaction scheduling in DRTDBS involves both CPU scheduling and data scheduling and is done according to the priorities assigned to transactions. The performance of DRTDBS is heavily affected by the method used in assigning the priority to the cohorts. Hence, we proposed a new scheme to determine the priority of cohorts to schedule CPU and data. The new scheme consists of determining initial as well as temporary intermediate priorities of cohorts. The initial priority is used to schedule CPU and temporary intermediate priority is used to schedule data items, in case of conflict.

2. The proposed priority assignment scheme is capable to cope with the starvation problem encountered by long transactions. The improvement in transaction Miss Percentage is up to 10.
3. A method is also developed to compute the deadline of global and local transactions in parallel execution environment.
4. The proposed priority assignment schemes have been compared with EDF priority assignment policy using S2PL concurrency control algorithm and 2SC commit protocol. We have implemented distributed real time simulator for main memory resident database. The simulation results show that the proposed scheme not only ensures fairness within the real time constraints, but also reduces Miss Percentage of transactions ranging from 3% to 10%.
5. DRTDBS uses a commit protocol to ensure transaction atomicity. Most of the existing commit protocols used in DRTDBS try to improve the system performance by allowing a committing cohort to lend its data to a lock requesting cohort, thus reducing data inaccessibility. This creates a dependency between the lender and the borrower. Further, we have proposed a static two phase locking with higher priority (S2PL-HP) based distributed real time commit protocol named as SWIFT.
6. SWIFT is based on redefined dependencies that are created when a lock holding cohort lends its locked data to some other cohorts for reading or updating. The WORKSTARTED message is sent just before the start of processing phase of the cohort in place of sending WORKDONE message at the end of processing phase. This improves performance of the system by overlapping the transmission time of WORKSTARTED message with the processing time of the cohorts.
7. SWIFT also reduces the time needed for commit processing by allowing commit dependent only borrower to send its WORKSTARTED message instead of being blocked.

8. To ensure non-violation of the ACID properties, checking of completion of processing and the removal of dependency of cohort are done before sending the YES VOTE message in SWIFT. The important point of SWIFT is that the required modifications are local to each site and do not require inter-site communications so free from message overhead.
9. The performances of SWIFT has been compared with the 2SC and PROMPT for the scenario where communication delays are negligible and when they are large by simulating a distributed real time database system consisting of N sites both for the main memory resident and the disk resident databases. Results of simulation show a performance improvement of the order of 5% - 10% in transaction Miss Percentage.
10. The performance of SWIFT has also been analyzed for partial read-only optimization, which minimizes intersite message traffic, execute-commit conflicts and log writes consequently resulting in a better response time. The effect of partial read only optimization has been studied both for the main memory and the disk resident databases at communication delay of 0ms and 100ms. The performance improvement in transaction Miss Percentage varies from 1% to 5%.
11. The impact of permitting the communication between the cohorts of the same transaction (sibling) in SWIFT has also been analyzed both for the main memory and the disk resident database at communication delay of 0ms as well as 100 ms. The cohort sends the abort messages directly to its siblings as well as its coordinator. A little improvement in transaction miss percentage was observed, i.e., up to 3%.
12. At last, a memory efficient commit protocol (MECP) has been proposed after redefining all kind of dependencies that may arise by allowing a committing cohort to lend its data to an executing cohort under both update (read-before-write) model and blind write (write not ever read) model.

13. A new locking scheme has also been proposed for MECP, in which a lock not only shows the lock obtained by the lender but also the lock obtained by the borrower.
14. The new locking scheme also ensures that a borrower can't be a lender simultaneously at the same site. This relieves the system from the burden of checking that a borrower is not trying to lend as compared with PROMPT and 2SC.
15. As a result of the new locking scheme, in MECP, each site maintains only a single set of borrowers in comparison to PROMPT and 2SC, where two different sets are required.
16. The performance of MECP is compared with PROMPT and 2SC and is marginally better with these commit protocols in term of Miss Percentage of the transaction, but it reduces the memory requirement to a great extent. This makes it suitable for data intensive applications with high transaction arrival rate where system's main memory size is a bottleneck.

In nutshell, we have developed, implemented and evaluated the policies/protocols that have been devised to deal with aforementioned issues.

6.2 SCOPE FOR FUTURE RESEARCH

The work presented in this thesis is only a starting point. Many other issues are still to be resolved and warrant further investigation. Following are some suggestions to extend this work.

- (i) Alternative approaches such as analytical methods and experiments in actual environment can be used to evaluate the effects of the proposed priority assignment policies, deadline computation method and commit protocols on the performance of DRTDBS.
- (ii) Our performance studies are based on the assumption that there is no replication. The replicated database systems are of special interests to real time applications because they possess several desirable features:

enhanced reliability, improved responsiveness, no directory management, and load balancing. Hence, a study of relative performance of various topics discussed here deserves a further look under assumption of replicated data.

- (iii) The integration and the performance evaluation of proposed commit protocols with 1PC and 3PC protocols.
- (iv) Although tremendous research efforts have been reported in the hard real time systems in dealing with hard real time constraints, very little work has been reported in hard real time database systems. So, the performance of the proposed work can be evaluated for hard real time constrained transactions.
- (v) Our work can be extended for Mobile DTRDBS, where memory space, power and communication bandwidth is a bottleneck. The MECP will be well suited to hand held devices and possibility of its use for commit procedure can be explored.
- (vi) The fault tolerance and the reliability are highly desirable in many real time applications because in these applications, continued operation under catastrophic failure and quick recovery from failure is very crucial. These aspects may also be dealt.
- (vii) In our work, we assumed that each site has a system with a single processor. An obvious extension of our work is for multiprocessor environment.
- (viii) More work is needed to explore the impact of communication in between cohorts of the same transaction (siblings) on the overall system performance.
- (ix) Our research work can also be extended for grid database systems.

REFERENCES

1. Abbott Robert and Garcia - Molina H., "Scheduling Real - time Transactions with Disk Resident Data," Proceedings of the 15th International Conference on Very Large Databases, Amsterdam, The Netherlands, pp. 385 - 395, 1989.
2. Abbott Robert and Garcia - Monila H., "Scheduling Real - Time Transaction: a Performance Evaluation," Proceedings of the 14th International Conference on Very Large Databases, pp. 1 - 12, August 1988.
3. Abdallah Maha, Guerraoui R. and Pucheral P., "One - Phase Commit: Does It Make Sense," Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS'98), Tainan, Taiwan, Dec. 14 - 16, 1998.
4. Agrawal Divyakant, Abbadi A. El and Jeffers R., "Using Delayed Commitment in Locking Protocols for Real - Time Databases," Proceedings of the ACM International Conference on Management of Data (SIGMOD), San Diego, California, pp. 104 -113, June 2 - 5, 1992.
5. Agrawal Divyakant, Abbadi A. El, Jeffers R. and Lin L., "Ordered Shared Locks for Real - time Databases," International Journals of Very Large Data Bases (VLDB Journal), Vol. 4, Issue 1, pp. 87 - 126, January 1995.
6. Aldarmi Saud A. and Burns A., "Dynamic CPU Scheduling with Imprecise Knowledge of Computation Time," Technical Report YCS - 314, Department of Computer Science, University of York, U. K., 1999.
7. Aldarmi Saud A., "Real - Time Database Systems: Concepts and Design," Department of Computer Science, University of York, April 1998.
8. Al - Houmaily Yousef J. and Chrysanthis P. K., "Atomicity with Incompatible Presumptions," Proceedings of the 18th ACM Symposium on Principles of Database Systems (PODS), Philadelphia, June 1999.
9. Al - Houmaily Yousef J. and Chrysanthis P. K., "In Search for An Efficient Real - Time Atomic Commit Protocol,"
Url = citeseer.nj.nec.com/47189.html.
<http://www.cs.bu.edu/techreports/pdf/1996-027-ieee-rtss9.pdf/13.ps>
10. Al - Houmaily Yousef J., Chrysanthis P. K. and Levitan Steven P., "Enhancing the Performance of Presumed Commit Protocol," Proceedings of the ACM Symposium on Applied Computing, San Jose, CA, USA, February 28 - March 1, 1997.
11. Al - Houmaily Yousef J., Chrysanthis P. K. and Levitan Steven P., "An Argument in Favor of the Presumed Commit Protocol," Proceedings of the

- IEEE International Conference on Data Engineering, Birmingham, April 1997.
12. Arahna Rohan F. M., Ganti Venkatesh, Narayanan Srinivasa, Muthukrishnan C. R., Prasad S. T. S. and Ramamritham K., "Implementation of a Real - time Database System," *Information Systems*, Vol. 21 , Issue 1, pp. 55 - 74, March 1996.
 13. Attaluri Gopi K. and Salem Kenneth, "The Presumed - Either Two - Phase Commit Protocol," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 14, No. 5, pp. 1190 - 1196, Sept. - Oct. 2002.
 14. Audsley Neil C., Burns A., Richardson M. F. and Wellings A. J., "Data Consistency in Hard Real - Time Systems", YCS 203, Department of Computer Science, University of York, June 1993.
 15. Audsley Neil C., Burns A., Richardson M. F. and Wellings A. J., "Absolute and Relative Temporal Constraints in Hard Real Time Databases," *Proceedings of the 4th Euromicro Workshop on Real - time Systems*, IEEE Computer Society Press, Athens, pp. 148 – 153, June 1992.
 16. Bestavros Azer, "Advances in Real - Time Database Systems Research," *ACM SIGMOD Record*, Vol. 24, No. 1, pp. 3 - 8, 1996.
 17. Bestavros Azer, Lin K. J. and Son S. H., "Real - Time Database Systems: Issues and Applications," Kluwer Academic Publishers, 1997.
 18. Bowers David S., "Directions in Databases," *Lecture Notes in Computer Science*, 826, Springer - Verlag, pp. 23 - 54.
 19. Burger Albert, Kumar Vijay and Hines Mary Lou, "Performance of Multiversion and Distributed Two - Phase Locking Concurrency Control Mechanisms in Distributed Databases," *International Journal of Information Sciences*, Vol. 1 - 2, pp. 129 - 152, 1997.
 20. Carey Michael J., Jauhari R. and Livny M., "Priority in DBMS Resource Scheduling," *Proceedings of the of 15th VLDB Conference*, August 1989.
 21. Chakravarthy Sharma, Hong D. and Johnson T., "Real Time Transaction Scheduling: A Framework for Synthesizing Static and Dynamic Factors," TR - 008, CISE Dept., University of Florida, 1994.
 22. Chen Hong - Ren, Chin Y. H. and Tseng Shin - Mu, "Scheduling Value - Based Transactions in Distributed Real-time Database Systems," *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, pp. 978 - 984, April 23 - 27, 2001.
 23. Chen Yu - Wei, and Gruenwald Le, "Effects of Deadline Propagation on Scheduling Nested Transactions in Distributed Real - Time Database

- Systems," *Journal of Information Systems*, Vol. 21, No. 1, pp. 103 - 124, 1996.
24. Chrysanthis Panos K., Samaras G. and Al - Houmaily Y. J., "Recovery and Performance of Atomic Commit Processing in Distributed Database Systems," *Performance of Database Recovery Mechanism*, Editors: V. Kumar and M. Hsu, Prentice Hall, pp. 370 - 416, 1998.
 25. Datta Anindya, Mukhejee S., Konana F., Yiguler I. R. and Bajaj A., "Multiclass Transaction Scheduling and Overload Management in Firm Real - Time Database Systems," *Information Systems*, Vol. 21, No.1, pp. 29 - 54, 1996.
 26. Datta Anindya, Son S. H. and Kumar Vijay, "Is a Bird in the Hand Worth More Than Two in the Bush? Limitations of Priority Cognizance in Conflict Resolution for Firm Real - Time Database Systems," *IEEE Transactions on Computers*, Vol. 49, No. 5, pp. 482 - 502, May 2000.
 27. Davidson Susan B., Lee I. and Wolfe V., "A Protocol for Timed Atomic Commitment," *Proceedings of the IEEE 9th International Conference on Distributed Computing Systems*, pp. 199 - 206, June 5 - 9, 1989.
 28. Davidson Susan B., Lee I. and Wolfe V., "Timed Atomic Commitment," *IEEE Transactions on Computer*, Vol. 40, Issue 5, pp. 573 - 583, May 1991.
 29. DeWitt David J., Katz Randy H., Olken Frank, Shapiro Leonard D., Stonebraker Michael R. and Wood David, "Implementation Techniques for Main Memory Database Systems," *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, Boston, Massachusetts, pp. 1 - 8, 1984.
 30. DiPippo Lisa C. and Wolfe Victor F., "Real - Time Databases," *Database Systems Handbook*, Multiscience Press, 1997.
 31. Dogdu Eordgon & Ozsoyoglu Gultekin, "Real - Time Transactions with Execution Histories: Priority Assignment and Load Control," *Proceedings of the 6th International Conference on Information and Knowledge Management*, Las Vegas, Nevada, United States, pp. 301 - 308.
 32. Elmasri Ramez and Navathe Shamkant B., "Fundamentals of Database Systems," Addison - Wesley, 2000.
 33. Fortier Paul J. and Turner Patrick, "A Simulation Program for Analysis of Distributed Database Processing Concepts," *Proceedings of the 19th Annual Symposium on Simulation*, Tampa, Florida, United States, pp. 105 - 126, 1986.
 34. Garcia - Molina Hector and Lindsay Bruce, "Research Directions for Distributed Databases," *ACM SIGMOD Record*, Vol. 19, Issue 4, December 1990.

35. Garcia - Molina Hector and Salem K., "Main Memory Database Systems: An Overview," IEEE Transactions on Knowledge and Data Engineering, Vol. 4, Issue 6, pp. 509 - 516, Dec. 1992.
36. Gehani Narain, Ramamritham K., Shanmugasundaram J. and Shmueli O., "Accessing Extra Database Information: Concurrency Control and Correctness," Information Systems, Vol. 23, Issue 7, pp. 439 - 462, Nov. 1996.
37. George Binto, "A Secure Real - Time Transaction Processing," PhD Thesis, Supercomputer Education and Research Centre, I.I.Sc. Bangalore, India, Dec. 1998.
38. Ginis Roman and Wolfe Victor Fay, "Issues in Designing Open Distributed Real - Time Databases," Proceedings of the 4th IEEE International Workshop on Parallel and Distributed Real - Time Systems, Honolulu, HI, USA, April 15 - 16, pp. 106 - 109, 1996.
39. Gray Jim and Reuter A., "Transaction Processing: Concepts and Technique," Morgan Kaufman, San Mateo, CA, 1993.
40. Gray Jim, "Notes on Database Operating Systems," Operating Systems: an Advanced Course, Lecture Notes in Computer Science, Springer Verlag, Vol. 60, pp. 397 - 405, 1978.
41. Gray Jim, "The Transaction Concept, Virtues and Limitations," Proceedings of the 7th VLDB Conference, Cannes, France, pp. 144 -154, 1981.
42. Gupta Ramesh and Haritsa J. R., "Commit Processing in Distributed Real - Time Database Systems," Proceedings of the National Conference on Software for Real - Time Systems, Cochin, India, pp. 195 - 204, January 1996.
43. Gupta Ramesh, "Commit Processing in Distributed On - Line and Real - time Transaction Processing Systems," Master of Science (Engineering) Thesis, Supercomputer Education and Research Centre, I.I.Sc. Bangalore, India, 2000.
44. Gupta Ramesh, Haritsa J. R. and Ramamritham K., "More Optimism About Real - Time Distributed Commit Processing," Technical Report TR - 97 - 04, Database System Lab, Supercomputer Education and Research Centre, I.I.Sc. Bangalore, India, 1997.
45. Gupta Ramesh, Haritsa J. R. and Ramamritham K., "Revisiting Commit Processing in Distributed Database Systems," Proceedings of the ACM International Conference on Management of Data (SIGMOD), Tucson, May 1997.

46. Gupta Ramesh, Haritsa J. R., Ramamritham K. and Seshadri S., "Commit Processing in Distributed Real - Time Database Systems," Proceedings of Real - time Systems Symposium, Washington DC, IEEE Computer Society Press, San Francisco, Dec.1996.
47. Gupta Ramesh, Haritsa J. R., Ramamritham K. and Seshadri S., "Commit Processing in Distributed Real - Time Database Systems," Technical Report TR-96-01, Database System Lab, Supercomputer Education and Research Centre, I.I.Sc. Bangalore, India, 1996.
48. Hansen Gary W. and Hansen James V., "Database Management and Design," Prentice - hall of India, 2000.
49. Haritsa Jayant R. and Ramamritham K., "Adding PEP to Real - Time Distributed Commit Processing," Proceedings of the 21st IEEE Real - time Systems Symposiums, Orlando, USA, pp. 37 - 46, Nov. 27 - 30, 2000.
50. Haritsa Jayant R. and Ramamritham K., "Real - Time Database Systems in the New Millennium," Journals of Real Time Systems, Vol. 19, No. 3, pp. 1 - 5, September 2000.
51. Haritsa Jayant R., Carey M. J. and Livny M., "Data Access Scheduling in Firm Real - Time Database Systems," Journal of Real - Time Systems, Vol. 4, No. 3, pp. 203 - 242, 1992.
52. Haritsa Jayant R., Carey M. J. and Livny M., "Value - based Scheduling in Real - time Database Systems," Technical Report TR - 1204, CS Department, University of Wisconsin, Madison, 1991.
53. Haritsa Jayant R., Carey Michael J. and Linvy Miron, "Dynamic Real - Time Optimistic Concurrency Control," Proceedings of the 11th Real - Time Systems Symposium, Dec. 1990.
54. Haritsa Jayant R., Carey Michael J. and Linvy Miron, "On being Optimistic about Real - time Constraints", Proceedings of the ACM PODS Symposium, April 1990.
55. Haritsa Jayant R., and George B., "Secure Real-Time Transaction Processing," (book chapter) Real-time Database Systems: Architecture and Techniques, Kluwer Academic Publishers, vol. 593, Kluwer International Series in Engineering and Computer Science, eds. Tei-Wei Kuo and Kam-Yiu Lam, pp. 141-157, 2001.
56. Haritsa Jayant R., Livny M. and Carey M. J., "Earliest Deadline Scheduling for Real - Time Database Systems," Proceedings of 12th IEEE Real - Time Systems Symposium (RTSS), San Antonio, Texas, USA, pp. 232 - 242, December 1991.

57. Haritsa Jayant R., Ramamritham K. and Gupta R., "Characterization and Optimization of Commit Processing Performance in Distributed Database Systems," Technical Report, University of Massachusetts, March 1998.
58. Haritsa Jayant R., Ramamritham K. and Gupta R., "Real - Time Commit Processing," Real - time Database Systems: Architecture and Techniques, Kluwer Academic Publishers, Vol. 593, Kluwer International Series in Engineering and Computer Science, eds. Tei - Wei Kuo and Kam - Yiu Lam, pp. 227 - 243, 2001.
59. Haritsa Jayant R., Ramamritham K. and Gupta R., "The PROMPT Real - time Commit Protocol," IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 2, pp. 160 - 181, 2000.
60. Hong Dong - Kweon, "Real Time Transaction Scheduling: A Cost - Conscious Approach," M. Sc. Thesis, Graduate School of University of Florida, 1992.
61. Hong Dong - Kweon, Johnson Theodore and Chakravarthy Sharma, "Real - Time Transaction Scheduling: A Cost Conscious Approach," Proceedings of the SIGMOD Conference, pp. 197 - 206, 1993.
62. Huang Jiandong and Stankovic John A., "Real - Time Buffer Management," COINS TR 90 - 65, August 1990.
63. Huang Jiandong, "Real Time Transaction Processing: Design, Implementation and Performance Evaluation," PhD thesis, University of Massachusetts, May 1991.
64. Huang Jiandong, Stankovic John A., Ramamritham K. and Towsley D., "On Using Priority Inheritance in Real - time Databases," Proceedings of the 12th Real - Time Systems Symposium, pp. 210 - 221, Dec. 4 - 6, 1991.
65. Hung Sheung - Lun, Lam K. W. and Lam K. Y., "Efficient Technique for Performance Analysis of Locking Protocols," Proceedings of the 2nd IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Durham, NC, USA, pp. 276 - 283, Jan. 31 - Feb. 2, 1994.
66. Kao Ben and Garcia - Molina H., "Deadline Assignment in a Distributed Soft Real - Time System," Proceedings of the 13th International Conference on Distributed Computing Systems, pp. 428 - 437, 1993.
67. Kao Ben and Garcia - Molina H., "Subtask Deadline Assignment for Complex Distributed Soft Real - time Tasks," Technical Report 93 - 149, Stanford University, 1993.
68. Kao Ben and Garcia - Monila H., "An Overview of Real - time Database Systems," Advances in real - time systems, pp. 463 - 486, 1995.

69. Kayan Ersan and Ulusoy O., "An Evaluation of Real Time Transaction Management Issues in Mobile Database Systems," *Computer Journal*, Vol. 42, No. 6, 1999.
70. Kim Young - Kuk and Son S. H., "Predictability and Consistency in Real - Time Database Systems," Editor S. H. Son, *Advances in Real Time Systems*, Prentice Hall, New York, pp. 509 - 531, 1995.
71. Kim Young - Kuk, "Predictability and Consistency in Real Time Transaction Processing," PhD thesis, University of Virginia, May 1995.
72. Lam Kam - Yiu and Kuo Tei - Wei, "Real - Time Database Systems: Architecture and Techniques," Kluwer Academic Publishers, 2001.
73. Lam Kam - Yiu, "Concurrency Control in Distributed Real - Time Database Systems," PhD Thesis, City University of Hong Kong, Hong Kong, Oct. 1994.
74. Lam Kam - Yiu, Cao Jiannong, Pang Chung - Leung and Son S. H., "Resolving Conflicts with Committing Transactions in Distributed Real - time Databases," *Proceedings of the Third IEEE International Conference on Engineering of Complex Computer Systems*, Como, Italy, pp. 49 - 58, Sep. 8 - 12, 1997.
75. Lam Kam - Yiu, Hung S. L. and Son S. H., "On Using Real - Time Static Locking Protocols for Distributed Real - Time Databases," *Real - Time Systems*, Vol. 13, pp. 141 - 166, 1997.
76. Lam Kam - Yiu, Law Gary C. K. and Lee Victor C. S., "Priority and Deadline Assignment to Triggered Transactions in Distributed Real - time Active Databases," *Journal of Systems and Software*, Vol. 51, No.1, pp. 49 - 60, April 2000.
77. Lam Kam - Yiu, Lee Victor C. S., Kao Ben and Hung S. L., "Priority Assignment in Distributed Real - time Databases Using Optimistic Concurrency Control," *IEE Proceedings on Computer and Digital Techniques*, Vol. 144, No. 5, pp. 324 - 330, Sept. 1997.
78. Lam Kam - Yiu, Pang C., Son S. H. and Cao J., "Resolving Executing - Committing Conflicts in Distributed Real - time Database Systems," *Journal of Computer*, Vol. 42, No. 8, pp. 674 - 692, 1999.
79. Lam Kwok - wa, Lee Victor C. S. and Hung S. L., "Transaction Scheduling in Distributed Real - Time Systems," *International Journal of Time - Critical Computing Systems*, 19, pp. 169 - 193, 2000.
80. Lampson Butler and Lomet D., "A New Presumed Commit Optimization for Two Phase Commit," *Proceedings of the 19th International Conference on Very Large Data Bases*, Dublin, Ireland, 1993.

81. Lee Inseon and Yeom Heon Y., "A Fast Commit Protocol for Distributed Main Memory Database Systems," Proceedings of the 16th International Conference on Information Networking (ICOIN-16), Cheju, Korea, January 2002.
82. Lee Inseon and Yeom Heon Y., "A Single Phase Distributed Commit Protocol for Main Memory Database Systems," 16th International Parallel & Distributed Processing Symposium (IPDPS 2002), Ft. Lauderdale, Florida, USA, April 15 -19, 2002.
83. Lee Inseon, Heon Y and Park Taesoon, "A New Approach for Distributed Main Memory Database Systems: A Causal Commit Protocol," IEICE Transactions on Information and System, Vol. E87-D, No. 1, pp. 196-204, January 2004.
84. Lee Juhnyoung, "Concurrency Control Algorithms for Real - time Database Systems," PhD Thesis, Department of Computer Science, University of Virginia, 1994.
85. Lee Victor C. S., Lam K. Y., Kao Benjamin C. M., Lam K. W. and Hung S. L., "Priority Assignment for Sub - Transaction in Distributed Real - time Databases," 1st International Workshop on Real - Time Database Systems, 1996.
86. Lee Victor C. S., Lam Kam - Yiu and Kao B., "Priority Scheduling of Transactions in Distributed Real - Time Databases," International Journal of Time-Critical Computing Systems, Vol. 16, pp. 31 - 62, 1999.
87. Lee Victor C. S., Lam Kwok - wa and Hung S. L., "Concurrency Control for Mixed Transactions in Real - Time Databases," IEEE Transactions on Computers, Vol. 51, No. 7, pp. 821 - 834, July 2002.
88. Lin Kwei - Jay and Son S. H., "Real - time Databases: Characteristics and Issues," Proceedings of the 1st IEEE Workshop on Object - Oriented Real - Time Dependable Systems, Dana Point, CA, pp. 113 - 116, Oct. 1994.
89. Lindsay Bruce G., Haas Laura M., Mohan C., Wilms Paul F. and Yost Robert A., "Computation and Communication in R*: A Distributed Database Manager," in ACM Transaction on Computer Systems (TOCS), Vol. 2, No. 1, pp. 24 - 38, Feb.1984.
90. Lindström Jan and Raatikainen Kimmo, "Using Importance of Transactions and Optimistic Concurrency Control in Firm Real - Time Databases," Proceedings of the 7th International Conference on Real - Time Computing Systems and Applications (RTCSA'2000), Cheju Island, South Korea, December 12 - 14, 2000.
91. Lindstrom Jan, "Optimistic Concurrency Control Method for Distributed Real Time Database Systems," PhD Thesis, Report A-2003-I, Helsinki University, January 2003.

92. Lindström Jan, "Using Priorities in Concurrency Control for RTDBS," Seminar on Real - Time and Embedded Systems, Department of Computer Science, University of Helsinki, autumn 1999.
93. Liu C. L. and Layland J. W., "Scheduling Algorithms for Multiprogramming in a Hard Real - time Environment," *Journals of the ACM*, Vol. 20, No. 1, pp. 46 - 61, Jan. 1973.
94. Misikangas Pauli, "2PL and Its Variants," Seminar on Real - Time Systems, Department of Computer Science, University of Helsinki, 1997.
95. Mittal Abha and Dandamudi Sivarama P., "Dynamic versus Static Locking in Real - Time Parallel Database Systems," *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, New Mexico, April 26 - 30, 2004.
96. Mohan Chandrasekaran and Dievendorff D., "Recent Work on Distributed Commit Protocols, and Recoverable Messaging and Queuing," *Data Engineering Bulletin*, Vol. 17, No. 1, 1994.
97. Mohan Chandrasekaran, "An Overview of Recent Data Base Research," *Journals of Database*, Volume 10, No. 2, pp. 3 - 24, 1978.
98. Mohan Chandrasekaran, Lindsay B. and Obermarck R., "Transaction Management in the R* Distributed Database Management Systems," *ACM Transactions on Database Systems*, Vol. 11, No. 4, pp. 378 - 396, December 1986.
99. Mohania Mukesh K., Kambayashi Yahiko, Tjoa A. Min, Wagner Roland and Bellatreche Ladjel, "Trends in Database Research," *DEXA*, pp. 984 - 988, 2001.
100. NG Pui, "A Commit Protocol for Checkpointing Transactions," *Proceedings of the 7th Symposium on Reliable Distributed Systems*, Columbus, OH, USA, pp. 22 - 31, Oct. 10 - 12, 1998.
101. O'Neil Patrick, Ramamritham K. and Pu C., "Towards Predictable Transaction Executions in Real - Time Database Systems," *Technical Report 92 - 35*, University of Massachusetts, August, 1992.
102. Ozsoyoglu Gultekin and Snodgrass Richard T., "Temporal and Real - Time Databases: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No. 4, pp. 513 - 532, August 1995.
103. Pang Chung - Leung and Lam K. Y., "On Using Similarity for Resolving Conflicts at Commit in Mixed Distributed Real - time Databases," *Proceedings of the 5th International Conference on Real - Time Computing Systems and Applications*, Oct. 27 - 29, 1998.

104. Pang HweeHwa, "Query Processing in Firm Real - Time Database Systems," PhD Thesis, University of Wisconsin, Madison, 1994.
105. Pang HweeHwa, Carey Michael J., and Livny Miron, "Multiclass Query Scheduling in Real - Time Database Systems," IEEE Transactions on Knowledge and Data Engineering, Vol. 7, No. 4, pp. 533 - 551, August 1995.
106. Park Taesoon and Yoem H. Y., "A Distributed Group Commit Protocol for Distributed Database Systems," Seoul National University, Korea.
107. Purimetla Bhaskar, Rajendran M., Sivasankaran K., Ramamritham K. and Stankovic John A., "Real Time Databases: Issues and Applications," Advances in Real Time Systems, Prentice Hall, 1996.
108. Qin Biao and Liu Y., "High Performance Distributed Real - time Commit Protocol," Journal of Systems and Software, Elsevier Science Inc., Vol. 68, Issue 2, November 15, pp. 145 -152, 2003.
109. Ramakrishnan Raghu and Gehrke Johannes, "Database Management System," McGraw Hill Publication, January 2003.
110. Ramakrishnan Raghu and Ullaman Jaffrey D., "A Survey of Research on Deductive Database Systems,"
www-db.stanford.edu/~ullman/dscb/ch1.pdf
111. Ramamritham Krithi and Chrysanthis P. K., "A Taxonomy of Correctness Criteria in Database Applications," Journal of the VLDB, 5, pp. 85 - 97, 1996.
112. Ramamritham Krithi and Chrysanthis P. K., "In Search for the Acceptability Criteria: Database Consistency Requirements and Transaction Correctness Properties," Proceedings of the International Workshop on Distributed Object Management, Canada, pp. 211 - 229, August 1992.
113. Ramamritham Krithi and Sen Rajkumar, "DELite: database support for embedded lightweight devices," EMSOFT, Pisa, Italy, pp. 3 - 4, Sep. 27 - 29, 2004.
114. Ramamritham Krithi, "Real - time Databases," Distributed and Parallel Databases, Special Issue: Research Topics in Distributed and Parallel Databases, Vol. 1, Issue 2, pp. 199 - 226, April 1993.
115. Ramsay S., Nummenmaa J., Thanisch P., Pooley R. J. and Gilmore S. T., "Interactive Simulation of Distributed Transaction Processing Commit Protocols," Department of Computer Science, University of Edinburgh. In P. Luker (ed.): Proceedings of Third Conference of the United Kingdom Simulation Society (UKSIM'97), pp. 112 - 127, 1997.

116. Ryu In Kyung and Thomasian A., "Analysis of Database Performance with Dynamic Locking," *Journals of the ACM*, Vol. 37, No. 3, pp. 491 - 523, July 1990.
117. Samaras George, Britton K., Citron A. and Mohan C., "Two - phase Commit Optimizations and Tradeoffs in the Commercial Environment," *Proceedings of the 9th International Conference on Data Engineering*, pp. 520 - 529, April 19 - 23, 1993.
118. Seshadri Thomas S. and Haritsa J. R., "Integrating Standard Transactions in Real - Time Database Systems," *Information Systems*, Vol. 21, No. 1, pp. 3 - 28, March 1996.
119. Sha Lui, Rajkumar R., Son S. H. and Chang C. H., "A Real - time Locking Protocol," *IEEE Transactions on Computers*, Vol. 40, No. 7, pp. 793 - 800, 1991.
120. Shih Stuart, Kim Young - Kuk and Son S. H., "Performance Evaluation of a Firm Real Time Database System," *Proceedings of the 2nd International Workshop on Real - Time Computing Systems and Applications*, pp. 116 - 124, Oct. 25 - 27, 1995.
121. Shioh_Chen and Li Victor O. K., "Performance Analysis of Static Locking in Distributed Database Systems," *IEEE Transactions on Computers*, Vol. 39, No. 6, pp. 741 - 751, June 1990.
122. Shu Lih Chyun, Young Michal and Rajkumar R., "An Abort Ceiling Protocol for Controlling Priority Inversion," *Software Engineering Institute, Carnegie Mellon University*, 1994.
123. Silberschatz Abraham, Stonebraker M. and Ullman J., "Database Research: Achievements and Opportunities into the 21st Century," *ACM SIGMOD Record*, Vol. 25, No. 1, 1996.
124. Son Sang H., "Real - time Database Systems: Present and Future," *Proceedings of the Second International Workshop on Real - Time Computing Systems and Applications, Tokyo, Japan*, pp. 50 - 52, Oct. 25 - 27, 1995.
125. Soparkar Nandit, Levy E., Korth H. F. and Silberschatz A., "Adaptive Commitment for Real - Time Distributed Transaction," *Technical Report TR - 92 - 15, 1992, Dept. of Computer Science, University of Texas, Austin* and also in the *Proceedings of the 3rd International Conference on Information and Knowledge Management, Gaithersburg, Maryland, United States*, pp. 187 - 194, 1994.
126. Srinivasa Rashmi, "Network - Aided Concurrency Control in Distributed Databases," *PhD thesis, University of Virginia*, Jan. 2002.

127. Stamos James W. and Cristian Flaviu, "A Low - Cost Atomic Commit Protocol," Proceedings of the 9th IEEE Symposium on Reliable Distributed Systems, Huntsville, Al, USA, pp. 66 - 75, Oct. 9 - 12, 1990.
128. Stankovic John A., "Misconception about Real - Time Computing," IEEE Computer, pp. 10 - 19, Oct. 1988.
129. Stankovic John A., Ramamritham K. and Towsley D., "Scheduling in Real - Time Transaction Systems," Foundations of Real - Time Computing: Scheduling and Resource Management, edited by Andre van Tilborg and Gary Koob, Kluwer Academic Publishers, pp. 157 - 184, 1991.
130. Stankovic John A., Son S. H. and Hansson J., "Misconception about Real - Time Database," IEEE Computer, Vol. 32, No. 6, pp. 29 - 36, June 1999.
131. Taina Juha and Son S. H., "Towards a General Real - Time Database Simulator Software Library," Proceedings of Active and Real - Time Database Systems, 1999.
132. Takkar Sonia & Dandamudi Sivarama P., "An Adaptive Scheduling Policy for Real - Time Parallel Database System," www.mpcs.org/MPCS98/Final_Papers/Paper.48.pdf
133. Tay Yong Chiang and Suri R., "Choice and Performance in Locking for Databases," Proceedings of the 10th International Conference on Very Large Databases, August 1984.
134. Tay Yong Chiang, "Some Performance Issues for Transactions with Firm Deadlines," Proceedings of Real - Time Systems Symposium, Pisa, Italy, pp. 322 - 331, Dec. 1995.
135. Tay Yong Chiang, Goodman N. and Suri R., "Locking Performance in Centralized Database," ACM Transactions on Database Systems, Vol. 10, No. 4, pp. 415 - 462, 1985.
136. Tay Yong Chiang, Suri R. and Goodman N., "A Mean Value Performance Model for Locking in Databases: the No - Waiting Case," Journal of the ACM (JACM), Vol. 32, Issue 3, pp. 618 - 651, July 1985.
137. Thomasian Alexander and Ryu I. K., "Performance Analysis of Two - phase Locking," IEEE Transactions on Software Engineering, Vol. 17, No. 5, pp. 386 - 402, 1991.
138. Thomasian Alexander, "Concurrency Control: Methods, Performance, and Analysis," ACM Computing Surveys (CSUR), Vol. 30, Issue 1, pp. 70 - 119, March 1998.
139. Thomasian Alexander, "Two - phase Locking Performance and Its Thrashing Behavior," ACM Transactions on Database Systems, Vol. 18, No. 4, pp. 579 - 625, 1993.

140. Ullman Jeffrey D., "Principle of Database Systems," Galgotia Publication Pvt. Ltd. 1992.
141. Ulusoy Ozgur and Belford G., "Real - Time Transaction Scheduling in Database Systems," Information Systems, Vol. 18, No. 8, pp. 559 - 580, Dec.1993.
142. Ulusoy Ozgur and Belford Geneva G., "A Simulation Model for Distributed Real - time Database Systems," Proceedings of the 25th Annual Symposium on Simulation, Orlando, Florida, United States, pp. 232 - 240, 1992.
143. Ulusoy Ozgur and Buchmann A., "A Real - Time Concurrency Control Protocol for Main - Memory Database Systems," Information Systems, Vol. 23, No. 2, pp. 109 - 125, 1998.
144. Ulusoy Ozgur, "A Study of Two Transaction Processing Architectures for Distributed Real - Time Database Systems," Journal of Systems and Software, Vol. 31, No. 2, pp. 97 - 108, Nov. 1995.
145. Ulusoy Ozgur, "Analysis of Concurrency Control Protocols for Real Time Database Systems," Information Sciences, Vol.111, No.1 - 4, 1998.
146. Ulusoy Ozgur, "Concurrency Control in Real - time Database Systems," PhD Thesis, Department of Computer Science, University of Illinois, Urbana-Champaign, 1992.
147. Ulusoy Ozgur, "Processing Real - Time Transactions in a Replicated Database System," Distributed and Parallel Databases, Vol. 2, No. 4, pp. 405 - 436, October 1994.
148. Ulusoy Ozgur, "Research Issues in Real - time Database Systems," Information Sciences, Volume 87, Issues 1 - 3, pp. 123 - 151, November 1995.
149. Vrbsky Susan V. and Tomic Sasa, "Satisfying Timing Constraints of Real Time Databases," Journal of Systems and Software, Vol. 41, pp. 63 - 73, 1998.
www.mpcs.org/MPCS98/Final_Papers/Paper.48.pdf
150. Yoon Yongik, Han Mikyung and Cho Juhyun, "Real - time Commit Protocol for Distributed Real - time Database Systems," Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems, pp. 221 - 225, Oct. 21 - 25, 1996.
151. Yu Philip S., Wu Kun - Lung, Lin Kwei - Jay and Son S. H., "On Real - Time Databases: Concurrency Control and Scheduling," Proceedings of the IEEE, Volume 82, No.1, pp. 140 - 157, Jan. 1994.

AUTHOR'S RESEARCH PUBLICATION

- (1) "Distributed Real Time Database Systems: Background and Literature Review" International Journal of Distributed and Parallel Databases, Springer Verlag (communicated).
- (2) "A Fast Distributed Real Time Commit Protocol" Journal of Computer Science & Informatics, Computer Society of India (Accepted)
- (3) "SWIFT- A New Real Time Commit Protocol", International Journal of Distributed and Parallel Databases, Springer Verlag (online on May 26, 2006).
- (4) "The MEWS Distributed Real Time Commit Protocol" WSEAS International Transactions on COMPUTERS, Issue 7, Volume 4, pp. 777-786, July 2005.
- (5) "Priority Assignment Heuristic and Issue of Fairness to Cohorts Executing in Parallel", WSEAS International Transactions on COMPUTERS, Issue 7, Volume 4, pp. 758-768, July 2005.
- (6) "Some Performance Issues in Distributed Real Time Database Systems" Proceedings of the VLDB PhD Workshop, The Convention and Exhibition Center (COEX), Seoul, Korea, Sept. 11, 2006 (communicated).
- (7) "Fair Scheduling Policy for Real Time Cohorts Executing in Parallel" Proceedings of the 4th International Conference on Computer Application (ICCA2006), University of Computer Studies, Yangon, Myanmar, February 23-24, 2006.
- (8) "A Research Perspective for Distributed Real Time Database System" Proceedings of National Seminar on Electrical Power Technology, Management and IT Applications (EPTMITA-06), Department of Electrical Engineering, M. M. M. Engineering College Gorakhpur, India, Feb. 17-18, 2006.
- (9) "OCP- the Optimistic Commit Protocol", Proceedings of the 17th Australasian Database Conference (ADC 2006), Hobart, Tasmania, Australia, Jan. 16-19, 2006.
- (10) "A Memory Efficient Fast Distributed Real Time Commit Protocol", Proceedings of the 7th International Workshop on Distributed Computing (IWDC 2005), Indian Institute of Technology Kharagpur, India, pp. 500-505, Dec. 27-30, 2005.
- (11) "Dependency Sensitive Distributed Commit Protocol", Proceedings of the 8th International Conference on Information Technology (CIT 05), Bhubaneswar, India, Dec. 20 – 23, 2005.

- (12) "Priority Assignment Heuristic to Cohorts Executing in Parallel", Proceedings of the 9th WSEAS International Conference on COMPUTERS Vouliagmeni, Athens, Greece, July 14-16, 2005.
- (13) "Memory Efficient Distributed Real Time Commit Protocol", Proceedings of the 9th WSEAS International Conference on COMPUTERS Vouliagmeni, Athens, Greece, July 14-16, 2005.
- (14) "Optimizing Distributed Real-Time Transaction Processing During Execution Phase" Proceedings of the 3rd International Conference on Computer Application (ICCA2005), University of Computer Studies, Yangon, Myanmar, pp. 1-7, March 9-10, 2005.
- (15) "Priority Assignment to Cohorts Executing in Parallel" Proceedings of the 3rd International Conference on Computer Application (ICCA2005), University of Computer Studies, Yangon, Myanmar, pp. 39-45, March 9-10, 2005.
- (16) "A New Commit Protocol For Distributed Real-Time Database Systems" Proceedings of the IASTED International Conference on Databases and Applications (DBA 2005), Innsbruck, Austria, pp. 122-127, Feb. 14-16, 2005.
- (17) "A Modified Distributed Real-Time Commit Protocol" Proceedings of the International Conference on Systemics, Cybernetics and Informatics, Hyderabad, India, pp. 783-786, Jan 6-9, 2005.
- (18) "Hard Real-Time Distributed Database Systems: Future Directions" Proceedings of All India Seminar on Recent Trends In Computer Communication Networks, Department of Electronics & Computer engineering, Indian Institute of Technology Roorkee, India, pp. 172-177, Nov. 7-8, 2001.

