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.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

**Controlling Shared Memory Contention on a Multiprocessor-based
Telephone Switch**

By

Naveenta Verma, B.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of

Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada K1S 5B6

January 2000

© Copyright 2000, Naveenta Verma.

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.

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 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.

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.

0-612-48475-0

Canada

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

**Controlling Shared Memory Contention on a Multiprocessor-based
Telephone Switch**

Submitted by Naveenta Verma. B.Eng. (Computer Eng.)
in partial fulfillment of the requirements for the degree of
Master of Engineering

Chair, Department of Systems and Computer Engineering

Professor Shikharesh Majumdar, Thesis Supervisor

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
January 2000.

Abstract

This thesis is concerned with the performance of a scalable multiprocessor based telephone switch. Simulation modeling is an effective technique for understanding the performances of large and complex systems such as multiprocessor systems. A simulation program for the call control module of a multiprocessor-based system has been designed, developed, and tested for this thesis study. Capacity analysis of the multiprocessor system is done and the performance of the system under different workloads is presented. The multiprocessor system comprises of three different types of processes that can run in parallel: payload (that does call processing functions) and two non-payload types (one handling maintenance related activities and the other handling operational functions). Contention occurs when multiple processes try to access the same shared memory line. At that time, one process continues execution and the other one rolls back depending upon the decision made by a rollback algorithm. Three rollback strategies: Bounded-Priority Scheme, Age-Based Rollback Scheme and Rollback-Payload Scheme are simulated and their impacts on the system capacity are analyzed. The simulator system is written in C++ using PROPHET (a Nortel proprietary simulation tool). One of the major contributions of the thesis is the observation that the combination of charging rollbacks to class and the rollback strategies tested in the system breaks the CPU shares allocated by the scheduler. Also, out of the three rollback strategies, the bounded-priority rollback scheme exhibits the best scalability. A tool for estimating memory contention described in this thesis was highly useful to Nortel Networks for capacity analysis.

Acknowledgements

This research has been successful due to partial funding from CITO (Communication and Information Technology of Ontario).

I would like to thank Professor Shikharesh Majumdar for his continued guidance and supervision of this thesis. I would also like to take an opportunity to thank Brian Carroll and Dale Streibel from Nortel Networks who have always been a source of excellent help and inspiration for me and my work throughout the course of this thesis. The challenge and enthusiasm of working on this research would not have been so much fun without the support of staff and facilities at Nortel Networks.

My heartfelt thanks to my dad, mom, sister and her family in Ottawa who have always been there for me, encouraging me at every single step and pushing me whenever I felt stumbled or disappointed with anything at all during this course of time. I wouldn't have been able to make it to this day without them all.

I would like to thank my loving husband, Vishal. I thank him for being extremely understanding and patient when I couldn't spend our newly married days with him in San Francisco, due to my thesis work in Ottawa. Also I would like to give the credit of successful completion of my thesis to him since he made my life simpler by moving to Ottawa making sure I don't give up. Thanks for believing in me till the last moment.

Finally, I would like to dedicate my thesis to my guru: my mother, Chander Verma. 'You have been my idol since my childhood and the best teacher I have ever come across in my life. You are the best!'

Table of Contents

1.	Introduction.....	1
1.1	Motivation.....	1
1.2	Related Work	3
1.3	Contributions.....	5
1.4	Outline of the thesis	7
2	A Multiprocessor based Telephone Switch	9
2.1	Capacity needs on existing node	9
2.2	Objectives of moving to a Multiprocessor-based Architecture.....	10
2.3	XA-Core Architecture.....	11
2.3.1	Types of processes.....	13
2.4	Method of Operation	14
2.5	Memory Ownership	16
2.5.1	Blocking	17
2.5.2	Rollbacks and Commits	18
2.5.3	Scheduling of processes	21
3	Simulator Model	24
3.1	Client-Server Model.....	24
3.2	Modeling with PROPHET	26
3.3	Scheduler.....	29
3.4	Inputs of the System.....	31
3.4.1	PROPHET Configuration.....	31
3.4.2	Memory Traces.....	34
3.5	Outputs of the System	34
3.6	Architecture.....	35
3.7	Programming Paradigm	37
3.8	Simulation Software.....	39
4	Rollback Policies	40
4.1	Introduction.....	40
4.2	Age-Based Rollback Scheme	41
4.3	Bounded-priority Rollback Policy	42
4.4	Rollback-Payload Priority Scheme	43
4.5	Rollbacks Charges.....	44
4.6	Rollbacks Charged to System	44
4.7	Rollbacks Charged to Class	45
4.8	Charge to Class chosen for this study	46
5	Results of Simulation Experiments	47
5.1	Performance Metrics	47
5.2	Model Validation	50
5.2.1	Validation Technique 1	50
5.2.2	Validation Technique 2	51
5.2.3	Validation Technique 3	52
5.2.3.1	Sensitivity Analysis 1.....	52
5.2.3.2	Sensitivity Analysis 2.....	57

5.3	Results of Experiments	62
5.3.1	Scenario 1	63
5.3.1.1	Discussion	65
5.3.2	Scenario 2	65
5.3.2.1	Discussion	69
5.3.3	Scenario 3	70
5.3.3.1	Discussion	74
6	Payload and Non-Payload contention	77
6.1	Contention between payload and non-payload	77
6.2	The qnp factor	78
6.2.1	Algorithm for qnp script	79
6.2.2	The “qnp” Measuring Tool	81
6.2.3	Recalculated qnp value	82
7	Conclusions	84
7.1	Discussion	84
7.1.1	Rollbacks Charging Policy	84
7.1.2	Scalability of the system	86
7.1.3	Comparison of Rollback Policies	87
7.2	Scope of Future Research	88
7.3	My ideas to future research	89
7.3.1	Algorithm 1	89
7.3.2	Algorithm 2	90
	Appendix	94

List of Figures

Figure 1: Shared memory architecture of a multiprocessor-based system.	12
Figure 2: Illustration of blocking in the Shared memory.	20
Figure 3: Illustration of Commit and Rollback in the Shared Memory architecture.	23
Figure 4: Client/Server Model.	25
Figure 5: System Modeling Paradigm in terms of PROPHET.	27
Figure 6: Graphical Representation of a Scheduler.	29
Figure 7: CPU shares by Scheduler Classes.	30
Figure 8: The PROPHET Configuration file.	32
Figure 9: Simulation results for sensitivity analysis 1 showing (a) UPP (b) RAP.	55
Figure 10: Simulation results for sensitivity analysis 2 showing (a) UPP (b) RAP.	56
Figure 11: Simulation results for Scenario 1 showing (a) RAP, (b) UPP.	67
Figure 12: Simulation Results for scenario 2 showing (a) RAP (b) UPP.	71
Figure 13: Simulation Results for scenario 3 showing (a) RAP (b) UPP.	76
Figure 14: Overlap of data domain of a payload & non-payload process running in parallel.	79
Figure 15: The effect of decrease in value of qnp on RAP.	83
Figure 16: Class Diagram of object classes used in the simulator.	94

List of Tables

Table 1: Decision Table for the Age-Based algorithm.....	41
Table 2: Decision Table for the Bounded-Priority Algorithm.	43
Table 3: Decision table for the Rollback-Payload algorithm.	45
Table 4: Results of Validation Technique 2.....	51
Table 5: Simulation Results for Sensitivity Analysis 2 for 3 PEs.....	59
Table 6: Simulation results for Sensitivity Analysis 2 for 5 PEs.	59
Table 7: Measured CPU shares for different rollback policies for scenario 1.	63
Table 9: Measured CPU shares for different rollback policies for scenario 3.	72

Chapter 1

Introduction

Performance analysis and resource management of multiprocessor systems are important areas to be explored in achieving a high capacity system. All computing systems are continually challenged to deliver higher processing capacity for improving the response of existing applications and realizing new applications of higher complexity. In this research we are dealing with the call control module of a telephone switch that is based on shared memory multiprocessor architecture. Memory contention results from multiple processes accessing the same variable stored in the shared memory of the system. Shared memory contention is a source of capacity limitation on the switch. Different techniques associated with the resolution of such contentions are the focus of attention for this thesis. Performance analysis of the techniques is achieved through a trace-driven simulator as a part of this study. This chapter presents an overview of the system, the motivations of the research, followed by the brief description of the overall architecture of the system model and contributions made to the thesis.

1.1 Motivation

This thesis is concerned with the simulation of the call control module of a shared-memory multiprocessor-based telephone switch system. The main components of a telephone switch include line interfaces, the switching fabric, and the call control module.

In our study, our primary focus is on the call control module. The multiprocessor-based call control module is responsible for various types of functions like call connection and setup, call maintenance, billing etc. There are two categories of processes that perform the above functions. The first category comprises of *payload* processes, which are responsible for processing a call. The other category consists of *non-payload* processes which are responsible for operational, administrative and maintenance functions of the switch. Call processing is an example of payload process and maintenance, audit are examples of non-payload processes. Our simulation study involves analysis of the multiprocessor system in which three types of processes run in parallel. Call models exist that are based on the skeleton of real call mixes that can run on a telephone switch. For the purposes of simulations, a mix of calls is broken down into subsets called *subcalls*. Each subcall handles the processing of an event such as sending dialed digits, call release, and call hookup occurring on the switch. Execution of a subcall is called a *transaction*. Based on the incoming messages to the system, there is one subcall per incoming message. It is determined by the system architecture as to how many peripherals send messages to the call control module. Executing subcalls may need to access the same addresses in the shared memory at some point of execution. During the lifetime of a call, a system maintenance (non-payload) process may collide with a call processing process (payload). At the point of collision, a decision has to be made as to which process should rollback. These decisions are made with the help of a rollback algorithm. The rollback algorithms can potentially have a strong impact on system performance. We have analyzed three rollback algorithms in our study: Bounded-Priority algorithm, Age-Based algorithm and Rollback-Payload algorithm. A detailed description of these algorithms is

provided in Chapter 4. A simulator tool written in C++ using the PROPHET simulation engine (Nortel Networks proprietary) is used in this thesis (details in Chapter 3). The aim of this thesis is to understand the behavior of the multiprocessor system with three different types of processes running in parallel, compare the performance of different rollback strategies under different workload and system parameters, compute the amount of scalability that can be achieved out of the system. Scalability is an important feature in this system since more processor cards can be added to the switch (with no cost involvement of redesigning) if scalability is assured.

1.2 Related Work

Other studies have illustrated work that exists in the areas of memory contention, rollback algorithms, scheduling algorithms and capacity analysis for shared memory multiprocessor systems as well as distributed systems.

A considerable amount of work exists in the area of techniques for the control of resource contention. A representative subset of the existing work is presented. Most work related to resource contention on multiprocessor systems has focussed on the control of shared memory contention [Maj99.1]. Memory hotspots and their impact on system performance for binary hypercube networks are discussed in [Dand92]. The problem of saturation of the network connecting the processors to memory due to hotspots and a request combination technique to reduce its effect on performance are discussed in [Pfis95]. Management of processing resources on multiprocessor systems has spanned two different domains: the allocation of processors to the threads in a given parallel

computation and the scheduling of processors to multiple parallel applications running simultaneously on the system. One of the early works in the area of mapping threads in a single application to processors proposed a heuristic strategy called "multifit" that considered the allocation of processors to applications with inter-communicating threads is presented in [Coff78].

Majumdar talks about controlling shared memory contention through scheduling in [Maj97]. He proposes different scheduling policies such as Serial (where all non-payload processes runs serially). Independent (all payload processes run in parallel). Gang (processes are ganged together) and Hybrid (engineered mix of two or more of the previous policies).

Using locking and rollbacks for controlling access to shared resources on general purpose computing systems is described in [Silb94]. Different aspects of rollback strategies have been researched.

Koo and Toueg [Koo87] have proposed a rollback recovery algorithm for distributed systems to take care of failures during execution and also to prevent concurrent invocation of rollback recovery algorithms. This algorithm takes a consistent set of checkpoints and avoids livelock problems during recovery. The algorithm's approach is said to be synchronous as the processes involved coordinate their local checkpointing actions such that the set of all recent checkpoints in the system is guaranteed to be consistent.

In most rollback recovery schemes, a failed process is restored to a checkpoint, and the process receives messages in the exact same order as it received them before failing. To recover messages lost due to failure, logging is the most commonly used technique. Jalote

[Jal89] shows the above approach to recover messages is stricter than necessary to save the rollback overhead on the system. He proposes new techniques in [Jal89] to implement fault-tolerant processes that can handle multiple process failures due to collisions. These references are taken from [Sing94].

Another algorithm is proposed by Juang and Venkatesan [Juan91] is for recovery in a system that employs asynchronous checkpointing. The algorithm assumes that a processor upon restarting broadcasts a message that it had failed. Under the asynchronous approach, checkpoints at each processor are taken independently without any synchronization among the processors. Asynchronous checkpointing has its disadvantages against synchronous checkpointing but also avoids the synchronization overhead cost in this method.

Previous work in Nortel Networks has addressed the modeling of shared memory contention on the multiprocessor-based telephone switch. Drweiga [Drw96.1] describes a method of calculating the capacity impact of memory contention between payload and non-payload processes in XA-Core (Nortel Networks). He developed an analytical model of bounded-priority rollback policy (defined in Section 4.3). From this model, a set of equations is solved to find the probability of rollback and the processing power used by rollbacks caused. A method of estimating the contention probability is proposed.

1.3 Contributions

Previous work existed that simulated a multiprocessor system that could handle only payload class of input data [Stre97]. This work was extended to simulate a multiprocessor

system that could simulate more than one class that can run parallel to each other. The simulator developed in this thesis is for a multiprocessor system that handles three classes of processes and can be extended easily to handle more number of classes as well. The three process classes studied in this thesis are Call Processing (CALLP), Maintenance (MAINT), and Audit (AUDIT). The Call Processing processes belong to payload category of process class whereas MAINT and AUDIT belong to the non-payload classes.

PROPHET (Nortel Networks) library is the backbone of the simulator model. The simulator uses PROPHET (Nortel Networks proprietary simulation tool) routines. It provides the rules, concepts and standard functions, to simulate the real-time model of a telephone switch. The multiprocessor system has a scheduler that is responsible for scheduling the different types of processes running in the system. A partial design of the simulator scheduler for a single class existed. It is re-written in this thesis to handle three classes. The simulator code is written in programming language C++. The existing single class system implemented the strategy of charging the rollbacks to the system/class.

Being a single class system, both charging policies produce the same results. The simulator described in this thesis implements the rollbacks charged to class policies. Two rollback strategies existed before for charge to system: Age-Based algorithm and Bounded-Priority algorithm. A charge to class version is implemented for each of the strategies. Another rollback strategy called Rollback-Payload is developed as a part of this study. A description of this strategy is included in [Maj99.2]. The simulation results are illustrated with their explanations in chapter 5. The analysis of the results obtained out of the system have proven that the strategy of charging rollbacks to class (referred to as charge to class) breaks the scheduler shares allocated to the processes running in the

system. Thus the processes are not able to achieve their actual CPU shares. This study was useful to Nortel Networks although previous results based on analytical models indicated this fact, this thesis provided an experimented confirmation. The results of the simulation experiments described in the thesis provide valuable insights into system scalability and the relative performances of the rollback strategies.

Payload and non-payload classes are equally important for maintenance of reasonable capacity on the telephone switch. Payload is responsible for processing the call that goes through the switch and non-payload is responsible for the maintenance and other subsidiary activities of the switch. Thus contention between payload and non-payload processes has been a major concern. This thesis includes the development of a trace-based tool that computes the level of contention between payload and non-payload. The tool identifies the addresses that contend the most. As a part of this study, a feature in the tool was implemented which excluded the highly contentious addresses from the set of memory addresses accessed by the application. By re-engineering the code to avoid contention for these memory locations, the blocking level could be lowered and a higher capacity could be achieved.

1.4 Outline of the thesis

Chapter 2 describes the architecture of the multiprocessor based telephone switch involved in the study. Brief description of the architecture and modeling of the system under study is presented. Chapter 3 describes the simulator model designed and developed in this thesis. It describes the simulation techniques applied to simulate the multiprocessing features of the telephone switch with the help of PROPHEET. Chapter 4

presents the rollback strategies implemented in the system. Chapter 5 describes the simulation results obtained with three different sets of input traces for different rollback schemes. Chapter 6 describes another tool that calculates the level of contention between payload and non-payload processes. Conclusions and directions for future research are described in Chapter 7.

Chapter 2

A Multiprocessor based Telephone Switch

2.1 Capacity needs on existing node

Computing systems have moved from uniprocessing to multiprocessing to produce higher processing power to improve functionality and working of existing applications. In a multiprocessor system a number of processors incorporate together to solve a given problem. The existing uniprocessor-based telephone switch architecture for Nortel Networks, referred to as SuperNode, typifies this trend with constant demand for increased call volumes and advanced call services. [Ben97] The new switch based on multiprocessor architecture is a product to provide SuperNode capacity evolution to higher capacity. This evolution improves on real-time processing, memory, mass storage, and input/output capacity limitations of the existing core. The capacity requirements are derived from a mix of economic, regulatory, and technology factors. Those requirements tend to generate three broad categories of capacity growth in existing product:

- **Port Growth:** Due to pressure on Telco operation costs, smaller switches are being consolidated into fewer big ones to reduce staffing and operations costs. The aim is to achieve switches that can handle higher toll call volumes and still support the same kind of interconnection networks.
- **Service Complexity:** Regulatory changes are redefining the industry, spurring demand for many new integrated applications. Services such as integrated voice

and data and personal communication services are processing intensive, and their mass deployment will require far greater computing power than before.

- **Software Evolution:** The economics of software development, production and support, as well as market responsiveness, are constant drivers for improving software-programming methodologies of product's architecture frameworks. For example, Object Oriented programming can make software more manageable, more modifiable and easier to debug for shorter time to evolve new features. But in most cases the trade off is that the software programs consume more processing power, and this trend is likely to continue with the growing challenge of computing systems to deliver high and higher processing power.

2.2 Objectives of moving to a Multiprocessor-based Architecture

The following are the key objectives that have served to resolve trade-off and compromises toward the overall product design of the new architecture for the telephone switch [Ben97]:

- **High Capacity:** The new multiprocessor-based product aims to re-architect the existing node for sustaining significant growth in workload.
- **High Reliability:** The new system employs new standards through the design of state-of-the-art architectural frameworks for reliable computing.
- **Scalability:** This implies flexibility to increment capacity without long-term planning and complicated hardware surgery. The new product based on multiprocessor architecture provides the ability to extend processing capacity.

memory, message throughput, and mass storage by simple and straight additions to the existing one.

- **Compatibility:** The goal is to run all applications that are currently running on the existing Uniprocessor-based application architecture and to hide the changes in the processing environment from other system components.

The new system XA-Core is proposed that present the opportunity of meeting the above requirements [Ben97]. XA-Core adopts the shared-memory multiprocessor architecture because it offers compatibility with legacy software that ran on a uniprocessor based system. A description of the XA-Core architecture is presented next.

2.3 XA-Core Architecture

XA-Core has a shared-memory architecture that attains not only higher capacity, but also higher reliability than is possible with the duplex processor design of the existing switch product. The memory capacity, the computing capacity and the input/output capacity are scalable by inserting more memory cards and processors. A high-level diagram of the Multiprocessor-Based telephone switch is given in Figure 1. The architecture consists of a redundantly implemented Shared Memory, a series of Processing Elements (PEs) to provide computing capacity, and a number of I/O processors (IOPs) to provide input/output interfaces and mass storage. The internal communication network that connects all of these components together is called the Interconnect.

1. *Shared Memory*: consists of independent memory cards that prevent the conflicting use of memory locations by multiple processors. It operates in duplex for fault tolerance.
2. *PE (Processing Element)*: is a simplex processor. All are functionally identical, executing processes in parallel for fully symmetric operation.
3. An *IOP (I/O processor)*: is a simplex processor responsible for mass-storage or for communication with the outside world like disk, tape etc.
4. The *Interconnect*: is an internal point-to-multipoint network that allows any PE or IOP to access any memory card.

Figure 1: Shared memory architecture of a multiprocessor-based system.

The Shared Memory supports a concept of dynamic memory ownership, which allows a PE to operate on data structures in memory while preventing any other PE from reading or writing the same memory locations and causing conflicts. This gives compatibility with legacy software, including payload activities in a call. However some fine-tuning is required in order to reach capacity targets. The memory allocation done to the processes can be static or dynamic.

Shared Memory contains Data Store (DS) memory, which is fully addressable by every processor. Shared Memory also contains a master copy of Program Store (PS) memory. A local copy of the PS is made available a priori and is used by PEs to ensure maximum execution speed. A Processing Element (PE) is a simplex processor. The general-purpose capability is provided by the PEs. Each PE executes a process in parallel with other PEs. All PEs are functionally equivalent and execute the same software load during normal operation. The input/output interface is provided by an IOP processor, a simplex processor supporting disk and tape units for mass-storage capability or for communication with the outside world.

The Interconnect is an internal point-to-multipoint network that allows any PE or IOP to access any memory card. The main advantage of an interconnection network is its ability to cause communication in the architecture and extendibility, i.e., the possibility of adding or removing functional units easily.

2.3.1 Types of processes

There are mainly two types of processes in the multiprocessor system.

- **Payload:** This refers to application software that implements services that a Telco sells to its customers. The payload software includes functionalities such as call processing, responsible for actually executing a call made on the switch and billing.
- **Non-payload:** This refers to switch support software, consisting primarily of operation, administration, and maintenance features. For example, the non Call-processing such as MAINT, AUDIT classes are responsible for maintenance and operational functionalities of a call made on switch.

Payload and Non-payload are complementary terms for categorizing processes or general software functionality exclusive of the operating system.

2.4 Method of Operation

Multiprocessing in XA-Core is achieved with the same multiple processes as in the pre-existing architecture by executing them in parallel (concurrently) on separate PEs. The obstacle in this strategy is that parallel threads of execution violate historical assumptions about mutual exclusion. Reference of material in this section is obtained from [Ben97]. Utilities that include the control of preemptibility (for example MUTEXON), merely prevent a process from being preempted inside a critical section. MUTEXON is a semaphore used to ensure that only one process can enter the critical section at one time. It is used for mainly disabling the interrupt so that the processor is not taken away in the middle of execution. They were not designed to stop other processors from accessing memory at the same time. In order to achieve correct operation on the switch, it is required that the implied mutual exclusion of a non-preemptable process be preserved

when true concurrency is created. There must be a way of ensuring that a non-preemptable process retains complete control of all data it reads or writes until it is once again preemptable. Other processes must be locked out from using the same data, but it should not be prevented from accessing other areas of memory in parallel. XA-Core provides precisely this capability. The Shared Memory hardware implements an ownership mechanism that automatically claims ownership of all memory that a process accesses when running. That memory remains inaccessible to other processes until the owning process finishes its work and suspends. Meanwhile, another process that uses a different set of data is able to access the Shared Memory in parallel. With this memory-access strategy in mind, several principles of operation emerge from the XA-Core architecture:

- Multiple PEs executes processes in parallel.
- Shared Memory is fully addressable by every PE and continues to be the sole repository for all data preserved when a process suspends, completes its execution.
- The Shared Memory supports dynamic memory ownership, which prevents destructive interference between parallel processes attempting to access common data.
- PEs are symmetric; that is all PEs are functionally equivalent and execute the same software load. A Software program achieves the same result no matter which PE it runs on.

- The system scheduler runs on all PEs. As before, when the current process is finished the scheduler is invoked (within a critical section) to select the next process to execute.
- Any single process will run on only one PE at a time, but the process can run on a different PE every time. As calls are processed at the same time, call processing can execute on all PEs simultaneously.
- The work accomplished by a process does not become visible to the rest of the system until it commits.

2.5 Memory Ownership

A shared-memory multiprocessor system must prevent parallel processes from conflicting use of shared memory locations. The shared memory supports a concept of memory ownership that provides this ability. The underlying principle of operation is that two processes can run in parallel provided that they don't read or write any common memory locations. To implement this, each process that is executing in a PE is assigned a unique identifier, and this identifier is used as a tag of claim ownership of all locations in Shared Memory that are read or written by the process. Any other process attempting to access a memory location that is already owned is blocked from proceeding. A blocked process aborts what it was doing and restarts from the beginning at a future point in time. The outcome is that those processes requiring the same Shared Memory locations are, in effect, serialized by the system, whereas independent processes are allowed to execute in parallel. A blocked process is able to start over because the Shared Memory never allows

a process to permanently update any memory location until it commits. In other words, signals its successful completion. All updates are recorded in the Shared memory as pending changes only, and the pending changes do not become official until commit time, the time when the job is successfully accomplished. If the process blocks, the pending changes are discarded and the process is rescheduled for another attempt to run to completion later, after the blocking process had committed and cleared its ownership tags. This ability to discard pending changes and return to previous state is called a *rollback*. The rollback is a very powerful capability that provides data integrity and is well suited to fault-tolerant processing. In the event of a PE failure, it is possible for the process that was executing at the time of failure to roll back and restart in another PE. Software functions are now effectively immune to a processor crash. Freedom of starvation is ensured on the system through rollback policies described in Section 4.2 through 4.4.

2.5.1 Blocking

The Shared memory provides ownership tags internally for all memory lines (a memory line is 32 bytes long). The tags allow the Shared Memory to keep track of which process is accessing which memory locations. Whenever a process executes, all memory lines that it reads or writes are locked, on the first access. A memory line is tagged as being temporarily owned by that process. The ownership tags remain in effect until the process finishes its work and commits. At that time, ownership is released and other processes can claim the memory lines. Only one process can own a memory line at a time. The

Shared memory will *block* any other process from accessing a memory line that is already tagged. When blocked, a process rolls back: that is, it aborts what it was doing, all work performed up to the point of the block is discarded, and it is given another chance some time later to execute without blocking, starting over from the point of the last commit. The terms “process contention” and “process collisions” are used synonymously with the term “process blocking”. Every time two processes contend for the same memory line, one of them must roll back and start over. They are effectively forced to execute serially so that mutual execution is preserved. See illustration of blocking between two processes in Figure 2. In Figure 2, A and B are acronyms used for process A and B respectively.

2.5.2 Rollbacks and Commits

There are two copies of every memory line in Shared Memory. The Commit copy is the official copy that records the most recently committed update. The Active copy is where a process temporarily records an update until it commits. When a process commits, the two copies exchange roles such that the Active copy in each tagged memory line becomes the new Commit copy and the old Commit copy becomes the new Active copy to be updated the next time. When a process rolls back, changes made to Active copies are just ignored, effectively erasing any trace of the process’s execution. In either case, ownership tags are released so other processes can use the memory lines. Commits and rollbacks are not system wide but rather function on a per process basis – many processes can be executing at any one time and commit or rollback independently of one another.

Commit and rollback (Figure 3) might appear to be intricate but in practice they are simple. They are activated by a mere state change of the tag identifier in the Shared Memory. There is no immediate need to walk through the memory to update all the memory lines that are affected. A memory line will be adjusted on its next access.

The commit procedure is associated with the normal completion of a process's thread of execution. The process stops running on the PE and all cached information is flushed from the PE back to the Shared Memory. The rollback procedure, on the other hand, is associated with the abnormal termination of the thread. The process stops running on the PE, the cached information is invalidated, and the process is rescheduled for another chance at executing to completion. Thus, a process that rolls back must behave as if it never executed since its last commit. There is also no guarantee that the rolled-back process once rescheduled, will execute exactly the same sequence of instructions as before: it might branch onto a different path of execution than before.

Figure 2: Illustration of blocking in the Shared memory.

In shared memory systems, several processes cooperate by exchanging information from shared memory to accomplish a task. In such concurrent systems, all cooperating processes need to establish recovery points, where one process has to rollback when a failures occurs. A process in XA-Core executes a number of subcalls. XA-Core implements a concept of Commit and Continue to ensure the recovery of the system in case of memory contentions. Here a process commits at the end of every subcall (transaction) it executes before starting the next one. The process releases the memory addresses it owned in the previous transaction and does a context switch to the next transaction. This way, when the process fails (memory contention occurs) and has to roll back, it has saved all the processing power accomplished till the last commit and only the transaction it was currently processing goes to waste.

2.5.3 Scheduling of processes

The scheduler bundles processes into classes for managing the fair allocation of CPU time across processes. Class shares are engineered to a maximum percentage of total CPU time when the system is running at full capacity. A principle of this system is that all PEs are equal. A scheduler in the form of a scheduler template defines the order in which the processes are scheduled. Each category of process is referred to as a class in the scheduler template. So, for example if there are six different types of processes running in the system, there are six classes in the system. When the process at hand finishes and suspends, the scheduler is invoked to select the next process to run, using the class specific ready queues and other scheduling information residing in the scheduler template

at the scheduler level. The system has a capability of guarding against two PEs invoking the scheduler at the same instant and picking the same process to run. The scheduler has a small critical section that can run in only one PE at a time. A scheduler template exists where each class, out of the n classes, is allotted some percentage of the total share. The scheduler then picks up the particular class according to the scheduler share and schedules a process of that class to run. The fact that a process can roll back and start over means that it might suffer some delay before completion, especially if it rolls back more than once in a row. The scheduler ensures that steady progress is made and the system is engineered to meet grade of service standards. The scheduler template is designed in such a way that it makes sure each class of application is guaranteed some share of time. Thus the work definitely progresses.

Figure 3: Illustration of Commit and Rollback in the Shared Memory architecture.

Chapter 3

Simulator Model

This chapter describes the design and implementation of the three class simulator model. It also discusses about the methodology used to model the system: basic principles behind the modeling, inputs and outputs of the system, and working of the system in detail.

3.1 Client-Server Model

The simulator for the XA-Core call control module is designed in the terms of a three class *Client-Server* system with a capacity to run multiple PEs at the same time. The numbers of PEs that can be running in the system vary from 2 to 11. In the Client-Server model, the Server manages the data and processing is accomplished at the Server end. Clients represent the peripherals on the real XA-Core switch and Servers are used to model PEs. A Client generates messages at regular intervals of time. Interaction between Client and Server takes place through a messaging interface. Incoming requests are queued up and processed one by one, avoiding data contention. Figure 4 shows a Client-Server Model. There are three types of processes in the system namely CallProcessing (CALLP), Maintenance (MAINT) and Audit (AUDIT).

Figure 4: Client/Server Model.

Each process type has its own Client and Server. The Clients generates messages whereas the Server processes them. For example, CALLP client sends payload messages to the CALLP server. Similar is the action of messaging for the other two processes. Multiple instances of the same process can be executed by more than one PE at a time. Thus for every PE present in the system there is an instance of each Server type present. There is a concept of per-process data, a data construct that provides multiple copies of the data structures, one for each process instance. Per-process data works best for independent

processes, though communication and synchronization among them can be achieved with inter-process messages.

3.2 Modeling with PROPHET

PROPHET (Proprietary Hierarchical Evaluation Tool) is a library of C++ classes and functions for the development of simulation programs. It is a design capture and performance analysis toolkit for multi-tasking/multi-processor systems. It has been designed to offer modeling flexibility, speed of execution, and ease of use. The current system's modeling is also done using PROPHET [Prop96]. PROPHET executes the model and collects statistics that can be used to analyze or predict the performance of the system. To develop the simulator with the tool, a C++ program is written that describes the real-time behavior of the system to model, using basic constructs from the library. The modeling is done in terms of a Client-Server system in PROPHET. Client and Server are modeled as tasks, the most important fundamental components in PROPHET. They usually contain most of the information in the model. Tasks communicate by exchanging messages. Each message is assigned a numerical identifier, which determines its type. In our model, there are three types of messages sent from three types of processes. A sending task (Client) sends a message to a queue in receiving task (Server). Message queues hold the messages sent to a task until the task accepts them. The Server receives it and processes it. The Server processing simulates the execution of a particular class of application process. PROPHET's modeling paradigm is depicted in the form of modeling

layers. The three layers are namely applications, scheduling, and hardware. Figure 5 shows how the modeling is done for our system in terms of these mentioned layers.

Figure 5: System Modeling Paradigm in terms of PROPHET.

The application layer represents the software applications in the system. It is the set of concurrent tasks communicating with messages. A task owns a message queue to hold the messages it receives. Each task defines the control logic, real-time delays, and messaging of a part of the system being modeled. A task instance is an independent executable image of the task. Each task instance is given an order number to distinguish it from other instances of the same task. Instances of the given task can be assigned to different

processors. A task has a priority and a time slice, which is used by the scheduler to arbitrate contention for the processor. For example, in our case we have three types of processes running in our system. Out of CALLP, MAINT and AUDIT, Nortel Networks designers have given a priority of 1,2,3 to each one of them respectively. To execute, applications require processors that compose the hardware layer. This layer is a set of processors running in parallel. A processor can be seen as a passive resource that a task needs to execute. It has a relative speed and a scheduling algorithm. All processors are "virtually connected", and messages at the application level can travel between them without restrictions. The scheduling layer is a set of schedulers. The purpose of the Scheduling layer is to resolve contention between applications or the processing resources. A scheduler defines how contention for processors is resolved. Its job is to determine which task will get the processor when several of them are ready to run on it. There is exactly one scheduler per processor. The scheduler is also represented by a software program and thus requires the processor to execute. If the scheduler holds the processor for a non-null period of time, it is possible for tasks to request the processor while the scheduler is busy. If that happens, the requests are stored in the queue of pending requests and are served in order of arrival when the scheduler is available again. Figure 6 is a representation of a scheduler. In the figure, the scheduler is shown to have ready queues to store tasks waiting to be executed. If the scheduler holds the processor for some period of time, it is possible for tasks to request the processor during busy period of the scheduler. The scheduler then hold the requests in a queue of pending requests (as shown in figure 6) and serves them in the order in which they arrived in when the scheduler is free again.

Figure 6: Graphical Representation of a Scheduler.

3.3 Scheduler

The scheduler bundles processes into classes for managing the fair allocation of CPU time across processes. Class shares are engineered to a maximum percentage of total CPU time when the system is running at maximum capacity. Scheduler classes can be used as a model for categorizing processes and weighing their contribution to the overall behavior of the system. This is a three-class scheduler, the classes being CALLP, MAINT and AUDIT. Here is a description of how the scheduler works. Logistically, there are several different classes grouped together in various combinations called *schools*. In our case, we have four schools where each school is a collection of classes. Schools are built in order to lower latency times of process classes. In the system's scheduler, the order of schools are CM, ACM, MA, A where C stands for CALLP, M stands for MAINT and A stands

for AUDIT. The schools are visited in a fixed order and this is called a *scheduler template*. Figure 7 shows an example of distribution of scheduler shares among process classes in our model, provided by Nortel Networks.

<u>SHARE:</u>	<u>PURPOSE:</u>
Payload (CALLP) 75%	Processing a call
Non Payload (MAINT) 16%	Critical network and peripheral maintenance
Non Payload (AUDIT) 9%	Polling, operational & administrative functions.

Figure 7: CPU shares by Scheduler Classes

Decision has to be made of the scheduler template depending on how we choose the input streams of the three class data respectively. To give a simple example, we are visiting the schools in the following order: CM, MA, A, ACM - CM, MA, A, ACM. That is, the scheduler visits the schools in the order shown viz. school CM, then MA, then A and then ACM and then again the same order repeated. This is one round of the scheduler. In the next round it will start the whole process again. The scheduler goes from school to school looking for work to do. Within the schools, it scans the class. If there is no work to do in that class, it goes to the next class within the same school. If there is no work to do in the whole school, it simply goes to the next school in the template. So it keeps repeating this

process until it finds some work to do in some school. Thus, unused processing time in one class is used up by other classes (like any other round-robin scheduler). If for some reason, a school exceeds its time slice, the excess amount is deducted from its time slice the next time the same school is encountered. There are counters for each instance of the class running at a time to keep track of the time slice used. If one class uses up all the its available time, the counter moves to the next class in the school.

Note that in this example, the MAINT class was present in all the schools. This was to ensure that the MAINT program runs frequently and provides a short recovery time. In the absence of a fault, the share for MAINT will be taken by CALLP. In our study, the schools are chosen in order for MAINT class to have low latency in the system.

3.4 Inputs of the System

This section describes the inputs to the simulator system, one done through PROPHET configuration setup and the other through memory traces.

3.4.1 PROPHET Configuration

PROPHET requires an input data file, which is the most important for providing input to the PROPHET-modeled simulation system. It is responsible for setting the configuration of the experiment and is referred to as PROPHET.config file [Prop96]. The configuration file contains parameters related to the experiments, such as warm up length, batch duration etc. This file is mandatory. There is no restriction on the naming conventions of this file. Figure 8 shows an example of PROPHET.config with a configuration format as

required for our simulation model. A description on each input parameter is provided next.

The Number of experiments indicates the number of experiments the user can make in a simulation for a particular configuration, the number can be n , where n is a positive integer. The Clients send messages to the Server at regular intervals of time. The inter-arrival rate of injecting messages at Client end is controlled by an input variable *Inter-Arrival Time*. The time duration between arrival of two messages is generated from an (negative) exponential distribution with a mean of Inter-Arrival time. The lower the Inter-Arrival time, the more work the Client is injecting into the system for the server to do.

<Number of experiments>
<Warm-up length>
<Number of batches>
<Batch duration>
<CALLP interarrival time>
<MAINT interarrival time>
<AUDIT interarrival time>
<Number of PEs>
<Rollback Scheme>
<CALLP Service Time>
<MAINT Service Time>
<AUDIT Service Time>

Figure 8: The PROPHET Configuration file.

To adjust the overall Utilization done by CPU, the inter arrival rate can be decreased or increased. Since we have a system with three different types of processes, we have a separate three inter-arrival time for each of CALLP, MAINT, and AUDIT. The number of processing elements (PEs) is controlled by an input variable *Number of PEs*. The number of PEs can vary in the system from 2 to 11. The higher the number of processors in the system, the more processing power is expected to be produced by the system because

there is more work being done. The *Service Time* is the time burnt in a rollback overhead when a rollback occurs. This has a deterministic distribution and Nortel Networks has provided the exact value for the service times to be used in simulation. This measure would replicate what is happening on the real switch in terms of where the time goes when a rollback occurs. The rollback is charged to each class whose process is rolled back. Out of this whole time, one-third of the time is spent in releasing memory addresses the rolling back PE owned. The other two-third is spent in the context switch that occurs when the roll back has taken place. There are three inputs for the Service time as well each for CALLP, MAINT and AUDIT. Another input parameter is the capability to be able to choose the *Rollback scheme*. The parameter can be set to 0,1 or 2 for Age-Based, Bounded-Priority or Rollback-Payload respectively. The *Warm-up* of the experiment can also be set in terms of number of commits that should occur by warm-up. For higher number of PEs, higher warm-up is required for the system to produce accurate results. The *Number of batches* decides the length of experiment. The experiments made in the study include a time batch means experiment engine provided by PROPHET. This consists of a warm-up (as mentioned above) and a number of batches. All the batches have a fixed duration set at the beginning of the experiment. This is stored in *Batch duration*. Statistics collection begins at the start of simulation. At the end of warm-up, all statistics collected so far are reset. Statistics are consolidated at the end of every batch. In our case, each batch ends with a commit.

3.4.2 Memory Traces

This is a trace-driven simulation and the experiment runs are based on traces as input to the system. Currently, the only available data on memory contention are the traces of transactions collected in labs on the real uniprocessor-based switches. The traces contain all shared memory addresses accessed by each accessed process class. They also show when in the course of executions are those addresses accessed. Three trace files fed into the system, one for each type of processes in the system. The current set of tracefiles that were available at Nortel Networks included payload traces only. Different trace files were selected from this set of available files to constitute the CALLP trace, MAINT trace, and AUDIT trace. A trace is a sequential list of all memory locations that have been accessed by a processor while executing a selected program segment. Traces of calls are processed to find common memory locations. It is possible, for instance, to separately take memory traces of two call types and compare them. If they have any data locations in common, those locations are potential points of data contention and the calls could block each other if executed concurrently. Using the knowledge of which subcalls share memory, how long memory ownership would be held, the average work time of subcalls, and the weight of the call types in the call model, the performance of the system can be predicted.

3.5 Outputs of the System

The aim of writing a simulation model for a given system is to collect statistics on the performance indicators of the system. Tasks, message queues, ready queues and processors all have statistics gathering built in. Users can define their own statistics and

how and when to update them. Statistics provided in PROPHET are of two types: Sample Statistics and Time Averaged Statistics. Sample statistics collect values independent of time whereas in Time Averaged Statistics the values are weighed by time. PROPHET provides two standard simulation engines to measure performance indicators of the system. The TimeBatch engine, which we have used in this study, consists of a warm-up period followed by a certain number of batches. All the batches have a fixed duration, which is specified, at the beginning of the experiment. Statistics collection begins at the start of the simulation. At the end of the warm-up period, all statistics collected till those points are reset. Statistics are consolidated at the end of every batch. The engine by default prints a batch report at the end of every batch and a consolidated report at the end of the experiment. The user is also able to print out user-defined statistics in the consolidated and batch report.

3.6 Architecture

As described in Section 3.1, the modeling of the system is done in terms of a multiple Client-Server system. The client sends a message to the server. Since three types of processes can run at the same time, client A sends message type A to the server type A. If the server is busy at that instant, the message is stored in the message queue until the server is free next to process it. When the server is free, it allocates a PE to the message to process it. The message then becomes a unit of execution and is referred to as a process. This is a trace-driven simulation and the experiment runs are based on traces as input to the system. Call models exist that are based on the skeleton of real call mixes that

can run on a switch. Each trace file is a mix of call models broken down into subsets called subcalls. Processing a call is performed in stages called subcalls which correspond to the different stages in the lifetime of a call such as going off-hook, sending dialed digits, call setup, call supervision, and call release. The subcalls access memory addresses. Each subcall has a weight associated with it, which is called the work time, indicating how long the subcall will run for. There is no fixed order to picking up subcalls. The process picks up a subcall randomly (similar to a Monte Carlo simulation). Thus as the process starts execution, it picks up a trace file based on a random number and starts reading the addresses in the trace. At the same time, another process could be running holding any class of data and it correspondingly could be reading another trace file. If the two processes running at the same time do not encounter an address that could be common between them, they run successfully to completion. There can be a possibility that process 1 is occupying one of the memory addresses that process 2 is also looking for to occupy during its course of execution. At that point, a collision occurs between the two. A decision has to be made as to who wins over that memory address. The decision depends on the decision making rollback scheme. If process 1 wins, it continues execution with the memory address and the process 2 rolls back. Process 2 that rolls back is sent to a box implemented as a queue called *penalty box* and stays there. The implementation is such that the process 1 that caused the rolling back of process 2 is responsible to free the process 2 waiting in the penalty box. When process 1 has committed, it makes sure that the process held in penalty box is freed. Thus, process 1 continues execution ahead. Once a process rolls back, it spends some time in cache flushing, in releasing the memory addresses it owned when it was executing. The rolled

back process also spends some time in context switching before the new process starts execution. These constitute the rollback overhead that every process that rolls back experiences. Any process stops execution either when it blocks with another process or when it commits. A single process can be involved in multiple collisions within the same simulation experiment. Measurements are observed and noted out of the PROPHET statistics reports. An ideal run is the one where CPU is utilized to a capacity of 95%. If the system would tune to 100%, in reality that would hit the overload control. But there are no measures taken yet to model overload controls. Thus the simulations have been recorded with a CPU capacity of approximately 95%.

The simulation run length is user controlled and the parameters can be adjusted using the PROPHET.config file. For all simulation runs, we controlled the run length by defining the number of commits in a batch. Each run ends when the simulation has reached 3000 payload commits. Running the simulation into 30 batches with 100 commits/batch does this. The number of batches is chosen to be 30 and the number of commits per batch is set to 100. These figures were used for simulation runs in XA-Core. These numbers suggested by Nortel Networks architects are adequate for simulation runs to meet their accuracy requirements [Stre97].

3.7 Programming Paradigm

The coding of this simulator model is done using the object-oriented programming language C++. A part of this code was obtained from a pre-existing single class simulator

written in C++. This section gives a description of using the object-oriented paradigm in the multiprocessor-based system.

Data contention is the interference between two processes that attempt to use the same data in memory. The programming requirement is not only to manage data contention using mutual exclusion techniques but also to reduce the amount of data contention that can occur. To meet such purposes, the programmer's job is to include the design of data architecture that limits data contention. Programming for a multiprocessor machine requires special care in the management of program data because of performance penalties associated with data contention, where parallel processes interfere with each other when attempting to access the same data structures in memory.

Object-Oriented programming (O-O) is a comprehensive design philosophy and methodology that offers a formal data-oriented approach to software design. New techniques are available for defining data domains and creating program infrastructure around them. A process's data domain is the set of all data that the process reads or writes during its execution. In O-O terminology, data is encapsulated meaning that the data structures are contained within objects and hidden from external view. Encapsulated data is only accessible via methods.

Thus, data management is a compelling reason for considering the object-oriented programming for multiprocessing applications. O-O offers better program comprehension, resilience to change, and code reuse. A class diagram of object classes used and constructed in the simulator code is presented in Appendix (Figure 16).

3.8 Simulation Software

The simulator software is a program of approximately 5000 lines, out of which about 1700 lines of code are inherited from the legacy software. The lines are separated into files where each file contains code that implements a distinct part of simulator. The following is a description of different files and functions of each file.

Sim-main.c: This is the main program body that defines the clients and servers classes that form the basic backbone architecture of the simulator.

Sched_main.c: This is the main body of the scheduler simulator that contains code-controlling factors like handling time slices, decision-making as to which class of process is scheduled to run etc. It defines the code that controls the pointer to the class next to run. The ready-to-run queues are defined and invoked here. These queues queue up all the jobs, which are ready to run and are waiting for the CPU to be free.

Sched_template.c: This contains the scheduler template. The program defines the order in which classes of processes are defined and allocated for execution. It includes definition of the schools, order of classes in the school and their order in which they would run.

Chapter 4

Rollback Policies

4.1 Introduction

The shared memory allots ownership tags (refer to Section 2.5 for details) to all memory lines contained in it. Only one process running at one time in the system can own a memory line at a time. If any other process tries to access a memory location in the same memory line, a collision occurs. Whenever a collision occurs one of the processes is chosen to be the victim and is rolled back. The choice of the victim depends on the rollback policy. The process aborts what it is doing and all work performed up to the point of the block is discarded. The process is given another chance some time later to execute without blocking, starting over from the point of last commit. They are effectively forced to execute serially this way so that mutual exclusion can be maintained in the system. Rollbacks expend work to maintain data consistency. If not controlled effectively, such rollbacks can hinder system throughput and reduce scalability. There are three rollback policies implemented and tested in this 3-class simulator system:

- Age-Based rollback algorithm
- Bounded-priority algorithm
- Rollback-payload algorithm

4.2 Age-Based Rollback Scheme

The Age-Based rollback policy is based primarily on the age of the processes running a particular class of processes. Here is an example of how the scheme works. For example, p1 and p2 are two processes running in the system. When they start, their priority is set to zero. If a collision occurs between the two processes and p1 rolls back, then priority for p1 is increased by 1 and p2 continues execution. Now say collision occurs again between p1 and p2 with the same status of priority. p1 wins due to higher value of priority. The priority can go up to any limit. Thus, when a collision occurs between two processes, the one with the higher value of priority wins. The age of a process is used in the following way when two processes with the same priority collide. When a process starts executing it is given a fresh time stamp indicating its age. When a message arrives in the system, its age is zero. When two processes that have rolled back more than once collide and have the same priority level, then according to the age-based algorithm, their time-stamps are compared and the older one wins. The older job is thus assured higher priority and the youngest one rolls back. Table 1 depicts the Age-Based priority algorithm showing combinations of two processes with priorities.

Table 1: Decision Table for the Age-Based algorithm.

	Process 1 [Priority 0]	Process1 [Priority 1]
Process 2 [Priority 0]	First come first serve	Process 1 wins
Process 2 [Priority 1]	Process 2 wins	Oldest wins

4.3 Bounded-priority Rollback Policy

This policy involves a priority algorithm with a limit on a priority that a process can reach. The bound on the priority value is a positive integer and is fixed to be 2 in current simulations. To limit the number of times a transaction can be rolled back, this policy has been applied to XA-Core. A rolled back process cannot be restarted as long as the blocking transaction is executing. This prevents rollbacks from coming back too soon with higher priority and rolling back the blocking process. When a process is scheduled in for the first time, its priority is set to 0. When two processes of priority 0 collide over a memory access, the first continues its execution. The other process is rolled back and its priority is increased by 1. Similarly, when two processes of priority 1 collide, the one that owns the address continues, and the other is rolled back and is given a priority of 2. Let us consider a system like ours in which the upper bound on priority is 2. When a collision occurs between them and they have different priority levels, the higher priority process wins. The important feature of this rollback scheme is that the machine serializes when a process with priority 2 is running in the system. When processes reach a priority value of 2, they run serially, which implies that a priority 2 process cannot collide with any other process of priority 2 running at the same time. If a priority 2 job is running and there is another priority 2 job ready to run, the ready to run job will wait in a queue until the first job finishes and then gets its chance to run. Thus all processes of priority 2 are serialized so that they never collide with each other. They can only collide with a process of a lower priority, therefore they always continue their executions. This way a process cannot be rolled back more than twice. The following table describes a combination of various

possibilities of collisions that can occur between processes running in the system with different priorities.

Table 2: Decision Table for the Bounded-Priority Algorithm.

	Process1 [Priority0]	Process1 [Priority1]	Process1 [Priority2]
Process2 [Priority 0]	First come first serve	Process1 wins	Process1 wins
Process2 [Priority 1]	Process2 wins	First come first serve	Process1 wins
Process2 [Priority 2]	Process2 wins	Process2 wins	Serialization Occurs

4.4 Rollback-Payload Priority Scheme

Under this policy when a collision occurs between a payload and a non-payload process, the payload process is rolled back. The policy is applicable only if none of the contending process has a priority level of 2. If there is a level 2-priority process, the steps outlined in section 4.3 are followed. The rationale behind this rollback strategy is provided. There is a quick response to faults that are processed by non-payload processes and thus fewer calls are lost under faults. In this rollback strategy, when two processes of the same type collide and their priority is less than 2, the one, which has occupied the memory location first, wins the situation and the latter one rolls back. In the scenario where there is a collision between two processes running Payload and non-Payload types of processes and their priorities is 0 or 1, the payload (Call-processing) one always rolls back. In the case where a priority 2 process collides with a lower priority process, the priority 2 process always wins irrespective of the fact if it is carrying a payload or non-payload job. This

way a process cannot be rolled back more than twice. This is a way to set the limit on number of times a process can roll back in XA-Core before it is provided full liberty to execute without any interruptions due to collisions. Table 3 describes different possible situations that can occur in the system.

4.5 Rollbacks Charges

Rollbacks that occur in a system cause wastage of CPU power. When a process rolls back there is a cost associated with the rollback. In this simulation model, the rollback has an associated real-time overhead in addition to the lost work. The total rollback overhead includes the overhead incurred by the process in releasing the memory addresses it owned when it was executing and the other part of the overhead is constituted by the time that is spent in context switching from one process to another.

4.6 Rollbacks Charged to System

In this scheme, the rollbacks are charged to the whole system. If a CALLP rollback occurs, it effects not only the CALLP class but all the other classes as well and the effect is seen system-wide. Similar scheme is used with other classes running in the system.

Charging rollbacks to the system time means that all scheduler classes are guaranteed the predetermined shares of the useful processing time. The useful processing time is the total system time minus the time spent in rollbacks.

Table 3: Decision table for the Rollback-Payload algorithm.

	Process1 (priority0) Call-P	Process1 (priority1) Call-P	Process1 (priority 2) Call-P	Process1 (priority0) Non Call-P	Process1 (priority1) Non Call-P	Process1 (priority2) Non Call-P
Process2 (priority0) Call-P	First come first serve	Process1 wins	Process1 wins	Process1 wins	Process1 wins	Process1 wins
Process2 (priority1) Call-P	Process2 wins	First come first serve	Process1 wins	Process1 wins	Process1 wins	Process1 wins
Process2 (priority2) Call-P	Process2 wins	Process2 wins	Serialization	Process2 wins	Process2 wins	Serialization
Process2 (priority0) Non Call-P	Process2 wins	Process2 wins	Process1 wins	First come first serve	Process1 wins	Process1 wins
Process2 (priority1) Non Call-P	Process2 wins	Process2 wins	Process1 wins	Process2 wins	First come first serve	Process1 wins
Process2 (priority2) Non Call-P	Process2 wins	Process2 wins	Serialization	Process2 wins	Process2 wins	Serialization

4.7 Rollbacks Charged to Class

In this kind of scheme, the rollbacks are charged to individual classes. If a CALLP rolls back, rollback is charged to only the CALLP class not effecting other classes that are running in the system. The scheduler share allocated for the process class includes with the useful processing time as well as the rollback overhead incurred by the class.

4.8 Charge to Class chosen for this study

Charge to class was chosen for this study for the following reason. When a rollback is occurring in a class, it does not impact other classes at all. This strategy does not affect all the classes at a fair level but charges the class that rolled back with the rollback cost rather than making the other classes victims who are not involved in the rollback. The charge to class strategy is motivated by the effective incorporation of the same strategy in charging scheduler overheads. In the Shared Memory architecture, CPU time is charged per class by the scheduler. Scheduling overhead is also charged per class. These charging mechanisms firewall the processes of one class from another.

Thus the CPU time per payload process can be guaranteed in order to engineer payload capacity and grade of service. Similarly, the real time for a non-payload process and its grade of service can be guaranteed. By analogy, if the rollback cost and rollback overheads were charged to class, this would fire wall one class from another. However, this will only be true if the rollback algorithm does not affect the scheduler share per class. However, with the rollback schemes tested in this study, we have concluded that if the rollbacks are charged to the class, the scheduler shares cannot be maintained and thus are broken. A detailed discussion is provided in chapter 5.

Chapter 5

Results of Simulation Experiments

Results of simulation experiments conducted for investigating the performance of the rollback strategies are presented in this chapter. Before describing these, the experiments done for validating the simulation model are described.

5.1 Performance Metrics

The results of the simulation experiments conducted for analyzing the performance of the rollback policies are presented in this chapter. The capacity of the three class multiprocessor system is analyzed in terms of two metrics: Useful Processing Power and Rated Available Power. These performance metrics are defined as follows. A number of experiments were performed using the simulation model. Results of the experiments conducted for analyzing the performance of the three rollback policies for different scenarios are presented in Section 5.3.

- **Rated Available Power:** Rated available power is a measure of the real work done by payload processes. The unit of RAP is in terms of PEs. It is the average number of PEs that are busy per unit time in processing the useful work done by payload processes.

Thus $RAP = (CP_commits * CP_AWT) / Elapsed_Time$

Where: CP_Commits is the number of payload (CALLP) commits that occurred during the course of simulation experiment.

Elapsed_Time is the total elapsed time of the simulation experiment.

CP_AWT is the average work time of a payload (CALLP) sub call measured in the experiment.

NumberofPEs is the count of processing elements running in the system in the particular simulation experiment.

- Useful Processing Power: The useful processing power is the sum of the useful work done per unit of time by all process classes. Both the hardware and the software architectures impose overheads that can be attributed directly to multiprocessing. Useful Processing Power is the component of the total work done per unit time on the switch, that is left over to do the useful work after multiprocessing overheads has been subtracted out.

The formula for UPP calculation is

$$UPP = \text{System work time} / \text{Elapsed Time}.$$

Where: System work time is the measured time the system has spent in doing real payload and non-payload work excluding work expended due to rollbacks and its overheads. Elapsed time is the total elapsed time for the simulation experiment.

RAP and UPP are measures of utilization's that indicate system capacities. As captured in Little's Law [Jain91], system throughput is directly proportional to system utilization.

The contention between different process classes is characterized by three contention parameters that are defined next. Q_{pp} is the conditional probability of blocking when a payload process runs concurrently with another payload process. Q_{np} is the conditional probability of blocking when a non-payload process runs concurrently with another payload process. Q_{nn} is the conditional probability of blocking when a non-payload process runs concurrently with another non-payload process. An operational definition of q_{np} and an algorithm for its estimation is presented in Section 6.2.1.

Blocking content in a scenario is defined by blocking level. The blocking probability is an indicator of the number of common addresses that exist between process classes that can lead to blocking when these processes run simultaneously on the system. Higher the blocking probability, more common addresses exist that would cause more collisions in the system.

We chose three input traces of data collected on an uniprocessor-based switch of varying blocking levels. The different input traces are referred to as “scenarios” in the chapter. For this study, we chose three scenarios as inputs to our system: scenario 1, scenario 2, and scenario 3. Scenario 1 is the one with lowest blocking of 3.8%, where $q_{np} = q_{pp} = q_{nn} = 0.038$. Scenario 2 is the one with medium blocking probability of 6.6% where $q_{np} = q_{pp} = q_{nn} = 0.067$ and Scenario 3 is the one with highest blocking probability of 16.6% where $q_{np} = q_{pp} = q_{nn} = 0.167$. These input streams of varying blocking probabilities are chosen in order to study the performance of the system under different scenarios that produce a range of low to high blocking probabilities.

These scenarios are used to analyze the different rollback algorithms: the age-based algorithm, the bounded-priority algorithm and the rollback-payload algorithm that are described in chapter 4.

5.2 Model Validation

The goodness of a simulation model is measured by the closeness of the model output to that of the real systems. Validation refers to ensuring that the assumptions used in developing the model are reasonable in that, if correctly implemented, the model would produce results close to that observed in real systems. In some cases, validation of the model is infeasible because either the real system does not exist or because the measurements may be too expensive to carry out. In such cases it may be possible to analytically model the system to a certain extent. The similarity of analytical predictions and simulation results is used to validate the simulation model [Jain91]. Three different techniques were used for validating the model and are discussed briefly in the following subsections.

5.2.1 Validation Technique 1

One way of validating the model is to compare the results of the model with measurements made on the real switch under the same workload conditions. In our case, this method of validation is not actually feasible because the real telephone switch was still under development when the research was carried out. An analytical model does exist that predicts that combination of certain rollback algorithms with the policy of

charging rollbacks to class could break the scheduler shares. Our simulation results also demonstrate the same as discussed in Section 5.3.

5.2.2 Validation Technique 2

Previous work existed where a single class simulator was used to simulate the execution of Call-processing processes on a multiprocessor-based switch. Thus to validate our multi-class simulator model against the existing single-class simulator system, we had to disable the two non-payload (MAINT and AUDIT) classes in our system and had just the Call-processing processes enabled. Thus, we were able to compare the two single-class systems in terms of various parameters such as CPU utilization, queue sizes, CPU idle time and average waiting time for a process to execute. We were successfully able to validate our model with results such as RAP, UPP that closely matched to those of the existing model. A sample set of results comparing the two simulators for bounded-priority rollback algorithm is presented in Table 4.

Table 4: Results of Validation Technique 2.

	# of PEs	Single Class Simulator (PEs per unit time)	Multi Class Simulator (PEs per unit time)
RAP	3	2.77	2.8
UPP	8	6.85	6.84

5.2.3 Validation Technique 3

Sensitivity Analysis is used to check the model's face validity. It is a check to make sure that the model behaves in the expected way when one or more input variables is changed. Based on observations and experience on the real single processor-based system, there is a notion at least of the direction and change in model output when an input variable is increased or decreased. For the validity of our simulation model, we performed two sensitivity analysis experiments by changing the workload for the non-payload processes. The results for Sensitivity Analysis 1 and 2 agree with the expected notions based on the multiprocessor system. They are discussed in details in Section 5.2.3.1 and Section 5.2.3.2.

5.2.3.1 Sensitivity Analysis 1

Sensitivity analysis was performed with different blocking probabilities for 7 PEs in the system. For the purposes of measuring the sensitivity to a change in the non-payload work times in the multiprocessor system, the subcall work times of the non-payload traces were doubled. Since the non-payload work times are doubled, non-payload processes will require more time to run. Hence non-payload will experience more collisions. Moreover, payload gets less chance to run and as a result RAP decreased by nearly 40% to 45%. Results are depicted as in Figure 9 a) and b). We measured the sensitivity of the system with respect to two things a) Sensitivity of system performance to change in work times for different rollback schemes and blocking levels and b) Sensitivity of performance of rollback schemes to different blocking levels. The original simulation measurements are

referred to as generic measurements in the following subsections. The following abbreviations are used in the Figures presented in this chapter.

- 1) RAP-Rated Available Power
- 2) UPP-Useful Processing Power
- 3) BP-Bounded-priority algorithm
- 4) RP-Rollback-payload algorithm
- 5) AB-Age Based algorithm

I. Sensitivity of system performance to change in work times for different rollback schemes and blocking levels.

UPP values for scenario 1 are higher by 0.1 to 0.3 PEs and UPP values for scenario 2 are lower by 0.1 to 0.15 PEs than the generic values as presented in Section 5.3. Refer to Figure 9 b) and Figure 10 b). This variation is because of the difference in amount of useful work being done by the system. Scenario 1 is a low blocking scenario, thus in this case where non-payload work times are doubled, the effect of larger non-payload work times is slightly less than scenario 2 which exhibits a higher blocking as compared to scenario 1. As seen in Figure 9 a), the RAP values for the three rollback algorithms have reduced by 40-45% in comparison to generic cases shown in Figure 11a) and Figure 12a). Due to larger work times of non-payload, the payload gets less chance to run and as a result RAP has decreases.

II. Sensitivity of performance of rollback schemes to different blocking levels.

This section describes the effect of change in blocking levels on the performance of the different rollback algorithms. Figure 10 a) and Figure 10 b) present the UPP and RAP values achieved with the higher non-payload work times. Figure 10 a) shows that the low blocking scenario i.e. Scenario 1 enables the system to achieve higher processing power as compared to Scenario 2, which exhibits a higher blocking case. As the blocking in the system increases, the processing power of the system decreases for all rollback algorithms. As the blocking increases in the system, there are more collisions between non-payload to non-payload processes and non-payload to payload processes. The effect is stronger in comparison to the generic case because non-payload processes run for longer periods here. This accounts for decrease in RAP of the system. (Refer to Figure 10 a) and b).

(a)

(b)

Figure 9: Simulation results for sensitivity analysis 1 showing (a) UPP (b) RAP.

(a)

(b)

Figure 10: Simulation results for sensitivity analysis 2 showing (a) UPP (b) RAP.

5.2.3.2 Sensitivity Analysis 2

The blocking of the payload and non-payload processes in the simulation is controlled via input addresses in the memory traces fed into the multiprocessor system. In this sensitivity analysis, we are testing the sensitivity of the system to qpp, qnn, qnp.

Scenario 2 has been chosen for the study of this analysis. Scenario 2 is a medium blocking case with $qnp=qnn=qpp=0.067$. The sensitivity analysis is done for 3 and 5 PEs in the system.

In terms of implementation in the system, the existing multiprocessor simulator is modified with a new feature that controls the qpp, qnp and qnn of the system. A new control is added to the system that can simulate a particular blocking probability to be zero in order to ignore it during the course of the simulation experiment. This new feature extends the capability of controlling the blocking between processes of two classes. For example, if the user chooses to ignore the blocking between payload and payload processes, the collision between a payload and payload process is ignored. As two payload processes collide with each other, the execution control is sent to the end of the loop in the code that processes the collision. The server pulls the next message from the queue that holds the ready-to-run processes thus ignoring the previous collision that occurred. Similarly, a payload - non-payload collision is ignored if the user chooses to ignore blocking between payload and non-payload processes. And a non-payload - non-payload collision is ignored if the user chooses to ignore blocking between non-payload and non-payload processes. The results of the sensitivity analysis are presented in a tabular fashion for 3 and 5 PEs in Table 5 and 6 respectively.

Collision Stat is an output parameter of the system that counts the number of collisions that took place in the system during the course of the single simulation experiment. This statistic also includes the collisions that were ignored by choosing to ignore a particular type of blocking in the system. Here P stands for a payload process and N stands for a non-payload process. PP Collision Stat is the number of collisions that took place between two payload processes during the whole experiment. PN Collision Stat is the number of collisions that took place between payload and non-payload processes during the whole experiment. NN Collision Stat is the number of collisions that took place between non-payload and non-payload processes during the whole experiment. In the Table 5 and Table 6, column 1 defines the number of PEs (# of PEs), column 2 indicates the rollback algorithm chosen for the simulation experiment, column 3 indicates which one of the contention types (payload - payload denoted as P-P, non-payload – non-payload denoted as N-N and non-payload – payload denoted as N-P) is ignored. This column can also have a value 'Generic' which states the normal operation of the simulation experiment with non-zero blocking probability (none of the blocking levels are ignored in the system). Column 4 and 5 indicate the rated-available power (RAP) and useful processing power (UPP) achieved by the system respectively. Columns 6, 7 and 8 indicate the PP Collision Stat, PN Collision Stat, and NN Collision Stat for the particular simulation experiment.

Table 5 shows the results of the sensitivity analysis tested in a multiprocessor system with 3 PEs for scenario 2. The RAP and UPP values for all the simulation experiments shown in Table 5 do not depict a high variability. This is because the inter-arrival times of messages have not changed in between the experiment for the same rollback algorithm.

Table 5: Simulation Results for Sensitivity Analysis 2 for 3 PEs

# of PEs	Rollback Policy	Blocking ignored?	RAP	UPP	PP Collision Stat	PN Collision Stat	NN Collision Stat
3	Age-based	Generic	1.165	2.54	600	22	645
	Age-based	P-P	1.12	2.48	785	65	751
	Age-based	N-P	1.15	2.54	642	119	924
	Age-based	N-N	1.12	2.48	420	98	1411
3	Bounded-Priority	Generic	1.16	2.56	724	12	712
	Bounded-Priority	P-P	1.19	2.55	1202	104	740
	Bounded-Priority	N-P	1.08	2.42	428	190	786
	Bounded-Priority	N-N	1.19	2.55	505	45	1549
3	Rollback-Payload	Generic	1.14	2.53	669	6	398
	Rollback-Payload	P-P	1.21	2.57	1056	128	571
	Rollback-Payload	N-P	1.24	2.56	584	14	749
	Rollback-Payload	N-N	1.19	2.55	721	163	1734

Table 6: Simulation results for Sensitivity Analysis 2 for 5 PEs.

# of PEs	Rollback Policy	Blocking ignored?	RAP	UPP	PP Collision Stat	PN Collision Stat	NN Collision Stat
5	Age-based	Generic	2.45	3.7	1038	47	754
	Age-based	P-P	2.5	3.8	2025	123	391
	Age-based	N-P	2.6	3.6	1230	313	654
	Age-based	N-N	2.55	3.6	1074	76	332
5	Bounded-Priority	Generic	2.27	3.5	1023	134	648
	Bounded-Priority	P-P	2.37	3.5	1974	184	403
	Bounded-Priority	N-P	2.2	3.6	989	385	787
	Bounded-Priority	N-N	2.2	3.5	1080	273	1269
5	Rollback-Payload	Generic	2.2	3.6	1236	179	699
	Rollback-Payload	P-P	2.19	3.5	2154	391	366
	Rollback-Payload	N-P	2.2	3.6	989	385	787
	Rollback-Payload	N-N	2.2	3.5	735	568	2100

As we ignore the blocking between payload and payload processes, a growth is observed in the number of overall collisions in the system between payload and non-payload processes for age-based, bounded-priority and rollback-payload algorithms. With no blocking between payload -payload processes in the system, more payload processes get a chance to execute quickly leading to a rise in number of payload - payload collisions (PP Collision Stat). Note that the payload-payload blocking was ignored in order to simulate the activities on the switch with $qpp=0$. Fast processing of payload processes gives a chance to non-payload processes to run through the system, thus a higher count of non-payload - payload and non-payload - non-payload collisions (PN Collision Stat and NN Collision Stat).

By ignoring the blocking between payload and non-payload processes, in age-based rollback algorithm, this bumps up the count of payload - non-payload collisions as the processes are running through the shared addresses at a fast pace, leading to high number of collisions. Thus the collisions would grow even between non-payload - non-payload processes. But in case of bounded-priority and rollback- payload algorithms, there is different effect observed with the status of collisions between payload - payload processes. Both the algorithms involve priority 2 serialization in the machine. Due to highest share of payload processes in the scheduler, there are high numbers of priority 2 payload processes in the system. With the serialization of machine, there are no collisions between priority 2 processes and thus lower overall system collision counts.

When the blocking between non-payload and non-payload is ignored, the count of non-payload - non-payload collisions grow up because of fast processing through the shared addresses between two non-payload processes, even though no rollbacks actually take

place in the system. Measured results also show a jump in the value of NN Collision Stat for all the rollback algorithms. Thus payload processes, due to their large share in the scheduler template get to run in parallel more with non-payload processes and thus collide with each other more often. This increases the count of payload – non-payload collisions as well in the system (PN Collision Stat) from 6 to 163. Since the rollbacks are charged to individual classes in the system, ignoring non-payload and non-payload blocking should not have any effect on the collisions between payload – payload processes. Though the count of PP Collision Stat seems to have been decreased, it is not an effect of qnn, it is simply an effect of less payload work done in the system.

Table 6 illustrates the results of the sensitivity analysis for 5 PEs in scenario 2. The measured values for RAP and UPP for all the simulation experiments, as in Table 6 do not show a high degree of variance. This is because the demand on the system has not changed in between the experiments.

As blocking increases in the system due to higher number of PEs in the system, the effect of ignoring any type of blocking does not cause any variance on the collisions between two non-payload processes in the case of age-based rollback algorithm. As more processes block, the count of collisions increases in the system. Due to high share of payload processes in the scheduler, there are more payload rollbacks under all situations. Also in age-based algorithm the older process wins any collision that occur in the system. The probability of a payload process winning a collision is higher than a non-payload process. Since the rollbacks are charged to class, non-payload processes do not get the allocated CPU share and thus do not get a chance to run.

By ignoring blocking between payload - payload processes in the system, more payload processes get a chance to execute quickly leading to a rise in number of payload - payload collisions (PP Collision Stat). Measured results in Table 6 indicate this to be true for all rollback algorithms. Refer to value of PP Collision Stat for generic case and case with where payload - payload blocking is ignored, the count of collisions rises by an average of 44%. By ignoring the blocking between payload – payload processes, a difference in results is noticed for 5 PEs as compared to 3 PEs in the system. With 5 PEs in the system, the effect of charging rollbacks to class shows a strong impact (due to higher blocking of processes in the system with more number of PEs) and as a result payload which owns a large scheduler share gets to run for longer periods, thus more payload work is done in the system. Due to this, the non-payload processes that own a small CPU share in the scheduler anyway do not get their complete share. Bounded - priority algorithm and rollback-payload algorithm show similarities in measured results for the sensitivity analysis for a higher blocking system (5 PEs). See NN Collision Stat and PN Collision Stat in Table 6 for bounded-priority and rollback-payload algorithms. The similarity in the two rollback algorithms is due to serializing priority 2 processes in the system.

5.3 Results of Experiments

Results of the simulation experiments are presented. A discussion of the results obtained with each scenario is presented in separate sub sections as follows.

5.3.1 Scenario 1

This is a low blocking scenario. Figure 11 a) and 11 b) show the graphs displaying the relationship of rated available power and useful processing power with the number of PEs. In Figure 11 a) and 11 b), it is observed from the simulator results that the scheduler shares as defined in the scheduler template are not maintained. Refer to Table 7 and Figure 7 (Section 3.3) for details. Table 7 describes the measured data during simulation whereas Figure 7 presents the actual scheduler shares. The main reason behind this is that rollbacks are charged to class. The class, which is charged, gets punished and spends real time in terms of rollback overhead. Thus the scheduler shares are broken and the measured CPU shares depend on the count of rollbacks each class experiences.

Although the system attempts to run the different process classes in accordance with the pre- allocated scheduler shares, the different number of rollbacks incurred by the different classes lead to a deviation from these pre allocated shares.

Table 7: Measured CPU shares for different rollback policies for scenario 1.

Rollback Scheme	# of PEs	CPU Util %	CP Util %	MAINT Util %	AUDIT Util %
BP	3	95.6	69.0	12.5	14.1
	5	94.05	69.8	11.5	12.6
	7	95.9	80.1	6.3	9.37
	9	94.02	77.8	8.8	7.3
AB	3	94.7	57.9	17.02	19.8
	5	94.9	71.9	10.33	12.6
	7	93.8	76.5	6.6	10.7
	9	95.8	79.2	7.2	9.36
RP	3	94.3	56.5	17.9	17.8
	5	94.75	77.4	7.3	10
	7	93.6	77.6	5.6	10.4
	9	92.8	76	7.8	8.9

In Figure 11 a), it is observed that for the case of 3 PEs, the RAP value for rollback-payload scheme and age based schemes are almost the same. As we increase the number of PEs that is for the case of 5, 7 and 9 PEs, the age-based scheme shows a lower RAP as compared to the rollback-payload policy. In the age-based rollback scheme, due to larger scheduler share for payload processes, there is a high probability of non-payload winning a contention (probability of being the older job) between a payload and non-payload process. As the number of processors increase, the system experiences more blocking and thus more payload rollbacks according to the algorithm.

In Figure 11 b), UPP is quite stable for all rollback algorithms. Simulation measurements show that the bounded-priority algorithm achieves the best scheduler shares for payload and non-payload (closest to the actual CPU shares in the scheduler) as shown in Table 7. It demonstrates the best performance in terms of UPP and RAP values out of the three rollback algorithms for scenario 1.

Scenario 1 is a low blocking scenario. Since there are fewer collisions, more processes are able to commit successfully and thus more work gets done out of the system. Scenario 1 is able to achieve the highest rated available power out of the three scenarios tested for this study. It achieves a rated available power of total 4.5 PEs with the rollback strategy of bounded-priority algorithm for a system with 9 PEs.

5.3.1.1 Discussion

Preservation of Shares: The CPU shares achieved to different classes do not agree with the intended CPU shares. Refer Table 7 for CPU shares for scenario 1 and Figure 7 for actual CPU shares allocated by the scheduler.

Scalability: As expected both the RAP and UPP increase with an increase in the number of processors. The system is scalable as the number of PEs increase from 3 to 9. (see Figure 11)

Performance Comparison: Bounded-priority strategy demonstrates the best performance in terms of RAP. Using knowledge of the age of jobs or its class does not seem to produce any benefit.

The UPP attained with all the algorithms are comparable to one another. Thus the policies that produce lower capacity for payload give rise to a higher non-payload capacity. This is intuitive in case of rollback payload: by favoring a non-payload process the algorithm gives rise to a higher non-payload capacity, which is accompanied by a proportional decrease in payload capacity. Since the rate of arrival for payload processes is much larger than non-payload, a payload process is likely to be less old than a non-payload process when a collision occurs between the two. As a result payload processes are rolled back more often leading to a lower RAP and a higher non-payload capacity for the age-based algorithm in comparison to the other two.

5.3.2 Scenario 2

This is a medium blocking scenario. Figure 12 a) and 12 b) show the effect of the number of PEs on the rated available power and useful processing power respectively. In this

scenario, the blocking level is higher than scenario 1. Thus the RAP and UPP values for scenario 2 are lower than scenario 1 for a given rollback strategy or given number of PEs. For example, as shown in Figure 11 a), the maximum RAP achieved with 9 PEs in scenario 1 is 4.5 PEs, whereas as shown in Figure 12 a), the RAP for 9 PEs in scenario 2 is 3.9 PEs. In scenario 2, it is again observed that the CPU shares of the classes are not maintained. See Table 8 for details. The reason behind this is that rollbacks are charged to class. The class, which is charged, spends real time in terms of rollback overhead. Thus the scheduler shares are broken.

In scenario 2, as the number of processors in the system increase from 5 to 9, the rollback-payload algorithm performs the best out of the three rollback algorithms with increasing number of PEs in the system. This is because this algorithm attempts to reduce the number of payload to non-payload contentions. Since a non-payload process always wins at priority 0 and 1 levels, the proportion of time both payload and non-payload run on the system is reduced and there by the number of payload-non-payload contentions is reduced. Thus, rollback-payload policy shows the highest RAP values out of the three rollback algorithms whereas bounded-priority rollback algorithm shows the lowest values of RAP for 5, 7 and 9 number of PEs in scenario 2.

(a)

(b)

Figure 11: Simulation results for Scenario 1 showing (a) RAP, (b) UPP.

Table 8: Measured CPU shares for different rollback policies for scenario 2.

Rollback Scheme	# of PEs	CPU Util %	CP Util %	MAINT Util %	AUDIT Util %
BP	3	91.5	41.7	25.9	23.8
	5	91.5	63.2	12.9	14.3
	7	92.5	80.6	5.9	6.4
	9	90	81.9	3.8	2.8
AB	3	98.4	60.1	20.8	17.4
	5	97.2	74.1	11.2	11.8
	7	96.5	85.4	4.8	6.7
	9	98.1	86.2	5.5	6.3
RP	3	95.7	52.1	22.1	22.4
	5	96.5	77.5	10.4	8.5
	7	94.5	87.01	4.2	3.4
	9	96.3	91.4	1.2	3.6

Age-based has performed well for scenario 2 as well. it produces RAP and UPP values in between bounded-priority scheme and rollback-payload scheme. For 5 and 7 PEs in the system, age-based algorithm shows similar performance as rollback-payload scheme. This is because, age-based scheme also favors non-payload processes (similar to the behavior in rollback-payload scheme) thus accounting for close RAP values for 5 and 7 PEs.

As shown in Figure 12 b), the values of useful processing power for all the three rollback algorithms are reasonably close to one another. For an average level of blocking case like scenario 2, measurements show that the number of non-payload commits is high for all three rollback schemes, compared to scenario 1. As the blocking increases, RAP goes down, but the increase in the number of non-payload commits compensates for the decrease.

5.3.2.1 Discussion

Preservation of Shares: The CPU shares achieved by different classes do not agree with the intended CPU shares. Refer to Table 8 for CPU shares for scenario 2 and Figure 7 for actual shares allocated by the scheduler.

Scalability: The RAP achieved with the bounded priority and the rollback payload policies increase with an increase in the number of processors. Although initially, the RAP produced by the age based algorithm increases with an increase in the number of processors, the RAP achieved with 9 PE's (RAP = 3.26 PEs) is not significantly larger than that achieved with 7 PE's (RAP = 3.24 PEs). The rationale for such a behavior is similar to that observed in Scenario 3 and is discussed in detail in the following Section 5.3.3. Bounded-priority scheme and rollback-payload schemes make the system quite scalable as the RAP and UPP increase with increase in number of PEs.

Performance Comparison: Rollback payload demonstrates the best RAP especially at higher number of processors. It is interesting to observe that although rollback payload discriminates against payload processes it achieves the best RAP. This is because this algorithm attempts to reduce the number of payload to non-payload contentions. Since a non-payload process always wins at priority 0 and 1 levels, the proportion of time both pay load and non-payload run on the system is reduced and there by the number of payload-non-payload contentions is reduced. As discussed in the context of Scenario 1, the age-based algorithm also favors non-payload processes and produces a performance that is close to that of rollback payload.

All the three strategies attain a comparable UPP for any given number of processors. This indicates that although the rollback algorithms have an impact on RAP the overall system performance (UPP) remains unaltered when the rollback algorithm is changed.

5.3.3 Scenario 3

This is a high blocking scenario. Figures 13 a) and 13 b) show the effect of the number of PEs on the rated available power and the useful processing power, respectively. In this scenario, in the simulation experiments with the age-based rollback scheme, the system failed to adjust to a CPU utilization of 95%. In a few cases, the system runs out of memory and cannot allocate any more memory. As observed in scenario 1 and scenario 2, in cases of bounded-priority rollback policy and rollback-payload policy, the scheduler shares are broken as well (See Table 9 below and Figure 7 in Section 3.3).

Another important observation out of scenario 3 is that the system is scalable upto 7 PEs for bounded-priority scheme. With increase in number of PEs to 9 in the system, the UPP and RAP drop down. The system is not scalable with rollback-payload scheme because UPP and RAP of the system decrease with increase in number of processors.

Figure 12: Simulation Results for scenario 2 showing (a) RAP (b) UPP.

Table 9: Measured CPU shares for different rollback policies for scenario 3.

Rollback Scheme	# of PEs	CPU Util %	CP Util %	MAINT Util %	AUDIT Util %
BP	3	97.5	87.3	0.2	10.04
	5	98.4	95	0.22	3.5
	7	91.5	88.4	2.5	0.39
	9	92.5	86.7	0.26	5.45
RP	3	96.5	82.5	4.5	10.5
	5	95.5	81.4	6.1	8.0
	7	92.5	83.3	3.2	9.0
	9	93.0	87.2	1.5	4.3

As in the previous scenarios, payload share is larger in the scheduler template. Thus payload gets more chance to run. According to the age-based rollback scheme, the older job wins and there is a higher probability that an older job in the situation of a collision would be a non-payload process. Thus, chances of payload losing collisions are more. Since rollbacks are charged to class, payload processes are charged more as they roll back more often. Thus the scheduler shares are not guaranteed. Being a worst case scenario with high contention level, the count of overall collisions in the system is quite high. There is so much work in the work queues that the simulator runs out of virtual memory. Figure 13 b) shows the relationship between useful processing power and the number of PEs. An observation from Figure 13 a) and b) is that there is more non-payload work done in case of rollback-payload algorithm. This is because payload rolls back more often due to the design of the respective rollback strategy. This aspect of rollback payload is more pronounced in this scenario which corresponds to the highest blocking level, thus non-payload gets to run more and more non-payload work gets done out of the system. The RAP and UPP in the case of rollback-payload algorithm go down with an increase in number of the processors in the system. Bounded-priority policy demonstrates better RAP

for 5,7,9 PEs than the rollback-payload policy for scenario 3. The measured results indicate that for high number of PEs, the bounded-priority algorithm can achieve higher rated available power than rollback-payload algorithm in a high blocking scenario (refer to Figure 13 a). At higher number of PE's, as more payload processes are run simultaneously on the system, payload-payload contention becomes more dominant in comparison to non-payload-payload contention. In rollback-payload strategy, the payload is made to roll back in a collision between payload and non-payload processes. This factor is more prominent in a high blocking case like scenario 3 and thus accounts for decrease in RAP and UPP with increasing number of PEs. This does not happen in bounded-priority scheme. Here a higher priority process always wins a collision between payload and non-payload process. This improves the RAP achieved with the bounded priority algorithm in comparison to rollback payload at higher number of PE's.

Figure 13 a) shows that the bounded-priority rollback scheme results in a bell shaped curve for the rated available power values for 3,5,7,9 number of PEs. The rated available power increases with an increase in number of PEs until the number of PEs in the system is 7, and then it drops for 9 PEs. Scenario 3 is a high blocking scenario. Due to high blocking, the count of priority 2 processes in the system is more. Since priority 2 processes in the system run serially, the machine is serialized for longer periods thus giving less chance to priority 0 and 1 processes to run to completion (priority 2 process always wins over lower priority processes in a collision between the two). Thus the RAP values for 3, 5, 7 and 9 PEs are low. With the increase in number of processors from 3 to 7, as the blocking level increases, the bounded-priority scheme improves in performance for a high blocking case. In this rollback policy, a higher priority process wins the

collision over a low priority process. As the number of processor increase in the system from 3 to 7 PEs, due to the priority scheme (where a higher priority process wins over a lower priority process), the RAP of the system increases from 0.22 to 2.79 PEs. That is, with increase in number of processors in the system, the number of payload priority 2 processes also increases, resulting in more payload wins over non-payload processes as the number of processors go up in the system, thus accounting for rise in RAP values. The observation from Figure 13 a) and b) shows that the system reaches a threshold at 7 PEs and the RAP drops down again when the system encounters extreme case of blocking at 9 PEs. Measured results show a high number of payload rollbacks for 9 PEs. A similar pattern is observed in Figure 13 b) where UPP shows a growth till 7 PEs and then drops down for 9 PEs. This indicates that for a system exhibiting the blocking characteristics of scenario 3, using unblocking or other more effective rollback policies are necessary.

5.3.3.1 Discussion

Preservation of Shares: The CPU shares achieved by different classes do not agree with the intended CPU shares. Refer to Table 9 for CPU shares for scenario 2 and Figure 7 for actual shares allocated by the scheduler.

Scalability: Both the RAP and UPP attained with the rollback payload algorithm decrease with an increase in the number of processors. The RAP and UPP for the age based algorithm increases initially with an increase in the number of PE's upto 7 PEs but decreases when the number of PE's is changed from 7 to 9. A brief explanation for this anomalous behavior is provided. As the number of processors increases there is more available processing power to do useful work. As more number of processes run on the

system however, the number of collisions increases for a given set of contention levels. The system capacity attained is a tradeoff between these factors. For rollback payload the second factor dominates and performance degrades with an increase in the number of PE's. In case of the bounded-priority algorithm, although the first factor dominates initially, the second factor becomes more prominent with 9 PE's and both RAP and UPP are observed to decrease.

Performance Comparison: The UPP attained with rollback-payload is higher than that attained with bounded priority especially when the number of processors is small. By favoring non-payload processes, the duration of time during which non-payload processes and payload processes run simultaneously on the system is reduced; as a result the number of non-payload-payload contentions is reduced leading to a higher UPP for the rollback payload algorithm. At higher number of PE's more payload processes run on the system and payload-payload contention dominates. As a result the superiority of rollback payload is reduced and both algorithms demonstrate a comparable UPP.

For reasons described in the previous paragraph, rollback payload demonstrates a substantially higher RAP in comparison to bounded priority for 3 PE's. At higher number of PE's, as more payload processes are run simultaneously on the system, payload-payload contention becomes more dominant in comparison to non-payload-payload contention. By using a priority level 2 queue the bounded priority algorithm makes sure that the priority level 2 processes complete without blocking. This improves the RAP achieved with the bounded priority algorithm in comparison to rollback payload at higher number of PE's.

(a)

(b)

Figure 13: Simulation Results for scenario 3 showing (a) RAP (b) UPP.

Chapter 6

Payload and Non-Payload contention

This chapter describes a method and tool for calculating the level of blocking caused by memory contention between payload and non-payload processes.

6.1 Contention between payload and non-payload

Two processes collide over an access to a shared memory address, if one of them requests an address that is already owned by another process. The problem is resolved by rolling back one of the processes while the other continues execution. If the blocking between payload and non-payload processes is too high, it becomes a limiting factor in terms of system capacity. As more payload addresses block with non-payload addresses, more collisions occur in the system and the system capacity comes down. Payload processes consume the larger share of processing resources and have the highest impact on the system performance. Payload must have a low blocking probability most of the time, and thus the code is re-engineered to reduce the amount of run-time data contention that produces blocking and rollbacks. CALLP process in our case is the payload process that is responsible for connection and billing of telephone calls, communication setup etc. Unlike payload, non-payload software is extremely diverse. It is multidisciplinary, with a myriad of architectures and processes. Currently, there are no performance models on which to base a performance prediction. A blocking analysis is difficult to conduct. Non-payload processes are equally important for maintenance of the switch in case a switch

component breaks down [Ben97]. MAINT is responsible for network and peripheral maintenance, collection of operational measurements, TCP/IP communication etc.

AUDIT is a background process that scans the integrity of the application or resource data, detects data inconsistency, detects invalid states of the machine etc. With shared memory architecture, contentions cannot be avoided. Thus techniques are developed that can enable the system to experience fewer contentions. In this chapter, such a technique is presented where the highly contentious groups of shared-memory variables are identified for elimination from the input data traces in order to bring down the contention level between payload and non-payload.

6.2 The qnp factor

In order to control the payload - non-payload contentions, an estimate of the probability of blocking between the two process classes is required. The blocking level is measured in terms of a factor referred to as qnp, which is the conditional probability of blocking when a payload process runs concurrently with a non-payload process. This probability factor is conditional upon the fact that payload is running on one processor and non-payload is running on another. Measures have been taken in order to reduce the value of qnp, which results in higher capacity of the system. A tool that computes qnp from a trace collected on a multiprocessor-based telephone switch is presented in this Section 6.2.2.

6.2.1 Algorithm for qnp script

The following technique is used to calculate blocking factor between payload and non-payload processes.

Figure 14: Overlap of data domain of a payload & non-payload process running in parallel.

In the Figure 14, PE1 and PE2 are running payload and non-payload processes respectively. Figure 14 illustrates an example that shows that payload and non-payload are running in parallel to each other. A subset of data exists that is shared between them and causes blocking between them. The motivations for the tool discussed in this chapter is to identify the set of shared data and reduce the level of blocking between payload and non-payload processes [Kols97].

An operational definition of qnp is provided by Drweiga [Drw97] and the following expression is used for the estimation of qnp:

$$qnp = 1 - \prod (1 - A_i)$$

Where $A_i = B_i * C_i$

For each location i shared by payload and non-payload.

B_i = (total lock time of i by payload / total payload time),

C_i = (total lock time of i by non-payload / total non-payload time), and denote

$$productA = \prod (1 - A_i)$$

The set i is a collection of memory addresses shared by payload and non-payload processes. As shown in Figure 14, the memory locations shared by payload and non-payload processes can be bundled in a set obtained with the overlap of their data domains.

Lock time is defined as the time period in units for which a process (payload or non-payload) locks a particular memory address during its ownership of the particular address.

Total payload or non-payload times are defined as the execution time, i.e. total time in time units that the respective payload and non-payload processes consume during their execution.

A_i is the probability that a memory location i is held by a process. The value of $\{1 - A_i\}$ is defined as the probability that the memory address is not held by any process, and $productA$ gives the probability of all the memory addresses shared by payload and non-payload processes not being held by any process (no blocking).

Thus the value of $\{1 - productA\}$ is the probability that blocking would occur between payload and non-payload process over the shared memory addresses, where i denotes each shared address. Note that a uniform probability distribution is assumed: a memory

address is accessed uniformly over the time payload and non-payload processes run in parallel [Drw97].

6.2.2 The “qnp” Measuring Tool

A script was developed in the language Perl for implementing the algorithm presented in the previous sub-section. Data is fed into the script in the form of traces containing memory addresses, that is a mix of data for payload and non-payload processes, collected on a real telephone switch. [Drw97] Nortel Networks use the script in the following way. Dump files are collected in the form of memory addresses that were accessed on the real telephone switch. The program script is able to identify context switches that occurred in the system during the run, the memory addresses accessed between one context switch to another, and the scheduler class each process belonged to. Also it has a feature of identifying the type of process, whether it is a payload or non-payload process. With the help of XA-Core team of developers, another procedure has been developed that enables the user to rank contentious data by how often the data blocks. Nortel Networks designers are developing a process called unblocking by which it is possible to re-engineer the existing code and eliminate blocking at a specific set of memory addresses. It is not necessary to unblock all contentious data, rather only critical ones those block over and over again. Contentious data items are sorted by their holding times form a hotlist and the most contentious items near the top of the list are known as hotspots. The tool provides an automated means of generating a hotlist. After converting the location address to program symbols, designers identified the contentious data structures and could evaluate

how critical they were to the blocking removal program. The list of the most contentious addresses referred to as the “hottest addresses” is created. In order to investigate the impact of code re-engineering for removing contention on this address set, the hottest addresses were removed from the input traces. As a part of this study, the tool was modified such that when the program runs through the unblocked traces again, it excludes every time it comes across an address out of the hot address list. The performance calculations were redone every time certain locations were redeemed to be unblocked and removed from the list of contentious locations. With this iterative process, it was possible to arrive at an optimization program that could meet capacity objectives by bringing down the value of qnp. In the experiments done for a particular set of input data traces, the qnp value was reduced from 0.57 to 0.17 with the help of this tool.

6.2.3 Recalculated qnp value

We experimented with two sets of input data one with $qnp = 0.172$ and the other with $qnp = 0.036$. With the reduction in the value of qnp from 0.1727 to 0.036 (with the above technique), the rated available power of the system increases from 3.1 PEs to 4.5 PEs for a 9 PE system. RAP of the system is calculated with the formula discussed in Section 5.2 of chapter 5. Figure 15 presents the RAP values. Capacity of the system shows an improvement because of reduction in blocking levels between payload and non-payload processes. The lower the levels of blocking, the lesser the number of collisions, the greater the amount of useful work done and the higher the capacity of the system.

Figure 15: The effect of decrease in value of qnp on RAP.

Chapter 7

Conclusions

The results from the simulation model of the multiprocessor based telephone switch are presented in Chapter 5. Performance analysis is an important area of study for engineering a system to meet certain performance criteria such as certain level of system capacity. Investigation of rollback strategies on the multiprocessor-based telephone switch, presented in this thesis study, has been extremely useful in the redesigns of the new rollback policies imposed on the system for the purposes of capacity improvements. Also the thesis has been very helpful for Nortel Networks to understand the differences between different rollback charging policies. The conclusions of the study are presented in the following sections.

7.1 Discussion

This section is divided into three subsections, each describing conclusions regarding a particular aspect of system management and performance.

7.1.1 Rollbacks Charging Policy

As shown in Chapter 6, the rollbacks are charged to class in the system model. Due to this fact, the scheduler shares are broken and the process utilization for CALLP, MAINT and AUDIT classes do not meet the actual allocated CPU shares in the scheduler. This is due to the fact that when a rollback is charged to an individual class, it burns up the real time

from its allocated time in the scheduler share and spends it in cost associated with the rollback. This cost incurred by each class is not necessarily proportional to the corresponding class share allocated by the scheduler. Thus the class shares as allocated in the scheduler template cannot be accomplished and are broken. There are three rollback strategies investigated in this thesis. These are the age-based rollback scheme, the bounded-priority rollback scheme and the rollback-payload scheme. Due to the combination of charging rollbacks to class and the design algorithms of rollback schemes, it was noticed that the scheduler shares were broken. This violates one of the most important requirements of the telephone switch.

The factor p_value is defined as a ratio of elapsed time when blocking occurs, measured from the subcall starting time and the execution time. If a process starts and collides right away or blocks in close vicinity of its starting point, its p_value is small. In case of age-based rollback policy, the older job wins the collision. Payload being the class having the largest scheduler share gets the maximum chance to execute. In the case of age-based rollback algorithm, when a fresh payload process starts execution and blocks with a non-payload process, there is a high probability of payload having a smaller p_value as compared to non-payload process. The probability of a non-payload process winning the collision between payload and non-payload process is more, as non-payload process would normally be the older job (due to lower share allocated to non-payload processes). Thus payload is charged more rollbacks the cost of which is not proportional to its scheduler shares. As a result the rollback policy is unable to achieve the designated scheduler shares.

In case of bounded-priority rollback algorithm and rollback-payload algorithm, the combination of rollback algorithms and charging rollbacks to class breaks the scheduler shares as well. Both the rollback schemes involve a concept of serialization of priority 2 processes. While a priority 2 process is running, another priority 2 process cannot run. Other lower priority processes can run but if they collide with a priority 2 process then the higher priority process wins. This factor causing serialization of processes overpowers the allocation of scheduler shares and is a reason for breaking them. This implies that due to the serialization of a priority 2 process, many other priority 2 processes could be waiting in the queue to be run. Since all these processes need to be executed serially, the processors in the system can be forced to be idle for longer periods. This expends CPU time of the system and makes it hard for the scheduler to provide the allocated CPU share for each class.

7.1.2 Scalability of the system

The system is scalable for age-based rollback policy, bounded-priority policy and rollback-payload policy for low (scenario1) and medium (scenario2) blocking scenarios. For high blocking scenario (scenario3), an observation is made that age-based rollback scheme does not work due to extreme blocking. The reason behind this is the fact that age-based policy favors non-payload processes. Details are presented in Section 5.3.3. Though, bounded priority policy results in low RAP of the system, the system is still scalable upto 7 PEs and the RAP drops down for higher number of PEs. Results show that in case of very high blocking (scenario3), rollback payload is not scalable at all and

RAP decreases with an increase in number of PEs. This is due to the fact that the rollback payload favors the non-payload processes.

Thus bounded-priority algorithm is the only one that assures scalable systems up to 9 PEs in case of low and medium blocking scenarios and up to 7 PEs in high blocking scenario. Due to proprietary issues, a statement cannot be made on number of calls that go through the actual XA-Core telephone switch. Thus no further comment can be made on scalability of the switch.

7.1.3 Comparison of Rollback Policies

As said above, none of the rollback policies help in maintaining the scheduler shares of classes. The UPP attained with all the algorithms are comparable to one another. Thus the policies that produce lower capacity for payload give rise to a higher non-payload capacity. This is intuitive in case of rollback payload: by favoring a non-payload process the algorithm gives rise to a higher non-payload capacity, which is accompanied by a proportional decrease in payload capacity. Also, age based favors non-payload processes. Thus, knowledge of age of the process or its class type does not seem to add much benefit to the overall system performance as captured by UPP. Bounded-priority policy performs well in low and medium blocking scenario at high number of PEs and contributes to scalability of the system. In the worst case of blocking (scenario 3), bounded priority policy shows a good performance. It produces low RAP in the system but does scale in systems with upto 7 PEs. Thus bounded priority policy can be stated to perform the best in case of high blocking in the system. Rollback payload policy is the next best policy for

low and medium-blocking cases of input out of the three policies experimented in the system.

7.2 Scope of Future Research

Explained in the Section 7.1 are the conditions that prevent the simulation system from reaching the capacity goals. A valuable area of research would be the design of new rollback algorithms that would ensure none of the above situations that have a tendency of breaking the scheduler shares. Another area that can be explored could be charging rollbacks to system instead of class. Another researcher is extending the present multiprocessor simulation, to be able to charge rollbacks to the system. This would be a way to compare the new rollback-charging scheme with existing rollback schemes and analyze capacity of the system. It also gives a chance to experiment with the new charging system with new rollback policies (being designed at Nortel Networks) at a later point.

The design of rollback policies is a very sensitive area to explore. The thesis results have shown that the combination of charging rollbacks to class and particular rollback designs could break the scheduler shares. This has encouraged Nortel Networks to explore new designs for rollback policies. New rollback algorithms are under study by some developers at Nortel Networks. Due to non-disclosure matters, the details of such algorithms cannot be disclosed.

7.3 My ideas to future research

7.3.1 Algorithm 1

An area of future research that I would like to suggest is towards design of a new rollback algorithm called “Share-check before rollback” where the following rules apply for all types of processes running in the system with priority level 0 and 1. The scheduling layer would require most of the re-designing for this algorithm. At all times, the share accomplished by a given class of processes is noted and calculated as to how far is it from occupying its total share in the scheduler template. The following example is provided to explain the concept of the new rollback algorithm. Consider that blocking occurs between process A and process B, where process A has utilized say 22% of its share out of total allocated share of say 66% in the scheduler template. On the other hand, process B has utilized 14% out of 36% of its share. Thus till the point of blocking, process A has utilized 33.3% of its share whereas process B has taken up 36.6% of its share. Process A is lagging behind process B in terms of utilizing its share and therefore process A should be allowed to win the collision. This way we can assure processes being able to maintain their scheduler shares. Note that in this algorithm, a priority 2 process always wins a collision with a priority 0 or priority 1 process. Also a priority 2 job may serialize the machine as it does in bounded-priority algorithm.

7.3.2 Algorithm 2

This is another algorithm called “Predict completion time” that is based mostly on prediction. Let us define another factor: c_value whose definition is as follows. It is the ratio of time left to the predicted completion time to the predicted total completion time. If there is a collision between two processes of different priority levels, their c_values are computed. The process with lower c_value should be allowed to win the collision. Lower c_value implies that the process is closer to completion than the other process involved in collision. In our case, completion point of a process is not predictable. In simulations, we can do that by checking out the weight associated with each subcall in the memory traces which basically indicates how long the subcall would run for. Simulation measurements can also calculate average work times (AWT) of a subcall. Prediction on the basis of measured AWT of payload and non-payload processes can be done to judge how far the subcall is from completion. For example, in our simulation measurements, measured AWT for a payload process is about 20 time units and varies from +5% to -5% between experiments. Similarly AWT for non-payload also lies within a range of values. Using this knowledge of values, the subcall lengths can be predicted.

References

- [Ben97] Beninger B. "XA-Core: Guide to Multiprocessing". Nortel Networks. 1997.
- [Coff78] Coffman E.G., Carey M.R., Johnson D.S., "An application of Bin Packing to Multiprocessor scheduling", SIAM Journal of Computing, vol.7, February 1978, pp.1-17.
- [Dand92] Dandamudi D., Eager D.L., "Hot-Spot contention in Binary Hypercube Networks". IEEE Trans. On Computers, vol. 41, no.2, February 1992.
- [Drw96.1] Drweiga T. "Calculating capacity degradation by memory contention between non-payload and payload". Nortel Networks. 1996.
- [Drw96.2] Drweiga T. "Shared Memory Contention and its Impact on Multiprocessor Call Control Throughput", 15th International Teletraffic Congress, June 1997, pp.22-27.
- [Drw97] Drweiga T. "Formulas for Calculating qnp", Nortel Networks. 1997.
- [Jain91] Jain R., "The Art of computer Systems Performance Analysis", Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley Professional Computing Publications. 1991.
- [Jal89] Jalote P. "Fault Tolerant Processes, Distributed Computing, Springer-Verlag, New York, vol.3, March 1989, pp. 187-195.
- [Juan91] Juang T. and Venkatesan S., "Crash Recovery with little Overhead", Proceedings of the 11th International conference on Distributed Computing Systems, May 1991, pp. 454-461.
- [Koo87] Koo R., and Toueg S., "Checkpointing and Rollback-recovery for Distributed Systems", IEEE Transactions on Software Engineering, vol. 13, no.1, January 1987, pp23-31.

- [Kim88] Kim, K.H.. "Programmer-Transparent Coordination of Recovery Concurrent Processes: Philosophy and Rules for Efficient Implementation." IEEE Transactions on Software Engineering, vol.14, no.6, pp810-821, June 1988.
- [Kols97] Kolski S. "Determining Data Domain Overlap-Analysis of GSM and NA100 Traffic Data". Nortel Networks, 1997.
- [Maj97] Majumdar S.. "Effectiveness of Scheduling Policies in Controlling Resource Contention in Multiprocessor Systems". December 1999, 21 pages [submitted for publication]
- [Maj99.1] Majumdar S.. "Scheduling Strategies for Controlling Resource Contention of Multiprocessor Systems". Proceedings of the 6th International Conference on High Performance Computing (HiPC '99), Calcutta, India, pp. 151-157, December 1999.
- [Maj99.2] Majumdar S., Streibel D., Beninger B., Carroll B., Verma N., Liu M., "Controlling memory contention on a scalable multiprocessor-based telephone switch". Extended Abstract: Industrial Experiences, Proceedings of ACM SIGMETRICS'99 Conference, Atlanta, May 1999, pp228-229.
- [Pfis95] Pfister C., Norton V.A., "Hot-Spot Contention and Combining in MultiStage Interconnection Networks". IEEE Trans. on Computers, vol. C-34, No.10, October1995.
- [Prop96] "PROPHET User Guide and Reference Manual". Release 2.0.3, Document Issue 1.3, Nortel Networks, 1996.
- [Silb94] Silberschatz A., Calvin P.B., "Operating Systems Concepts (Fourth Edition)". Addison-Wesley, 1994.
- [Sing94] Singhal M., Shivaratri N.G., "Advanced Concepts in Operating Systems". McGraw-Hill Publications 1994.

[Stre97] Streibel Dale. "Simulation Study of a single class multiprocessor system". Nortel Networks, 1997.

[Wood93] Woodside C.M. and C.G. Monforton. "Fast allocation of Processes in Distributed and Parallel Systems", IEEE Trans. On Parallel and Distributed Systems, vol.4, no.2, Feb.1993, pp.164-174.

[Wu90] Wu K.L. and Fuchs W.K., "Error Recovery in Shared Memory Multiprocessor Machines using Private Caches." IEEE Transactions on Parallel and Distributed Systems, vol.1, no.2, April 1990, pp.231-240.

Appendix

Class Inheritance in the 3-class simulator code in our study:

The following diagram shows a pictorial view of inheritance and relationship between object classes as structured in the simulator code. The base class in a Prophet model that defines a task is the class Task. SOS Task is the class that inherits characteristics from the

Figure 16: Class Diagram of object classes used in the simulator.

Base class in Prophet and possesses SOS behavior. The CALLP, MAINT and AUDIT servers are object classes constructed in the code that inherits the attributes and characteristics of a SOS Task. Description of Clients and Servers are provided in Section 6.3.