INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

Rollback Strategies for Controlling Memory Contention in a Multiprocessor-Based Telephone Switch

By
Ying Qiao, B.Eng.,

A thesis submitted to the
School of Graduate Studies and Research
In partial fulfillment of the requirements for the degree of

Master of Engineering
in Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University

May, 2001

© 2001, Ying Qiao, Ottawa, Canada

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-66880-0

Canada

The undersigned recommend to
the Faculty of Graduate Studies and Research
Acceptance of the thesis

**Rollback Strategies for Controlling Memory Contention in a
Multiprocessor-Based Telephone Switch**

Submitted by
Ying Qiao, B.Eng.,

In partial fulfillment of the requirements
For the degree of Master of Electrical Engineering

Chair, Department of Systems and Computer Engineering

Professor Shikharesh Majumdar, Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
May, 2001

Abstract

The thesis focuses on memory contention control on a multiprocessor-based telephone switch. As concurrent transactions run in parallel on multiple processors, memory contention occurs if they access a common memory line. To preserve data integrity, a lock and rollback mechanism is adopted for resolving this memory conflict. A transaction locks a memory line before accessing it. When a contention occurs, one of the contending transactions is rolled back. The rolled back transaction releases the memory line held and restarts from the beginning at a future point in time. The transaction to be rolled back is decided by a rollback strategy. Rollbacks expend work to preserve data integrity. A good rollback strategy should be efficient in decreasing the negative effect of rollback on the system throughput.

This thesis presents experimental results from a trace driven simulation of a number of rollback strategies. The rollback strategies implemented in the simulator are: Time Stamped, Age based, Bounded Priority, and Reverse Share. System mean response time, useful processing power, rollback wasted power, produced by the different strategies are compared under different blocking level workloads, and various values of rollback processing overheads.

Acknowledgements

I would like to express my gratitude to my supervision Prof. Shikharesh Majumdar for his constant guidance, inspiration, advice, and encouragement throughout my course study and this thesis work.

I would also like to thank Mr. Brian Carroll and Mr. Dale Streibel for giving me valuable advice when I did this thesis project in Nortel. I also appreciate Mark Christopher, Ping Wu, Ali Labed, and Minru Liu for their help and information to this project work in Nortel.

I would like to acknowledge the supports of Natural Science and Engineering Research Council (NSERC) of Canada and Communications and Information Technology Ontario (CITO).

I would also like to take this opportunity to thank my parents and my sister for their continued encouragement. Thanks to my husband, Hua Cheng, for his assistance and support.

Table of Contents

Chapter 1 Introduction	1
1.1 Motivation	1
1.2 Objective.....	3
1.3 Thesis Organization.....	5
Chapter 2 Background	6
2.1 Telephone Switch Architecture	6
2.2 Multiprocessor Core Architecture	7
2.3 Scheduler in Core Operating System.....	10
2.4 Message Processing in Core	11
2.5 Related Works	13
Chapter 3 Simulation Model and Rollback Strategies.....	17
3.1 Rollback Strategies.....	17
3.1.1 Time Stamped Rollback Strategy	17
3.1.2 Age Based Rollback Strategy	18
3.1.3 Bounded Priority Rollback Strategy	19
3.2 Workload Characterization.....	19
3.3 Simulation Model	23
3.4 Simulator Design and Software Architecture.....	28
3.5 Verification and Validation	31
3.6 Simulator Output	32
3.7 Design of Experiments	36

Chapter 4 System and Workload Parameters.....	38
4.1 Impact of Messages Arrival Rate	38
4.2 Impact of Number of Processors	42
4.3 Impact of Workload.....	44
4.4 Impact of Rollback Overhead.....	50
4.5 Discussion.....	55
Chapter 5 Performance Comparison of Rollback Strategies	56
5.1 Experiments on Single Class Systems.....	56
5.1.1 Low blocking level workload and low rollback overhead	57
5.1.2 High blocking level workload and low rollback cost.....	61
5.1.3 Low blocking level workload and high rollback overhead.....	67
5.1.4 High blocking level workload and high rollback overhead	70
5.1.5 Discussion of the results of experiments on single class systems	73
5.2 Experiments on Multi-class System	75
5.2.1 Low blocking level workload and low rollback overhead.....	79
5.2.2 Low blocking level workload and high rollback overhead.....	80
5.2.3 High blocking level workload and low rollback overhead.....	86
5.2.4 Discussion on two-class system experiment.....	86
Chapter 6 Conclusions	88
6.1 Research Summary	88
6.1.1 Single class system	89
6.1.2 Two-class system	90
6.2 Thesis Contribution	92
6.3 Suggestions for Future Research	93
References.....	95
Appendix 1	98
Appendix 2.....	99

List of Figures

Figure 2.1 Telephone switch architecture.....	7
Figure 2.2 Architecture of Multiprocessor-based core	8
Figure 2.3 Scheduler template of the core operating system on DMS switch.....	10
Figure 2.4 Mapping of total share of Payload and Non-Payload processing time	10
Figure 3.1 Simulation model.....	24
Figure 3.2 Process states in system.....	25
Figure 3.3 Component of time consumed by transactions.....	27
Figure 3.4 Simulator software architecture.....	29
Figure 3.5 Sequence diagram of Case 1: TransactionB requests memory failure	30
Figure 3.6 Sequence diagram of Case 2: TransactionB requests memory successfully	31
Figure 3.7 The system capacity	35
Figure 4.1 The increase in mean response time with arrival rate (3 processors, low blocking level workload, and low rollback overhead).....	39
Figure 4.2 The relationship between throughput and arrival rate (3 processors, low blocking level workload, and low rollback overhead).....	39
Figure 4.3 Impact of arrival rate on processing powers (3 processors, low blocking level workload, and low rollback overhead).....	40

Figure 4.4 Impact of arrival rate on collision rates (3 processor, low blocking level workload, and low rollback overhead).....	41
Figure 4.5 Comparison of mean response times achieved with 3 and 9 processors (low blocking level workload, and low rollback overhead).....	42
Figure 4.6 Comparison of collision rates achieved with 3 and 9 processors (low blocking level workload, low rollback overhead).....	43
Figure 4.7 Comparison of processing powers (low blocking level workload, low rollback overhead).....	44
Figure 4.8 Comparison of mean response times for low and high blocking level workload (3 processors, and low rollback overhead)	45
Figure 4.9 Comparison of collision rates for low and high blocking level workload (3 processors, and low rollback overhead).....	45
Figure 4.10 Comparison of processing power between low and high blocking level workloads at the usable capacity point (3 processor, and low rollback overhead).....	46
Figure 4.11 Comparison of collision rates under low and high blocking level workloads (low rollback overhead)	47
Figure 4.12 Comparison of processing powers achieved with high blocking level workload (low rollback overhead)	48
Figure 4.13 Comparison of collision rates under low and high rollback overheads (low blocking level workload)	51
Figure 4.14 Comparison of rollback wasted power under low and high rollback overheads (low blocking level workload)	51

Figure 4.15a Comparison of useful processing power with low and high rollback overheads (low blocking workload).....	53
Figure 4.15b Comparison of useful processing power with low and high rollback overhead (high blocking workload)	53
Figure 4.16 Comparison of total processing powers under low and high rollback overheads (high blocking level workload).....	54
Figure 5.1 Comparison of total processing power for rollback strategies (low blocking level workload, low rollback overhead).....	57
Figure 5.2 Comparison of useful processing power for rollback strategies (low blocking level workload, low rollback overhead).....	58
Figure 5.3 Comparison of collision rates for rollback strategies (low blocking level workload, and low rollback overhead).....	60
Figure 5.4 Comparison of total processing powers for rollback strategies (high blocking level workload, low rollback overhead).....	62
Figure 5.5 Comparison of useful processing power for rollback strategies (high blocking level workload, low rollback overhead).....	62
Figure 5.6 Comparison of rollback wasted power for rollback strategies (high blocking level workload, low rollback overhead).....	63
Figure 5.7 Comparison of collision rates for rollback strategies (high blocking level workload, and low rollback overhead).....	66
Figure 5.8 Comparison of total processing powers for rollback strategies (low blocking level workload, and high rollback overhead).....	67

Figure 5.9 Comparison of useful processing powers for rollback strategies (low blocking level workload, and high rollback overhead).....	68
Figure 5.10 Comparison of collision rates for rollback strategies (low blocking level workload, and high rollback overhead)	69
Figure 5.11 Comparison of total processing power for rollback strategies (high blocking level workload, high rollback overhead)	71
Figure 5.12 Comparison of useful processing power for rollback strategies (high blocking level workload, high rollback overhead)	72
Figure 5.13 Comparison of collision rates for different rollback strategies (high blocking level workload, and high rollback overhead).....	73
Figure 5.14 Comparison of processing powers for rollback strategies (7 processors, low blocking level workload, low rollback overhead, and rollback is charged to class)	79
Figure 5.15 Comparison of Payload processing powers for rollback strategies (7 processors, low blocking level workload, low rollback overhead, rollback is charged to class).....	80
Figure 5.16 Comparison of processing powers for rollback strategies (7 processors, low blocking level workload, low rollback overhead, and rollback is charged to system)	82
Figure 5.17 Comparison of Payload useful processing powers for rollback charged to class and charged to system (7 processors, low blocking level workload, low rollback overhead).....	82
Figure 5.18 Comparison of processing power for rollback strategies(7 processors, low blocking level workload, high rollback overhead, and rollback is charged to class)	83

Figure 5.19 Comparison of processing powers for rollback strategies (7 processors, low blocking level workload, high rollback overhead, and rollback is charged to system).... 85

Figure 5.20 Comparison of Payload useful processing power between rollback charged to class and charged to system (7 processors, low blocking level workload, low rollback overhead)..... 85

List of Tables

Table 3.1 Number of memory request per transaction	22
Table 3.2 Input parameters for the single class system	36
Table 3.3 Input parameters for the two-class system.....	37
Table 4.1 Comparison of arrival rates and collision rates under low and high blocking level workloads (low rollback overhead).....	48
Table 4.2 Comparison of rate of increase in useful processing power (power/PE) (low and high blocking level workload, and low rollback of overhead)	50
Table 4.3 Comparison of the collision rate to the arrival rate, when system is working with low and high rollback overheads (high blocking level workload).....	54
Table 5.1 Comparison of parameters for BP02 and BP05 (low blocking level workload, low rollback overhead)	59
Table 5.2 Comparison of parameters for BP02 and BP05 (high blocking level workload, low rollback overhead)	65
Table 5.3 Comparison of parameters for BP02 and BP05 (low blocking level workload, high rollback overhead).....	69
Table 5.4 Comparison of parameters for BP02 and BP05 (high blocking level workload, high rollback overhead)	72

Table 5.5 Comparison of Payload share in total processing power with that in useful processing power (7 processors, low blocking level workload, low rollback overhead, rollback is charged to class)..... 81

Table 5.6 Comparison of Payload share in total processing power with that in useful processing power (7 processors, low blocking level workload, high rollback overhead, rollback is charged to class)..... 84

Chapter 1 Introduction

Multiplicity is a way to supply extensive power. A multiprocessor computer system, a parallel I/O subsystem, and cluster of workstations are examples of such multiplicity deployed for achieving higher performance. The improvement in performance however comes at the cost of increasing system complexity. The major issues underlying management of the system including synchronization, load balancing and management of shared resource contention. Shared resources can be files in a distributed computing system, physical links in a network system, and memory lines in a shared memory system. The contention for shared resources often constrains system efficiency and scalability. This thesis focuses on research on the control of memory contention in a shared memory multiprocessor-based telephone switch.

1.1 Motivation

Network traffic has been increasing continuously for meeting the growing requirements of real world business application. As an important component of the telecommunication network, a voice and data switch needs to supply a large call-handling capacity to meet this demand. Efforts have been made in utilizing high speed CPU to improve processing ability, separating functions into multiple application modules with high-speed inter-processor communications, and adopting multi-stage memory for fast retrieval of instructions and data [Keis95]. This research focuses on an environment in which a

multiprocessor system is employed for improving the performance of the uniprocessor-based legacy switch.

The core of the switch is responsible for computing and message processing for call switching as well as for performing system management functions [Keis95]. The advantage of introducing a multiprocessor core with concurrent processes is that it provides more computing capacity. When call processing transactions are executed simultaneously in a multiprocessor machine, concurrent control must be carried out to guarantee the correctness and the integrity of the system. Many proposed concurrency control methods proposed are based on blocking and restart [Ulso92]. In these methods, a shared resource is locked by transactions using the resource. When a requested shared resource has been locked, the requesting transaction either blocks until the resource is unlocked, or rolls back after releasing the locks it already holds. Much research has been undertaken in blocking, restart, or a combination of both in database systems.

In the telephone switch, multiple processors share the global memory through a shared bus. A research project that focuses on controlling memory contention in this multiprocessor-based telephone switch is in progress at Nortel. It reduces memory contention in two ways: by avoiding large numbers of collisions through appropriate scheduling contentious transactions [Maju99a] [Liu00], and by deploying effective rollback strategies [Verm00] [Maju99b].

Real-time systems are usually categorized as either 'hard' or 'soft'. In a hard real time system, it is necessary to meet all time constraints or deadline. Although failing to meet the time constraints will not cause disaster in a soft real time system, processing requests

in a timely manner is still a primary issue [Stan96]. As a soft real time system, the performance of a multiservice telephone switch becomes important because it directly affects revenue and business efficiency for a network carrier company. Its customers will not be patient in waiting for a long latency dial tone and call setup. It requires the switch producer to provide a reliable, efficient, scalable and controllable system.

Avoiding deadlocks is important for responding to requests in a timely manner. Blocking a transaction on the occurrence of a memory contention may lead to a deadlock. Two transactions may wait for memory lines held by each other forever. Nortel has adopted a lock-rollback mechanism that does not suffer from this deadlock problem [Beni98].

1.2 Objective

As transactions run in parallel on multiple processors, memory contention occurs when two transactions access a common memory location. In order to preserve data consistency, transactions accessing the same memory locations are serialized by using a lock-rollback technique. The requesting transaction locks a memory location before accessing it. Locks are released when the holding transaction commits. When a transaction attempts to access a locked memory location, one of the transactions is rolled back. The rolled back transaction releases all the locks it is holding and restarts at a future point in time. The choice of the transaction to be rolled back is made by the rollback strategy. The objective of this research is to study rollback strategies and their effect on system performance.

The simplest of all the rollback strategies is First Come First Served (FCFS): the transaction that has generated the resource request first wins whereas the transaction that has generated a later request rolls back. Other rollback strategies we study are the Time Stamped, Age Based, Bounded Priority, and Reverse Share. The system performance achieved with these rollback strategies is studied with the help of a trace driven simulator. Some of these strategies are available in the literature on databases whereas the others are proposed by Nortel Networks.

Because this research is concerned with a telephone switch, a trace driven simulation is conducted to gain credible results [Carr99]. The input traces are collected from the switch and the traces reflect the real memory contention among transactions running on the system. Traces with various levels of memory contention are used. The output from the simulator are the mean response time, throughput, and system processing power.

Two related issues are also examined in this thesis: system scalability with increasing of number of processors, and the effect of different rollback costs on the performance of the different rollback strategies.

Two classes of processes run on the switch: Payload and Non-Payload. Examples of Payload processes are call processing and billing whereas Non-Payload processes are concerned with subsidiary activities such as switch maintenance. The main focus of this thesis is on single class systems, because it enables capturing the impact of the stronger workload component, Payload, on performance. Analysis of a two-class system that demonstrates the effect of interaction of two workload classes on system behavior and performance is also presented.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 introduces the multiprocessor-based telephone switch system, its scheduler, and message processing on it. Related existing work on concurrency control in transaction processing systems is also described. Chapter 3 describes the Time Stamped, Age Based, Bounded Priority, and FCFS rollback strategies. Workload characterization, simulator architecture, and design of experiments are also presented in this chapter. The factors affecting system performance are described in Chapter 4. The first part of Chapter 5 presents the results of experiments on a single class system. The second part of Chapter 5 presents the Reverse Share rollback strategy as well as the results of experiments on two-class systems. Finally, conclusions from this research, the contribution of the thesis, and suggestions for further research are presented in Chapter 6.

Chapter 2 Background

This chapter is intended to provide background of this thesis work by introducing the telephone switch architecture, the multiprocessor-based core system, and its processes, scheduler and message processing. Related previous work on the concurrency control in transaction processing systems is discussed in the later part of this chapter.

2.1 Telephone Switch Architecture

A block diagram of the studied telephone switch architecture is shown in Figure 2.1 (adopted from [Keis95]). The highest layer is the Service Processing layer, containing the Core, Bus, and application processors for Line and Trunk services, Network Services, and Operations Management. The second layer, the signaling and connectivity layer, contains the Link Peripheral Processors, for signaling, interfacing and control, and the switching network (ENet). The lowest layer, the Physical Access Layer, contains line and trunk interfaces to both analog and digital facilities [Keis95][Nort95].

The service-processing layer takes charge of central call handling, and system management. Its higher level functions are associated with the wide variety of telecom services provided by Nortel's DMS switch [Nort95]. Signaling and connectivity layer is the major internal messaging component and contains the call switching fabric. It transfers traffic from an incoming trunk to an outgoing trunk. The physical access layer is responsible for terminations, signal processing, service-specific protocol handling and

multiplexing. It provides interfaces and processors having access to external signaling and data networks, and allows lines and trunks carrying voice or data traffic to connect to the DMS switch.

Figure 2.1 Telephone switch architecture

2.2 Multiprocessor Core Architecture

The core is the principal computing and memory resource of the switch. It performs call processing and system management. The software of the core plays a major role in Operations, Administration and Maintenance. It translates call address digits, selects outgoing trunks, processes call features, and supervise a call. It has functions in controlling the enhanced switching network (ENET), maintaining core and its links,

loading switch software and monitoring processor sanity. It also loads the software of switch peripherals [Beni98].

Intending to increase its processing capacity, a shared memory multiprocessor Core (MultiCore) was designed by Nortel. The MultiCore consists of Processing Elements (PEs), I/O Processors (IOPs), a shared memory and shared Bus, as shown in Figure2.2.

Figure 2.2 Architecture of Multiprocessor-based core

A PE is a Central Processing Unit (CPU) card with private memory storage. It is responsible for computing, and the processes running on it mainly include the call processing processes, system operations processes, and maintenance processes. The private memory storage has two components: a Program Store (PS) and a Data Store (DS). PS holds a copy for each program, which runs in PEs. CPU fetches instructions from local PS directly. Data store is local data memory storage [Beni98].

The MultiCore, containing multiple PE cards, is a symmetric multiprocessing system. All PE cards are equal in that they can each run the operating system and application processes. Thus, any work that needs to be done in the system can be performed by any of the processors. Because each PE card has a kernel image in its own PS, all PE cards can run the kernel and can do so simultaneously. The application processes, which are the call processing and maintenance processes, can run simultaneously on different PE cards.

The shared memory is also separated as Program Store (PS) and Data Store (DS). Programs are loaded into global PS when the system is initialized. A PE runs the system maintenance process to refresh the program copy in its local PS. In the global DS, each memory line has an extra byte as an ownership tag, implemented by hardware, as well as the data bytes. When a PE issues an operand fetch or store request to a memory line in shared memory for the first time, a cache miss will occur, and the request will be passed through to the shared memory. If the memory line is free and the ownership tag is empty, the request will succeed, and an ownership tag for the memory line will be created for the requesting PE. When the process finishes executing, a commit operation will be executed. A commit operation is a checkpoint of an executing process. It flushes all the cache line back to the shared memory, and clears cache DS. The ownership tags of memory lines in shared memory will also be cleared [Beni98].

Shared memory on the multiprocessor is duplicated as Active memory and Commit memory. One of the memory planes is actively used by a transaction at a given point in time. On a commit changes made by the completed transaction are made available in the commit memory. On a rollback changes made by the rolled back transaction are

discarded and the initial values from the Commit memory are retained. A detailed discussion is provided in [Beni98].

2.3 Scheduler in Core Operating System

Payload and Non-payload are two categories of the processes executed in the core. Payload processes do call-processing, and take charge of call setup, billing and provide various customer services. Non-payload processes include the Audit processes, that monitor the system, Maintenance processes, processes that load new programs to local PSs of PEs, and so on. Network Carrier companies get almost all their revenue, and all their profit from their customer for payload working, so this thesis will focus on the useful system capacity divested to payload processes.

The scheduler in the operating system of the core assigns the CPU time among Payload and Non-Payload processes by time-sharing (see Figure 2.3). Figure 2.3 shows the representation of a scheduler template. Each component of the pie represents a proportion of CPU time divested to a particular process type. Figure 2.4 shows the overall proportions of assigned CPU time between Payload and Non-Payload:

Figure 2.3 Scheduler template of the core operating system on DMS switch

Figure 2.4 Mapping of total share of Payload and Non-Payload processing time

Currently, in approximately one round of template, Payload processes use 70% of CPU time, while Non-Payload processes consume the remaining 30%. Each PE card runs processes using its own copy of scheduler template.

2.4 Message Processing in Core

The application software running on the switch core is message driven. Processing time for a message is much shorter than a process's time slice. During one time slice, a process works on several messages by invoking one transaction for each message. Thus a process consists of many transactions and a transaction is invoked when the process retrieves a message from the message queue.

A transaction must work successfully until its end, or abort totally and recover the system state if it is rolled back, so that the system can remain in a consistent state [Beni98].

In the example below, T1 and T2 are two transactions with some common memory line requests.

<i>T1:</i>	<i>T2:</i>
<i>Read address1 into A;</i>	<i>Read address3 into C;</i>
<i>Read address2 into B;</i>	<i>Write C into address1;</i>
<i>A=A+B;</i>	<i>Read address2 into D;</i>
<i>Write A into address2;</i>	<i>D=C+D;</i>
	<i>Write D into address2;</i>

Address1, address2, address3 are the memory addresses in global memory, and, *A*, *B*, *C*, *D* are the local variables of transactions. If these two transactions run concurrently, accessing to the common shared memory lines needs to be serialized.

In MultiCore, multiple PE cards are inserted into a switch cabinet, and shared memory is the only data storage for the system. Processes run in parallel and invoke transactions simultaneously. The transactions must behave as if they run in sequence. To ensure memory integrity during running, a lock mechanism is implemented by an ownership tag in memory line. The actions for a memory request become lock and read, as well as lock and write. A transaction does Begin, Commit, and Rollback operations in its runtime. A number of memory requests can be performed by each transaction. Those locks held by the transaction are released implicitly during its rollback or commit operation.

In the previous example, the run time actions of T1 are shown as below:

T1:
Begin;
Lock address1 and Read address1 into A;
Lock address2 and Read address2 into B;
A=A+B;
Lock address2 and Write A into address2;
Commit; {locks are released implicitly}

In this case, T1 runs in parallel with T2, and T1 requests and locks address1 before T2. When T2 requests address1, a contention will occur and one of the transactions will be rolled back. In this example we assume that the rollback strategy chooses T2 to rollback. The actions of T2 are shown as below:

T2:
Begin;
Lock address3 and Read address3 into C;
Lock address1 and Write C into address1;
Rollback; {locks are released implicitly}

During the Rollback operation, T2 releases the memory lines locked by it, and issues a delay time to restart. The delay time prevents the same collision occurring again before T1 finishes. T1 will continue to run until it ends with a commit operation. Commit is a last operation for a successful execution of a transaction. The running process writes back all the updated memory from its own DS, unlocks all the memory previously locked, and clears its local DS cache for next transaction.

2.5 Related Works

When a number of different transactions are being executed at the same time in a concurrent system, special techniques must be carried out to ensure the correctness when multiple transactions update a shared storage resource simultaneously.

Research in the area of concurrency control for transaction systems has led to development of many concurrency control algorithms. Most of the algorithms are based on one of three basic mechanisms: locking, timestamp, and optimistic concurrency control (also called commit-time validation or certification).

Locking: Before accessing a data item, transaction must first lock it. If the data item is not currently locked, then the lock is granted, and the transaction can lock it and access it. Otherwise, the transaction has to wait until the lock is released [Silb98].

Timestamp: Each transaction is associated with a unique fixed timestamp. The timestamps of transactions are used to determine the serialization order of transactions on the system. Methods for implementation of timestamp based concurrency control strategies are discussed in [Silb98].

Optimistic: Transactions are allowed to execute to completion and are validated only after they have performed their operation and are about to commit. A transaction is restarted at its commit point if it finds that any data item that it used has been changed by another transaction that committed during its lifetime [Kung81].

Restarted concurrency control algorithm is one of algorithms studied in detail through simulation in [Agra87]. If a requested resource is locked, the requesting transaction is aborted and restarted after a restarted delay. The delay period prevents the same conflict from occurring repeatedly. The restart algorithm is studied together with locking and optimistic algorithms. The results show that the performance of the restart algorithm is sensitive to the restart delay time. The delay of about one transaction response time is best, and the throughput begins to drop off rapidly when the delay exceeds more than a few transaction times. Also, at a low multiprogramming level, these algorithms affect system performance in a similar way.

Restart-oriented locking methods combine blocking and abort to cope with the performance limitations of standard locking [Thom98]. This allows a transaction encountering lock conflicts to be blocked. When multiple transactions block for a given data item, the level of lock contention is reduced by aborting some of these transactions. The aborted transactions release all the locks that they currently hold. The authors conclude that restart-oriented locking methods reduce the level of lock contention at the cost of additional processing. In a system characterized by a high level of lock contentions a significant increase in the maximum throughput is possible.

A real-time transaction system is a transaction system designed to provide information in real-time. It should satisfy the properties of both real-time systems and conventional transaction system, by providing real-time response for queries and updates while maintaining the consistency of data. In [Ulso92], the author presents a lock-based protocol which prevents the priority inversion problem by scheduling the lock requests based on priorities associated with data items. Extending the basic timestamp-ordering method by using real-time priorities of transactions in scheduling the data accesses is also discussed. These two methods as well as previous proposed concurrency control protocols, such as lock-based protocols, and timestamp-ordering protocols are compared through a simulation.

In real time transaction processing systems, in order to meet the time criteria, scheduling strategies are introduced into concurrency control algorithms. Stankovic and Zhao [Stan 88] proposed several scheduling methods for soft real time transactions. The methods attempt to make scheduling decision based on the real time requirement of the transaction. Majumdar [Maju99a] and Liu [Liu00] studied scheduling strategies in reducing the shared resource contention in multiprocessor-based multi-class transaction systems.

Real-time transaction processing on centralized and distributed databases has been studied by a number of researchers (see [Hari91], [Hong93] for example). Scheduling of hard real time transactions on shared nothing parallel database systems in which scheduling decisions are made locally at each node is discussed in [Takk98]. Using

priority caps to bound the number of priority inversions on parallel hard real time database systems is proposed in [Kuo00].

Comparatively little research exists on pure restart algorithms, which choose one of two transactions to be rolled back when collision occurs between two of them. In this thesis, five pure restart algorithms, Bounded Priority, Age based, Time Stamped, FCFS, Reverse Share and their effect on system performance under different contention levels and different number of processors are compared.

Chapter 3 Simulation Model and Rollback Strategies

A simulation study is conducted to compare and analyze the effects of the rollback strategies on system performance. As sets of memory traces are collected from a real telephone switch, it is possible to undertake a trace driven simulation and obtain a credible result. Before describing the workload captured by the traces and the simulation model, short descriptions of the rollback strategies are presented.

3.1 Rollback Strategies

When a collision occurs between two transactions, one of the transactions will be rolled back according to the rollback strategy. The rollback strategies presented in this chapter include Time Stamped (TS), Age Based (AG), and Bounded Priority (BP). Reverse Share (RS) is dedicated for multi class system and is presented in Section 5.2. Experiments with the FCFS strategy are also carried out as performance benchmarks for the other strategies. The FCFS strategy rolls back the transaction that attempted to lock the desired memory location later and allows the transaction that is currently holding the lock to continue. The other roll back strategies are discussed in the following subsections.

3.1.1 Time Stamped Rollback Strategy

Time Stamped rollback strategy gives higher priority to the transactions with longer waiting time messages. As an input message comes to the system, a time stamp is

attached to a data structure associated with it. A process starts a transaction and the time stamp of the message is associated with the transaction processing it. If a collision with another transaction happens during execution, time stamps of both transactions are compared, and the transaction with the earlier time stamp wins, and the transaction with a later time stamp is rolled back.

3.1.2 Age Based Rollback Strategy

The Age Based rollback strategy makes decisions based on the message's previous rollback times as well as their arrival times. In a message's data structure, its arrival time, as well as the number of times it has rolled back are recorded. There are only two levels of priority, 0 and 1. Priority 0 is for the transaction processing a message which has not been rolled back before. Priority 1 is for the transactions processing a message which has been rolled back before. When a collision occurs between the same priority transactions, the messages' arrival times are compared as in the Time Stamped rollback strategy, and the transaction with the earlier arrival time wins. When a collision occurs between different priority transactions, the priority 1 transaction wins, and the priority 0 transaction is rolled back [Verm00].

The difference between Age based strategy and Time Stamped strategy is that, Age Based strategy considers the messages' previous rollbacks and gives a higher priority to the transaction with the message that had caused an earlier roll back.

3.1.3 Bounded Priority Rollback Strategy

In the strategies discussed earlier, a message may cause a few rollbacks before its final commit. Because rollback processing consumes CPU processing power, higher roll back times means wasting more CPU power. The Bounded Priority strategy controls the wasted CPU power.

The Bounded Priority strategy is associated with an integer N . With a Bounded Priority N strategy, the highest priority of a transaction on the system is set to N . A message's data structure contains a field for storing the associated number of previous rollbacks. The number of rollbacks field is increased by one for each rollback. When the transaction runs, its priority number is given by the content of the number of rollbacks field. Priority level is in the range of 0 and N . A higher priority transaction wins over a lower priority transaction. If the priorities of transactions are the same, the transaction with the earlier arrival time wins.

Only one of the priority N messages can be processed at a time; that is, the other priority N messages must wait in message queue until the transaction processing a priority N message completes. Multiple lower priority processes can however run concurrently with the highest priority process. Because only one priority N transaction runs in the system, it always wins the contest when a collision occurs and runs to commit.

3.2 Workload Characterization

This research is based on trace driven simulation. The workload processed by the simulator consists of a list of service requests to the system, and it represents a system

usage in real life. The system behavior depends on its workload. Workload characterization is the process of studying the real user environments, identifying the key characteristics of the workload, with the aim of developing a workload model with workload parameters that can be used repeatedly in experiments. Once a workload model is available, the effect of changes in the workload and system can be studied in a controlled manner by simply changing workload parameters.

The workload corresponds to the memory requests sent from transactions to shared memory. A given transaction's memory requests are stored in one trace. At present, as many as 347 traces are available; these traces have been collected from a real system by Nortel Networks. A trace includes the memory address requested by the caller, the CPU time for the transaction, and the probability of the transaction running on the system.

Below is an example of a trace file.

Transaction T_i : 20 ms 0.00005

02DF3451 0.05

0434E322 0.87

The first line contains the transactions' name, its service demand and the probability that it will run on the system. Since each transaction corresponds to a message type, the probability is equal to the probability of arrival for a message of the corresponding type on the system. The subsequent lines are characterized by two columns. The first column is the memory address of a request, and the second column is the access time for the request normalized with respect to the transaction's run time. In this example, transaction T_i runs for 20ms on the system. It has a probability of 0.00005 to run on the system, and

generating two memory requests with addresses 02DF3451, and 0434E322. The transaction sends the first memory request 1 ms ($0.05 \times 20\text{ms}$) after starting its execution, and the second 17.4ms ($0.87 \times 20\text{ms}$) after its execution.

The workload has two parameters, service demand and blocking level. The service demand is the total CPU time as the transaction runs on a uniprocessor system. The blocking level is a measure of the level of contention for common memory address accessed by the different transactions. A high blocking level workload, potentially gives rise to more rollbacks in comparison to the low blocking level workload.

The mean value of the service demand for transactions used in our experiments is 20.8520 ms. The variance is 318.228, and its standard deviation is 17.8390. Coefficient of variation is the ratio of standard deviation and mean. It indicates the variability in transaction execution time. For the transactions used in our experiments, coefficient variation of their service demands is 0.85555.

Another parameter for the workload of the system is the blocking level. Blocking levels can be obtained by making measurement on the real system. Different products may be characterized by different blocking levels. Traces with different memory blocking levels captured on real switches are available. There are two types of workloads that will be used in the experiments reported in this thesis: low memory blocking level workload, and high memory blocking level workload. Each workload type is characterized by different demands on memory. Statistics on the number of memory requests generated by the two-workload types are shown in Table 3.1.

Table 3.1 Number of memory requests per transaction

	LOW BLOCKING LEVEL	HIGH BLOCKING LEVEL
Mean	10.6364	43.588
C.V	3.798	1.5174

In comparison to service demand, the blocking level is expected to have a large impact on the number of collisions. Both kinds of workloads have the same service demand time with different blocking levels.

From the trace files, the collision group of each transaction can also be determined. The collision group of a transaction T_i is a set of transactions that have common memory requests with T_i . $G_{T_i} = \{T_j \mid D_{T_j} \cap D_{T_i} \neq \emptyset\}$; where G_{T_i} is the collision group of transaction T_i , and D_{T_i} is the set of memory location accessed by transaction T_i .

By the definition,

- 1) $T_i \in G_{T_i}$;
- 2) if $D_{T_j} \cap D_{T_i} \neq \emptyset, (i \neq j)$, $T_j \in G_{T_i}$
- 3) if $T_j \in G_{T_i}, i \neq j$ then $T_i \in G_{T_j}$

p is defined as the probability of collision when two transactions run in parallel on two PEs:

$$p = \sum_{i=1}^N (p_i \times \sum_{j=1}^{M_i} p_j);$$

N is the total number of transaction types and is equal to 347 in the system. M_i is the number of transactions in the collision group of transaction T_i . p_i is the probability that

transaction T_i is running on the system. In the low blocking level workload the value of p is 0.0421. In the high blocking level workload the p value is 0.2093. Thus, the probability of collision between two parallel transactions is 0.0421 for the low blocking level workload, and 0.2094 for the high blocking level workload.

3.3 Simulation Model

The simulation model is based on an open queuing model shown in Figure 3.1 [Jian00]. When a message comes to the system, it is appended to the end of the message queue and waits for processing. The process running on a PE will invoke a transaction to process the message in the queue. The transaction sends out a memory request to the shared memory while running; if the memory line is free, the transaction is allowed to access it, and lock it. The transaction will continue to run. If the requested memory is locked by the other transaction, a collision occurs, and the rollback strategy chooses one of the colliding transactions to be rolled back. After a commit or rollback, the process will release the locked memory lines, clear its cache and invoke a new transaction from the current head of the message queue. If there is no message waiting in the message queue, the process will be swapped out by a scheduler. Bounded Priority rollback strategy has an additional requirement that is explained in Section 3.1.3. When a process's time share is over, the scheduler lets the process complete the processing of the current message and swaps it out even if there are additional messages in the message queue.

Figure 3.1 Simulation model

The process state graph is shown in Figure 3.2. A process waits to run in the Ready state.

Figure 3.2 Process states in system

Scheduler assigns a process to run on a PE. The process comes back to Ready state from Run state when the share of running time in the scheduler template is over. It goes to Wait state if there are no more messages in the message queue. A process enters the Rollback transaction state when it does the rollback operation. After the rollback is finished, it can either return to the Run state for processing a message in message queue, or the Wait state if the message queue is empty. If the process's time slice is over it goes to the Ready state.

To prevent a rolled back transaction from colliding again with the same transaction that caused the rollback, the rollback message is not restarted immediately, but put in a

penalty queue where it waits until other transaction commits [Stre99]. Then it is put back to the system message queue and is available for processing.

The purpose of the simulation is to compare the system behavior under different rollback strategies. The overheads associated with commits, transaction swapping after committing, and process scheduling time do not change with the rollback strategy. Because they do not concern the relative performance of the policies, we do not consider them here.

The processing time measured during simulation includes the running time of committed transactions, the time of rollback operations and the running time of rolled back transactions.

Rollback time or cost consists of the memory releasing time, and the transaction swapping time. During a rollback operation, the memory locked by the rolled back transaction must be released, so that, other transactions can use it when the message is not processed. After that, the process must switch the message out, and start a new transaction for the message at the head of message queue. The time consumed by two types of transaction is shown in Figure 3.3. Two values of rollback cost are used in this simulation, 8.2 ms and 32.5148 ms. The 8.2 ms is the assumed value used in the beginning of the XA-Core project at Nortel [Stre99], and the 32.5148 ms is based on the measured value after the real hardware was developed. During the simulation run the value of the rollback overhead given in a configuration file is used to simulate the CPU time burnt on the occurrence of rollback.

A: a run-to-commit transaction (1): the total running time of transaction A

B: a rolled back transaction (2): the wasted running time

(3): Memory-releasing time

(4): Transaction-swapping time

Figure 3.3 Component of time consumed by transactions

We assume that a message arrives at the system in a totally random manner with a Poisson distribution. So, the inter-arrival time between two messages follows an exponential distribution.

3.4 Simulator Design and Software Architecture

The simulator is implemented with a simulation tool called PROPHET [Nort96] and the process-interaction approach introduced in [Bank99]. The tool provides a simulation engine and a multithread library that can be used to mimic the parallel running processes in a multiprocessor system. It also provides plenty of templates for developing simulator applications, and functions for collecting data statistics.

The simulator software is composed of two major parts: workload generator and switch. Figure 3.4 shows its architecture. The workload generator creates messages according to the trace files and sends them to the message queue of the switch. Each message contains memory addresses as well as their requesting time as described at Section 3.2. During the time between two sequential messages, the workload generator will sleep for an exponentially distributed duration with a given mean.

The simulator for the call-processing module in this switch contains processes, shared memory lines, rollback strategy and scheduler. Scheduler selects processes from the process pool to run on PE's. Currently, each PE has its own scheduler in the real system. With the exception of the scheduler, the Scheduler Template, and the Base OS, all the other modules in Figure 3.4 were implemented by the author of this thesis. The scheduler simulator designed by Verma [Verm00] was adopted to a two-class system where a rollback can be charged either to the system or to the class (see discussion in Section 5.2).

Figure 3.4 Simulator software architecture

The sequence diagrams in Figure 3.5 and Figure 3.6 show the implementation of rollback and commit for transactions in the processing unit in the simulator. It is assumed that TransactionA and TransactionB run in parallel and request the same memory line: memorylineI, during their execution. In Case 1 (shown in Figure 3.5), TransactionA has higher priority. TransactionA requests the memory line first and owns it by setting a tag in the memorylineI. It continues its running. Before it commits, TransactionB requests the memory line. Because the tag in memorylineI has been set, a collision occurs. TransactionB invokes the rollback strategy to make the decision on which transaction should continue to run. Rollback strategy chooses TransactionA. So, TransactionB releases all its locked memory by clearing the tag in memory line and put the processing message back.

Figure 3.5 Sequence diagram of Case 1: TransactionB requests memory failure

In Case 2 (shown in Figure 3.6), it is assumed that TransactionB has the higher priority. According to the decision made by the rollback strategy, TransactionB wins. Then, it sends a “rollback” message to TransactionA. TransactionA rolls back after it receives the message.

Figure 3.6 Sequence diagram of Case 2: TransactionB requests memory successfully

3.5 Verification and Validation

The simulation program is verified through the log files and the output report files. A log file records each activity, including the work generator generating messages, process invoking a transaction for a message, rolling back when collision occurs, and transaction committing. The performance metrics are reported in the output report file. They include the processor utilization, running time, message mean response time, and queue length.

By reading the log file, the correctness of message processing and the decision made by the rollback strategies are checked. Different test cases were used in the verification. The simplest case is the uniprocessor single class system. There is no memory contention in this case, and the transaction will not be interrupted during running. It tests whether the program can do the basic work, such as message generation, memory allocation, transaction commit, and so on, correctly. Various rollback strategies and number of processors are combined to test the correctness of the program. The most complicated case is the multi-class system with the maximum number of processors under the high blocking level workload.

The simulation is validated with the help of an analytic modeling tool [Wu00a] provided by Nortel. By using the same input values in number of processors, workload blocking level, and rollback strategies, the outputs of the tool were observed to be close to those of the simulator.

3.6 Simulator Output

When the simulation is running, the throughput, response time, processing time, useful processing time and rollback wasted time are recorded. Throughput is the number of messages finished in the system per second. Response time of a message is the time difference between the message arrival time and its completion time. Processing time is total CPU working time during the experiment. Useful processing time is the total time in which CPU process transactions successfully with commits. Rollback wasted time includes total time wasted in rollback overhead and the CPU time consumed by the

transaction which is rolled back in the middle of execution. Number of collisions is the total number of collisions that occurred during the experiment.

From the data collected from the experiment, the total processing power, useful processing power, and wasted processing power can be computed. Total processing power is the ratio of total CPU working time to the total run time of the experiment. It measures how busy the system is.

$$TOTAL_PROCESSING_POWER = \frac{CPU_PROCESSING_TIME}{EXPERIMENT_TIME};$$

The useful processing power is the ratio of the total CPU time used for executing committed transactions to the total run time of the experiment. It measures how much useful work is done by the system. It is also directly proportional to the throughput of the system.

$$USEFUL_PROCESSING_POWER = \frac{USEFUL_PROCESSING_TIME}{EXPERIMENT_TIME};$$

The rollback wasted power is ratio of the total wasted time to the total run time for the experiment. The wasted CPU time includes the rollback overhead and the CPU time already consumed by the transaction that is rolled back.

$$ROLLBACK_WASTED_POWER = \frac{WASTED_CPU_TIME}{EXPERIMENT_TIME};$$

Note that,

$$TOTAL_PROCESSING_POWER = USEFUL_PROCESSING_POWER + ROLLBACK_WASTED_POWER$$

Collision rate is the mean number of collisions that occur per time unit. So, it is the ratio of number of collisions to the total run time for the experiment.

$$COLLISION_RATE = \frac{NUMBER_OF_COLLISIONS}{EXPERIMENT_TIME};$$

Full capacity processing power is the maximum power achieved on the system. As a multiprocessor system with N processors, the system processing power is,

$$FULL_CAPACITY_PROCESSING_POWER = TOTAL_PROCESSING_POWER + IDLE_POWER;$$

where idle power is the ratio of the total CPU idle time to the total run time of the experiment.

Nominal capacity, usable capacity and knee capacity are the three points in the Useful Processing Power versus Load graph [Jain92]. Figure 3.7 shows the relationship between these three points. As the system load increases the useful processing power is observed to increase until the knee is reached. Beyond the knee the system tends to saturate as a result of which the slope of the curve is significantly reduced. The maximum achievable useful processing power under ideal workload conditions is called the nominal capacity of the system. The usable capacity is the maximum useful processing power achievable without exceeding a specified response time limit. The useful processing power at the knee of the Useful Processing Power versus Load graph is called the knee capacity. As a soft real time system, the message handling time of the telephone switch is specified to lie within an interval of 1617 ± 231.111 ms [Carr99]. In this thesis, we focus on the useful capacity of the system when the message mean response time lies in this interval. A tool is developed for controlling the experiments to achieve the usable capacity point (see Appendix 1). The useful capacity at the usable capacity point is a measure of the revenue that can be generated by the switch. Higher this capacity higher is the revenue

generated. That is why this performance metric is of interest to Telcos and switch manufacturers.

Figure 3.7 The system capacity

The transient state at the start of simulation is removed by a longer warm up time. The warm up time and the simulation run length were specified by Nortel to meet their demands on accuracy [Carr99].

3.7 Design of Experiments

Two categories of experiments are performed on this simulator. The first is concerned with single class systems. In this case, we compare rollback strategies in an environment with only one class of workload. We study the system performance under the assumption that the transactions are statistically identical with the same mean value on shared memory blocking level. The second is concerned with a two-class system. The processes running on a PE can be divided with two classes: Payload processes, and Non-Payload processes. As introduced in Section 2.2, Payload processes only deal with Payload messages, and Non-Payload processes deal with Non-Payload messages. They are assigned CPU running times by the CPU scheduler. Two separated trace flows are fed to system, one for each class.

In the single class system, blocking level is the main factor for controlling the workload on system. Traces with both low blocking level and high blocking level are used. The effect of the rollback time on performance is also studied. Due to constraints on physical size of a telephone switch shelf, the maximum number of PE's is 10. The input parameters used in the experiments on the single class system are shown in Table 3.2.

Table 3.2 Input parameters for the single class system

<i>Processor number: 3, 5, 7, 9;</i>
<i>Mean service demand time: 20.8 ms;</i>
<i>Rollback overhead: low (8.2 ms), high (32.5138 ms);</i>
<i>Workload blocking level: low, high;</i>
<i>Rollback strategy: Time Stamped (TS), Age Based (AG),</i>
<i>FCFS, Bounded Priority 2 (BP02),</i>
<i>Bounded Priority 5 (BP05)</i>

This thesis focuses primarily on single class system. A limited number of experiments are conducted on a two-class system. In the two-class system, Payload and Non-Payload messages will be separated into different message queues. A Payload or Non-Payload process retrieves messages from the message queue for its own class. By investigating the two-class system, we tried to study the impact of the different rollback strategies on Payload useful processing power with Non-Payload processes simultaneously running on the system. The input parameters for the two classes system are shown in Table 3.3.

In addition to five rollback strategies studied on the single class system, Reverse Share (RS) rollback strategy, which is proposed by Nortel, is also simulated for a two-class system. It intends to assign more time to Payload processes by rolling back a higher proportion of Non-Payload processes (see Section 5.2 for details).

Table 3.3 Input parameters for the two-class system

<p><i>Processor number: 7;</i> <i>Mean service demand time: 20.8 ms;</i> <i>Rollback overhead: low (8.2 ms), high (32.5138 ms);</i> <i>Workload blocking level: low, high;</i> <i>Rollback strategy: Time Stamped (TS), Age Based (AG),</i> <i>FCFS, Bounded Priority 2 (BP02),</i> <i>Bounded Priority 5 (BP05),</i> <i>Reverse Share (RS)</i></p>
--

Chapter 4 System and Workload Parameters

The factors affecting the system performance are message arrival rate, number of processors, workload blocking level, and rollback policy and the concomitant overhead. This chapter describes how these parameters impact system behavior. The FCFS rollback strategy acts as a benchmark in this thesis, so it is chosen as the base rollback strategy in this study. A number of basic experiments that display the impact of the different factors on performance of FCFS are described in this chapter. The different performance metrics used in this thesis are also discussed. Figure 4.1 to Figure 4.16 present the results of these experiments. The rollback strategy used in all these figures is FCFS whereas other system parameters are indicated in the figure captions.

4.1 Impact of Message Arrival Rate

In this simulation model, messages arrive on the system following a Poisson distribution. The arrival rate is the mean number of messages arriving per unit time. Figure 4.1 and Figure 4.2 show message mean response time curve increasing with the increasing of its arrival rate. A knee point exists in Figure 4.1 at an arrival rate of 130 messages per second approximately. Before the knee point, the system can process the messages causing a short latency, and throughput is equal to arrival rate, so that no message is resident in a queue for a long time. When the arrival rate is increased, the system load is

Figure 4.1 The increase in mean response time with arrival rate (3 processors, low blocking level workload, and low rollback overhead)

Figure 4.2 The relationship between throughput and arrival rate (3 processors, low blocking level workload, and low rollback overhead)

increased, and, throughput is increased smoothly (see Figure 4.2). After the knee point, the increasing rate of throughput drops. Throughput will not be equal to arrival rate any more, and, the queue length becomes infinite with messages accumulated in the queue. Meanwhile, the mean response time line climbs sharply.

As a contention system, a part of processing time is consumed in handling the contention rather than doing effective work. The total processing power of a system is composed of two parts: useful processing power and wasted power. Useful processing power is the time during which the system runs successful transactions. Wasted power is the time that includes time consumed by rolled back transactions and the rollback overhead. Figure 4.3 shows useful processing power, wasted power and total processing power of the system versus the arrival rate.

Figure 4.3 Impact of arrival rate on processing powers
(3 processors, low blocking level workload, and low rollback overhead)

The collisions occurring on the system are reflected in the collision rate versus message arrival rate graph in Figure 4.4. Collision rate is the mean number of collisions happening in a system per time unit. One collision causes one transaction rollback. The higher the number of collisions, the more rollback wasted time is incurred by the system. When the system works at an arrival rate as low as 20 messages per second, the collision rate is about 0.2 times per second. Thus, the rollback wasted power is very small, close to 0 in Figure 4.3. At high arrival rates, the system is busy in processing messages; the number of requests to shared resource is increased. So the shared resource contention is also increased producing a larger collision rate. At this time, the system consumes more time in dealing with memory contention. This is shown in Figure 4.3 as the increase of rollback wasted power with the increase of message arrival rate.

Figure 4.4 Impact of arrival rate on collision rates
(3 processor, low blocking level workload, and low rollback overhead)

4.2 Impact of Number of Processors

The number of PE cards is the maximum number of transactions that can run concurrently on the system. When multiple PE cards are added to the system, the processing power of the system increases; thus, the system can produce more throughputs. The response time in a 9 PE system is compared with that in a 3 PE system (see Figure 4.5). The knee point in the 9 PE system is approximately at the arrival rate of 360 messages per second, which is far away from 130 messages per second in the 3 PE system.

Figure 4.5 Comparison of mean response times achieved with 3 and 9 processors (low blocking level workload, and low rollback overhead)

Figure 4.6 shows the collision rate on a 3 PE and a 9 PE system. System load is the overall utilization of the PE's on the system. There are more collisions happening in the 9 PE system than that in the 3 PE system when they work at the same utilization level. In the 3 PE system, the increase in collision rate with system utilization is small. In the 9 PE system, the increase in collision rate with an increase in load is much sharper. At usable

capacity point (described in Chapter 3), there are 60 collisions per second happening in the 9 PE system, which is 20 times of that occurring on the 3 PE system.

Figure 4.6 Comparison of collision rates achieved with 3 and 9 processors (low blocking level workload, low rollback overhead)

Figure 4.7 shows the system processing power at usable capacity point as a function of number of PE cards. As long as the increase in throughput and useful processing power increases with an increase in the number of PE cards, the collision rate and rollback wasted power are also increased. In the 3 PE system, it wastes 3% of total processing power in rollback, but, in the 9 PE system, this value is increased to 15% of total processing power. By adding 2 more PE to the 3 PE cards system, 61.28% more successful transactions can be processed. By adding 2 more PE to the 7 PE cards system, only a further 21.132% more can be processed. However, the useful processing power increases with the increase in the system size. Thus the system shows a good scalability when working with the low blocking workload.

Figure 4.7 Comparison of processing powers
(low blocking level workload, low rollback overhead)

4.3 Impact of Workload

Different types of work on the system are produced by the different traces. With the high blocking level workload, transactions send more memory requests in comparison to the low blocking level workload. As well, a shared memory line in the system running heavy blocking workload is shared by more transactions.

Figure 4.8 compares the mean response times for two workloads at different message arrival rates. In the high blocking workload, system passes the knee point at 115

Figure 4.8 Comparison of mean response times for low and high blocking level workload (3 processors, and low rollback overhead)

Figure 4.9 Comparison of collision rates for low and high blocking level workload (3 processors, and low rollback overhead)

messages per second, which occurs earlier in comparison to the low blocking workload system. The collision rates versus message arrival rate of the system with low and high workloads are compared in Figure 4.9. For a given load, there are more collisions happening per time unit in the high blocking level workload system than in the low blocking level workload system. As expected, the collision rate with the high blocking level workload increases faster in comparison to the low blocking level workload with the increase of message arrival rate.

Figure 4.10 shows the processing power of the 3 PE system running with two kinds of workloads. At a low blocking level workload, the system runs 97% of its total processing power in executing transactions successfully at the usable capacity point. At a high

Figure 4.10 Comparison of processing power between low and high blocking level workloads at the usable capacity point (3 processor, and low rollback overhead)

blocking level, only about 78% of total processing power is useful. 22% of total processing power is consumed in the rolled back transactions and in rollback overheads.

The collision rate as a function of the number of PE's is shown in Figure 4.11. The difference in collision rates between two kinds of workload is dramatic. In the 3 PE system, a high blocking level workload produces 13 times the number of collisions produced by the low blocking level workload. The increase in collision rates with the increase in system size is sharp. The comparison of collision rate with the message arrival rate at the usable capacity point is shown in Table 4.1. With high blocking level workload, and for system with 5 or more PE's, the system collision rate is higher than the message arrival rate. The meaning of this is that multiple rollbacks are experienced by a message on such systems. At 9 PE, the collision rate is 2.5 times the arrival rate on the system.

Figure 4.11 Comparison of collision rates under low and high blocking level workloads (low rollback overhead)

Table 4.1 Comparison of arrival rates and collision rates under low and high blocking level workloads (low rollback overhead)

Low blocking level				
Number of processors	3	5	7	9
Arrival rate(msg/sec)	140.82	227.109	304.8437	371.8744
Collision rate(collision/sec)	3.49763	13.8298	31.34589	62.16188
High blocking level				
Number of processors	3	5	7	9
Arrival rate(msg/sec)	113.281	139.844	145.7031	143.75
Collision rate(collision/sec)	42.9252	143.967	275.4031	362.1422

The useful processing power of a system at usable capacity point is shown as a function of the number of processors in Figure 4.12. In the 5 PE system, only 58% of total processing power is useful. By adding two more PE cards to the 5 PE system, the proportion of useful processing power increases only marginally. After the 7 PE system, the wasted processing power exceeds the useful processing power, and adding more PE

Figure 4.12 Comparison of processing powers achieved with high blocking level workload (low rollback overhead)

cards only increases the rollback wasted power and idle time instead of the throughput.

This is because a large number of processes run more transactions concurrently which leads to an increased number of collision for the high blocking level workload. The useful processing power becomes a plateau after 7 PE. At high blocking workload, the system cannot work at full capacity point and maintain a given message latency requirement. Increasing the system load by adding traffic only increases the queue length and makes the message latency exceed the desired level. In the 7 PE system, it only can work at 95% of its full capacity processing power to meet these latency requirements. When the 9 PE system runs a high blocking workload, only 33% of system running time is used to do useful work, 11% of system time is wasted in latency control, and 56% of system time is used in rollback (see Figure 4.12). In order to ensure that the mean response times lie within the specified interval, the message arrival rate cannot be increased in an unconstrained fashion. This may result in idle duration for the system. This is the system time that is said to be lost due to latency control.

The number of processors affects the system performance along with workload interactively. The rates of increase in useful processing power with the increase in system size with low and high blocking level workloads are compared in Table 4.2. At low blocking level, increasing the system size from 7 PE to 9 PE, the useful processing power of the system will increase 0.668 power per PE. At high blocking level, we observe a lower rate of increase in power. Thus, the system is not scalable with the high blocking level workload.

**Table 4.2 Comparison of rate of increase in useful processing power (power/PE)
(low and high blocking level workload, and low rollback of overhead)**

Low blocking level			
Variation in number of processors	3~5	5~7	7~9
Useful Processing Power increasing rate	0.8918	0.8142	0.66805
High blocking level			
Variation in number of processors	3~5	5~7	7~9
Useful Processing Power increasing rate	0.27915	0.0568	0.00865

4.4 Impact of Rollback Overhead

High rollback overhead system wastes more processing time in handling rollbacks and the useful time in running concurrent transactions is reduced in comparison to system with a low rollback overhead. As a result, the memory contentions and collision rates are reduced. Two values of rollback overheads are set in this experiment: low overhead as 8.2ms and high overhead as 32.5148ms.

Figure 4.13 shows the collision rate at a usable capacity point versus the number of processors for two values of overheads under low blocking workload. There are fewer collisions occurring in the high rollback overhead system than in the low overhead system (see Figure 4.13). With the 3 PE system, the difference in the number of collisions between systems with difference overheads is small. When the number of memory contention increases with the increase of system size, the difference between the systems is also increased. Transactions in the low overhead system experience more rollbacks than the transactions in the high overhead system.

Figure 4.13 Comparison of collision rates under low and high rollback overheads (low blocking level workload)

Figure 4.14 Comparison of rollback wasted power under low and high rollback overheads (low blocking level workload)

Figure 4.14 compares the rollback wasted power of these two systems. Although the high rollback overhead system reduces the collision rates of the system, it consumes more time in rollbacks than the low rollback overhead system, and gives rise to a smaller throughput at the usable capacity point. The difference in performance among the two overhead systems is also increased with the increase of number of processors.

Figure 4.15 presents the useful processing powers of the multiprocessor system running with a specific workload for low and high rollback times. Figure 4.15a corresponds to the low blocking level workload, whereas, Figures 4.15b corresponds to the high blocking level workload.

At low blocking level, on a the 3 PE system, the useful processing power for high rollback overhead is 2.68% less than that achieved with low overhead. When increasing system size by adding processors, the memory contention is increased and the system with high rollback overhead produces 8.68% less than that achieved with low rollback overhead. The difference in useful processing power with two rollback overheads is increased with the increase of system size.

At high blocking level, when the useful processing power of the low rollback overhead system gets to its plateau at 5 PE, the useful processing power for the high rollback overhead system can still increase. As described in Section 4.3, in the low rollback overhead system, when the collision rate is higher than message arrival rate, the useful processing power does not increase with the increase in system size. In a high rollback overhead system, its collision rate is still lower than the message arrival rate (see Table 4.3). Figure 4.16 compares the total processing power for the low

Figure 4.15a Comparison of useful processing power with low and high rollback overheads (low blocking workload)

Figure 4.15b Comparison of useful processing power with low and high rollback overhead (high blocking workload)

and high rollback overhead system for 5 PE's working under a high blocking level workload. The high rollback overhead system uses more processing power to do work and spends less idle time for message latency control in comparison to the low rollback overhead system. So, its useful processing power can still increase with the increase in system size. The system has a better scalability with the high rollback overhead than with the low overhead when memory contention gets to the high level.

Figure 4.16 Comparison of total processing powers under low and high rollback overheads (high blocking level workload)

Table 4.3 Comparison of the collision rate to the arrival rate, when system is working with low and high rollback overheads (high blocking level workload)

	Number of Processors			
	3	5	7	9
Low rollback overhead				
Arrival rate	113.281	139.844	145.703	143.75
Collision rate	42.8429	143.605	275.481	362.279
High rollback overhead				
Arrival rate	96.9481	121.094	132.813	139.844
Collision rate	24.683	63.5258	106.834	153.66

4.5 Discussion

System working time is composed of transactions processing time and the rollback wasted time. The greater the numbers of transactions that are executed in parallel, higher is the number of collisions that occur on the system. This can have a detrimental effect on performance.

Blocking level reflects the memory sharing characterization of the workload. Higher blocking levels cause a large number of collisions and move the system saturation point to a smaller load while giving rise to a lower throughput.

When a system with a large number of PE's works with a high blocking workload, a part of time needs to be used in message latency control. This length of idle time increases with the number of processors.

Adding processors to the system will not always increase the system throughput especially with a high blocking workload. The scalability of the system is observed to deteriorate with a high blocking workload.

The collision rate in a high rollback overhead system is less than that in a low rollback overhead system, when the systems are identical in terms of other parameters. However, because of the high overhead time, the high rollback overhead system spends more power in rollback than that in the low rollback overhead system.

Chapter 5 Performance Comparison of Rollback Strategies

Rollback processing deals with the memory contention while transactions run concurrently on the system. It degrades the system performance because the work done by the rolled back transaction is wasted and it restarts at a future point in time. This chapter is divided into two parts: experiments on single class systems and experiments on two-class systems. The single class system experiment section describes the system behavior with different rollback strategies, when a single class of transactions runs on the system. Using a single class system allows us to analyze the impact of the rollback strategies given on performance in the absence of interference from Non-Payload processes. The effect of running Non-Payload and Payload simultaneously on the system is captured on the second part.

Note that an improvement produced by a rollback strategy in useful processing power by even few percent is important. Nortel spends approximately \$55,000 in terms of manpower for producing 1% improvement in capacity [Carr99].

5.1 Experiments on Single Class Systems

Four experiments, which use various combinations of the number of processors, rollback overhead and workload blocking level, are conducted. The rollback strategies studied

here are Bounded Priority with highest priority 2 (BP02) and 5 (BP05), Time Stamped (TS), Age based (AG), and FCFS.

5.1.1 Low blocking level workload and low rollback overhead

With a low blocking level workload, each transaction has a few shared memory requests, and the memory contention between two parallel transactions is small. The total processing power at the usable capacity point of systems with five different rollback strategies is shown in Figure 5.1. As described in Section 5.5, the usable capacity is the maximum useful processing power that can guarantee the average response time to lie within an interval 1617 ± 231.111 ms. A graph plotting the useful processing power of

Figure 5.1 Comparison of total processing power for rollback strategies (low blocking level workload, low rollback overhead)

these rollback strategies versus the number of processors is shown in Figure 5.2. With low blocking level, the difference in throughput and useful processing power is as small as 0.5%. With the increase in the number of processors, the difference tends to increase. At 5 PE, TS has 1% more useful processing power compared to FCFS. Because the system throughput has a direct relation with useful processing power, the throughput

Figure 5.2 Comparison of useful processing power for rollback strategies (low blocking level workload, low rollback overhead)

of TS system is also 1% higher than that of FCFS. TS shows a higher useful processing power consistently as the number of processors increases from 3 to 5 to 7 PE. With 9 PE, TS can produce 3% more useful processing power compared to FCFS because it wastes a smaller amount of work due to rollback. BP02 follows TS in performance with a 2.8% higher useful processing power than FCFS for the 9 PE system. AG and BP05 shows a

close performance and the useful processing power obtained with them are 1.56% higher than FCFS.

Only one highest priority transaction can run at a time in the system (see Section 3.2) in the BP rollback strategy. The highest priority is the highest priority allowed by the strategy. In BP02, the highest priority is priority 2, and in BP05, it is priority 5. Thus, when there is one highest priority transaction running in the system and all of the messages in the waiting queue are highest priority transactions, the spare processors must idle. We name this kind of waiting as serialization idle. Table 5.1 shows three measured factors for different number of processors. The first one is the percentage of the highest priority transactions in the waiting queue. The second one is the percentage of committed transactions that rolled back the highest times allowed by a BP strategy, (twice for BP02, and five times for BP05). The last one is the serialization idle time as discussed earlier. The higher these values, the larger the power wasted in idle waiting. As shown in the table the low blocking workload produces a small proportion of highest priority (Priority 2/Priority 5) transactions. As a result, a negligibly small serialization idle time is observed.

**Table 5.1 Comparison of parameters for BP02 and BP05
(low blocking level workload, low rollback overhead)**

Number of Processors	3	5	7	9
BP02				
Highest priority transaction in waiting queue(%)	0.5	0.61	0.57	0.96
Highest priority transactions that committed(%)	0.15	0.7	1.68	3.14
Serialization idle (power)	0	0	0.0003	0.0029
BP05				
Highest priority transaction in waiting queue(%)	0.13	0.24	0.24	0.2
Highest priority transactions that committed (%)	0	0.01	0.07	0.32
Serialization idle (power)	0	0	0	0

Figure 5.3 shows the collision rates versus number of processors under five rollback strategies. Collision rate increases with the increase in number of processors. With a 9 PE system, the collision rate of TS is as high as 54 per second. The differences in collision rates achieved with different strategies are observed to increase with an increase in the number of processors. When the system is controlled by the TS strategy, which chooses the rollback victims according to their arrival times, it has a small number of collisions happening per second. As shown in Figure 5.3 BP02 gives rise to a smaller collision rate in comparison to TS. However, the serialization idle incurred by the strategy seems to offset the advantages of the lower collision rate and the useful capacity achieved with TS is observed to be higher than that achieved with BP02. Thus, TS has the highest useful processing power at the useable capacity point (see Figure 5.2).

Figure 5.3 Comparison of collision rates for rollback strategies (low blocking level workload, and low rollback overhead)

AG, BP02 and BG05 produce comparable collision rates which are higher than that of TS. FCFS demonstrates the highest collision rate. The differences in useful processing powers achieved by the different policies are small, however, for this low blocking level workload (see Figure 5.2).

5.1.2 High blocking level workload and low rollback cost

In this experiment, the system works under the high blocking level workload. The memory contentions among transactions are extremely heavy. System total processing powers for different rollback strategies are shown in Figure 5.4. Unlike systems with low blocking level workload, they reach the usable capacity point at different power levels. That is they exhibit different system processing powers for attaining the given response time interval. With 3 PE's, the usable capacity point is reached at full processing capacity processing power. Full capacity processing power is the maximum processing power a system can have. For a system with N processors, the full capacity processing power is N (as described in Section 3.6). With 5 PE's, the system total processing power for BP02 saturates at 4.2 power. This means, if the system load is increased by increasing the message's arrival rate, more and more transactions are accumulated in the queue because of the large amount of collisions and rollbacks, and the mean response time exceeds the specified limit. Thus, with BP02 strategy, the 5 PE system only can work at 85% of its total power for guaranteeing message latency to lie in the required interval. With 9 PE's, 53% of total power is wasted in BP02 system. Although less in magnitude, such a power wastage is also observed for other policies. FCFS shows it cannot work at full capacity with 7 PE's, and it can only run at about 90% of its full capacity processing power at 9

Figure 5.4 Comparison of total processing powers for rollback strategies (high blocking level workload, low rollback overhead)

Figure 5.5 Comparison of useful processing power for rollback strategies (high blocking level workload, low rollback overhead)

PE's. BP05 drops its processing power to 81% of the full processing capacity at 9 PE's. TS and AG achieves 97.33% of full capacity processing power on a 9 PE system.

Figure 5.5 shows the useful processing power of the system, and Figure 5.6 shows the wasted rollback power. TS rollback strategy shows the best results with the highest useful processing power. Larger number of processors increases the difference in throughputs achieved by TS and FCFS. With 3 PE's, TS produces 2.5% more useful processing power than FCFS.

Figure 5.6 Comparison of rollback wasted power for rollback strategies (high blocking level workload, low rollback overhead)

With 9 PE's, the difference increases to 17.4%, and AG follows TS closely. BP05 produces 10.3% more useful processing power than FCFS. BP02 produces 2.86% power less than FCFS. At higher number of processors, BP02 provides the processing power that is 7.8% lower than that achieved with FCFS on a 7 PE system.

There are two factors that slow down the increase in useful processing power as the system size increases. The first is the large number of collisions causing a large number of rollbacks; thus, more power is wasted for dealing with collisions in larger systems. This reduces the increase in useful processing power that accompanies the adding of processors to the system. In addition to FCFS, which is already discussed in Section 5.2, this effect is observed in TS, AG strategies as well. Although with BP, the rollback wasted power is constrained to a certain level, a large amounts of BP idle waiting time occurs. This is the second factor that slows down the increase in useful processing power when more processors are added to the system.

Although the power wasted by BP02 due to serialization is small at smaller number of processors, it increases to a large value for 9 PE's (see Table 5.2). As shown in the last column, 99.72% of the waiting transactions have the highest priority in BP02 at 9 PE's. This indicates that most of the transactions in the queue have rolled back twice. Thus for a large proportion of time, only highest priority transactions are ready to run on the system. Since only one highest transaction can run on the system at a time, the other processors remain idle. As a result a high serialization overhead of 4.6162 is achieved at 9 PE's.

By increasing the priority number to 5, the proportion of highest priority transaction in the waiting queue is reduced to 97.77%. A substantial decrease in serialization idle is observed: at 9 PE's, a serialization idle of 1.6131 is achieved.

**Table 5.2 Comparison of parameters for BP02 and BP05
(high blocking level workload, low rollback overhead)**

Number of Processors	3	5	7	9
BP02				
Highest priority transaction in waiting queue (%)	0.62	99.65	99.86	99.72
Highest priority transactions that committed (%)	8.37	25.25	25.73	25.52
Serialization idle (power)	0	0.757	2.7881	4.6162
BP05				
Highest priority transaction in waiting queue (%)	0.52	0.41	2.59	97.77
Highest priority transactions that committed (%)	0.1	3.15	14.58	18.1
Serialization idle (power)	0	0	0.0053	1.6131

As shown in Figure 5.7, at higher number PE's, the collision rate produced by FCFS, TS, and AG are higher than those produced by BP02 and BP05. The collision rates produced by the first three tend to increase with an increase in the number of PE's where as the collision rates produced by BP02 and BP05 saturate at 5 and 7 PE's respectively. A lower amount of wasted power is thus produced by these two bounded priority policies in comparison to the others (see Figure 5.5). FCFS, TS and AG however are free from serialization idle which is significant in the case of the bounded priority policies. The useful processing power produced by each of these policies is affected by both of these factors. TS and AG are observed to produce a larger useful processing power in comparison to the bounded priority policies. Although FCFS performs better than BP02, its performance is inference to that of BP05 (see Figure 5.5).

Figure 5.7 Comparison of collision rates for rollback strategies (high blocking level workload, and low rollback overhead)

In a high blocking level workload with low rollback overhead, the useful processing power of the five rollback strategies drop significantly, in comparison to a low blocking level workload experiment. BP strategies show a plateau in useful processing power and throughput because of the serialization idle time. The highest useful processing power produced by BP02 is 2.7, and by BP05, is 3.3. BP05 can reach a higher throughput by wasting less time in serialization idle (see Table 5.2). FCFS shows a plateau at 3.0. With 9 PE's, TS and AG, produced useful processing powers that are only 45.1% of that achieved with the low blocking level workload, and they are not so scalable as on a system with a light workload level (see Figure 5.2 and Figure 5.5). The rollback policies exhibit a lower scalability for the high blocking level workload.

5.1.3 Low blocking level workload and high rollback overhead

Figure 5.8 shows the total processing powers at the usable capacity point. As in the case of the results displayed in Figure 5.1, comparable system performance metrics are achieved with these five rollback strategies. Rollback overhead is one of the factors affecting system performance. As expected, a high rollback overhead system produces a lower useful processing power than a low rollback overhead system. Figure 5.9 shows the useful processing power at usable capacity point. The shape of the figure is very close to that captured in Figure 5.2.

Figure 5.8 Comparison of total processing powers for rollback strategies (low blocking level workload, and high rollback overhead)

In a 3 PE system, the differences among the useful processing power achieved with different rollback strategies is very small. With 5 PE's, TS shows the best result, with a 0.7% higher useful processing power in comparison to FCFS. At 9 PE, TS displays a processing power that is 2.3% higher than that produced by FCFS. BP02 follows TS with

useful processing power that is 2.04% higher than that achieved with FCFS. AG and BP05 demonstrate a useful processing power that is higher than that achieved with FCFS.

Figure 5.9 Comparison of useful processing powers for rollback strategies (low blocking level workload, and high rollback overhead)

BP02 has the useful processing power as high as TS on a 9 PE system with a small rollback wasted processing power. BP05 has the producing priority bound set at a higher level, and shows a lower useful processing power than BP02 by more number of rollbacks and rollback wasted power (see Figure 5.10). As shown in Table 5.3, the serialization idle time becomes smaller and the proportion of committed highest priority transactions is less, compared to those in Table 5.1. Thus, the system wastes less time in serialization idle waiting in comparison to the system with a low rollback overhead.

**Table 5.3 Comparison of parameters for BP02 and BP05
(low blocking level workload, high rollback overhead)**

Number of processors	3	5	7	9
BP02				
Highest priority transaction in waiting queue (%)	0.52	0.52	0.51	0.34
Highest priority transactions that committed (%)	0.1	0.55	1.36	2.46
Serialization idle (power)	0	0	0	0.0005
BP05				
Highest priority transaction in waiting queue (%)	0	0.22	0.24	0.17
Highest priority transactions that committed (%)	0	0	0.02	0.11
Serialization idle (power)	0	0	0	0

AG produces more throughputs than FCFS. Its throughput is 1.3% smaller than that produced by TS (see Figure 5.9). In AG strategy, a transaction with a priority 1 message wins over a transaction with a priority 0 message, if they collide. When the priorities of the messages are the same, it follows the same rule used by TS. At low blocking level, the yielding of transactions with messages which have not been rolled back before, to the transactions, which have been rolled back before, reduces the system throughput.

**Figure 5.10 Comparison of collision rates for rollback strategies
(low blocking level workload, and high rollback overhead)**

Figure 5.10 shows the collision rates versus the number of processors. When the number of processors is as low as 3 or 5, the numbers of collisions occurring per second for the different rollback strategies are very close to one another. The highest collision rate is demonstrated by FCFS in 7 PE and 9 PE systems. FCFS wastes the largest amount of time in dealing with collisions. Although BP02 has the least collision rate, its useful processing capacity is less than TS because of the wastage of processing time in idle waiting. However by increasing the highest priority to 5, the collision rate of the bounded priority policy becomes close to that produced by AG. BP05 exhibits a smaller serialization idle time in comparison to BP02. The useful processing power for BP05 is observed to be close to that of AG (see Figure 5.9).

5.1.4 High blocking level workload and high rollback overhead

High blocking level workload makes the system waste more time on rollback processing. With a long rollback overhead, degradation of system performance becomes even more serious. Figure 5.11 shows the system processing power for different number of processors at the usable capacity point. BP02 reaches a plateau at 7 PE's with a 5.5 power, which is only 92.4% of the full capacity processing power on a 7 PE system. The total processing power of the system cannot increase with system size. With 9 PE's, a system using BP02 can only produce a useful work that is at 72% of the system full capacity processing power. Other strategies can achieve approximately 98% of the full system capacity.

The useful processing power is shown in Figure 5.12. TS shows the best result with the highest useful processing power and throughput. With 3 PE's, the useful processing powers of these five strategies are very close to one another. With 5 and 7 PE's, TS and FCFS systems show better results than other policy. When a collision occurs, TS gives a higher priority to the transactions with the earlier arriving times, and FCFS gives higher priority to transactions which has locked the blocking resource first. The closeness of these results indicates that, in many situations, a transaction with earlier arriving times locks the shared resources earlier, so that it wins on both the TS and FCFS systems. With 9 PE's, TS demonstrates the highest useful processing power, that is 2.4% higher than that of FCFS.

Figure 5.11 Comparison of total processing power for rollback strategies (high blocking level workload, high rollback overhead)

Figure 5.12 Comparison of useful processing power for rollback strategies (high blocking level workload, high rollback overhead)

Table 5.4 shows a number of metrics that are important to BP02 and BP05. In comparison to a system with low rollback overhead (see Table 5.2), the serialization idle incurred by BP02 and BP05 are smaller for any given number of processors. This is because of the lower collision rates produced by the policies for the system with a high blocking rollback overhead.

Table 5.4 Comparison of parameters for BP02 and BP05 (high blocking level workload, high rollback overhead)

Number of Processors	3	5	7	9
BP02				
Highest priority transaction in waiting queue (%)	0.88	1.6	98	99.85
Highest priority transactions that committed (%)	3.35	14.86	25.71	25.07
Serialization idle (power)	0	0.0013	0.5401	2.5219
BP05				
Highest priority transaction in waiting queue (%)	0.75	0.48	0.32	0.62
Highest priority transactions that committed (%)	0	0.16	1.25	3.92
Serialization idle (power)	0	0	0	0.0004

Figure 5.13 Comparison of collision rates for different rollback strategies (high blocking level workload, and high rollback overhead)

As Figure 5.13 shows, the collision rate in the BP02 system stops increasing at 7 PE. The serialization idle does not seem to significantly affect the collision rate for BP05 that produces a comparable performance with the other policies.

5.1.5 Discussion of the results of experiments on single class systems

Rollback preserves data integrity by expending processing power. At light blocking level, the difference in system useful processing power produced by the different rollback strategies is very small. As well, the system is scalable: system throughput increases with increase in number of processors. TS shows the best results with the highest useful processing power. BP02 follows it with a limited number of rollbacks and collisions, and achieves the same value in useful processing power at 9 PE's. BP05 makes transactions rollback more to reach the priority bound, and it shows close results with AG. FCFS is the worst with the largest wasted time in rollbacks.

rollback more to reach the priority bound, and it shows close results with AG. FCFS is the worst with the largest wasted time in rollbacks.

At heavy blocking level, when the system size is as small as 3 PE, the difference in useful processing power from different rollback strategies is small too. With the increase in system size, TS and AG shows highest useful processing power and throughputs. Systems with these two rollback strategies waste less time in latency control and rollback. FCFS wastes the largest amount of time in rollback, so it produces the least throughputs. For controlling the message latency, a part of time is wasted due to idling. If the system is too busy, large number of collisions makes the queue length for waiting message, and the message's mean response time cannot meet the specification.

BP05 is superior to BP02 on systems with a high blocking level workload and the number of PE's greater than or equal to 5. Serialization idle is observed to be small on systems with low memory contention as well as on systems with high memory contention and a small number of processors. In these situations, BP02 performs better than BP05 which allows a transaction to roll back more number of times in comparison to it.

In high rollback overhead system, the differences in the produced useful processing power by different rollback strategies become less than in those observed on a low rollback overhead system.

5.2 Experiments on Multi-class System

Hundreds of processes run on a real telephone switch. As discussed in Section 2.2, these processes can be divided into two major categories: Payload and Non-Payload. A scheduler assigns CPU shares among these processes. CPU time for Payload needs to be guaranteed and its processes are assigned 70% of CPU time. Maintenance and Background processes gain the remaining 30% of CPU time. A two-class system running Payload and Non-Payload processes is discussed in this section. As mentioned in Section 3.7, a limited number of experiments are carried out for investigating the two-class system.

A Rollback expends processing time to resolve memory contention. Two ways of charging rollbacks are implemented in this simulator: charging it to class, and charging it to system [Carr00]. In rollback charged to class, the wasted time due to rollback for each class is deducted from its designated CPU share. In the implementation of rollback charged to class, a process is scheduled out when its CPU share is used up or no new message from that class is present in the message queue [Verm00]. Using a template the scheduler runs the Payload and Non-Payload processes in cycles. In each cycle, 70% of the CPU time is given to Payload and 30% to Non-Payload. In a charge to class system, the classes are isolated by scheduler share. Thus the amount of processing time wasted due to Non-Payload rollbacks will not affect the CPU time given to Payload.

When a charge to system strategy is used, the system absorbs the wasted rollback time of each class. Thus, the proportion of useful processing time given to Payload is 70% and that given to Non-Payload is 30%. As in the case of charge to class, a process is swapped

out when its scheduler share is exhausted. But, the time wasted due to rollback is added back to its share in the next scheduler cycle [Klap00].

Experiments are conducted on a two-class system. Processes of two classes with identical (low or high blocking level) workload characteristics are run on the system. Both rollback charged to system and charged to class are investigated. Effect of low and high rollback overhead are also analyzed.

A Reverse Share (RS) rollback strategy is proposed by Nortel [Buda00] for a multi-class system. It intends to assign more processing power to Payload by rolling back Non-Payload transactions more often. The collisions in the system belong to two categories, intra-class and inter-class. Intra-class collisions occur between two transactions of the same class. Inter-class collisions occur between two transactions of different classes. As the name Reverse Share indicates, when inter-class collisions occur, the chance of a transaction being rolled back is related to the reverse of its processing share in the scheduler template. The algorithm attempts to produce a ratio of number of rollbacks between Payload class and Non-Payload class caused by the inter-class collisions that is equal to the reverse of the ratio of processing shares of their classes in the scheduler template. That is if the Payload and Non-Payload shares in the scheduler template are 70% and 30% respectively, for inter-class collisions, we expect to observe a payload transaction rollback 30% of the time and a Non-Payload transaction to rollback 70% of the time. Wu [Wu00b] compares RS with BP02 in rollback charged to class through an analytic model.

In the Reverse Share strategy, each class is awarded a priority. The priority of Payload is 70, and Non-Payload is 30. A transaction in its first run inherits this priority as its initial priority. When inter-class collision occurs, their priorities are compared. The higher priority transaction wins, and the lower priority transaction is rolled back with its priority increased by the value of its initial priority [Buda00]. Because Non-Payload transaction starts with priority 30, it wins in an inter-class collision after rolling back twice. At that time, its priority is increased to 90, which is higher than the Payload initial priority. If the priorities of the contending transactions are the same, the transaction which has locked the memory line first wins. The loser is rolled back with an increment in priority equal to its initial value. When a collision occurs between transactions in the same classes, the higher priority transaction wins. If their priorities are the same, the transaction which locked the memory line first wins. However after an intra-class collision, the loser's priority is increased by 1 only. A high level algorithm of the strategy is presented.

class initial priority:

payload_init: 70

nonpayload_init: 30

when a transaction runs first time, its priority= class initial priority;

If collision happens between same class transactions

if priorities are same

transaction that locked memory line first wins;

increment the loser's priority by 1.

else

higher priority wins

```

        lower priority transaction loses and its priority is increased by 1
    else
        if Payload transaction's priority > Non-Payload transaction's priority
            rollback Non-Payload transaction;
            Non-Payload transaction's priority += Non-Payload_init
        else
            if Payload transaction's priority < Non-Payload transaction's priority
                rollback Payload transaction;
                Payload transaction's priority += payload_init
            else { priorities are the same }
                transaction that locked memory line first wins;
                increment the loser's priority, by the corresponding class initial
                priority;
            endif
        end if
    end if
end if

```

In order to investigate the system in which 30% of the processing power is divested to Non-Payload processes, we first choose a Non-Payload arrival rate that utilizes 30% of the CPU time. The Payload useful processing power is the measured at the useable capacity point [Carr00], when the Payload message mean response time is located in the specific interval.

5.2.1 Low blocking level workload and low rollback overhead

Figure 5.14 shows the useful processing power and rollback wasted power for each class when the system is working at the usable capacity point, whereas rollback is charged to class. The total processing power for TS at the usable capacity point is 96.64% of the maximum capacity. For the other rollback strategies, the total processing power is 99% of the maximum capacity.

Figure 5.14 Comparison of processing powers for rollback strategies (7 processors, low blocking level workload, low rollback overhead, and rollback is charged to class)

Figure 5.15 shows the Payload processing power for each of the rollback strategies. BP02 has the highest Payload useful processing power. TS has the lowest useful processing power and the highest rollback wasted power. The performance of AG, FCFS, and BP05 are very close to one another. RS doesn't show any advantage, although its Payload

rollback wasted time is saved by rolling back a greater proportions of Non-Payload transactions.

Figure 5.15 Comparison of Payload processing powers for rollback strategies (7 processors, low blocking level workload, low rollback overhead, rollback is charged to class)

Table 5.5 shows the Payload share assigned from the total processing power and its share in useful processing power. The value for BP02, BP05, FCFS, and AG show that their assigned processing time is close to 70% of total processing time. They also give rise to approximately 70% of the total useful processing power for the system.

Table 5.5 Comparison of Payload share in total processing power with that in useful processing power (7 processors, low blocking level workload, low rollback overhead, rollback is charged to class)

Rollback strategy	BP02	RS	TS	AG	FCFS	BP05
Payload Share in Total Processing Power(%)	69.6261	66.93942	69.28796	69.6601	69.42029	69.41595
Payload Share in Useful Processing Power(%)	69.6199	68.99629	67.57684	69.31903	69.44281	69.44767

RS produces a greater portion of Payload share in useful processing power than its share in total processing time. It is caused by the rollback of more Non-Payload transactions. Although it gives more chance of commit for Payload transactions, the payload class gains smaller running time share in comparison to system with: BP02, BP05, AG and FCFS. The small portion of the share in running time means that Payload processes are swapped out for the lack of messages in queue before the Payload share is finished. The increased portion of the share in useful processing power doesn't compensate the loss of the share in the total processing power completely.

TS gains a smaller portion in useful processing power than in the total processing power. It indicates that Payload class bears more portions of rollbacks than its share in the collision with Non-Payload. Non-Payload residence time in its queue is expected to be larger than that of Payload. So, Non-Payload transactions have more chance to win in the inter-class collision.

Figure 5.16 shows the system processing power when rollback is charged to system. The shape of it is similar to that of Figure 5.14. Figure 5.17 compares the Payload useful processing power achieved with rollback charged to system and class. The figure

Figure 5.16 Comparison of processing powers for rollback strategies (7 processors, low blocking level workload, low rollback overhead, and rollback is charged to system)

Figure 5.17 Comparison of Payload useful processing powers for rollback charged to class and charged to system (7 processors, low blocking level workload, low rollback overhead)

indicates that when both Payload and Non-Payload classes are characterized by the same low blocking characterization, both charge to class and charge to system give rise to similar performance for the same rollback strategy.

5.2.2 Low blocking level workload and high rollback overhead

From Section 5.3 and 5.4 we know that the collision rate in high rollback overhead is less than that in low rollback overhead, and the rollback wasted power is increased in high rollback overhead. From Section 5.1.3, and 5.1.4, the difference among rollback strategies in high rollback overhead is less than that in low rollback overhead.

Figure 5.18 Comparison of processing power for rollback strategies (7 processors, low blocking level workload, high rollback overhead, and rollback is charged to class)

Figure 5.18 compares the processing power among rollback strategies, when rollback is charged to class. The relative performance of the strategies is similar to that captured in Figure 5.14, which corresponds to the low blocking level workload and low rollback

overhead. The Payload useful processing power achieved by each strategy is somewhat lower than those achieved with a low rollback overhead (see Figure 5.14 and Figure 5.18). The differences in useful processing power among rollback strategies are less than that in captured in Figure 5.14.

Table 5.6 presents the share of the processing powers consumed by Payload for each rollback strategy. With a low blocking level workload, the Payload shares are observed to be closed to the shares stored in the scheduler template (70%). The Payload shares achieved with a high rollback overhead is a little smaller than those achieved with a low rollback overhead (see Table 5.6).

Table 5.6 Comparison of Payload share in total processing power with that in useful processing power (7 processors, low blocking level workload, high rollback overhead, rollback is charged to class)

Rollback strategy	BP02	RS	TS	AG	FCFS	BP05
Payload Share in Total Processing Power(%)	68.1108	64.117	68.2337	68.2948	68.2948	68.2948
Payload Share in Useful Processing Power(%)	68.1285	67.4011	65.7539	67.7681	67.7681	67.7681

Figure 5.19 compares the processing powers, for rollback strategies when rollback is charged to system. The relative performances of the strategies are the same as captured in Figure 5.19. Figure 5.22 compares the useful processing power between rollback charged to class and charged to system. It shows that, for a low blocking level workload and high rollback overhead, the differences in useful processing power achieved with the two types of charging schemes are very small for most rollback strategies.

Figure 5.19 Comparison of processing powers for rollback strategies (7 processors, low blocking level workload, high rollback overhead, and rollback is charged to system)

Figure 5.20 Comparison of Payload useful processing power between rollback charged to class and charged to system (7 processors, low blocking level workload, low rollback overhead)

5.2.3 High blocking level workload and low rollback overhead

A preliminary investigation with a high blocking workload and low rollback overhead was conducted. The results indicate that TS and AG become unstable even before reaching the usable capacity point. The processing shares achieved by Payload and Non-Payload does not agree with the specified shares contained in the scheduler template. Further investigation of this forms an important direction for future research.

5.2.4 Discussion on Two-Class System Experiment

In two-class system, the experiments on BP02, BP05, TS, RS, FCFS are undertaken with rollback charged to class and charged to system under the cases: low blocking level workload and low rollback overhead, high blocking level workload and low rollback overhead, low blocking level workload and high rollback overhead.

In low blocking level workload, TS produces the lowest useful processing power, and its share in useful processing power does not meet the share in CPU running time contained in the scheduler template. BP02 produces the highest useful processing power, and its share in useful processing power meets the share in CPU running time contained in the scheduler template. RS produces less useful processing power than BP02 but higher than others. FCFS, AG and BP05 produce comparable useful processing powers. For a given rollback strategy the useful processing power achieved with rollback charged to class and system are comparable.

A preliminary investigation with high blocking level workload indicates that TS and AG become unstable before reaching the usable capacity point. The scheduler shares for Payload and Non-Payload are observed to be broken. Further analysis is required.

In the high rollback overhead system, the useful processing power is less than that in the low rollback overhead system. The differences in Payload useful processing powers achieved by the strategies are observed to be small for a high rollback overhead. The Payload share in total processing power and useful processing power are smaller than those achieved in the low rollback overhead system.

Chapter 6 Conclusions

6.1 Research Summary

This thesis describes a trace driven simulation-based investigation of the rollback strategies for a multiprocessor-based telephone switch system under different workloads, and rollback overheads. Contentions for memory lines are resolved by a lock-rollback mechanism. A memory line is locked by a transaction that is using it. When another transaction requests for a locked memory line, one of the transactions is rolled back. The rolled back transaction is restarted at future point in time. Currently, there is little work that focuses on the performance of such rollback and restart methods. This research project analyzes five rollback strategies: First Come First Served, Time Stamped, Age based, Bounded Priority, and Reverse Share. Both single class and two-class systems are investigated. In the two-class system, the impact of interaction between Non-Payload and Payload processes on the performance of different rollback strategies is captured.

System load is controlled by adjusting the arrival rate of messages coming into the system. System load affects the mean number of collisions in the system, and a heavy system load causes more collisions, and affects the useful processing power.

The number of processors determines the maximum number of transactions running concurrently on the system. A high number of processors in the system causes more

chance of collisions. As a result sometimes, useful processing power cannot match the increase in the number of processors in the system.

System performance is analyzed for both high and low blocking level workloads. In a low blocking level workload, increasing the number of processors significantly improves useful processing power that in turn leads to an increase in system throughput. In a heavy blocking level workload, the system is not easily scalable. The increase in the number of processors does not necessarily improve the useful processing power for the system. As shown in Section 3.2, guaranteeing message latency sometimes leads to an increase in the processor idle time.

Rollback overhead affects system performance in two different ways. A high rollback overhead is observed to reduce the transactions' collision rate on the system. Although the system gives rise to a lower number of rollbacks in comparison to a system with a low rollback overhead, overall processing power wasted on the system with a high rollback overhead is larger. As a result, the system with a high rollback overhead tends to produce a lower useful processing power in comparing to a system with a lower rollback overhead.

6.1.1 Single class system

The results of experiments on a single class system show that TS produces the highest useful processing power. The performance differences among the policies are small for a light blocking workload. The superiority of TS is more pronounced with the heavy blocking workload.

BP02 attempts to reduce the number of rollbacks by setting a bound of two on the maximum number of rollbacks experienced by a transaction. Although BP02 performs close to that of TS for a low blocking level workload, its performance is observed to deteriorate for a high blocking level workload due to larger periods of serialization idle occurring on the system. This leads to a poor scalability for BP02: the useful processing power is observed to saturate at 5 PE (see Figure 6.5). Raising the bound on the number of rollbacks to 5 seems to improve performance, and BP05 exhibits a higher useful processing power in comparison to BP02.

The performance of AG, FCFS, and BP05 are comparable with a low blocking level workload. AG is observed to produce a performance comparable to that of TS with a high blocking level workload.

In high rollback overhead system, the differences in performances among different rollback strategies are smaller than those achieved in low rollback overhead system.

6.1.2 Two-class system

Most of the observations on a two-class system were made for a low blocking level workload and are summarized. On two-class systems, TS becomes the worst strategy. The system assigns 69% of its total running time to Payload processes, but, only 67% of the total useful processing power is consumed by Payload. The Non-Payload class has a smaller CPU share than Payload class does. The residence time of Non-Payload messages in the message waiting queue is longer than that of Payload messages. So, when a collision occurs between Payload and Non-Payload transactions, the Payload transaction has a higher chance of being rolled back. It indicates that, on a multi-class system, TS is

unable to achieve the engineered ratio between Payload and Non-Payload processing powers.

BP02 is observed to produce the best results in a two-class experiment with low blocking level workload. Payload class achieves the share in useful processing power equal to the share in the scheduler template. It indicates that BP02 produces effective throughput by the limiting the number of rollbacks.

The Reverse Share strategy is also observed to be unable to achieve the designated processing share for the Payload processes. AG, FCFS and BP05 strategies show comparable results and nearly achieve the designated processing power shares for the two classes. Their Payload throughput shares are close to their Payload running time shares in the scheduler template. Their throughput values are very close to that of BP02.

A limited experimentation with a high blocking level workload indicates that TS and AG become unstable even before the system reaches the useable capacity point. For the heavy blocking level workload, the system cannot achieve the designated shares of processing power for Payload and Non-Payload contained in the scheduler template.

We have experimented with two types of rollback charging mechanisms: charge to class and charge to system. With low blocking level workload and a given rollback strategy both these mechanisms seem to produce a comparable performance.

6.2 Thesis Contributions

This thesis is concerned with rollback strategies in a multiprocessor-based real time transaction oriented system. In co-operation with Nortel, this research focuses on a telephone switch call processing module and uses real system traces for performance investigation. A flexible C++ simulator based on Nortel's model is implemented. The rollback strategies studied are Time Stamped Rollback (TS), Age Based Rollback (AG), Bounded Priority Rollback (BP02 and BP05), FCFS Rollback (FCFS), and Reverse Share Rollback (RS).

Contribution to knowledge

A number of experiments are conducted on a single class system for various combination of rollback strategies, system size, rollback overheads, and contention characteristics of workload. A limited set of experiments on a two-class system is also conducted. A number of important insights into the related performance of the strategies is reported. A brief summary of these insights is provided in the previous sections. These insights and the relative performance of the rollback strategies in particular, have been useful to Nortel in building their multiprocessor-based telephone switch (see Appendix 2). The important contributions to knowledge are briefly summarized.

- Overall, two strategies TS and BP are observed to be attractive for different types of real systems. Systems processing a single workload class or two-class systems in which one workload class uses up most of the processing power, TS is expected to produce the best performance. On multiclass systems in which both classes consume a significant portion of the processing power, Bounded Priority policies are expected to be superior. The choice between BP02 and BP05 seems to be dependent on the

blocking level. BP02 is observed to be better on systems with a low blocking level whereas BP05 is attractive for systems processing a workload characterized by a high blocking level.

- Although a high rollback overhead tends to decrease the collision rate on the system, the useful capacity produced by a given rollback policy is observed to deteriorate with an increase in rollback overhead.
- Previous work on telephone switches has focused on evaluation of system performance for a 95% processor utilization. Bounded Priority policies are observed to produced the best useful processing power at such utilization on single class systems. As explained in the first item of the list, a different observation is made in this thesis by evaluating the policies at the usable capacity point.

6.3 Suggestions for Future Research

Due to constraints on time, only two types of blocking levels were experimented. Analyzing the performance of the rollback strategies at medium blocking levels can improve our insights into system performance. Only a limited set of experiments on two-class system is reported in this thesis. Further investigation of multi-class system is warranted. Experiments with new rollback strategies form an important direction for future research. A strategy that uses the duration of time used by a transaction for choosing the rollback victim looks promising. With such a strategy the CPU time used by two contending transactions are compared. The transaction that has accumulated a small CPU time rolls back. This will lead to a decrease in the wasted work due to rollbacks. The implementation of this policy requires the capability of the system to efficiently

record the CPU times used by the transactions. The overhead associated with the strategy and the overall performance achieved by it needs investigation. This research has focused on a telephone switch. Investigation of the rollback strategies described in this thesis in the context of real time database system seems interesting.

References

- [Agra87] R. Agrawal, M. J. Carey, M. Livny, "Concurrency Control Performance Modeling: Alternatives and Implications", *ACM Transactions on Database Systems*, Vol. 12, No. 4, December 1987, pages 609-645.
- [Bank99] J. Banks, J. S. Carson, B. L. Nelson, "Discrete-Event System Simulation (Second Edition)", Prentice Hall, 1999.
- [Beni98] B. Beninger, "DMS SuperNode XA-Core Guide to Multiprocessing", Nortel Networks, 1998.
- [Buda00] I. de Buda, Personal Communication, Nortel Networks, May 2000.
- [Carr99] B. Carroll, Personal Communication, Nortel Networks, 1999.
- [Carr00] B. Carroll, Personal Communication, Nortel Networks, 2000.
- [Cata94] B. J. Catancaro, E. Cliffs, "Multiprocessor System Architecture", Prentice Hall, 1994.
- [Fran92] P. A. Franaszek, J. T. Robinson, A. Thomasian, "Concurrency Control for High Contention Environments", *ACM Transactions on Database Systems*, Vol. 17, No. 2, June 1992, pages 304-345.
- [Hari91] J. R. Harista, "Earliest Deadline Scheduling for Real-Time Database Systems", *Proceedings of the 12th Real-Time System Symposium*, San Antonio, Texas, December, 1991, pages 232-242.
- [Hong93] D. Hong, et al., "Real-Time Transaction Scheduling: A Cost Conscious Approach", *ACM SIGMOD Record*, page 197-206, May 1993, pages 197-206.
- [Jain91] R. Jain, "The Art of Computer Systems Performance Analysis", John Wiley & Sons, INC., 1991.
- [Jian01] X. Y. Jiang, Nortel Networks, Personal Communication, 2001.
- [Keis95] B. Keiser, "Digital Telephony and Network Integration", N.Y., Van Nostrand Reinhold, 1995.
- [Klap00] N. Klappholz, "Rollback Charged to System and Charged to Class in SOS scheduler", Nortel Networks, 2000.
- [Kung81] H. Kung, J. Robinson, "On optimistic methods for concurrency control", *ACM Transactions on Database Systems*, Vol. 6, No. 2, June 1981, pages 213-226.

- [Kuo00] T. W. Kuo, H. C. Hsih, "Concurrency Control in a Multiprocessor Real-Time Database System", *Real-Time Systems, 2000, Euromicro RTS 2000, 12th Euromicro conference on*, pages 55-62.
- [Liu00] M. Liu, "Scheduling on a Scalable Multiprocessor-Based Telephone Switch", M.Sc. Thesis, Dept. of System and Computer Engineering, Carleton University, 2000.
- [Maju99a] S. Majumdar, "Scheduling Strategies for Controlling Resource Contention on Multiprocessor Systems", *Proceedings of the 6th International Conference on High Performance Computing (HiPC'99)*, Calcutta, India, 1999, pages 151-157.
- [Maju99b] S. Majumdar, D. Streibel, B. Beninger, B. Carroll, N. Verma, M. Liu, "Controlling memory contention on a scalable multiprocessor-based telephone switch", Extended Abstract: Industrial Experience, *Proceedings of ACM SIGMETRICS'99 Conference*, Atlanta, May 1999, pages 228-229.
- [Nort89] "SOS operating system Guide", Nortel Networks, 1989.
- [Nort95] "DMS Thin Architecture Guide", Nortel Networks, 1995.
- [Nort96] "PROPHET User Guide and Reference Manual", Release 2.1, Nortel Networks, 1998.
- [Rosc97] A. W. Roscoe, "The Theory and Practice of Concurrency", Prentice Hall, 1997.
- [Silb98] A. Silberschatz, P. B. Galvin, "Operating system concepts, fifth edition", Addison Wesley, 1998
- [Stan88] J. A. Stankovic, W. Zhao, "On Real-Time Transactions", *ACM SIGMOD Record*, March 1988, pages 4-18.
- [Stan96] J. A. Stankovic, "Real-Time and Embedded Systems", *ACM Computing Surveys*, Vol. 28, No. 1, March 1996, pages 205-208.
- [Stei99] D. Streibel, Nortel Networks, Personal Communication, 1999.
- [Takk98] S. Takkar, S. P. Dandanmudi, "Performance of Hard Real-Time Transaction Scheduling Policies in Parallel Database System", *Proceeding of IEEE MASCOTS, 98 conference, Montreal*, July 1998, page 176-184.
- [Thom98] A. Thomasian, "Concurrency Control: Methods, Performance, and Analysis", *ACM Computing Surveys*, Vol. 30, No. 1, March 1998, pages 70-119.
- [Ulso92] O. Ulusoy, G. G. Belford, "Concurrency Control in Real-Time Database Systems", *Proceedings of the 1992 ACM annual conference on Communications*, 1992, pages 181-188.

[Verm00] N. Verma, “Controlling Shared Memory Contention on a Multiprocessor-based Telephone Switch”, M.Eng. Thesis, Dept. of System and Computer Engineering, Carleton University, 2000.

[Wu00a] P. Wu, Nortel Networks, Personal Communication, March 2000.

[Wu00b] P. Wu, “Preliminary Results for Charging Rollbacks to Classes According to their Reversed Share”, Nortel Networks, March 2000.

Appendix 1

Tool for determining the usable capacity point

A tool to determine the usable capacity point is implemented. The pseudo code for the tool is presented here.

Step1: get message high_bound (arrival rate), low_bound (arrival rate), response time interval for usable capacity point from user;

Step2: current message arrival rate=(high_bound+low_bound)/2;

Step3: run simulator;

Step4: check the mean response time from output file

If response time lies above the specific interval

Then high_bound= current arrival rate;

Goto step 2;

Else

If response time lies below the specific interval

Then low_bound= current arrival rate;

Goto step 2;

Step 5: stop experiment and report data.

Appendix 2

Summary of contributions to Nortel Networks

[Email from Brian Carroll, Nortel Networks]

In a telephone switch, there are primary processes (associated with call processing), and secondary processes (associated with maintaining the switch). In a parallel processor, data integrity is maintained through the rollback mechanism that prevents two processes from changing the same address location. Rollback costs decrease capacity of a parallel processor. One way to increase capacity is to prevent rollbacks between secondary processes. This can be accomplished by serializing the scheduling of secondary processes. The tradeoff is the introduction of a serialization cost of waiting to run secondary processes. This thesis explored a number of rollback strategies. The goal was to find a rollback strategy that would allow secondary processes to run in parallel, while limiting secondary-secondary rollbacks sufficiently to gain capacity. The simulations of these strategies saved over 10 person-years of effort that would have been required to implement, tune, and characterize on a real switch, the capacity impacts of even one more of these strategies.

Note: Primary and secondary process refer to Payload and Non-Payload process respectively.