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.

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

UMI®

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Load Sharing in Call Server Clusters

By

Asif Muhammad B.Sc. (EE)

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

Master of Applied Science

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
August, 2005

© Copyright 2005, Asif Muhammad

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Library and Archives Canada
Bibliothèque et Archives Canada

Published Heritage Branch
Direction du Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

0-494-08383-2

Your file *Votre référence*
ISBN:
Our file *Notre reference*
ISBN:

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

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

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

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

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

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

Load Sharing in Call Server Clusters

Submitted by
Asif Muhammad B.Sc. (EE)

In partial fulfillment of the requirements for
the degree of Master of Applied Science

Chair, Department of Systems and Computer Engineering

Dr. Shikharesh Majumdar, Thesis Supervisor

Carleton University
August, 2005

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

A Call Server in a telephone switch is responsible for handling call processing requests. Call Server clustering can provide scalability, reduced operational cost, fault tolerance and load sharing. This thesis focuses on load sharing in Call Server clusters. It investigates static and adaptive load sharing techniques for effectively handling the load spikes occurring on the system. It has been observed that adaptive policies that use load information of the overloaded Call Server (sender) and the under loaded Call Server (receiver) perform better for a wide range of the load. A detailed simulation model is developed using the OPNET tool. The model is used to study the performance of the Call Server clusters using various load sharing strategies under different load conditions. A comparison of the performances of the load sharing strategies and various insights gained into their behavior and performance are presented.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgments

I would like to express my sincere thanks to my supervisor, Professor Shikharesh Majumdar and the Nortel Networks coordinator, Gerald Kopec, for their invaluable guidance and continuous support over the period of last sixteen months. I would also like to take an opportunity to thank my manager at Nortel Networks, Mark Christopher, for his continuous support and excellent help.

I would also like to thank support and friendship from our RADS lab group members, especially Imran Ahmed, Umar Farooq and Toqeer Israr.

The financial supports from the NSERC and Faculty of Graduate Studies and Research, Carleton University are gratefully acknowledged.

Finally, I would like to show my deep appreciation and thanks to my mother and my wife, for their patience and strong support, without which this research could not have been possible.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Abbreviations and Acronyms

95th POD	95th Percentile Origination Delay
API	Application Programming Interface
CCB	Call Control Block
CS	Call Server
DMS	Digital Multiplex Switch
ENET	Enhanced Network
EngCap	Engineered Capacity
GB	Gigabyte
IP	Internet Protocol
MPLS	Multi-Protocol Label Switching
MS	Microsoft
OC3	Optical Carrier 3
PC	Personal Computer
PM	Peripheral Module
POTS	Plain Old Telephone Service
SCSL	Select Call Server with System Load
SCSO	Select Call Server with Originator Data
TID	Terminal Identifier
UML	Unified Modeling Language
XA-Core	Extended Architecture of DMS Core

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1 Introduction

One way to enhance the power of a system is through deploying multiple processing resources. These multiple processing resources can be provided through a multiprocessor system, parallel input/output system or through a cluster of workstations. Utilizing multiple resources leads to the improvement of performance and reliability. But the performance improvement is at the cost of increased complexity of the system. The major complex issues are data management, load balancing and accessing shared resources and data. This thesis has focused on handling the issues of load sharing in Call Server clusters.

1.1 Motivation

With increasing demands of the telecommunications services, wireline and wireless, the voice and data call switches have to supply a large call handling capacity. There are two new technologies that have increased the switching capacity of telecommunication servers. One is the internet. There has been an explosive increase in the usage of internet in the recent years. The other is mobile technology, which is becoming an integral part of every day life. Because of these, the switching capacities of the existing Call Servers are being pushed to their limits. Another important issue in the context of telecommunication systems is fault tolerance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In order to handle the increased capacity demand, provide the required fault tolerance, and minimize the operational costs for the phone companies, Nortel Networks and Carleton University are jointly investigating the performance advantages of Call Server clustering. Such clusters are to be able to work with both circuit switched networks and Voice over IP networks. In the context of this thesis, a *Call Server* (CS) is an element consisting of a call control engine that performs call processing. A Call Server cluster is a group of multiple individual Call Servers covering a designated geographical area. Call Server clustering provides the advantages of scalability and reliability.

Load sharing in a Call Server cluster can be achieved at two different levels. The first level can be provided by installing different peripherals to different Call Servers in a cluster. Every peripheral is associated or owned by one Call Server, so it can initiate calls for only those subscribers who are wired to that peripheral. Thus traffics from multiple peripherals are directed to multiple Call Servers. The next level of load sharing can be achieved at the Call Server itself. Call Servers in a cluster may be subjected to different loads at a given time. One of the Call Servers in the cluster may be overloaded while there are other Call Servers that are operating below their engineered load level. In case of such an unbalanced situation, it is possible to distribute the traffic across the different Call Servers in the cluster. This level of load sharing prevents a single Call Server becoming the performance bottleneck. This also provides fault tolerance. When one of the Call Servers in the cluster fails, the service provided by failed CS can fall over to another CS in the cluster automatically. When the failed Call Server is restored, the service can again be restored to it. This mechanism prevents having a single point of failure in a Call Server cluster and minimizes the impact of single Call Server failures.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.2 Research Outline

This thesis focuses on load sharing in Call Server clusters for achieving high system performance. There are generally two terms used in the context of distributing the load among different nodes of a distributed system. One is termed as *load balancing* and the other is usually known as *load sharing*. Is there any difference between these two terms? In literature, most of the time, these two terms are used interchangeably. But some authors [Fer86] have defined load balancing as a finer approach that equalizes the load among the nodes of a distributed system. On the other hand, load sharing is a coarser form of load distribution, where load is only transferred from overloaded nodes to idle nodes without attempting to equalize the loads of all the nodes.

Majumdar, Sun and Kopec have done some initial work on Call Server clusters [Maj04]. In a Call Server cluster, any Call Server can be selected to process any call, so the first action on receiving a new call is to select an appropriate Call Server that will process that call. The concept of Call Server cluster and the selection of a Call Server for handling a new incoming call in order to minimize remote data access are discussed in [Maj04]. The proposed Call Server selection strategies are based on the characteristics of the call. The Call Server, which has the ownership of the state data of the originator of the call, is selected for processing the new call.

In this thesis, we devise strategies which also consider the characteristics of the system to select a Call Server. In situations of load imbalance on different Call Servers of a cluster, the CS subjected to high volumes of calls may become overloaded and can cause degradation of the overall performance of the CS cluster. In this situation, we propose to select the Call Server based on the system load. Selection of Call Server based on the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

system load may use various load sharing strategies. In this thesis, we devise different load sharing strategies, static as well as adaptive, to distribute the load from heavily loaded CSs to the lightly loaded CSs. The performance of these strategies is investigated for different unbalanced load situations ranging from the small overloading levels to very large overloading levels.

1.3 Thesis Contributions

This thesis focuses on the incorporation of load sharing strategies in the Call Server clusters. The thesis covers distributed redirection strategies for distributing the new incoming calls to different Call Servers in the cluster in detail. The main contributions are summarized.

1. The thesis devised various strategies for load sharing in a Call Server cluster. Policies based on both static and adaptive techniques are investigated.
2. Based on the architectural design of current Nortel telephone switches, a simulation model using the OPNET tool is developed.
3. The different load sharing strategies are compared for different types of overloading situations. Adaptive policies are found to be more effective for a variety of different load compositions.
4. Effects of different system and work load parameters on performance are analyzed.

1.4 Thesis Composition

This thesis is composed of six chapters. Chapter 2 discusses some internal details of telephone switches, provides a background for Call Server clusters, and related work

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

concerning different components of load sharing. The devising of different load sharing strategies for Call Server clusters is described in detail in Chapter 3. Chapter 4 discusses the design and implementation of the simulation model, workload and system parameters as well as the performance metrics. Chapter 5 provides a thorough comparison and analysis of the experimental results for different workload parameters. Conclusions are presented in Chapter 6.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2 Background and Related Work

This chapter has three main sections. The first section gives a high level description of uni-processor and multi-processor based telephone switches. The scope of the second section is cluster computing, clustering of Call Servers and the related work in the field of Call Server clustering. The topic of the last section is load sharing, in which a brief overview of load sharing is presented first and then a detailed literature overview is given for different components of load sharing.

2.1 Telephone Switch

Call Server cluster can be built with various telephone switches. As an example, we describe a telephone switch on which our simulation model is based. The earliest version of Nortel Networks telephone switch was developed in 1970. The design of that switch used a single processor for call processing. This uni-processor based switch, also called Digital Multiplex System (DMS) Super Node, was not capable of handling constant demand of increased call volumes and advanced call services. So in the recent past, multi processors based telephone switches were designed. Nortel Networks extended the architecture of the DMS switch to make it a multiprocessor based switch. The following sections describe the architecture of the basic telephone switch and then the extended architecture (XA) of Nortel's telephone switch.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1.1 Architecture of the DMS Switch

The DMS telephone switch includes a Call Server and a *switching network* [Nor02]. The Call Server performs call processing, as well as various activities relevant to billing and maintenance whereas the switching fabric is responsible for providing a path for the connection of two subscribers. Once the identifications of both the originating and terminating subscribers are known, the Call Server instructs the appropriate gateway controllers to set up the path between the subscribers using the switching network.

A very high level architectural overview of a DMS telephone switch is presented in Figure 2-1 [Kei95]. The architecture of the switch is divided into three layers. Each of these layers is described in one of the next sub-sections.

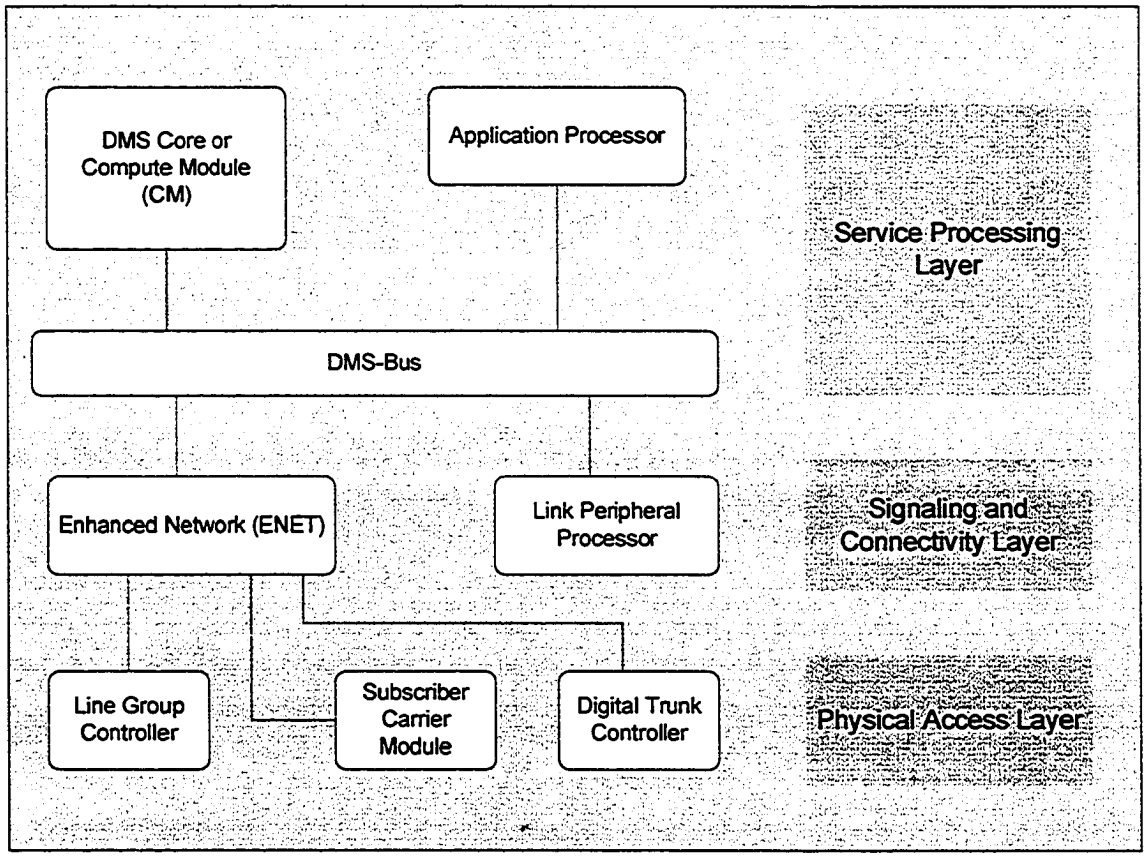


Figure 2-1: Architecture of a DMS telephone switch [Kie95]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Service Processing Layer

The service layer includes Computing Module or DMS-Core, the Application processor and the DMS-Bus. The *DMS-Core* is the computing and memory resource of a telephone switch. It also performs system management and billing related tasks in the DMS SuperNode applications. The *Application processors* provide Line and Trunk services, Network services and Operation Management functionality. The *DMS-Bus* interlinks different elements of the switch through a high speed inter-processor communication system. DMS-Bus provides a distributed internal message network and eliminates the need of any central control for inter-processor messages.

Signaling and Connectivity Layer

This layer contains two main components. First is the Link Peripheral Processor (LPP). LPP provides the protocol structure used on signaling links for inter-nodal communications. The second main component of this layer is the Enhanced Network (ENET). The ENET provides functionality of an interconnection network.

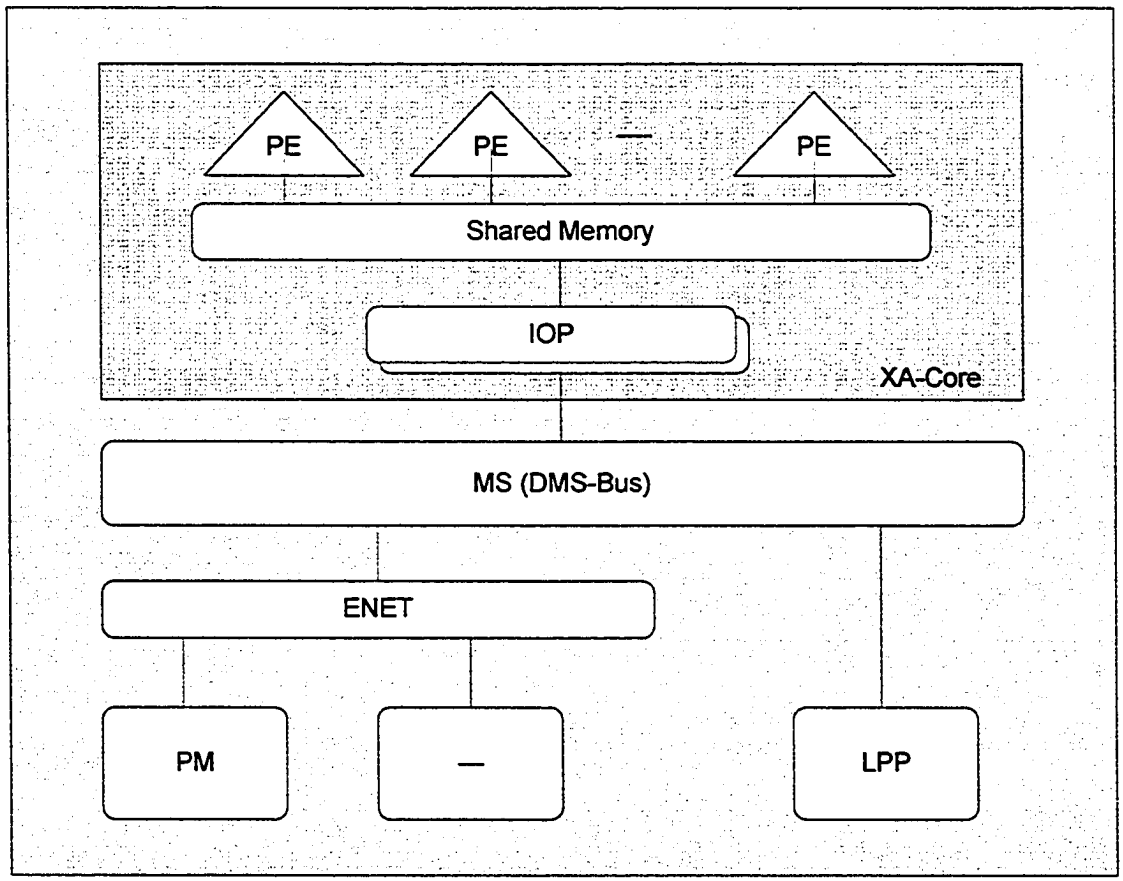
Physical Access Layer

This layer mainly contains Line (Line Group Controller, Line Connecting Module) and Trunk (Digital Trunk Controller) interfaces for both analog and digital facilities. The main component of this layer is the Subscriber Carrier Module that is generally referred to as the *Peripheral Module (PM)*. The Peripheral module provides a bridge between the DMS system and the voice and data traffic over trunks and lines. The PM is used for scanning lines for changes of circuit state; providing timing functions used for call processing, creating dial tones, sending, receiving signaling and controlling information to and from the DMS-Core.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1.2 The eXtended Architecture (XA) Core

The XA-Core based telephone switches use multiple processors for call processing (see Figure 2-2). The design of the state of the art XA-Core is such that it extends the existing architecture of the conventional DMS switch to a shared memory multiprocessor based platform. Only the DMS-Core part is replaced by the XA-Core architecture [Nor01].



PE → Processing Element
MS → Message Switch (DMS-Bus)
ENET → Enhanced Network
IOP → Input/Output Processor
PM → Peripheral Module
LPP → Line Peripheral Processor

Figure 2-2: XA Core architecture

The XA-Core consists of three major components; Shared Memory, Processing Elements (PEs) and the Input/Output processing elements (IOPs) (see Figure 2-2). These three

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

components communicate with each other through a common network called the Interconnect (not shown in Figure 2-2).

2.1.2.1 Shared Memory

Shared memory is the direct replacement of the CM (control module) memory. It is fully addressable by every processor and the data in the memory is shared by all processors. The ownership of shared memory is dynamic in nature and it prevents multiple processing elements to read or write the same memory locations at the same time. The network that allows any IOP or PE to access any memory module is the *Interconnect*. A detailed description of the design of shared memory is provided in [Ben98].

2.1.2.2 Processing Elements

A processing element includes a CPU, local memory and cache. A series of processing elements (PEs) provide general purpose processing power to XA-Core except Input/Output functionality. A time sharing scheduling policy is used. All PEs are functionally equivalent and execute the same software load. A software program achieves the same result no matter what PE it turns on. There is no special designation of any PE for a specific task. Any PE can perform any task. The advantage of this approach is that all PEs are loaded equally and processing load is balanced naturally.

2.1.2.3 I/O Processing Elements

The Input/Output interfaces in XA-Core are provided by I/O processing elements (IOPs). These devices provide three types of communications.

1. It provides Optical Carrier 3 (OC3) links to the Message Switch for regular communication with the rest of the DMS system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2. It supports disk and tape units for all DMS mass storage.
3. It can also provide links to other external high speed communication interfaces such as Ethernet and X.25.

Scheduling of multiprocessors is achieved through a scheduler. Once an active process is finished, then the scheduler is invoked to select the next process waiting to be executed in the queue. There is no specific rule that which process to run on which PE. Any process can run on any PE [Ben98].

2.2 Clustering of Call Servers

In this section, a brief introduction of different types of cluster computing is presented first. This is followed by a discussion of previous work on Call Server clustering.

2.2.1 Cluster Computing

The recent advances in high speed networks make it possible for clusters or networks of workstations to be more a cost effective parallel computing systems. Clusters are usually classified into three main categories: high performance computing, high throughput computing and high availability computing.

The *high performance computing* (HPC) class is concerned with how quickly a job can be completed. This can be achieved by using parallel processing on multiple high end processors [Buy99].

The *high throughput computing* (HTC) class focuses on how many jobs can be completed per unit time. The key to high throughput computing is to efficiently utilize all available computing nodes without adding new hardware resources [Liv97].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The *high availability* (HA) *Computing* class is concerned with the availability of resources in the cluster to handle a partial or nodal failure of the system. There are different ways to achieve high availability either by utilizing backup hardware or by sharing the resources owned by other nodes [Rob05].

This research mainly falls in the area of high throughput computing (HTC). Maintaining a desired level of latency is also an important requirement. Providing high availability is a secondary area of research. In case of accidental failure of one of the Call Servers in the cluster, other CS new to be deployed to serve the lines handled by the failed CS.

2.2.2 The Architecture of Call Server Clusters

The Call Server, which is deployed in the simulation model used in this thesis, is based on the XA-Core based DMS telephone switch. In the core part of the Call Server, a new process called *broker* (see Figure 2-3) was introduced in [Sun04]. The broker has been enhanced in this thesis to provide load sharing functionality (discussed in detail in Section 4.2.2).

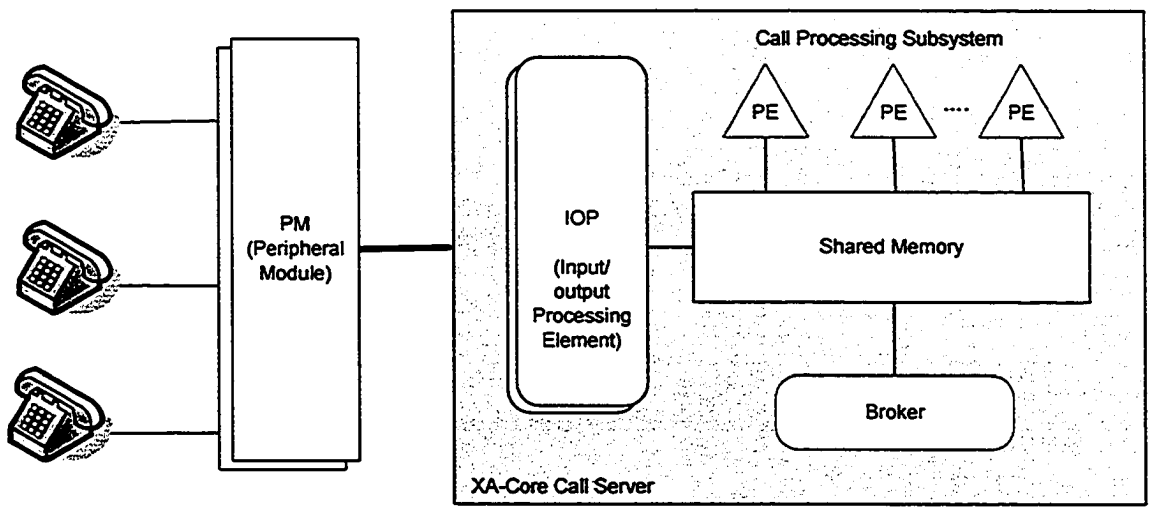


Figure 2-3: Modified architecture of XA-Core based DMS switch

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Figure 2-3 does not show those components of the switch that are not required in the simulation model. For simplicity, a PM is directly connected to the Call Server. The broker is an independent process like IOP and handles new incoming calls as they become available to it.

To build a Call Server cluster, multiple Call Servers are connected through a high speed network [Sun04]. All the communication between different Call Servers is handled by the IOP of each CS. A cluster of four Call Servers is shown in Figure 2-4.

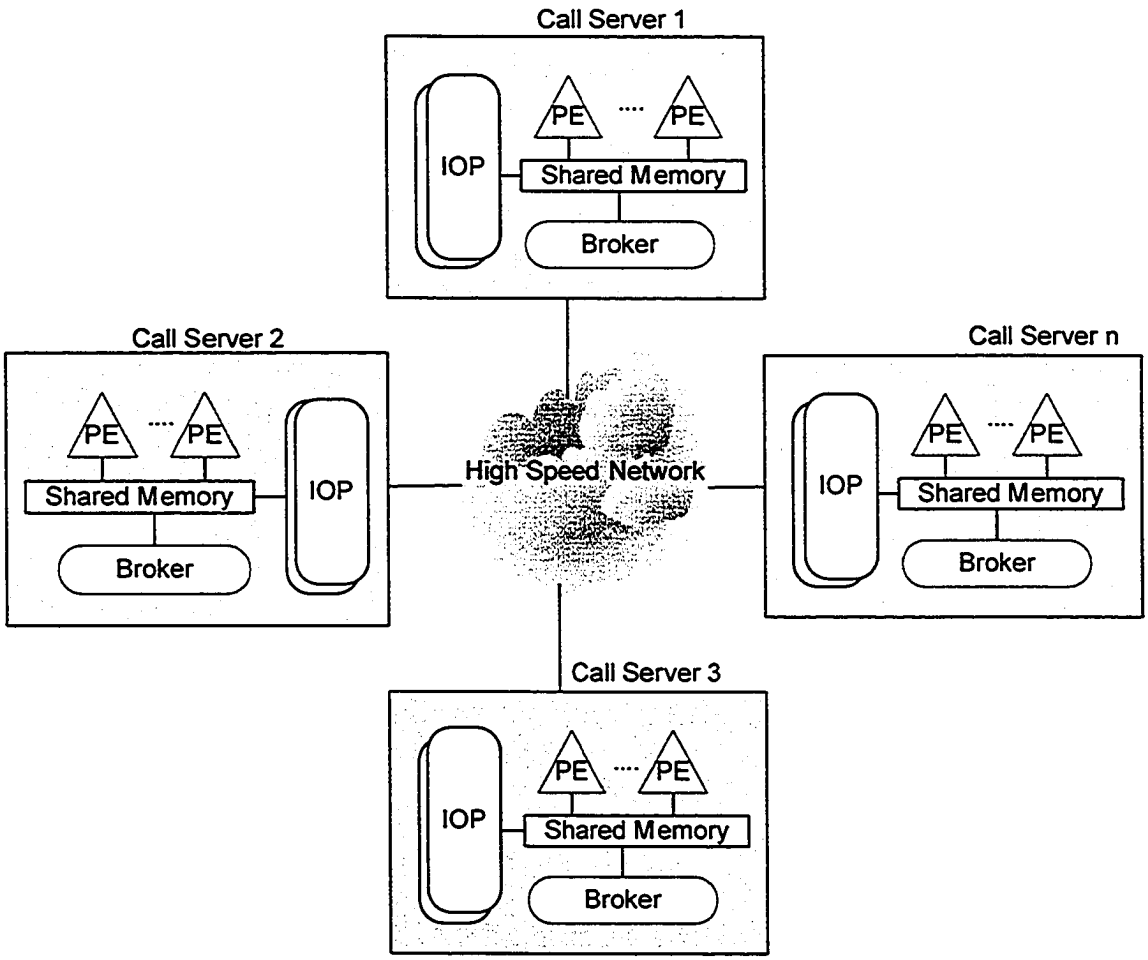


Figure 2-4: Call Server cluster

2.2.3 Objectives of Call Server Clustering

The following are the main objectives and advantages of Call Server clusters [Maj04].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

High Capacity: The Call Server cluster will be able to handle a higher number of calls as compared to a single Call Server.

Single Logical Switch View: A Call Server cluster-based system includes multiple Call Servers but functions as a single logical system. Such a cluster can be used to serve a larger geographic area than that currently handled by a single conventional CS.

Cost Reduction: Operational and maintenance cost will be reduced significantly, as the Call Server cluster can be viewed as a single telephone switch.

Scalability: A CS cluster may start with a small number of Call Servers and additional Call Servers can be incrementally added as the demand on capacity grows

Fault Tolerance (including Geographic Redundancy): Since any call can be processed by any CS in the cluster, the lines handled by a particular CS can be distributed among other Call Servers in the cluster when a Call Server fails. If the Call Servers are located sufficiently far from one another, the CS cluster can gracefully handle a Call Server failure due to a natural calamity or a terrorist attack.

Load Sharing: When incoming call volume is too high to achieve a desired performance level, the current independent Call Servers start rejecting new calls. In a Call Server cluster, new incoming calls on a heavily loaded Call Server can be forwarded to a lightly loaded Call Server that still has the spare capacity to handle new incoming calls.

2.2.4 Selection of Call Servers

There are number of issues that have to be addressed for clustering of Call Servers. These include the selection of a Call Server for call processing, communication topologies among Call Servers within a cluster, load sharing among Call Servers as well as billing and maintenance. In the first phase of this research, Majumdar et al. [Maj04]

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

investigated different Call Server selection strategies. These Call Server selection strategies lead to different data management policies among the Call Servers of cluster. There are two types of data in a telephone switch: static and dynamic. Static data does not change with time or changes infrequently in comparison to dynamic data. Examples of static data include subscriber features like call display or three way call features. Dynamic data includes data related to a subscriber state that changes when the subscriber participates in a call. As described in [Maj04], the static data for the entire system is replicated on all Call Servers but the dynamic data is only owned by a single Call Server. There is a provision that this ownership can be passed to one or more Call Servers, if that Call Server fails. Connection between subscribers that belong to a different Call Server cluster is performed via trunks that interconnect the different Call Server clusters. When a call needs to be set up between any two subscribers in a cluster, any one of the Call Servers in a cluster can be used to perform the call processing function.

Selection of Call Servers can be performed in four different ways. First, an independent strategy is proposed. With this strategy, a random selection of the Call Server is done for every new incoming call request. The second proposed strategy is based on the characteristics of the call. This strategy is investigated in detail by Majumdar et al. [Maj04]. Three different Call Server selection policies that are based on the originator's data are proposed. This class of policies is called *Select Call Server with Originator Data (SCSO)*. These SCSO strategies are described briefly in the next section. The third strategy for selection of a Call Server can be based on the characteristics of the system. The load on a Call Server is an example of system characteristics. A fourth strategy that uses both call and system characteristics to select a Call Server is considered in this

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

thesis. The main objective of this thesis is to investigate techniques for distributing the load among different Call Servers of a cluster based on their state.

2.2.4.1 Call Characteristic-Based Selection Strategies

As mentioned earlier, the static data is replicated on all the Call Servers. The dynamic state data corresponding to every subscriber is partitioned among the Call Servers. The state data for any subscriber is available initially on the Call Server that is connected to the PM that handles the given subscriber. The major motivation behind the devising of the SCSO strategies is to reduce the data access latencies. In all the three SCSO policies, the Call Sever which owns the state data for the originating subscriber is selected for processing of that call. The difference among the strategies stems from the way the state data for the terminating subscriber is handled when the terminator state data is not local and is owned by a different Call Server. Majumdar et al. [Maj04] have proposed three different strategies for accessing the state data of the remote terminating subscriber.

Remote Access: Whenever there is a read or write operation for the terminator data, the Call Server processing the call will send a message to the remote Call Server that owns the terminator state data.

Replication: In the replication policy, a copy of the terminator data is replicated on the current call processing server. This facilitates read operations from the local copy of the terminator data, but whenever there is a write operation, a message has to be sent to the remote Call Server.

Migration: In this strategy, the terminator state data is migrated from the remote Call Server to the current CS processing the call. This facilitates the current CS to do read and write operations locally. However, if another CS wants to access the terminator data, the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

request is forwarded to the CS currently owning the terminator data. When the call terminates, the state data for the terminator subscriber is sent back to the CS that is the original owner of the state data for that subscriber.

Simulation results have shown that the migration policy is marginally better than replication policy, and remote access policy is marginally better than both of replication and migration policy for a class of call processing transactions that perform a single read and a single write operation on the terminator data [Maj04]. In case of higher number of read and write operations, the migration policy has shown better performance in comparison to the two other policies.

2.2.4.2 System Characteristics-Based Selection Strategies

The goals and objective of this thesis are to explore Call Server selection strategies based on the characteristics of the system. The motivation is to distribute the load among the different Call Servers. The selection of a Call Server for a new incoming call is based on the loads of the different Call Servers in the cluster. This selection of a Call Server is performed by the overloaded node. For this purpose, Call Servers exchange state information with each other. The broker process in the core architecture of the Call Server is used to handle load sharing and to share load information. When the load on a Call Server exceeds a predefined threshold, then the broker tries to locate an appropriate receiver CS, which is running under the designed capacity. After identifying a receiver CS, the next most important step is to determine how much load is to be transferred to the receiver CS for this run of the broker. There are a number of redirection strategies that are proposed and investigated in this thesis. The details of these strategies are documented in Section 3.2.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Once a Call Server is selected as a receiver and a new incoming call is forwarded to it, an important issue is how to access the originator's dynamic data. There are again three options for accessing originating subscriber's state data. As discussed in the previous section, three different ways to manage state data of the terminating subscriber are proposed in [Maj04]. So for this thesis, there is an option of exploring all those three policies of data sharing for originating subscriber or choose the best policy based on the results shown in [Maj04]. In this thesis, we used the remote access policy as our default data sharing policy. But we have also investigated the other two policies to see their effects on system performance when load sharing is used.

There are different possible indicators of the load on Call Servers that includes response time (e.g. dial tone delay) for the new incoming calls or the average CPU utilization of a Call Server. Most of the work in the literature that focuses on load balancing or load sharing of conventional distributed processing uses the size of the waiting queue of jobs as a load indicator. But for a Call Server cluster, no such parameter which can be directly related to the load of a Call Server is available in the literature. In telephone switches, the 95th percentile delays are considered useful as an indicator of system performance [Ben98]. From the context of new incoming traffic, the 95th percentile origination delay is the best indicator and is in use for the detection of overload in Nortel's telephone switches [Kop05]. The *origination delay* is the time required by a new call messages to reach the call processing subsystem. A more precise definition of the term is provided in Section 4.3.3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.2.5 Related Work

There is hardly any work done in the area of clustering of Call Servers. Majumdar et al. [Maj04] started this research with collaboration of Nortel Networks in 2003. They have explored various aspects of call characteristics based selection of call servers in such clusters.

Cisco has also introduced clustering of callManagers for IP phones [Cis01]. The Cisco CallManager is the software-based call processing component of the Cisco enterprise IP telephony solution. The capacity of a single CallManager based server is only 7500 IP phones. Cisco CallManager clustering can yield scalability up to 30,000 IP phones per cluster. The subscribers are statically assigned to Call Managers that perform call switching. The CS clusters addressed in this thesis are expected to be scalable to hundreds of thousands of subscribers and perform intelligent dynamic Call Server selection as well as load sharing at run time while significantly reducing the operational and maintenance cost.

2.3 Load Balancing

There is a large body of literature available in the domain of load balancing or load sharing on conventional distributed systems. A representative set of the existing works is presented in this section.

2.3.1 Overview

Load sharing techniques are generally classified into two main categories; static and dynamic load sharing. Static techniques use predefined formulas or numbers without involving any run time state information of the system for decision making. The simple example of static load sharing is the distribution of load by a single broker on a round robin scheme. Static load sharing can only be effective, if there is only one centralized

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

broker or a task dispatcher and tasks are of the same size (equal processing time). This approach is not considered very useful for a distributed load balancing [Sit04, Dah99].

Dynamic load sharing techniques use state information of a system on the fly for decision making. The decision making is involved at different levels [Dah99]. For example, it is important to decide when to transfer the load from an overloaded node, where to transfer the load, and how much load to transfer. Dynamic load sharing techniques are also referred to as adaptive load sharing techniques in the literature [Eag85, Sit04, Das97]. In adaptive load sharing techniques, the performance of the system can be used to adjust the load sharing at run time. The two common strategies for adaptive load sharing are discussed and compared in [Eag85]. The *sender initiated* approach is applied to the overloaded nodes (senders) that try to locate a lesser loaded node for load sharing. But in the *receiver-initiated* approach, the light loaded nodes (receivers) search overloaded nodes (senders), from where load can be transferred. The sender-initiated approach is considered better when the loads of the nodes are in the range of light to moderate, whereas a receiver-initiated approach is better suited when the loads on different nodes are on the heavy side.

Load sharing has been investigated in the context of various environments. Chang et al. [Cha01] have proposed a two stage load balanced switch to scale up with the speed of fiber optics. They have shown that load balancing is more effective than the other approaches such as i-SLIP, a conflict resolution algorithm. In [Lon01], it is seen that for the MPLS traffic, dynamic load sharing algorithms are better in performance than the shortest path algorithm. Long et al. [Lon01] have also shown that dynamic load balancing algorithms are effective for both light and heavy load conditions.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.3.2 Literature Review

The load sharing policies available in the literature generally consist of four components. These are: the transfer policy, the selection policy, the location policy and the information policy. Related work on these individual components of load sharing is presented separately in the next sub-sections.

2.3.2.1 Transfer Policy

Transfer policy determines when to transfer the load to the other node. Normally, a load dispatcher or a broker makes this decision based on the load information of the local node. In [Bal04], different transfer policies are discussed. Two simple and effective policies, in addition to round robin and random, are based on a fixed threshold and an adaptive threshold. The former policy compares the current load information of the local node to a predefined fixed threshold. If the current load goes above the fixed threshold, it starts looking to transfer some of its load to another node. The dynamic or adaptive threshold based transfer policy first determines a threshold based on the load information of all the nodes in the system. Then this threshold is used in the decision making by the transfer policy. If the current load of a node is higher than this 'evaluated' threshold, then it starts transferring load to a lightly loaded node. The dynamic approach has an overhead of calculating the dynamic threshold. It is also observed that the simple fixed threshold policy gives better results than adaptive threshold based policy [Kru91].

2.3.2.2 Selection Policy

Selection policy determines which task is to better execute locally or remotely. There are two types of tasks for transfer: preemptive and non-preemptive. *Preemptive* tasks are those tasks which can be transferred at any time during the execution of the task. *Non-*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

preemptive tasks are newly arrived tasks which have not yet started execution. The scope of this thesis is only limited to *non-preemptive* tasks, because only new incoming calls are considered for the load sharing in Call Servers. Once processing of a call starts then it is not efficient to migrate the processing to another CS. Most of the work in the literature [Ju95, Wan93] for selecting new jobs is based on the criteria that use different characteristics of the task. These characteristics include memory requirement, frequency of I/O access and length of the execution time of the task [Wan93]. For example, if the new task has more I/O operations on the local disk, then it would not be good to transfer such task to a remote node. Similarly, a task that has shorter execution time is more suitable to execute locally, because the overhead of transferring such task may override the benefit [Wan93, Mar02]. Wang et al. [Wan93] have used intelligent techniques to select a task for transfer based on the knowledge of the characteristics of the task if it is run locally or remotely. In case of Call Server clusters, one such characteristic is the ownership of state data of the originator subscriber. Majumdar et al. [Maj04] used this locality characteristic of the subscriber's state data as selection criteria in the design of Call Server cluster. As already mentioned in Section 2.2.4, every subscriber's state data is owned by one Call Server only making it a default choice for call processing. In this thesis, the work of Majumdar et al [Maj04] is extended to use load sharing aspects within the Call Server cluster. The default selected Call Server for new incoming calls is again the Call Server that owns the originator subscriber state data. But if this owner Call Server is overloaded, then it can transfer the call to a remote Call Server within the cluster. Since the new calls available to process are in the form of a batch, the selection policy under consideration is not to decide the suitability of a single call to be processed

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

locally or remotely, but to deal with a number of call processing tasks that should be transferred to the remote Call Server. So in this thesis, the selection policy is a *redirection policy*, which determines the number of new incoming calls that should be redirected to other Call Servers to balance the load within the cluster.

2.3.2.3 Location Policy

In the literature, location policies are broadly categorized as sender-initiated, receiver-initiated and symmetrically-initiated. In the sender-initiated policies, the potential senders search for receivers where the load can be transferred while in the receiver-initiated policy, potential receivers search for senders. In a *symmetrically-initiated* policy, both senders and receivers search for complementary nodes. Sender-initiated has been extensively explored in [Bry81, Eag88, Ram84, Rot90], whereas details of the work for receiver-initiated policies can be found in [Liv82, 1, Rot90]. Rotihor et al. explored symmetrically-initiated policies in the [Rot90]. Kruger [Kru88] has also shown the advantage of using receiver-initiated policy over others at high loads due to the fact that the cost of search by a sender-initiated policy is more at very high load. But Kruger [Kru91] et al. have also shown that receiver-initiated policies are also prone to failure, particularly when arrival rates are not homogenous across the nodes of the system. In such a situation, it is quite difficult to locate overloaded nodes in the system. As a result, there is not much load sharing and the offered capacity of the distributed system will be lower than its computing capacity. Sender initiated algorithms perform better with non-homogeneous arrival rates, provided a suitable receiver is found efficiently. Symmetrically-initiated policies perform better than the other two, but these are also prone to failure in case of heterogeneous arrival rates across the nodes in the system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Krueger et al. [Kru88.1, Kru88.2] have proposed adaptive location policies by combining the benefits of all three: the sender-initiated, receiver-initiated and symmetrically-initiated strategies.

The key aspect of the sender-initiated location policy is the criteria of selecting a receiver. The first step is to look for potential receivers in the system. A threshold based approach is more common for selecting a potential receiver [Shi90]. If the load on the node is less than a certain threshold, then that node is considered as a potential receiver. The next step is to choose a single or multiple receivers from the list of potential receivers. The very important issue here is that multiple senders can choose one receiver at the same time, and start transferring tasks to it simultaneously. This will result in overloading of the receiver. This problem can be minimized by probing the receiver before transferring the load to it. Lee et al [Lee96] have proposed a prediction based adaptive location policy to handle this problem. They assumed all the nodes are connected in the form of a ring, and the sender will look from the list of potential receivers which is ahead of it in the ring. In [Gen00], Genova et al. suggested to select k receivers from the list of potential receiver randomly and then select the least loaded receiver from the list of k receivers. Eager et al. [Eag85] selected a node randomly and probes it to determine if it is a potential receiver based on the threshold policy. If it is a potential receiver, then load is transferred otherwise another node is selected randomly. This process continues until a suitable receiver is found. Eager et al. have also considered various location policies spanning a range of complexity and concluded that the use of a complex location policy yields only a slight improvement over the use of a simple location policy.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.3.2.4 Information Policy

The fourth component of load sharing is the sharing of load information among all nodes in the distributed system. There are three different types of information policies. The first approach is the *periodic* information policy, in which either every node transfers its load information to other nodes after a fixed period of time or every node probes other nodes in the system on a periodic basis to collect the load information. The communication cost of this strategy, for one run of sharing load information, is $\sum_{n}(n-1)$ messages, where n is the number of nodes in the system. The communication cost can be reduced by adopting a token based approach. All nodes will transmit load information to only one node which will have the token at that particular time. Once all load information is collected by the node with the token, then that node will broadcast complete load summary of the system to other nodes in the system. The communication cost for this type of load information sharing is *2(n-1).* One issue with this modification may be the extra delay introduced because the load information reaches to all destination nodes in two steps. As a matter of fact, a periodic approach always has communication cost more than any other approaches, but advantage is its simplicity and accuracy of the load information

The communication cost can be reduced significantly in a *demand driven* information policy. In this policy, load informations from other nodes are fetched only when they are required. If a node is overloaded, then it probes other nodes in the system to find out the best receiver, to which it can transfer the load. One disadvantage of this approach is the extra delay introduced, because the sender has to collect load information of other nodes at run time and this aspect of this policy might not be a very encouraging for systems

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

that has to make decision on the fly without waiting. One example of such a system is the Call Server cluster.

The third type of information policy, known as the *state change* based policy, tries to combine the good aspects of the previous policies. Every node evaluates its load state periodically, but broadcasts its load information only if the change in the load state is more than a fixed threshold. The main issue regarding all information policies is the accuracy of load information. The load information available to the sender at the time of load sharing decision should be consistent with the current load status of all nodes. Load balancing can degrade overall system's performance if the load information shared is not accurate enough. Dahlin et al. [Dah99] proposed load sharing policies which also consider aging of the load information. For example, if there are two nodes with the same load index, then the node, whose load information is more recent, receives more weight in decision making.

2.3.2.5 Load or Load Index

It has been seen that all dynamic load sharing policies use load state information as a key input for decision making. This load or state information is generally called *load index* or simply the *load.* It is important to choose the appropriate load indicator for a given system. But there is not much work available in the literature for selecting an appropriate parameter as a load. Many indices are suggested to represent the load on a node. The most common parameters are the CPU utilization, the length of the ready queue, the stretch factor (ratio between execution time of a task on a loaded machine and its execution time when the machine is idle), amount of memory available, average number of system calls per unit time and some derived forms of these basic variables. Although

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the most commonly used load indicators are the length of the ready queue [She02] and CPU utilization [Yos03], other indicators have also been used by researchers. For example, Ferrari et al. [Fer86] have investigated a response time based load index. Their hypothesis is based on the objective of the load sharing scheme to minimize the mean response time of the system. Jin et al. [Jin01] have introduced a workflow load index for distributed workflow management system. They used process execution times, an active process instance number and the workflow engine latency between two adjacent activities of a process instance to formulate the load index.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3 Load Sharing in Call Servers

This chapter is organized into two sections. Section 3.1 provides the technical overview of the different load sharing components in a Call Server cluster. Section 3.2 describes the design of the proposed redirection policies.

3.1 Load Sharing Components

The architectural design of Call Servers does not permit to plug in conventional load sharing strategies in the Call Server cluster. For example, the new tasks (incoming calls) are not served one by one, but these get served in a group or batch [Ben98]. So in case of Call Servers, the focus of the selection policy is not to decide whether a certain task should be executed remotely or locally. Rather, it deals with the number of new calls to be redirected from a sender Call Server to a receiver Call Server. That is why we refer to the selection policy as a redirection policy in the context of this thesis. In conventional load sharing strategies, the redirection policies are not explored much in great detail. Another difference in the context of Call Server cluster is that the load status of Call Servers is usually not determined by the size of ready queue or CPU utilization on a telephone switch. The 95^{th} percentile of origination delay is used as a load indicator of a Call Server.

Call Servers are designed for handling a specific input traffic intensity. In this thesis, a number of redirection policies are designed and investigated. The policies are based on

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the relationship between the origination delay and the input traffic. The policies for other components of load sharing are either adopted from the literature as is or are modified according to the requirements of Call Servers. A detailed discussion of redirection policies is presented after a brief description of the other components of load sharing.

3.1.1 Load Index

A candidate load index should be easy to compute and it should also be related to the main performance objective of the system such as the system's response time. It is seen that simple load indices are often more effective. The more complicated load indices usually do not give more benefits and sometimes also impose significant overheads. One effective load index is the length of the ready queue. Some systems such as Utopia [Zho92] and Condor [Lit92, Lit97] also use multiple load indices for making load sharing decisions. In case of Call Servers, there are principally two types of queues where tasks wait to get service. These queues are called the *origination queue* that holds the tasks for new calls and the *progress queue* that holds the tasks for calls in progress. The performance goal of the system is to minimize waiting time of new incoming calls, but the priority of the tasks on the progress queue is always high. Also the tasks waiting on the progress queue vary in execution times [Ben98]. Because of these reasons, it is difficult to find a direct relationship between the origination delay and the length of the origination queue. Another reason which makes it difficult to find a relation between the origination delay and the length of origination queue is that the tasks are processed in the form of batches. Thus, the origination queue is not a good indicator of the load on that particular Call Server. Current version of a number of telephone switches from Nortel use a moving 95^{th} percentile of the origination delay as the load indicator on a Call

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Server. In this research thesis, we also used the same parameter, moving 95th percentile origination delay (95th POD), as a load index.

The reason for using the moving 95th POD is because the average value of the origination delay or the simple 95th percentile origination delay lacks the precise and accurate indication of the current load on the Call Server. This is because of the lagging nature of these types of calculations. For example, if we use the average origination delay or the 95th POD, then after some period of time, it would not be possible to accurately capture rises or drops in the load status as soon the change occurs. It will capture a rise or fall in the load only if that change persists for some period of time. However, it might be too late for the system to appropriately respond to it. Similarly, if we use the instantaneous origination delay directly as the load indicator, then the system can respond to all types of transients in the load that may lead to an unstable system. So we use a moving 95th percentile origination delay as load index.

The calculation of the moving 95th POD is based on the statistics of the origination delay collected during a short period of time (T_{moving}). T_{moving} is referred to as the *Window Size* in this thesis. Window Size is the period of time for which the statistical data on the origination delay is stored. In order to compute the moving 95th POD, a number of buckets are used. Each bucket maintains a count associated with a given range of origination delay. If the origination delay for a particular call lies within a given range, the count in the corresponding bucket is incremented. On completion of a T_{moving} period of time, the moving 95th POD is computed from the counts stored in the buckets and the stored statistical data is discarded. But it has been observed in experiments that keeping some proportion of the statistical data of the origination delay from the previous Window

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

is helpful to avoid the effect of transients. The data that is carried forward from the previous Window to the next Window is called *History Data.* In a system in which x% of History Data is said to have been used, the counts in each bucket is multiplied by x/100 on completion of a T_{moving} period of time. These are the initial values of counts used in the subsequent window.

3.1.2 Transfer Policy

After analyzing different transfer policies discussed in the literature [Bal04, Wan93], a fixed threshold policy is selected as the transfer policy. A call server is operated at or below its engineered capacity. The load intensity on a Call Server which gives 95th POD equal to T_{ec} is referred to as the engineered capacity (EngCap). The value of T_{ec} varies according to the customer requirements and it also varies for different lines of products. The values of engineered capacity presented in this thesis are purely based on the output of the simulations. These values do not represent the engineered capacity of any lines of products of Nortel's Call Servers. The Call Server cluster investigated in this thesis uses a value of 140 msec for T_{ec} [Kop04.2]. The 95th percentile origination delay observed at the engineered capacity for a Call Server is the most logical choice for the fixed threshold. It has been seen in experiments that the threshold value slightly above the value of 95th percentile origination delay observed at the EngCap gives rise to a better system performance. This is discussed in more detail in Section 5.3.1. In this thesis, the threshold for the transfer policy is denoted by TH_T.

3.1.3 Location Policy

Location policy is applied if a Call Server decides to handover some of its load to another Call Server. The location policy is a *sender-initiated* policy and is applied after

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the call arrivals make the local CS a sender node. The process of locating a receiver Call Server is comprised of two steps.

The first step of the location policy is to identify the candidate receivers in the Call Server cluster. A threshold based policy is applied to search receiver Call Servers. For this purpose, a load table is maintained by every Call Server. This load table keeps updated load information of other Call Servers. This load information is available through the information policy. If the load index of a Call Server in the load table is less than a fixed threshold, then that Call Server is picked as a candidate receiver. In this way, a list of candidate receivers is compiled. The fixed threshold used in this policy is again based on the 95^{th} percentile origination delay at the EngCap of a Call Server. It has been seen in experiments that the threshold value slightly less than the value observed at the EngCap is a good candidate for a fixed threshold of location policy. This is discussed in more detail in Section 5.3.2. In this thesis, the threshold for the location policy is denoted by TH_L.

The second step of the location policy deals with the selection of a single potential receiver from the list of candidate receivers (obtained in the first step). The simplest choice is to select a least loaded receiver, but this strategy can cause a scenario of multiple Call Servers trying to throw their extra load to a single Call Server and in the end making it an overloaded Call Server. To avoid this, different strategies are discussed in detail in the Section 2.3.2. In this thesis, we use a new strategy which is based on the weighted random selection of the receiver from the list of the candidate receivers. Before going into the detail of this algorithm, we first define the available capability of a receiver. The available capability of a receiver is the difference between the threshold of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the location policy and the load index of the receiver CS. The method used in the weighted random selection policy is described. This algorithm gets a list of candidate receivers with their available capabilities (A_i) based on the threshold of the location policy. A receiver *i* in the list is selected randomly with a probability of $A_i / \sum_i A_n$.

3.1.4 Information Policy

Sharing of load information is the most critical component of load sharing, and there is always a trade off between accuracy of the load information and the overhead caused by the load information messages. For this thesis, a state change driven policy is adopted. On every invocation, the broker compares the current load index of the Call Server with the previously sent load index. If the change in the current load index is more than a fixed threshold, then this new value of load index is sent to the other Call Servers. On receiving a load information message from another Call Server, the broker of the current CS updates the load information of that source CS in the load table.

3.1.5 Data Sharing Policy

This policy has no direct relation with the load sharing. But as it has been discussed in Chapter 2, the dynamic state data of subscribers can be accessed in three different ways by the current Call Server. These three possible ways of data sharing are remote accessing, replicating or migrating originator's state data. We used the same policy for both the originator and the terminator data in one experiment. Most of the experiments in this thesis assume that the remote access policy for accessing originator and terminator state data in the Call Server cluster.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2 Design of Redirection Policies

The process of load sharing carried out by a broker is based on the flow chart presented in Figure 3-1. In Figure 3-1, δ is the *redirection factor* and it is a proportion of the total available new calls N_t that should be redirected to a remote receiver Call Server. n is the number of calls to be sent to the remote CS.

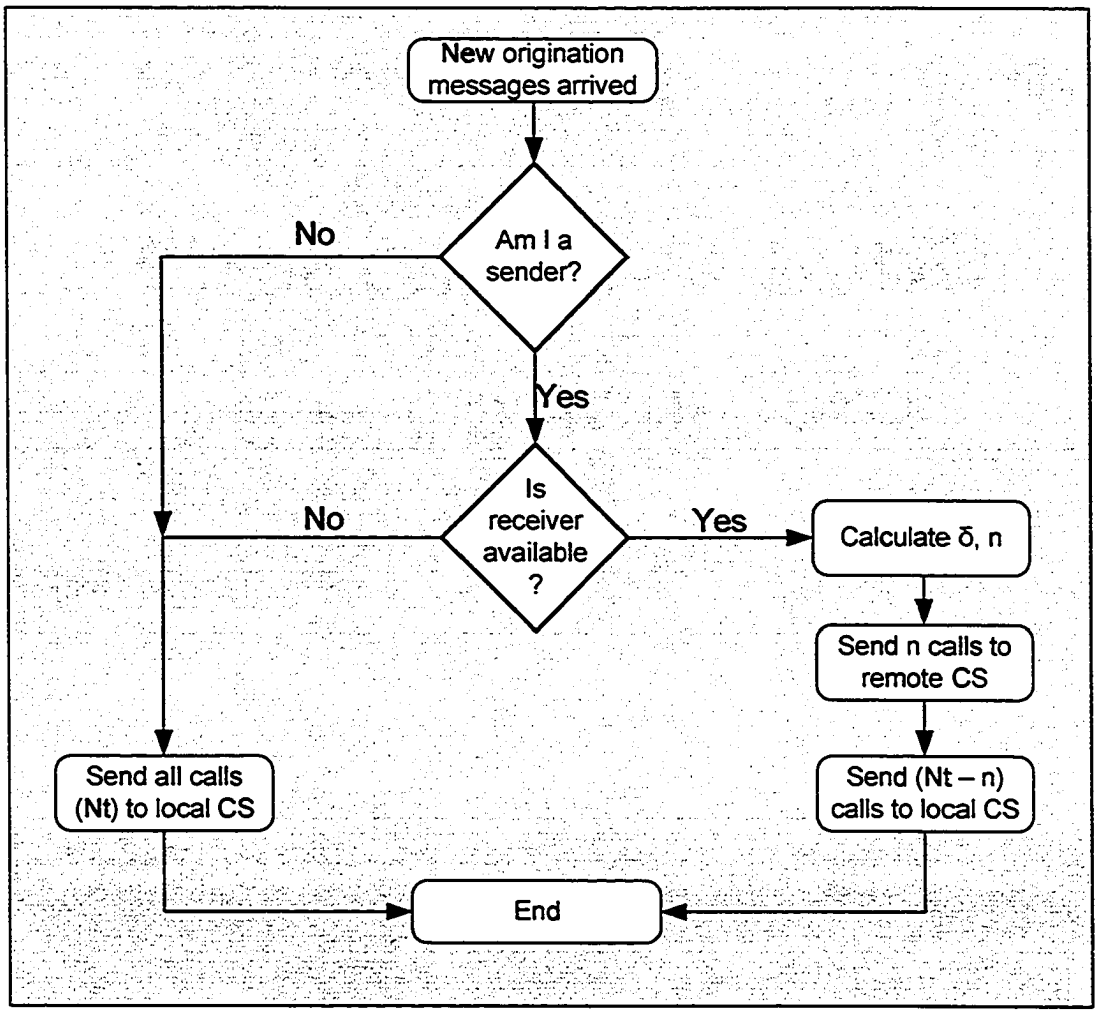


Figure 3-1: Load sharing flow chart

Figure 3-1 shows that every time the broker fetches new incoming calls (origination messages), it first decides whether the current CS is a sender or not (transfer policy). If the current CS is a sender node, then it looks for an appropriate receiver where it can

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

transfer the load of new incoming call messages (location policy). After locating a receiver successfully, the broker applies the redirection policy to determine the exact value of δ and n. n of the calls are transferred to the remote receiver CS while the rest of calls ($N_t - n$) are sent to the local CS. In case the local CS is not a sender node or there is no available receiver CS, then all the incoming calls are sent to the local CS.

In this thesis, a number of policies to calculate n are investigated. These policies are static as well as adaptive and are discussed in the following sub-sections. The naming of these policies is based on the redirection algorithm used.

3.2.1 Static Redirection Policy

The design of redirection policies is based on the idea of transferring a given proportion of the new incoming calls on an overloaded Call Server to a lightly loaded CS. The static or fixed redirection policy transfers a fixed proportion determined by the redirection factor, δ, of the total incoming calls, waiting to be processed by a call processing subsystem, to a receiver Call Server. This policy does not use any load information of either the sender or the receiver Call Server in determining δ. Thus,

$$n = \delta * N_t \qquad (1)$$

3.2.2 Adaptive #1: Sender Load Based Redirection Policy

This policy considers the load state of the sender CS to calculate the redirection factor, denoted by δ. The challenging part of this policy is to determine the redirection factor based on the extra load of the sender CS. This is because of the non-linear behavior of the Call Server when it is operating beyond its engineered capacity. So in such a situation, it is not wise to assume a linear relationship between δ and the load status of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

sender CS. A more accurate relationship is determined by analyzing the non-linear part of the curve shown in Figure 3-2. The graph presented in Figure 3-2 is obtained by running a simulation experiment with only one Call Server operating in the cluster and the behavior of the 95th POD is observed by varying the input load intensity.

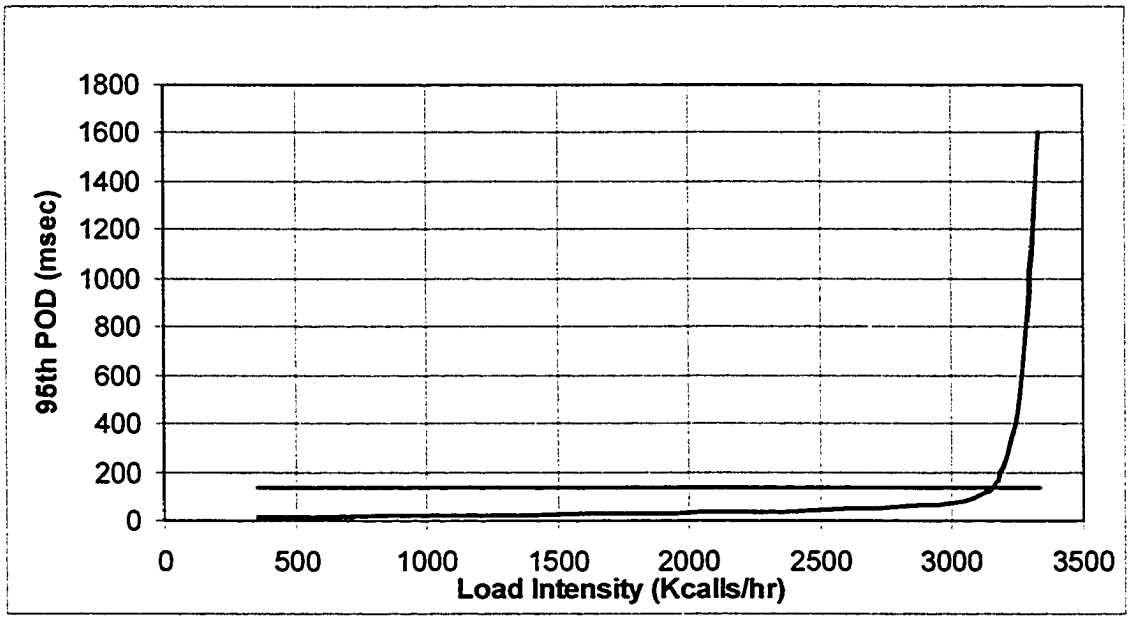


Figure 3-2: Variation in 95th POD (load index) with load intensity on a single CS

This graph shows how the 95th percentile origination delay (POD) varies with the load intensity of a Call Server. The solid horizontal line corresponds to T_{ec} and indicates the 95th POD of the Call Server when it is operating at the EngCap. The EngCap is observed to be 3157 Kcalls/hr in Figure 3-2. It is clear that the performance of the system degrades sharply as the input load goes beyond the engineered capacity. For further analysis, a microscopic view of a part of the curve is shown in Figure 3-3 to study the variation of the 95th POD beyond the engineered capacity in more detail.

The relationship between the 95th POD and the input load intensity seems to be approximately linear up to the engineered capacity of the Call Server. The 95th POD

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

increases non-linearly with load intensity beyond the engineered capacity of the Call Server. A 95th POD value of 1600 msec is considered to be very large for Nortel telephone switches. Distinguishing any values larger than 1600 msec are thus not important [Kop04.2]. Therefore the 95th POD is capped at 1600 msec. We denote this maximum possible value of 95th POD as T_M. The single Call Server based cluster achieves the limit of T_M for the input load intensity of 3327 Kcalls/hr (see Figure 3-3).

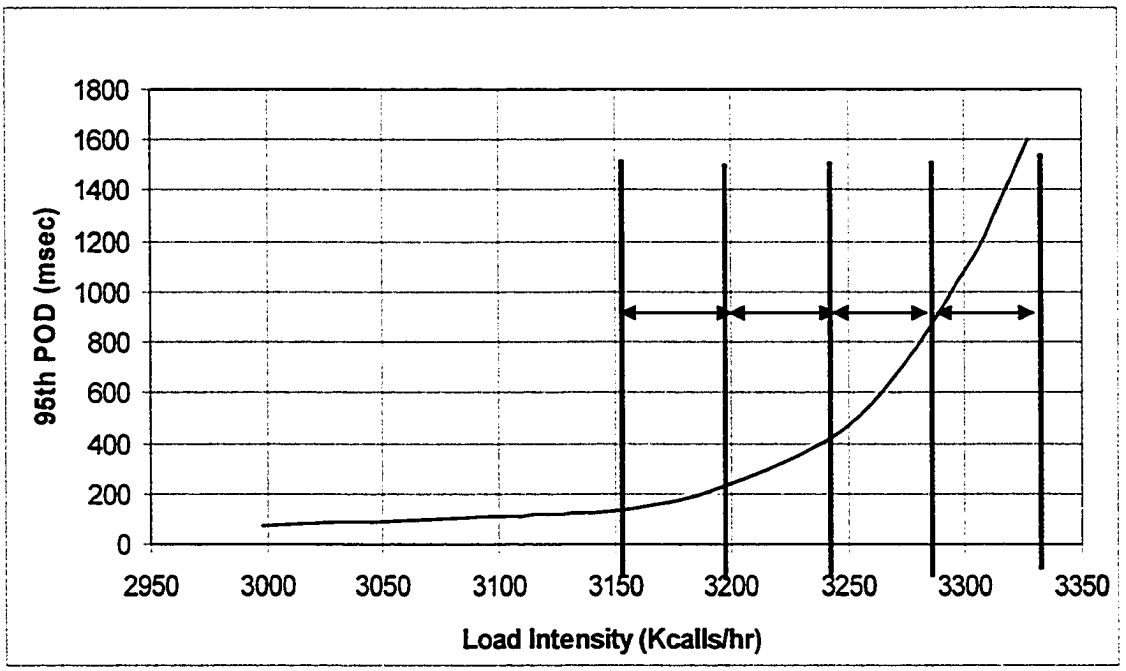


Figure 3-3: Detailed view of the relationship between 95th POD (load index) and load intensity

The additional load which pushes the 95th percentile origination delay from T_{ec} to T_M is only 170 Kcalls/hr (3327 – 3157). If we assume a linear relationship between the load index and the proportion of the redirected calls, the sender CS can forward a large number of calls to a remote CS. This can cause more overhead and may cause the remote Call Server to run into an overloaded situation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

To avoid this situation, the non-linear part of the curve beyond the engineered capacity is sub-divided into small linear segments. The exact relationship between δ and the load index of the sender CS is determined separately for each segment. In the graph shown in Figure 3-3, the non-linear part of the curve is divided into four segments which are named a, b, c and d in Table 3-1. The widths of these four segments are equal to one another and are obtained by dividing the additional load intensity of 170 Kcalls per hour into four equal parts. The load intensity ranges and the corresponding values of the 95th POD for the boundary points of these segments (x_1, x_2) are summarized in the Table 3-1.

Table 3-1: Segments of the non-linear part of the curve

Segment	**Load Intensity Range (Kcalls/hr)**	**Load Index Range (ms)**		**Change in Load Index (ms)**
		x_1	x_2	
a	3157 – 3199	140	230	90
b	3199 – 3241	230	440	210
c	3241 – 3383	440	850	440
d	3383 – 3325	850	1600	750

Using these four segments, we devise a relationship between δ and the load index of the overloaded CS. This is shown in Figure 3-4. The value of δ is computed by using the equation of a straight line for these four segments a, b, c and d (see Figure 3-4).

Depending on which segment the load index of the sender CS lies in, a specific equation is used to compute the value of δ. In order to determine the maximum value of δ for a given segment the possible range of δ (0 to 1) is divided into four equal parts. The maximum value of δ corresponding to segments a, b, c and d are set at 0.25, 0.5, 0.75 and 1.0 respectively (see Figure 3-4). If the load index of the sender Call Server, L_s, lies in the segment a, δ is given by:

$$\delta_a = 0.25^* (Ls - 140)/ (230 - 140) \qquad (2.1)$$

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

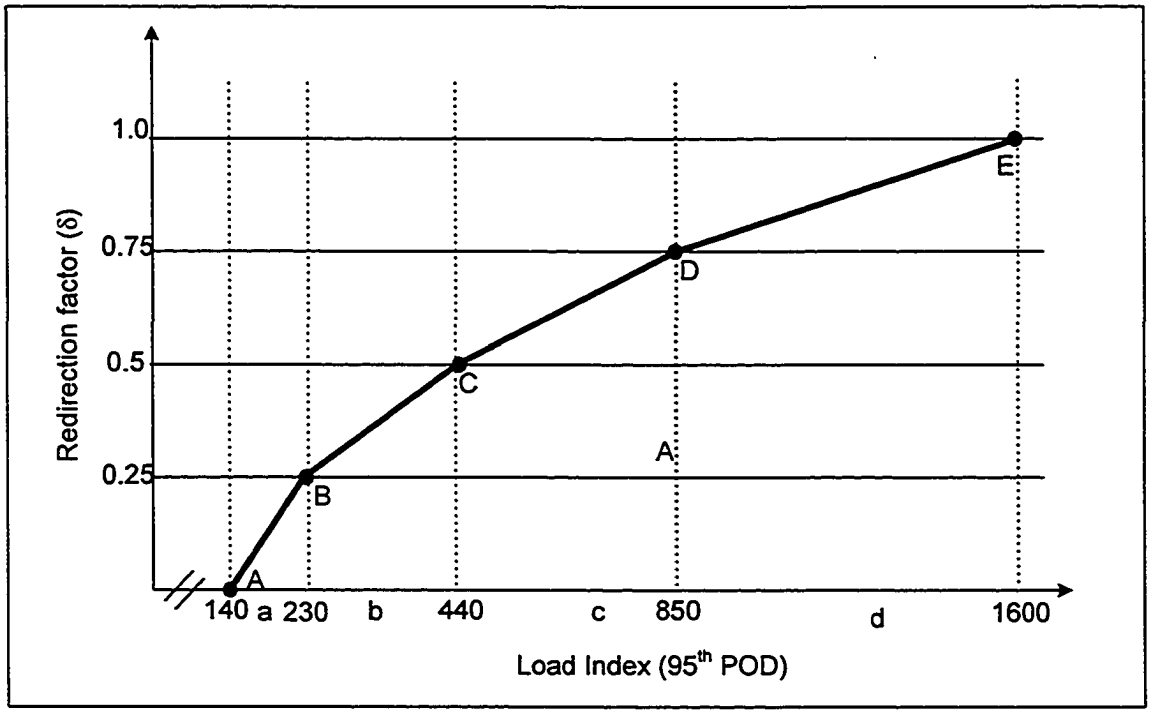


Figure 3-4: Relationship between δ and load index (95th POD) of the sender CS

A similar linear relationship is used for the other segments. Thus,

for segment b:

$$\delta_b = 0.25\ [1+ (Ls - 230)/ (440 - 230)] \qquad (2.2)$$

for segment c:

$$\delta_c = 0.25\ [2+ (Ls - 440)/(850 - 440)] \qquad (2.3)$$

for segment d:

$$\delta_d = 0.25\ [3+ (Ls - 850)/(1600 - 850)] \qquad (2.4)$$

Once we determine δ, we can calculate the exact number of calls to redirect n using this equation

$$n = \delta * P * N_t \qquad (3)$$

Where P is a constant which corresponds to the maximum redirected proportion of incoming calls that can be transferred to a remote Call Server. The value of the constant

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

factor P can be from 0 to 100%. The effect of the value of P on the performance of this redirection policy is investigated in more detail in Section 5.4.2. As shown in Figure 3-4, δ increases with an increase in the sender load and it also takes into account the non-linear impact of the load intensity on the load index. The rate of increase in the redirection factor gets smaller as the load on the sender Call Server becomes higher and higher.

3.2.3 Adaptive #2: Sender/Receiver Load Based Redirection Policy with Adaptive P

The sender load based redirection policy, discussed in the last section, uses information on the state of the sender CS to calculate the value of δ for forwarding some of the new incoming calls to the receiver CS. The second adaptive policy is the first version of the sender/receiver load based redirection policy. This policy uses the load information of both the receiver Call Server and the sender Call Server to calculate δ. The knowledge of the sender load is more recent and accurate than the load information of receiver because the sender load is fetched at the run time from the local Call Server and the accuracy of the receiver load information is dependent on when this information was received. So the calculation of δ again uses the same algorithm based on the sender load information, but the load information of the receiver is used to decide the appropriate value of the maximum redirected proportion P. This amount P is assumed to be a constant in the sender load based policy, but under this policy this parameter is adaptive to the load status of the receiver Call Server. The different steps involved in the algorithm of the sender/receiver based policy are described next.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Get Load information

As soon as the broker identifies the current Call Server as sender and also locates an appropriate receiver CS, the first step is to fetch the load information of the current local CS and the load information of the receiver CS.

Get Receiver Load Level

The receiver Call Server is chosen using a two step location policy as explained in Section 3.1.3. As we discussed in Section 3.1.3, the fixed threshold TH_L is close to the load index corresponding to the engineered capacity of the Call Server. We have also seen in the last section (see Figure 3-3) that the 95th POD increases non-linearly with an increase in load intensity beyond the engineered capacity of the CS. So the receiver CS operating slightly under the engineered capacity may become overloaded by just receiving a small number of new calls (local or remote). For this policy, we propose four levels of available capability that are simply referred to as availability levels. If the load index of a receiver CS is denoted by L_r, then the available capability, A, of that receiver CS is defined as the difference of TH_L and L_r

$$A = TH_L - L_r \qquad (4)$$

It has been seen in experiments that four availability levels are adequate. Using a higher number of levels does not provide any significant benefit. The four availability levels are defined as follow.

Availability Level 1: If $0 < A \leq (0.125 * TH_L)$, the receiver CS is categorized as operating at availability level 1.

Availability Level 2: If $(0.125 * TH_L) < A \leq (0.25 * TH_L)$, the receiver CS is categorized as operating at availability level 2.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Availability Level 3: If $(0.25 * TH_L) < A \leq (0.5 * TH_L)$, the receiver CS is categorized as operating at availability level 3.

Availability Level 4: If $(0.5 * TH_L) < A \leq TH_L$, the receiver CS is categorized as operating at availability level 4.

Get Maximum Limit for sender load policy

In this policy, P is assumed as a set of four values (P1, P2, P3, P4) in the increasing order (P1<P2<P3<P4). Each value of P in the set corresponds to one availability level of the receiver CS. For example, P1 corresponds to availability level 1 and P2 corresponds to availability level 2 and so on.

Once the availability level of the receiver CS is determined, then the next step is to choose a single value of P from the set (P1, P2, P3, P4) corresponding to the availability level of the receiver CS.

Apply sender load based policy

In the last step of calculating δ, we apply a method similar to the one used in the sender load based policy as explained in Section 3.2.2. The only difference is that P is a dynamic value which is chosen from a set based on the availability level of the receiver CS, as explained in the previous step. The advantage of this policy is that if the receiver is operating near the engineered capacity, then the sender CS will lower its maximum limit P to adapt the value of δ based on the both receiver load and the sender load. But if the receiver is operating at low load, then the sender CS will use higher values for the maximum limit P that will result in a more effective transfer of new calls to the remote Call Server.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.4 Adaptive #3: Receiver Load Based Redirection Policy

In the receiver load based policy, the sender CS only checks the load status of the receiver CS to calculate δ. To formulate a mathematical equation for δ, we need to look at the behavior of the 95th POD as a function of the load applied (see Figure 3-5). The graph shown in Figure 3-5 presents a magnified version of the initial part of the curve presented in Figure 3-2.

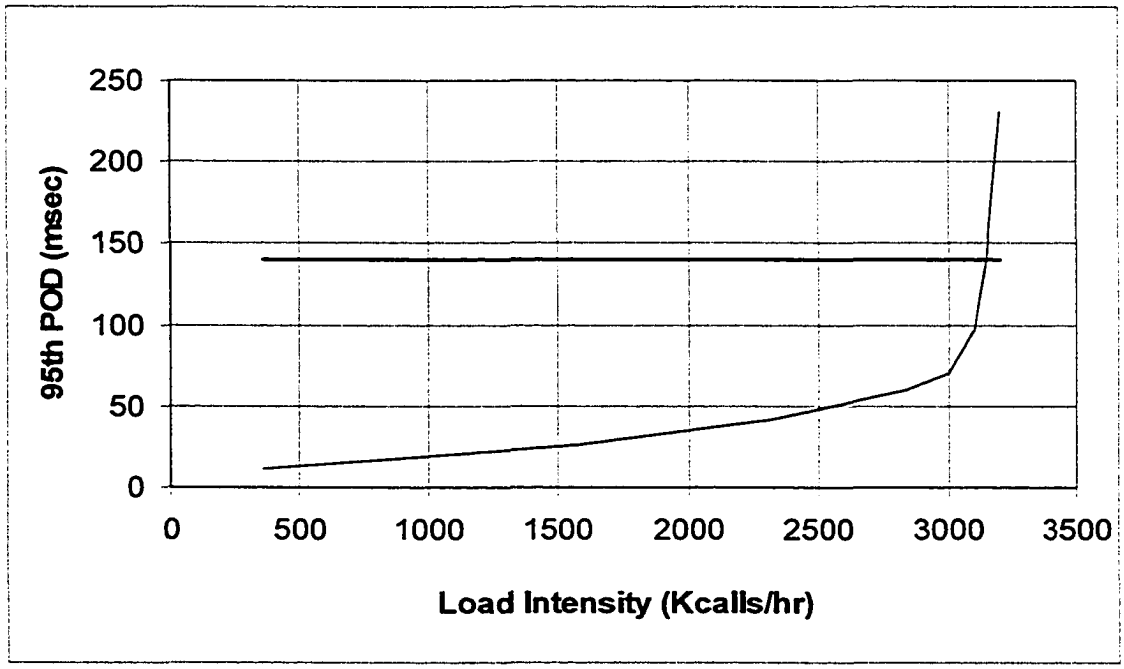


Figure 3-5: Variation in 95th POD (load index) with load intensity on a single CS: Initial Part

Up to a 95th POD of 70 msec, the relationship between this load index and load intensity is approximately linear. So we determine δ using a linear relation if the load index of the receiver CS is less than 70 msec. If we denote the load index of the receiver as L_r, available capability of the receiver CS as A and the threshold of the location policy as TH_L, then

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

$$\delta = A / TH_L \qquad [0 < L_r < 70 \text{ msec}] \qquad (5)$$

[where $A = TH_L - L_r$]

If the load index of the receiver is in the range of 70 msec and TH_L (140 msec in Figure 3-5), then it has been seen experimentally that if we transfer load by assuming a linear relationship between the 95th POD and the load intensity, then the receiver CS gets overloaded. To avoid this, we simply redirect a low fixed proportion of calls when L_r >70 msec. The pseudo code for determining δ is presented in Figure 3-6. If the load index of the receiver CS is higher than 70 msec, we transfer fixed proportions (10-25%) of new calls based on the available capability of the receiver CS, A. Higher the value of A, higher is the value of δ used. But these values of δ are lower than the values computed using equation (5).

```
If [ Lr < 70 ms]                          //Linear Part
{
        δ = A / THL
}
else                                      //Non-linear Part
{
        If [ A < 0.25* (THL - 70)]
                δ = 10%

        else if [ A < 0.50* (THL - 70) ]
                δ = 15%

        else if [ A < 0.75* (THL - 70) ]
                δ = 20%

        else if [ A < (THL - 70) ]
                δ = 25%

}
```

Figure 3-6: Pseudo code for determining δ

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Once we determine δ, we can calculate the exact value of n using equation (3). The value of P assumed for this policy is 1.

3.2.5 Adaptive #4: Uniform Sender/ Receiver Load based Redirection Policy

In the uniform sender/receiver load based policy, we calculate the redirection factor, δ, based on the sender load and the receiver load independent of each other. Since the product of δ and P can have a maximum value of 1, so we assume the sender and the receiver components are contributing equally to the determination of δ.Thus, n, the number of calls to be transferred is given by

$$n = \Delta * N_t$$

and

$$\Delta = 0.5*\delta_1 + 0.5*\delta_2 \qquad (6)$$

δ_1 is determined using the sender load based policy as described in Section 3.2.2 and δ_2 is calculated using the load status of the receiver CS (see Figure 3-6).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4 Simulation Model

This chapter describes the design and implementation of the simulator model of the Call Server cluster. It also discusses the basic methodology and principles used to model the Call Servers, the workload, system and performance parameters and the working of the system in detail. In the last section, the validation and the verification techniques, used in this thesis, are discussed.

4.1 OPNET Simulation Model

This thesis used OPNET Modeler, version 8.0.C [Opn02] to implement the simulation model of the Call Server cluster. OPNET provides a comprehensive development environment supporting the modeling of communication networks and distributed systems. Both the behavior and the performance of the modeled system can be analyzed by performing discrete event simulations. OPNET facilitates model design, simulation, data collection and data analysis in one software package. The key features of OPNET Modeler are summarized [Opn02].

1- Graphical environment for modeling real systems, devices and protocols.

2- Specialized in communication networks and distributed systems.

3- Flexibility to develop detailed custom models for communication and distributed systems with Finite State Machines and high level languages such as C/C++.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4- Designed models can automatically be compiled into executable, efficient, discrete event simulations.

5- Comprehensive library of standard protocols and devices, with completely open source code.

6- Facilitates graphical tools for post-simulation analysis of data.

7- Facilitates interactive analysis of the model with state of the art debugger.

8- Simulation runs can be automatically configured to generate animations of the modeled system.

9- Built-in tools for generation of customized packets and data for distributions other than standard distributions already available in the tool through API's.

4.1.1 Programming Paradigm in OPNET

OPNET uses two fundamental methods of describing a system through objects and procedures. Objects are used to refer to the structure of the system and procedures are used to express the system's behavior [Opn02]. The modeling of a system in the OPNET Modeler is mainly carried out hierarchically at three levels or domains. These hierarchical levels are referred to as the network model, the node model and the process model.

A *network model* defines the overall scope of the system to be simulated. The network model specifies the objects in the system, as well as their physical locations, interconnections and configurations. A network model may contain a single sub-network a single node or many interconnected nodes and sub-networks. The main building blocks of the network domain are sub-networks, nodes and communication links. For this thesis, a single Call Sever makes up a sub-network object in the network domain. The cluster is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

developed in the network domain by interconnecting multiple Call Servers through communication links.

A *node model* provides details of a single node or sub-network. The basic elements of the node model are processors, queue, transmitter and receivers modules. The communication paths between basic elements of the node model are provided through packet streams or static wires. Different components such as the peripheral module, the Input/Output process or the call processing system are modeled as processor model in this domain.

A *process model* defines the behavior of processes and queue models which exist in the node model. Process modeling has built-in graphical editor for creating Finite State Machines. The detailed behavior of the state machine is controlled through 'enter', 'exit' or transition level procedures. OPNET supports programming in C and C++ at this level.

4.2 XA-Core Model

For this thesis, XA-Core based Nortel's DMS switch is used as a basis for the simulation model of the Call Server. The key components of a real telephone switch are already discussed in chapter 2. The simulation model consists of two main sub-models. These sub-models are for the peripheral module (PM) and for the core Call Server. These two individual sub-models work as a client-server model. A very high level overview of the client server model used in the telephone switch is presented in Figure 4-1. The clients represent the multiple OPNET processes used for modeling peripheral modules and the OPNET processes in the XA-Core Call Server represent the servers in the client server model. The details of OPNET processes shown in Figure 4-1 are provided in the following sub-sections.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

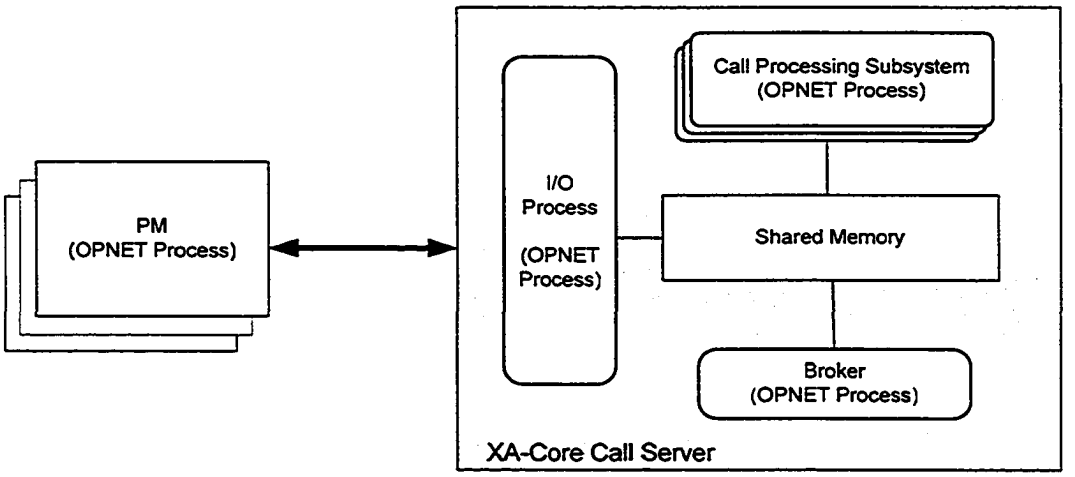


Figure 4-1: Client Server model of a Call Server

Different types of messages are used for different phases of the call. The key messages involved in call processing are described in Table 4-1.

Table 4-1: List of key call processing messages

Message Type	Description
Origination	The message is sent from subscriber who wants to initiate a new call (originator).
Start Digit Collection	If the resources are available, then the Call Server instructs peripheral module to collect digits from the initiator of the call.
Digits	Digits (of destination) are sent from the initiator of the call.
Apply Physical Ringing	Call server directs peripheral module to send a physical ring signal to the handset of the destination subscriber (terminator).
Stop Digit Collection	On receiving the Digits message, Call Server directs peripheral module to stop the process of digit collection.
Apply Audible Ringing	Call server directs peripheral module to send an audible ring to the handset of the initiator of the call.
Answer	Message sent from the destination subscriber, accepting the call.
Stop Audible Ringing	On receiving Answer message, Call Server sends this message to peripheral module to stop audible ringing on the handset of the initiator of the call.
Exit	Disconnect message sent either by the initiator or the destination subscriber to end up the call.
Apply Silence	A message sent from Call Server to peripheral module to apply silence on the handset of the initiator or the destination subscriber, because the other subscriber has already disconnected.
Idle	On receiving Exit message, Call Server asks peripheral module to send the Idle signal to the other subscriber. On receiving this message, peripheral module releases resources for that subscriber.
Load Information	This message carries load information of a Call Server.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.2.1 Peripheral Module

The peripheral module is modeled for plain old telephone service (POTS), which is used only for voice calls. The initial version of this model was originally developed by Bing Sun [Sun04], but it was redesigned in this thesis to accommodate new requirements. The key features of the original model are summarized.

- The original model uses a built-in traffic generator of OPNET for simulating new incoming calls. The inter-arrival time of input traffic is based on an exponential distribution with a given mean inter-arrival time.
- On receiving a new incoming call message from the traffic generator, the peripheral module creates an independent child process for that subscriber that serves as an independent client communicating with the Call Server.

- Figure 4-2 shows the state diagram of the call processing handled by the OPNET process of the peripheral module. This state diagram is valid for both the originator and the terminator of a call.
- Every peripheral module is owned by one Call Server. A peripheral module can only send messages to its default or owner Call Server, but it can accept messages from remote Call Servers as well.

In this thesis, we used the first three features as is and modify the last one. This modification was performed to enhance the PM's capability of communicating directly with remote Call Servers. In the original model, the response from a peripheral module to a remote Call Server is routed through its owner Call Server, which always introduces an extra delay. So in the current model, the peripheral module can communicate with any of the remote Call Servers in the cluster directly. To add this feature, every client in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the peripheral module has to keep an identification key of the state information of the call stored at the remote Call Server. This identifier is passed with every message (Answer or Exit message) sent from that client peripheral module to the remote Call Server. The remote Call Server uses that identifier to locate the state information of the call for which the message has been sent. This new model is referred to as a model for an *Intelligent Peripheral Module.*

The subtasks of a call processed by the peripheral module are explained in a state diagram shown in Figure 4-2. As already mentioned, this state diagram is for both the originator and the terminator of the call. As an originator, the peripheral module sends an 'Origination' message to its CS and then waits for the response in 'Origination' state. On receiving the 'Start Digit Collection' message, it starts collecting digits and sends those digits to the CS. Once the digits are sent, it waits in the 'Setup' state for setting up of the call by the CS. Once the call setup process is complete, the peripheral module receives a 'Stop Digit Collection' message and the line is connected to the destination line. The PM process of the caller makes a transition to the 'Connected' state and stays in the 'Connected' state as long the other participant of the call is connected as well. As soon one of the participants of the call disconnects, its peripheral module sends an Exit message to the CS and releases resources for that subscriber. The CS sends an 'Apply Silence' message to the peripheral module of the other participant of the call. That peripheral module applies silence to the handset of the subscriber and enters the 'Silenced' state. As soon as the other participant disconnects (second disconnect), its peripheral module also sends an Exit message to the Call Server.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

As a terminating subscriber, the peripheral module, on receiving an 'Apply Physical Ringing' message, sends a ring signal to the subscriber handset and enters the 'Be-Called' state. As soon as the phone is attended, an Answer message is sent to the CS and the PM process of that subscriber goes to the 'Connected' state.

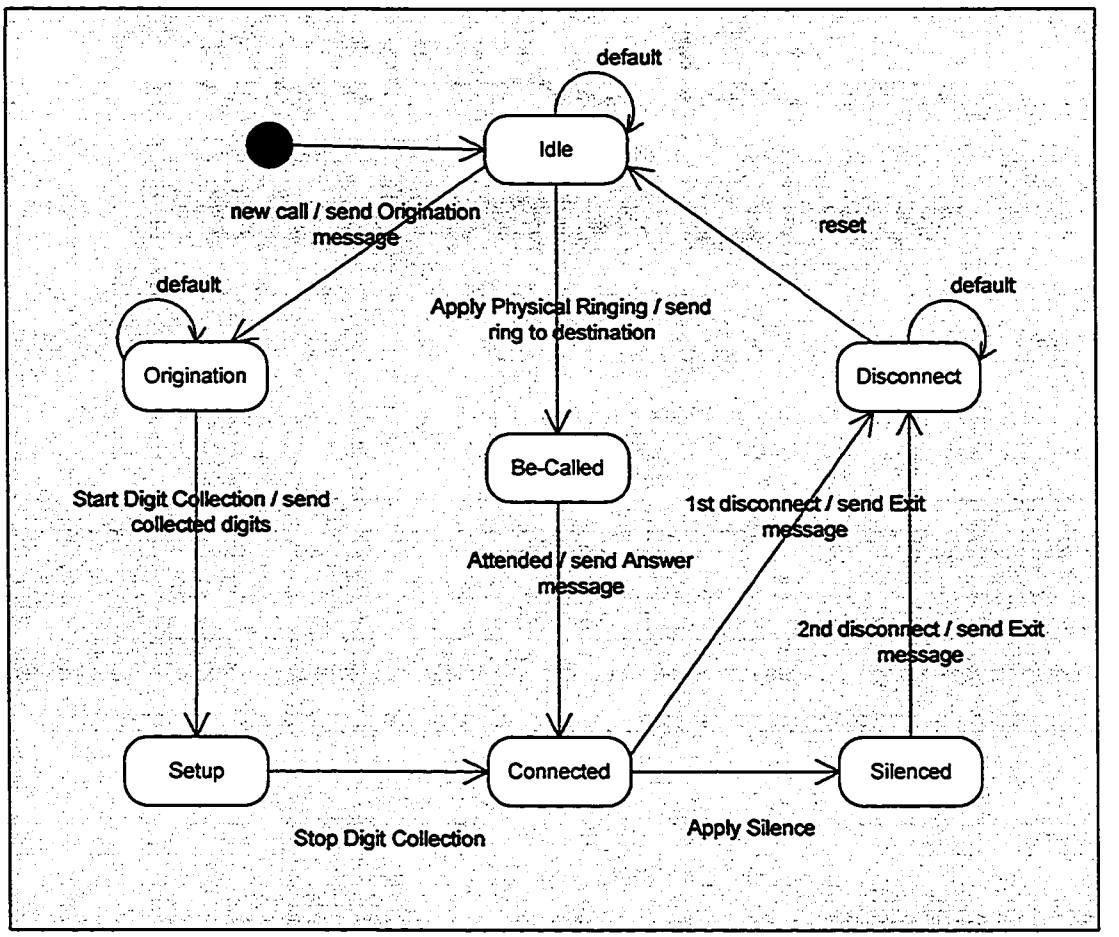


Figure 4-2: State diagram for PM call processing

4.2.2 Call Server Model

The model of the XA-Core Call Server is further decomposed into smaller component models. These components are modeled as processor models or queue models in the node domain of OPNET. Processor models are used to model Input/Output processes, the call processing subsystem and the broker. The simulation models of the Input/Output

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

and the broker processes are developed according to the requirements of this thesis. The model of the call processing subsystem was originally developed by Sun for the first version of simulation model [Sun04]. This is reused in the current version after some minor modifications. The queue models of the node domain are used to model the scheduler and different queues in the Call Server. The built-in queue model is based on the first-in first-out principle and its details are available in the OPNET library.

A UML collaboration diagram [Uml04] for the model of the Call Server is shown in Figure 4-3. The stereotypes used in this diagram are introduced to represent various OPNET processes and built-in queues. The numeric labels in the messages shown in Figure 4-3 are based on the assumption that the scheduler invokes Input/Output process first, followed by the broker and the call processing subsystem processes. But in reality, the order of the invocation of these processes is not fixed. In the next sub-sections, we give a brief overview of the software components shown in Figure 4-3 and also explain the messages involved in this collaboration diagram.

Input Queue

This queue is shown as 'Input Q' in Figure 4-3. The messages sent from external modules are stored in this queue. An Input/Output process extracts these messages every time it is invoked by the scheduler.

Output Queue

This queue is shown as 'Output Q' in Figure 4-3. The messages, which are to be sent outside the Call Server, are put in the output queue. The call processing subsystem and the broker process put outgoing messages in the output queue. Again, An Input/Output process processes these messages every time it is invoked by the scheduler.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Broker Input Queue

This is shown as XB-Q in Figure 4-3. The new origination and the load information messages are to be processed by the broker process in the Call Server cluster. An Input/Output process puts these messages in the broker input queue. The broker process extracts these messages every time it is invoked by the scheduler.

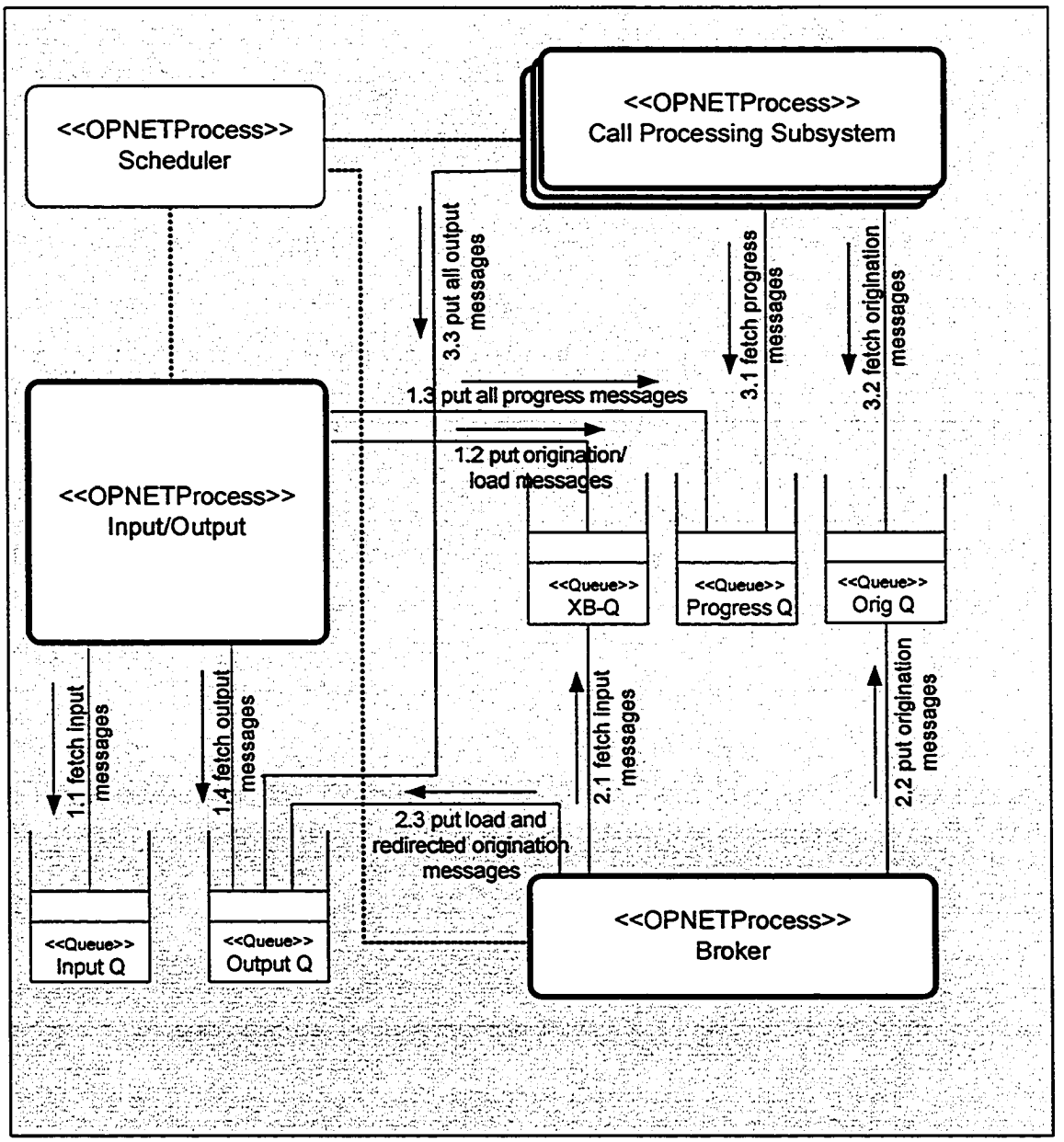


Figure 4-3: Collaboration diagram of Call Server simulation model

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Origination Queue

This is shown as 'Orig Q' in Figure 4-3. The broker process puts origination messages in this queue for local handling of the new call. The origination messages wait in this queue for processing by the call processing subsystem.

Progress Queue

All the messages other than the origination messages are put in this queue (shown as 'Progress Q' in Figure 4-3) by an Input/Output process. The progress queue is the second input queue for the call processing subsystem.

Input/Output Process

The model of the Input/Output process used in Figure 4-3 is redesigned to accommodate additional messages introduced for load sharing among Call Servers. The earlier version [Sun04] of this Input/Output process model can only receive messages from the local or default peripheral module and the remote Call Server. If the message is for a call being processed locally, then the message is put either in the XB-Q queue or in the progress queue. In case the call is being handled by a remote CS, the Input/Output process reroutes these messages to that remote CS. In the current version of the Input/Output process, we not only eliminate the rerouting of messages by introducing an intelligent peripheral module, but also add a few extra features that are summarized.

1- The new model can handle messages from remote peripheral modules as well as from remote Call Servers.

2- The outgoing messages can have multiple destinations such as the local peripheral module, the remote peripheral module and the remote Call Server. The

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Input/Output process uses message type and destination category to identify the exact location for the delivery of the message.

3- The load information messages are exchanged between Call Servers for sharing of updated load state information of every Call Server. The Input/Output process broadcasts these messages to other Call Servers. On receiving a load information message, an Input/Output process delivers this message efficiently to the broker to update the load database table.

In the simulation model, the Input/Output process sends a message to the scheduler to demand a PE whenever its self interrupt is invoked. On availability of the processor, the Input/Output process handles all the new input messages first and then the outgoing messages. In the end, it schedules itself for the next invocation and sends a message to the scheduler to free up the PE.

Broker Process

The broker process is added to the Call Server simulation model to incorporate different components of load sharing in the Call Server cluster. Whenever it is invoked, it processes the input messages from the input broker queue (XB-Q in Figure 4-3). Broker input messages can be divided into three categories. The first category is the load information messages. The broker updates the load data base of other Call Servers on receiving the load information message. The second category corresponds to forwarded origination messages that are transferred from an overloaded remote Call Server. After allocating the initial resources, these messages are put in the origination queue. The third category corresponds to origination messages from the local peripheral module. Load sharing is applicable for only these origination messages. If the transfer policy identifies

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the current Call Server as a sender, then the broker process uses the location policy to locate a suitable receiver for this sender. If a receiver is found, then next step is to determine how many origination messages should be forwarded to the remote receiver Call Server. The exact number of origination messages is determined by the redirection policy. With every invocation of the broker, it also checks the load of the current Call Server. If the change in load is significant as compared to previously broadcast load information, then a load information message is also sent to the output queue.

Call Processing Subsystem

The call processing subsystem uses multiple OPNET processes for processing of different call messages waiting in the progress and origination queues. Processing of a call in the call processing subsystem concerns the service of various messages such as Origination, Digits and Exit. The handling of a single message by the call processing subsystem can be referred to as a *subtask*. The processing of all messages of a single call is usually referred as a task [Ben98]. Since the processing time slice of a call process is large compared to the service time of these individual subtasks, the call process tries to string together the execution of several messages to fill out the time slice. This allows the efficient use of the allotted processing time by avoiding excessive context switching [Nor02]. As soon as a call process is invoked, it loads a fixed number of subtasks or messages from the progress and origination queues. As the call process loads messages to be served, the call related information required for these messages is made available to the call process. If a call process is not able to serve all messages in its processing time slice, then the messages are handled by the same call process during its next invocation. Because of the availability of multiple processors, another call processing process can be

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

activated by the scheduler, in parallel, to handle the pending messages on the progress and the origination queue. This design ensures mutual exclusion on sharing data in the call's information block.

Scheduler

The scheduling of the tasks in the telephone switch is autonomous. There is no central processor which does scheduling of the processes among different processing elements (PEs). Each individual process runs on a PE and is responsible for its scheduling. When a process on a PE suspends itself, the scheduler is invoked to select the next process to run. The model of this autonomous scheduler is implemented in OPNET by modifying a built-in queue model. The process that requires a PE sends a message to the scheduler. On receiving the message, the scheduler checks the number of available PEs. If the count is non zero, then it assigns a PE to the process and decrement the count. If the count is already zero, then it puts the message as a request in the process ready queue. When a process suspends its processing, it sends another message to the scheduler. On receiving this message, the scheduler first checks if there is any process waiting in the process ready queue. If there is a process waiting in the queue, the process is immediately assigned to it or it increments the count of available number of PEs.

4.3 Simulation Parameters

The simulation parameters include both the workload and the system parameters. First, we introduce workload parameters followed by the system management policies and system parameters. We then define performance indices which are observed to study the performance of the Call Server clusters.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.1 Workload Parameters

The workload parameters are varied for different sets of experiments to investigate the effect of work load on the behavior of the Call Server cluster. Most of the workload parameters are submitted as an input to the 'Simulation Sequence' tool provided by OPNET. This tool can create different instances of a simulation experiment for different sets of input parameters. In addition to the workload parameters, the user can also set the seed, the scalar and vector output file names as well as debugging and optimization configuration levels. The important workload parameters are presented next.

Load Intensity: The load intensity on a Call Server is the rate of calls that arrive at a Call Server per unit time. The call arrival process is assumed to be a Poisson process. That is, the time duration between the arrivals of two new calls on a Call Server is generated by an exponential distribution with a mean of inter arrival time.

Think Time: There are four events during the life cycle of a call that depends on the response of a subscriber. These four events are: dialing a destination number, answering a call, talking and hanging up (second disconnect in Figure 4-2). The think times model the time taken by the subscriber to respond to the events and are generated by using truncated exponential distributions with given means.

Service Time: Service times are the times required to process different call processing messages. These services times are different for different messages and Nortel Networks has provided the exact value for the service times to be used in simulation for different call processing messages. The values of the service times can not be disclosed due to a non disclosure agreement with Nortel.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Locality Factor: This parameter represents the probability that terminator state data is available locally on a Call Server that is processing the call. A detailed analysis of the effect of locality on performance is done by Majumdar et al. [Maj04]. We specified the probability of the terminator state data on the local Call Server through this parameter.

Read/Write Operations: The number of read and write operations on a remote Call Server is provided as input parameters.

4.3.2 System Management Policies and System Parameters

Data sharing Policies: The terminator state data can be managed in three different ways as discussed in Section 2.2.4. These are remote access, replication and migration. The choice of using which strategy in the simulation experiment is provided through a 'Strategy' parameter using value 1, 2 or 3.

Redirection Policies: The choice of redirection policy is used for load sharing. We devise one static and four adaptive redirection policies. These are listed in Table 4-2. The details of these redirection policies are available in Section 3.2.

Table 4-2: List of redirection policies

Policy	Description
Static Policy	Static fixed redirection policy
Adaptive #1	Sender load based redirection policy
Adaptive #2	Sender/ receiver load based redirection policy with Adaptive P
Adaptive #3	Receiver load based redirection policy
Adaptive #4	Uniform sender/receiver load based redirection policy

Number of PEs: The number of processing elements available on the multiprocessor based Call Server is provided through this parameter.

Cluster Size: The number of Call Servers in a cluster is a configurable through this input parameter.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Transfer Policy Threshold (TH_T): Although we use a fixed threshold based transfer policy, but we provide the value of the threshold through an input parameter.

Location Policy Threshold (TH_L): The threshold of the location policy is provided through this input parameter.

Load Index Attributes: As mentioned in Section 3.1.1, the load sharing techniques use a moving 95th POD as the load index. The calculation of the moving 95th POD involves two attributes: Window Size and History Data. These are explained in Section 3.1.1. The effective values for these two attributes are determined through experiments. The detailed results and the criteria used in the selection of values for these attributes are described in Section 5.2.

4.3.3 Performance Indices

A number of performance indices are logged into scalar and vector output files. The advantage of logging these parameters in the scalar file is that OPNET saves the output of multiple experiments in one scalar file which can be viewed as different data points in a graph using an analysis tool provided by OPNET. The output parameters, which are traced with respect to time, are logged into a vector file. OPNET's analysis tool also shows graphical view of the data logged in the vector file. The important performance indices computed from the simulation results are described.

Dial Tone Delay: It is the delay between the picking up of a phone receiver and the hearing of the dial tone. In our simulation model, this delay is calculated as a difference between the time when a peripheral module receives the 'Start Digit Collection' message from the Call Server and the time when a new 'Origination' message is sent by the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

peripheral module to the Call Server. Both the 95th percentile and the average value of the dial tone delay are calculated.

Origination delay: It is the key output parameter and the main indicator of the performance of the Call Server. This delay is the difference between the time when an 'Origination' message gets served by the call processing subsystem and the time when the Input/Output process in the Call Server receives that 'Origination' message. Again the 95th percentile and average origination delays are observed and calculated. These parameters are calculated for every Call Server in the cluster.

Average progress delay: It is the average time taken by the progress messages to travel from the Input/Output process to the call processing subsystem. All messages other than the 'Origination' message are the progress messages. Examples of progress messages include 'Digits', 'Answer' and 'Exit' messages.

Average setup delay: It is the average time taken to set up a call. This delay is measured as a difference between the time when the peripheral module starts hearing audible ringing of the destination phone and the time when a 'Digits' message is sent from a peripheral module to the Call Server.

Average Process Length: It is the average time taken by a process whenever it is invoked by the scheduler. Average process length or process time is measured for the Input/Output, the broker and the call processing processes separately.

Average CPU utilization: It is the average CPU utilization of a process. It is measured as the percentage of total processing time of a process spent on any of the available processing element. It is calculated for the Input/Output, the broker and the call processing processes separately.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Average inter-invocation time: It is the average waiting time between the two consecutive invocations of a process. This metric is computed for the Input/Output and the broker processes.

Overall 95th Percentile origination delay: Overall or aggregated 95th percentile origination delay of the Call Server cluster is calculated from the data collected for all the individual Call Servers. This is the 95th percentile origination delay for all the calls processed by the set of all Call Servers in the cluster.

Origination Delay Improvement: For Nortel Networks' telephone switches, the origination delay is a key performance index which governs the quality of the service provided by the switch for every new incoming call. The effectiveness of load sharing techniques is measured in terms of how much the 95th percentile origination delay is reduced in comparison to the value observed without using load sharing techniques. So we define the improvement in the 95th percentile origination delay as the ratio of the 95th POD without load sharing to the 95th POD with load sharing. This is called as Origination Delay Improvement (ODI) factor. Thus, for a given load sharing strategy,

$$ODI = 95^{th} POD_{NO\text{-}LS} / 95^{th} POD_{LS}$$

Where 95th $POD_{NO\text{-}LS}$ is the overall 95th percentile origination delay of the Call Server cluster without load sharing and 95th POD_{LS} is the overall 95th percentile origination delay of the Call Server cluster with load sharing.

ODI should be more than 1 for a better load balanced system. Higher the value of ODI factor the better is the load sharing across different Call Servers of the cluster.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3.4 Duration of Simulation Experiments

The duration of a simulation experiment is long enough so that the output is stabilized. The transients in the output of the system can be eliminated by ignoring the output data of an initial period of time [Jai91]. This initial period of time is referred to as *warm up time*. The value for this warm up duration is such that after this initial period of time, variation in the output data is not very significant. In accordance with the advice of Nortel engineers, we simulated call processing activities in the switch for eight minutes and the output data for the first four minutes is discarded [Kop04.1]. In order to verify the adequacy of the chosen time periods, we conduct an experiment in which the simulation was run for a long period of time (simulating 20 minutes of call processing activities). This experiment corresponds to Call Servers operating in a cluster and using one of the load sharing techniques. One of the CSs is overloaded 5% above the EngCap and the other two CSs are operating 5% below the EngCap. The values of the 95^{th} POD and the average origination delay obtained after discarding the data corresponding to the first four minutes of the run were observed as a function of the simulated time. The values of the 95^{th} POD and the average origination delay were observed to stabilize even before the simulation time of eight minutes was reached.

This demonstrates that the simulation of eight minutes of call processing activities in the switch and discarding the data corresponding to the first four minutes in computing the performance indices are adequate for analyzing system performance. For these simulation experiments, two Intel Pentium 4 based machines running MS Windows 2000 are used. The physical memory available on each of the machine is one gigabyte. A

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

typical simulation of eight minutes on one machine takes more than twenty hours for three Call Servers based cluster.

4.4 Model Verification and Validation

4.4.1 Verification

For verification of the model, the following techniques are used.

1- Simulations are run for shorter periods of time with controlled input parameters. The different states in all state diagrams are verified by checking the logged messages and state information stored in the simulator log files.

2- Little's Law is applied to the input, output, broker and origination queues [Jai91]. The relationship is observed to hold for a number of different parameters sets.

3- The model output is checked for reasonableness under a variety of settings of the input parameters. The simulator program logs a wide variety of output statistics for verification purposes. The Utilization Law is deployed to verify the utilization of the processing element.

4- The input parameters are also logged at the end of the simulation to verify that that input parameters are not changed during the experiment.

5- Arrival rate and inter-arrival rate time of incoming calls are logged to vector files of OPNET. The analytical tool of OPNET uses the vector file as input to visualize incoming traffic. This helps to verify that the input traffic streams arriving at different Call Servers are independent of each other.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.4.2 Validation Process

The correctness of a simulation model is measured by the accuracy of the model output to the output of a real system. Validation of a model confirms the assumptions used in the developments of the model if the outputs of the model are close to the outputs of the real system [Jai91]. In the current context, comparison of the outputs of the simulation model and the real system was not possible because of the non-existence of a real Call Server cluster. So an analytical model developed in the first phase of this research by Majumdar [Maj04] was used by Sun to validate the first version of the simulation model developed by Sun [Sun04]. The similarity between the analytical predictions and the simulation results can be used to validate the simulation model [Jai91]. There were two types of validations which were carried out by Sun et al. for the first version of the simulation model. They validated the output results with the predicted results of the analytical model. Secondly they also developed a simulation model for the original system without clustering. The output of the simulation model based on the original system was found to be in good agreement with the output of the real system. The call processing functionality of the simulation model used in this thesis is the same as the first version of the simulation model. The current version developed for this thesis includes a number of additional and revised modules such as the broker, the Input/Output and the intelligent PM. The design of these new simulation models, especially the UML diagrams and the state transition models are approved by the designers of Call Servers at Nortel. These diagrams were discussed and verified with Nortel Networks experts before starting the implementation. The output of the simulation model, before the incorporation of the load sharing strategies, is compared for

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a selected set of parameters to the output of the first version of the simulation model. The outputs of both the simulation models are found to be very close to each other and differ only by a maximum of 2%. This small variation is expected and is due to the modification made to the first version of the simulation model.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5 Results of Experiments

In this chapter, the results of simulation experiments are presented and discussed. These experiments are conducted to investigate the performance of the load sharing strategies. Note that all the load sharing strategies differ only in terms of redirection policies. The other components of the strategies are the same. The impacts of different workload parameters on the performance of the strategies are also discussed. A factor at a time approach is used: One parameter is varied at a time while the others are held at their default values presented in Table 5-1. A brief summary of experiments is presented in the following paragraph.

The effects of History Data and Window Size which are used for the calculation of moving 95^{th} percentile origination delay are explained in Section 5.2. Section 5.3 describes the effect of the transfer policy threshold and the location policy threshold on the performance of the system. The attributes such as δ and P, associated with different redirection policies, are investigated in Section 5.4. The effectiveness of all the redirection policies is compared in Section 5.5 at different overloading levels by varying the load intensity of the overloaded CS and the lightly loaded CSs. Section 5.6 explains the effect of the composition and the size of the cluster on the performance results. The effects of different data sharing policies, duration of different call events and the locality factor are discussed in Sections 5.7, 5.8 and 5.9 respectively.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.1 Performance Metrics

A number of performance indices are observed for these experiments. The details of these performance indices are discussed in Section 4.3.3. In this chapter, the results presented include only overall 95th percentile origination delay (95th POD) and Origination Delay Improvement factor (ODI). Additional results with other performance indices such as the average origination delay are presented in Appendix A and Appendix B.

5.2 Analysis of Load Index Attributes

The selection of the moving 95th percentile origination delay as the load index and the two attributes associated with it are explained in detail in Section 3.1.1. The attributes are History Data and Window Size. In this section, we investigate the effect of these two attributes in two different sets of experiments. In the first set of experiments, we analyze the performance of the cluster system for different proportions of History Data. In the second set, we explore different values of Window Size (T_{moving}) to study its effect on the performance of the cluster system.

These two sets of experiments assume default values for the input parameters listed in Table 5-1. One CS (CS1) is overloaded slightly above the EngCap (2%) and the other two CSs (CS2 and CS3) are operating 2% below the EngCap.

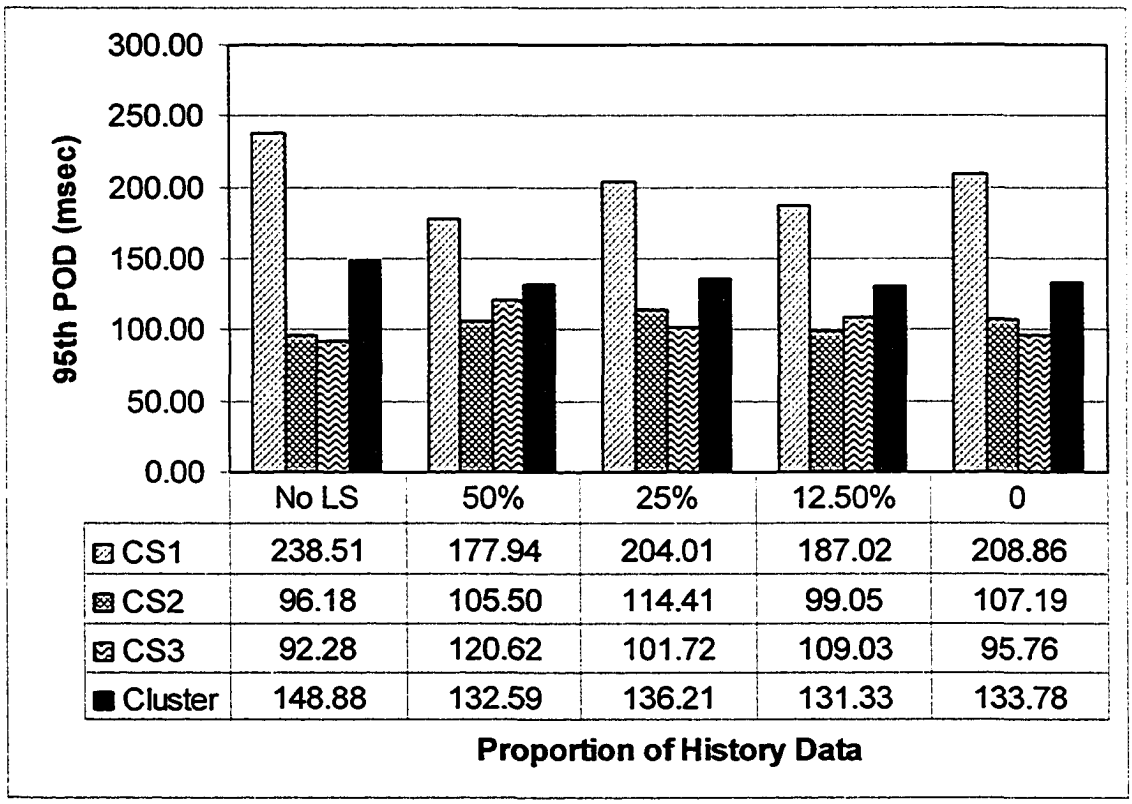
5.2.1 Effect of History Data

For this set of experiments, we assume a Window Size of 0.25 sec and vary the proportion of History Data from 0 to 50%. The results collected for these experiments are presented in Figure 5-1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5-1: Default values of input parameters

Parameter	Value
Cluster size	3
Load intensity on CS1	3210 Kcalls/hr
Load intensity on CS2 and CS3	3100 Kcalls/hr
Redirection policy	Adaptive #2
Window Size	1 sec
History Data	50%
Call event durations (dialing, answering, talking and hanging up)	Truncated exponential (mean of 5sec)
Simulation duration	8 minutes
Warm up time	4 minutes
Number of PE	3
Service time	Table 4-2
Data sharing policy	Remote access
Locality factor	50%
Transfer policy threshold	175 msec
Location policy threshold	105 msec



	No LS	50%	25%	12.50%	0
CS1	238.51	177.94	204.01	187.02	208.86
CS2	96.18	105.50	114.41	99.05	107.19
CS3	92.28	120.62	101.72	109.03	95.76
Cluster	148.88	132.59	136.21	131.33	133.78

Figure 5-1: Comparison of system performance for different proportions of History Data

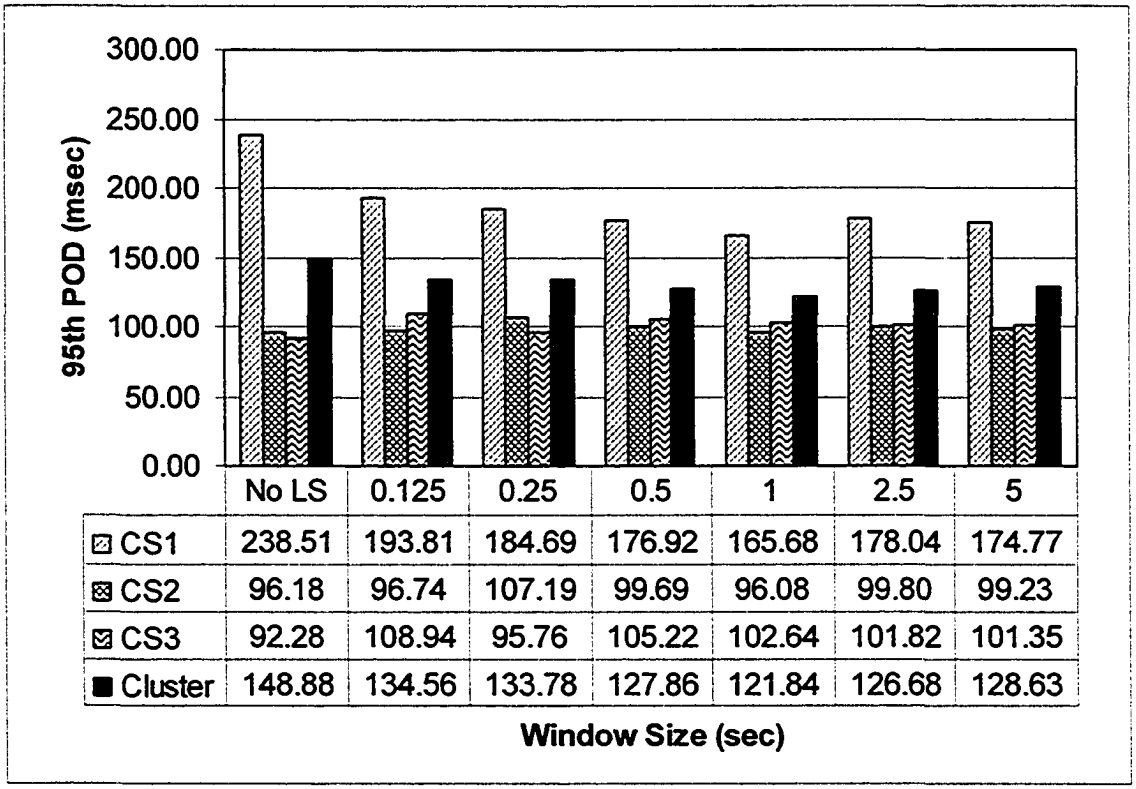
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The first column in the table of Figure 5-1 shows the results when no load sharing technique is used and it involves no History Data. The first conclusion is that the use of History Data is not degrading the overall performance of the cluster. The overall 95th POD of the cluster is almost the same for all History Data values. The 95th PODs of the individual Call Servers are slightly different for different values of the History Data. But there is not a very well defined pattern observed in these results. The more close values of the 95th POD of individual Call Servers and of the cluster indicate that a more effective load sharing has been achieved among the Call Servers of the cluster. The 95th POD values on the different CSs are closest to one another for the History Data of 50% as compared to the other values of History Data. So we chose 50% of History Data to be used in the further experiments.

5.2.2 Effect of Window Size

A very small window may make the load sharing strategy act prematurely due to an instantaneous change in load. A very large window on the other hand, may make the system too sluggish to react to changes in system load. So the Window Size is very important to avoid such situations. To study the effect of Window Size on the performance of the system, we use different Window Sizes for this set of experiment. The default values of the other input parameters are the same as listed in Table 5-1. The results are presented in Figure 5-2. The first column in the table of Figure 5-2 shows the results of the CS cluster without using any load sharing technique and it involves no Window Size parameter. Figure 5-2 shows that the overall performance of the cluster with any Window Size is always better than a no load sharing system.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



	No LS	0.125	0.25	0.5	1	2.5	5
CS1	238.51	193.81	184.69	176.92	165.68	178.04	174.77
CS2	96.18	96.74	107.19	99.69	96.08	99.80	99.23
CS3	92.28	108.94	95.76	105.22	102.64	101.82	101.35
Cluster	148.88	134.56	133.78	127.86	121.84	126.68	128.63

Figure 5-2: Comparison of system performance for different values of Window Size

The 95th POD of the cluster system and the overloaded CS (CS1) are reduced gradually as the Window Size in increased from 0.125 sec to 1.0 sec. The results obtained with the Window Size of 1.0 sec are the best as compared to the results with other values of Window Size. As the Window Size is increased beyond 1.0 sec, the 95th PODs of the overloaded CS and the cluster start increasing again. This confirms that the duration of the Window Size should neither be too small nor too large.

The smaller durations of the Window Size imply that the load index can include more transient behavior and the load sharing can become active more often than desired. This is why as we initially increase the duration of the Window Size, the values of 95th PODs of all CSs become closer to one another. We chose 1.0 sec as the default value of Window Size for further experiments.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3 Threshold Analysis

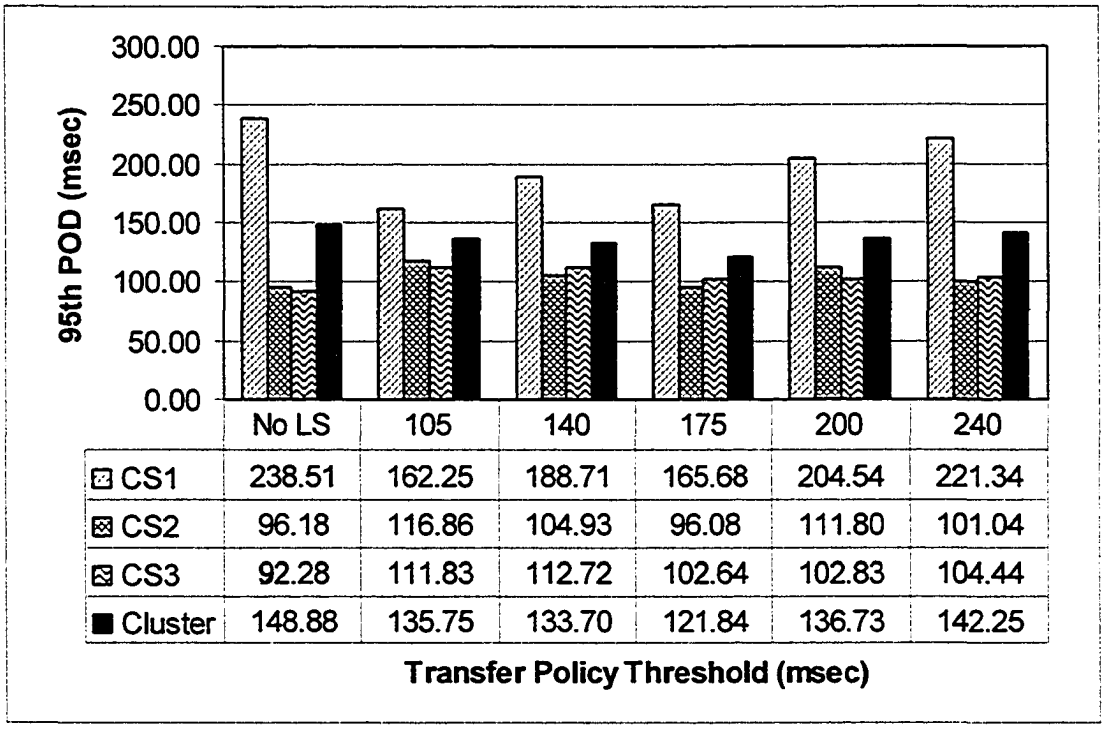
As already discussed in Section 4.3.2 of the last chapter, there are two types of thresholds corresponding to two different components of load sharing. One threshold is for the transfer policy and the other is for the location policy. These thresholds should correspond to the border line between an overloaded system and the non-overloaded system. For telephone switches, this border line is the engineered capacity of the system and the 95th POD on this border line is denoted by T_{ec}. A Call Server that has a 95th POD more than T_{ec} is considered overloaded and the one that has 95th POD below T_{ec} is considered operating as a lightly loaded CS.

The set of experiments described in this section assumes default values of the input parameters as listed in Table 5-1 except thresholds of the transfer policy and the location policy. We vary the threshold of the transfer policy in the first experiment and then the threshold of the location policy in the next experiment.

5.3.1 Effect of Transfer Policy Threshold

In the first experiment, we keep the threshold of the location policy equal to the 95th POD corresponding to the engineered capacity of a CS (T_{ec} or 140 msec) and vary the threshold of the transfer policy from 100 msec to 240 msec. To see the effectiveness of load sharing, the load intensity of one CS (CS1) is kept slightly above (2%) the EngCap while the other two CSs (CS2 and CS3) are operating 2% below the EngCap. The results of this experiment are presented in Figure 5-3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



	No LS	105	140	175	200	240
CS1	238.51	162.25	188.71	165.68	204.54	221.34
CS2	96.18	116.86	104.93	96.08	111.80	101.04
CS3	92.28	111.83	112.72	102.64	102.83	104.44
Cluster	148.88	135.75	133.70	121.84	136.73	142.25

Figure 5-3: Comparison of system performance for different values of the transfer policy threshold

The first column in the table of Figure 5-3 (with No LS label) shows the results when no load sharing technique is used. The results obtained, using load sharing, show improved system performance as compared to the no load sharing situation. For a threshold value of 105 msec, the 95th POD of the overloaded CS (CS1) is reduced significantly but it also pushes the 95th PODs of the lightly loaded CSs (CS2 and CS3) to higher values. So the overall 95th POD of the cluster is marginally higher as compared to results with other values of the threshold. A similar behavior is observed in results of the threshold of 140 msec.

For a threshold value of 175 msec, the 95th POD of the CS1 is reduced significantly without pushing the 95th PODs of the lightly loaded CSs to higher values. So, the overall 95th POD of the cluster is better than the lower threshold values (105ms and 140 msec). This is because the overloaded CS avoids transferring the calls due to small spikes in the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

load of the CS. For higher values of threshold such as 200 msec and 240 msec, there is not much reduction in the 95th POD of the overloaded CS (CS1) and therefore the overall 95th POD of the cluster is high as well.

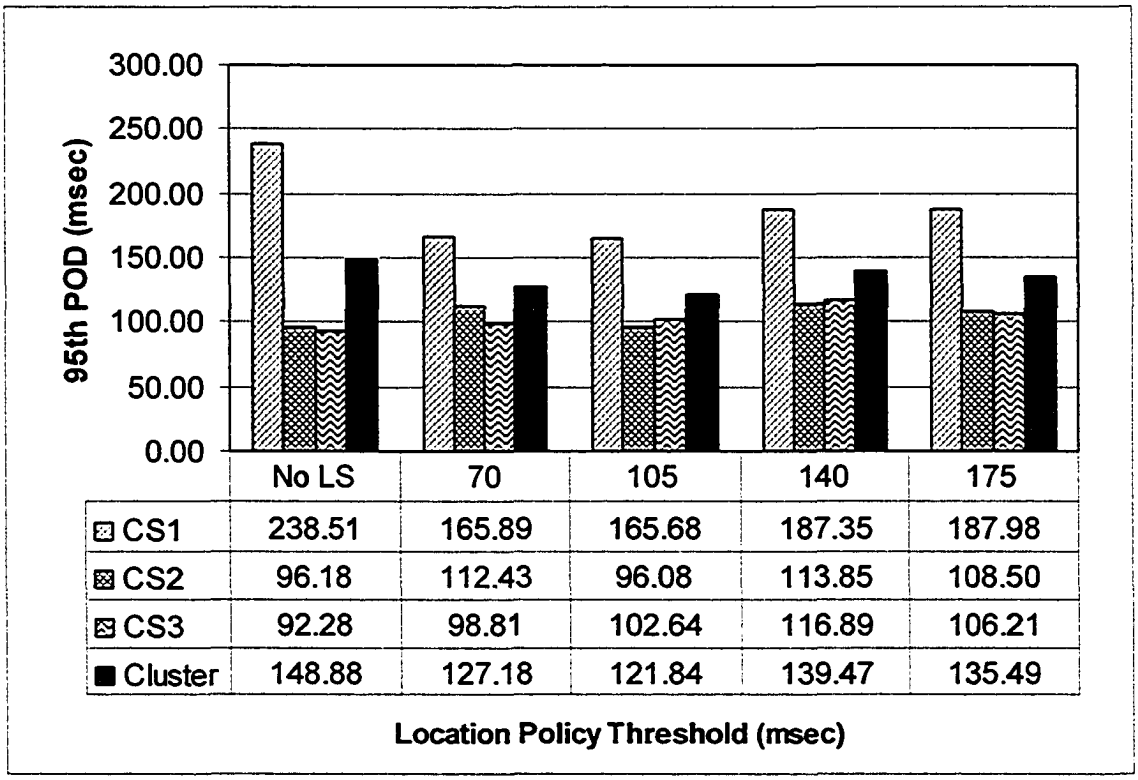
Figure 5-3 shows a graphical view of the results of the experiment. The four bars (from left to right) are for CS1 (overloaded), CS2, CS3 and the cluster respectively. The closer the 95th PODs, the better the load sharing achieved in the system. We can see that the bars are closest to one another for the threshold values of 175 msec and 105 msec. But the overall 95th POD of the cluster is better for the threshold value of 175 msec. For the rest of the experiments described on the following pages, we use a threshold value of 175 msec because of its immunity against small and temporary spikes in the load.

5.3.2 Effect of Location Policy Threshold

The second experiment investigates the effect of the location policy threshold on the overall performance of the Call Server cluster. As discussed in Section 3.1.3, the location policy threshold is used to identify the potential receiver Call Servers. In this experiment, we keep value of the threshold for the transfer policy equal to 175 msec and vary the value of the threshold for the location policy from 70 msec to 175 msec. Figure 5-4 shows the 95th POD results for the individual CSs and the cluster.

The first column in the table of Figure 5-4 (No LS) shows results when no load sharing is used. Again, the results for all values of the threshold show better performance than the results without load sharing. But some values of threshold produce better results than the others.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



	No LS	70	105	140	175
CS1	238.51	165.89	165.68	187.35	187.98
CS2	96.18	112.43	96.08	113.85	108.50
CS3	92.28	98.81	102.64	116.89	106.21
Cluster	148.88	127.18	121.84	139.47	135.49

Figure 5-4: Comparison of system performance for different values of location policy threshold

The results for threshold values of 70 msec and 105 msec are better than the results achieved with threshold values of 140 msec and 175 msec. This is because of the more sensitive behavior of Call Servers when they operate near the EngCap that corresponds to the knee of the curve between 95th POD and the load intensity on a single Call Server. A small increase in the number of new incoming calls from a remote sender CS can overload such a receiver CS. So the threshold for the location policy should be chosen to be lower than the 95th POD corresponding to the EngCap of a CS.

The overall 95th PODs for the threshold values of 70 msec and 105 msec are within ±5% of one another. We choose 105 msec as the threshold of the location policy because of the fact that it will increase the probability of finding the receiver CSs in the Call Server cluster in comparison to the situation in which a 70 msec value is used as the threshold.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.4 Analysis of Redirection Policies

In this set of experiments, we investigate the performance of the first three redirection policies, proposed in Chapter 3. We analyze these policies for different values of the key attributes associated with them such as the different values of the redirection factor for the static redirection policy. Similarly, the sender load based and the sender/receiver load based redirection policies are analyzed for different values of the maximum redirected proportion P. The results of these experiments provide insights into system performance as well as help in choosing suitable values for these key attributes to be used in further experimentations. The default values of input parameters used in these experiments are listed in Table 5-1. Note that one CS (CS1) is overloaded 2% above the EngCap and the other two CSs (CS2 and CS3) are operating 2% below the EngCap.

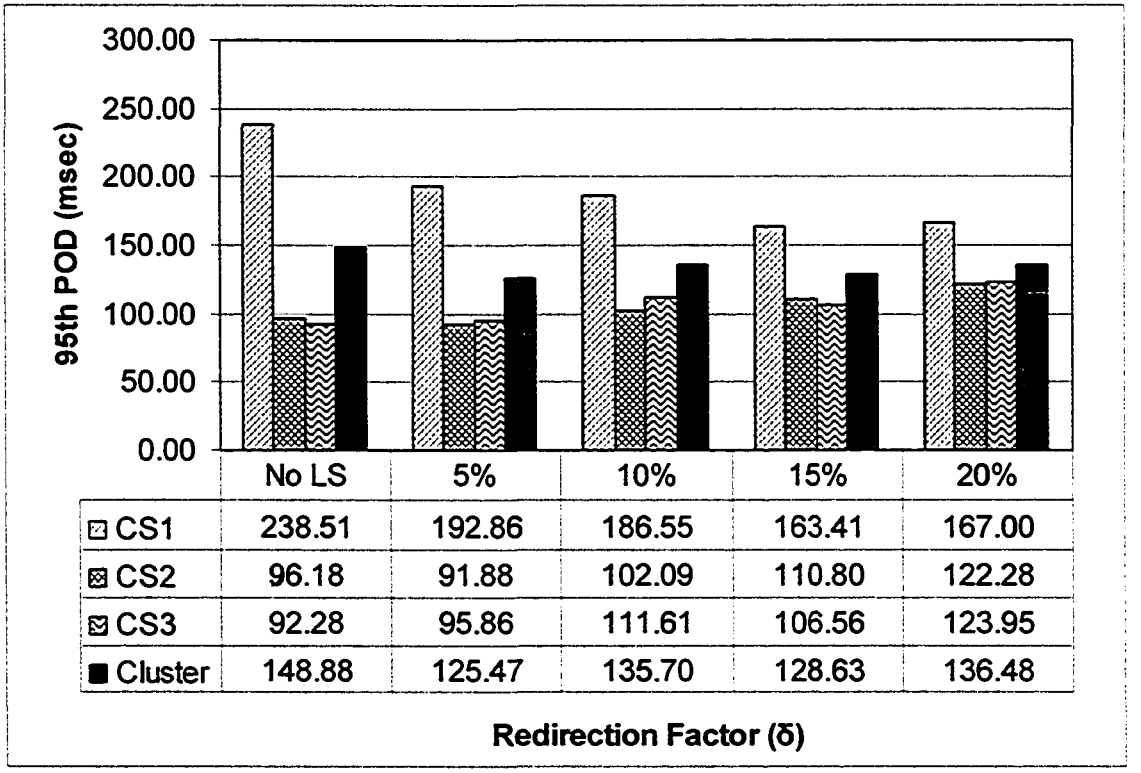
5.4.1 Static Redirection Policy

In the fixed or static redirection policy, a fixed proportion of the new incoming calls, redirection factor (δ), is forwarded to the receiver Call Server. In this experiment, we analyze different values of the redirection factor, δ, ranging from 5-20 %. The results are summarized in Figure 5-5.

The first column in the table of Figure 5-5 shows the results without load sharing. The results for different values of δ are improved as compared to the results obtained without load sharing. The overall 95th PODs of the cluster for all values of δ differ only by $\pm$ 5%. But the 95th POD of the overloaded CS (CS1) decreases as the value of δ is increased from 5% to 15%. For lower values of δ (5% and 10%), the improvement of 95th POD for the overloaded CS is less because of the lesser transfer of new calls to the remote CSs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

That is why the 95th PODs of the lightly loaded CSs (CS2 and CS3) are also not increased substantially (see Figure 5-5).



	No LS	5%	10%	15%	20%
CS1	238.51	192.86	186.55	163.41	167.00
CS2	96.18	91.88	102.09	110.80	122.28
CS3	92.28	95.86	111.61	106.56	123.95
Cluster	148.88	125.47	135.70	128.63	136.48

Figure 5-5: Effect of δ on the performance of the Call Server cluster using the static policy as the load sharing technique

For δ = 20%, the improvement in the 95th POD of the overloaded CS (CS1) is not better than the 95th POD of CS1 with δ = 15% although the 95th PODs of the lightly loaded Call Servers show higher values. The higher values of 95th PODs of the lightly loaded CSs are indicators of the higher number of calls received. This can lead to *thrashing* that is explained in the next paragraph.

If an overloaded node transfers a large proportion of its tasks to a lightly loaded node, then it is possible that the lightly loaded node may become overloaded because it is receiving too many tasks. Since the overloaded node has transferred bulk of its tasks, so

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

this node may be identified (temporarily) as the lightly loaded node or receiver in the cluster of nodes and might start receiving tasks from other overloaded nodes of the cluster and may again become an overloaded node. This situation leads to the degradation of the performance of not only the individual nodes but also the performance of the overall cluster. This phenomenon is called thrashing. At a δ=20%, the overloaded CS (CS1) seems to be redirecting too many new calls and overloading the lightly loaded CSs (CS2 and CS3). This is clearly indicated by higher values of 95th POD of lightly loaded Call Servers (CS2 and CS3) for δ = 20% (see Figure 5-5). But due to thrashing, the 95th POD of the overloaded CS (CS1) is not reduced accordingly.

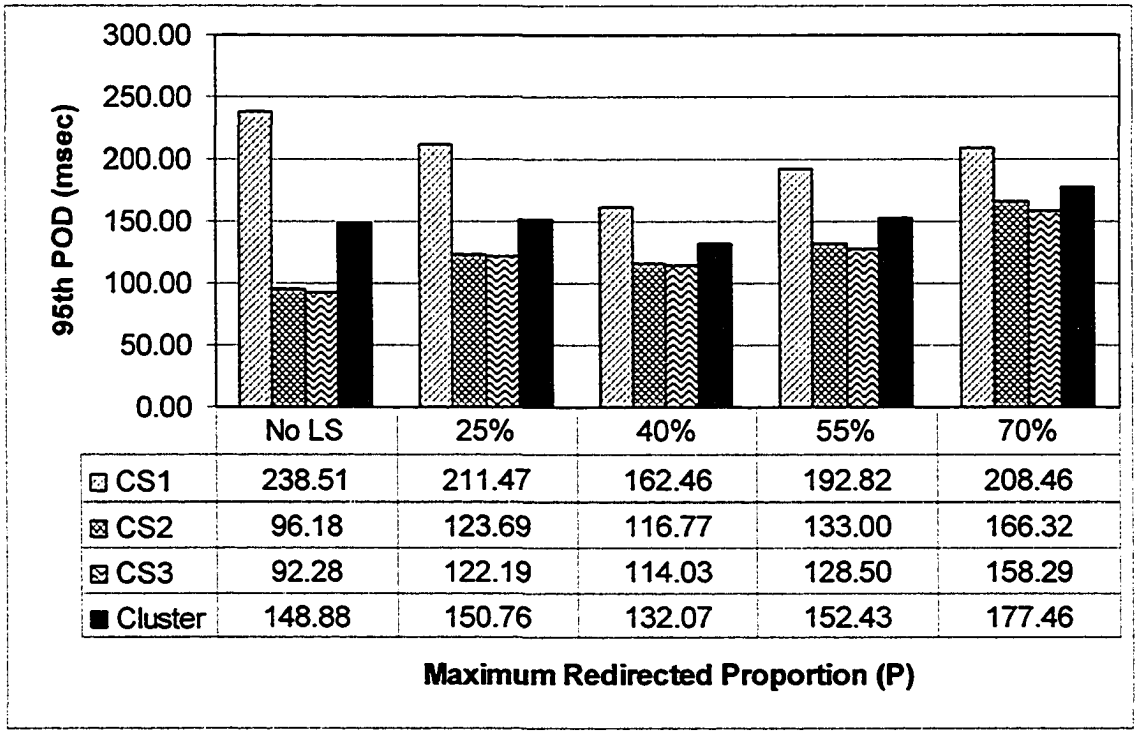
5.4.2 Adaptive #1: Sender Load Based Redirection Policy

In the sender load based redirection policy, the redirection factor, δ, is adaptive and changes with the current load status of the sender CS. The higher the load on the sender CS, the higher the number of calls transferred to the receiver CS. As discussed in Section 3.2.2, P is constant which corresponds to the maximum redirected proportion of incoming calls. The value of P can be anything between 0 and 100%. For this experiment, we analyze the effect of different values of P on the performance of the system. We started with a value of 25% and then increment in steps of 15%. The results are summarized in Figure 5-6.

The first column in the table of Figure 5-6 (No LS) shows the results of an experiment without load sharing. Using the sender load based redirection policy as a load sharing technique, the best results are observed at P = 40%. For all other values of P, the 95th POD of the overloaded CS (CS1) is improved but the overall 95th POD of the cluster is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

degraded in comparison to the overall 95th POD without using any load sharing technique.



	No LS	25%	40%	55%	70%
CS1	238.51	211.47	162.46	192.82	208.46
CS2	96.18	123.69	116.77	133.00	166.32
CS3	92.28	122.19	114.03	128.50	158.29
Cluster	148.88	150.76	132.07	152.43	177.46

Figure 5-6: Effect of P on the performance of the Call Server cluster

For a P = 25%, the number of redirected calls from the overloaded CS (CS1) to the lightly loaded CSs (CS2 and CS3) is not large enough. Thus, the 95th POD of the overloaded CS (CS1) is improved only slightly and the 95th PODs of the lightly loaded CSs (CS2 and CS3) are also not increased much. The overall 95th POD of the cluster is observed to be slightly worse in comparison to the situation without using load sharing.

For a P = 40%, both the 95th POD of the overloaded CS and the overall 95th POD of the cluster are improved significantly as compared to the situation without using the load sharing technique.

For higher values of P such as 55% and 70%, a degradation in performance is observed. The 95th PODs of the lightly loaded Call Servers (CS2 and CS3) are pushed to quite high

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

values but the 95th POD of the overloaded CS (CS1) is not reduced accordingly. This is because of the thrashing phenomenon explained in the last section. The overloaded Call Server (CS1) tries to send too many calls to the receiver CSs (CS2 and CS3). This overloads the receiver Call Servers that start sending back some of their calls to the Call Server that was originally overloaded (CS1). The phenomenon of thrashing affects the performance of the system in two different ways. First it can cause transfer of new calls from the Call Servers that were originally lightly loaded. Secondly, it hinders transfer of new calls from the originally overloaded Call Server because the number of potential receiver CSs is now reduced.

In Figure 5-6, the 95th POD values are closest to one another for P = 40%. For rest of the experiments, we use P = 40% for the sender load based redirection policy.

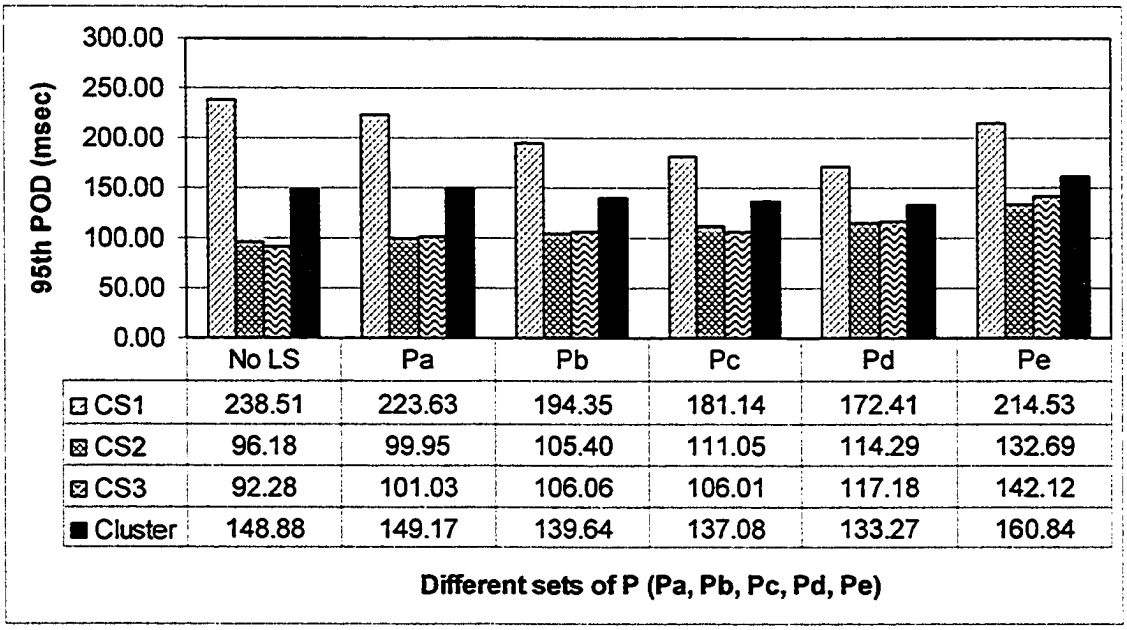
5.4.3 Adaptive #2: Sender/ Receiver Load Based Redirection Policy

As described in Section 3.2.3, this policy considers load status of both the sender and the receiver CS to calculate the redirection factor, δ, of incoming calls at run time. For this policy, we assume P as the set of four values (P1, P2, P3, P4) in increasing order and each value of P in the set corresponds to one availability level of the receiver CS as explained in Section 3.2.3. Based on the availability level of the receiver CS, a corresponding value of P is selected and applied in equation (3) to calculate n. So in this experiment we investigate the effect of P (shown in Table 5-2) on the performance. These five sets of P are labeled as Pa, Pb, Pc, Pd and Pe. The results are shown in Figure 5-7.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5-2: Different sets of P

Set Labels	Set of P (P1,P2,P3,P4)
Pa	(6,12,18,25)
Pb	(10,20,30,40)
Pc	(14,28,42,55)
Pd	(18,36,54,70)
Pe	(25,50,75,100)



	No LS	Pa	Pb	Pc	Pd	Pe
CS1	238.51	223.63	194.35	181.14	172.41	214.53
CS2	96.18	99.95	105.40	111.05	114.29	132.69
CS3	92.28	101.03	106.06	106.01	117.18	142.12
Cluster	148.88	149.17	139.64	137.08	133.27	160.84

Figure 5-7: Effect of different sets of P on the performance of the Call Server cluster

The first column with 'No LS' label in the table of Figure 5-7 shows the results without using any load sharing technique. As seen in Figure 5-7 the results for the set Pa are not good because of the lesser transfer of new calls from the overloaded Call Server (CS1) to lightly loaded Call Servers (CS2 and CS3). The 95th POD of CS1 is only marginally improved and the effect on the 95th PODs of lightly loaded CSs is also negligible as shown in the results. The overall 95th POD for this set of P also shows that there is no improvement in performance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

As we increase the values in these sets of P, more improvements in the performance of the Call Server cluster are observed. For set Pb, the 95th POD of the overloaded CS (CS1) is reduced to 194.35 msec and the overall 95th POD is 139.64 msec. The set Pc uses higher values of P (14,28,42,55). The 95th PODs of the overloaded CS and of the overall cluster are improved further to 181.14 msec and 137.08 msec respectively. The improvement continues with set Pd (18,36,54,70). But this trend does not continue with Pe that includes the highest level of the redirected proportion (100%). The results obtained for Pe (see Figure 5-7) indicate that these higher values degrade the performance of the cluster. Actually, more transfer of new calls from the overloaded CS (CS1) overloads the lightly loaded Call Servers (CS2 and CS3). It can be seen from the higher values of the 95th PODs of the lightly loaded CSs (132.69 msec and 142.12 msec respectively). But the 95th POD of CS1, which was originally overloaded, is only reduced from 238.51 msec to 214.53 msec. This is possibly because of the transfer of calls back to CS1 from CS2 and CS3 in the situation when CS1 has become lightly loaded for a temporary period of time. This phenomenon of thrashing has already been discussed in the previous sections. From this analysis, we choose set Pd for further experimentations.

5.5 Effectiveness of Redirection Policies at Different Load Levels

The core set of experiments presented in this thesis is to study the effectiveness of different redirection policies for various unbalanced load configurations in a Call Server cluster. In this section, we present three sets of experiments for these unbalanced load configurations. The first set of experiments is carried out to investigate effectiveness of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

all the redirection policies, when the load intensity of one overloaded CS is varied above the EngCap by small amounts (0 – 7.5%). In the second set, the load intensity of the lightly loaded Call Servers is varied to observe its effect on the performance of redirection policies. These two sets of experiments are labeled as "One Overloaded Call Server Scenario" and "Variable Lightly Loaded Call Servers Scenario". In the third set (Special Scenario), we run a few experiments to see the effectiveness of all the redirection polices for very high levels of overloading such as 25% and 30% above the EngCap.

Another issue is the number of overloaded and lightly loaded Call Servers in a cluster. For example, in a cluster of three Call Servers, we can vary the load on one or two Call Servers at a time. In this chapter, we only included results of experiments that have one overloaded Call Server. Either we vary the load intensity of the overloaded CS or the load intensity of the lightly loaded CSs. The results of experiments with two overloaded Call Servers in the cluster have also shown the same pattern of performance improvement but not as much as we will see in the following sub-sections for one over overloaded CS. This is due to only one lightly loaded CS in the cluster. The experiments with two overloaded Call Servers use same default value of input parameters listed in Table 5-1 and their results are summarized in Appendix C and Appendix D.

5.5.1 One Overloaded Call Server Scenario

In these experiments, a cluster of three Call Servers is used. The load intensities of the two Call Servers (CS2 and CS3) are maintained at 5% below the engineered capacity for all these experiments. But the load intensity of the third Call Server (CS1) is varied from 0% to 7.5% above the EngCap. Table 5-3 shows different load configurations with one

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

overloaded Call Server and two lightly loaded Call Servers. All the other input parameters are held at their default values described in Table 5-1.

Table 5-3: Load configurations for the one overloaded Call Server scenario

Load Configuration	Load Intensity (Kcalls/hr)	
	Overloaded CS (CS1)	Lightly loaded CSs (CS2 and CS3)
1OL-2LL_1	3150 (EngCap)	3000
1OL-2LL_2	3220 (2.5% above EngCap)	3000
1OL-2LL_3	3300 (5% above EngCap)	3000
1OL-2LL_4	3400 (7.5% above EngCap)	3000

95th POD

For these experiments, we present overall 95th POD of the cluster for various load configurations as a graphical and tabular form in Figure 5-8. The results of the average origination delay show the same pattern of performance for all redirection policies. These results are presented in Appendix A.

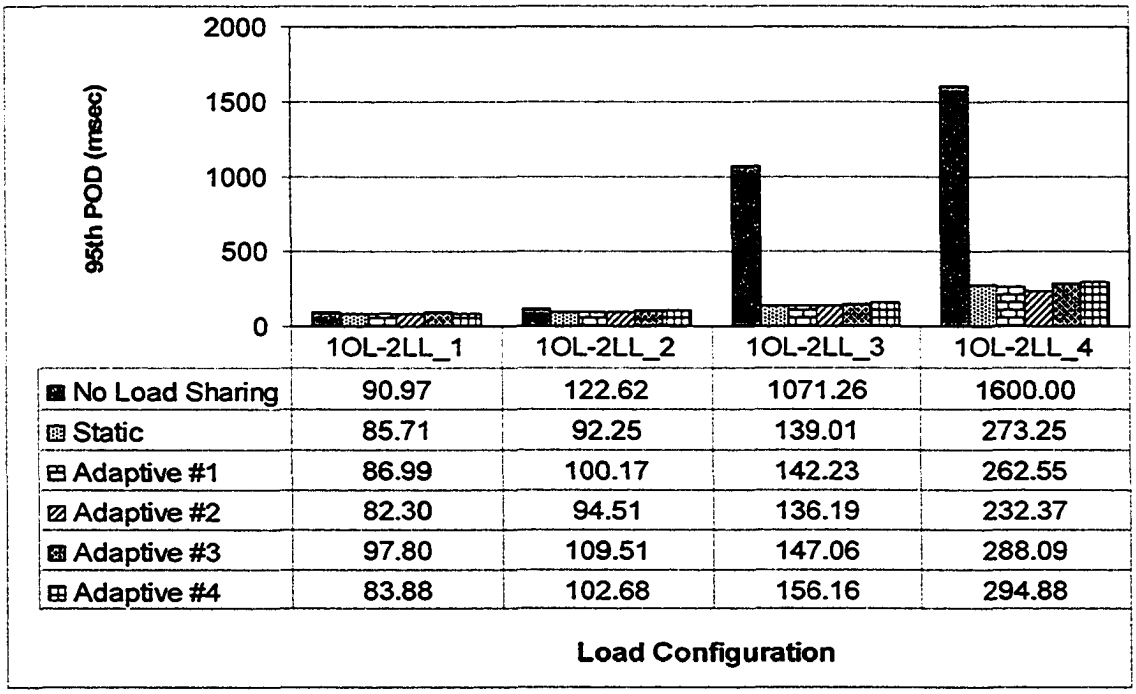


Figure 5-8: Comparison of the 95th PODs achieved using different redirection policies for the one overloaded Call Server scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The static policy has shown reasonable performance for the first three load configurations. But it could not maintain the 95th POD of the cluster under T_{ec} for the 1OL-2LL_4 configuration. This is because of the fixed value of the redirection factor. The load intensity of the overloaded CS in this configuration is 7.5% above the EngCap, and the redirection factor of the static policy (δ = 15%) may not be large enough to stop elevating the 95th POD of that CS.

Adaptive #1 also performs reasonably well for most of the load configurations. The 95th POD for the last load configuration (1OL-2LL_4) is again high but marginally better than the 95th POD obtained with static redirection policy. Adaptive #2 performs very well for all the load configurations. This might be because of its capability to adapt the value of δ according to the load status of both the sender and the receiver CS. So for this range of load configurations, we can conclude that the sender/receiver load based redirection policy shows better performance as compared to all other redirection policies. Adaptive #3 and Adaptive #4 show satisfactory performances except for the last load configuration (1OL-2LL_4). Both policies show lower improvements in 95th POD for this configuration. This is because both the policies tend to redirect calls more aggressively than the other policies. The overloaded CS is operating 7.5% above the EngCap and the lightly loaded CSs are operating only 5% below the EngCap. So in an attempt to redirect calls at a faster rate, these policies end up overloading the receiver.

In Figure 5-8, the first bar at each load configuration shows the 95th POD of the cluster without using any load sharing technique. The rest of the bars in the graph represent 95th POD for different redirection policies. The differences between first bar and the rest of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the bars indicate how effectively these redirection policies have reduced the 95th POD of the cluster as compared to the 95th POD achieved without load sharing.

ODI Factor

The improvement in the 95th POD is also measured by the ODI factor. The comparison of the ODI factor at different load configurations is shown in Figure 5-9. Although marginally, but the ODI factor achieved with Adaptive #2 is the winner at all load configurations.

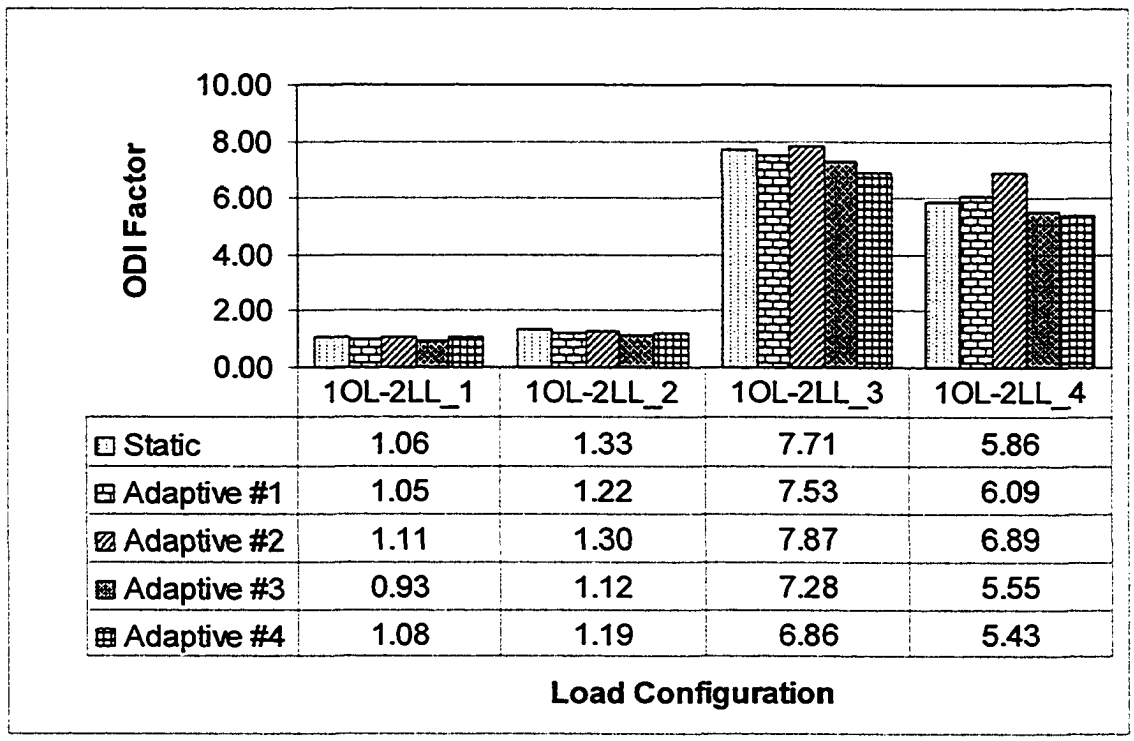


Figure 5-9: Comparison of the ODI factors achieved using different redirection policies for the one overloaded Call Server scenario

5.5.2 Variable Lightly Loaded Call Servers Scenario

In the scenario presented in the last section, we explored the effectiveness of the redirection policies for different load levels of the overloaded Call Server. With the scenario presented in this section, we investigate how the redirection policies perform if

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the lightly loaded Call Servers are operating at different levels. For these experiments, we assume one overloaded Call Server (CS1) in the cluster is operating 5% above the EngCap, and the load intensities of the two lightly loaded Call Servers (CS2 and CS3) are varied. All the other input parameters are held at their default values described in Table 5-1.

Based on the different levels of load intensity of the lightly loaded Call Servers, we define four load configurations in Table 5-4. The load intensities on the lightly loaded Call Servers are highest for the first load configuration (1OL-2LL_1) and decreases for the following load configurations.

Table 5-4: Load configurations for the variable lightly loaded Call Servers scenario

Load Configuration	Load Intensity (Kcalls/hr)	
	Overloaded CS (CS1)	Lightly loaded CSs (CS2 and CS3)
1OL-2LL_1	3300	3150 (EngCap)
1OL-2LL_2	3300	3080 (2.5 % below EngCap)
1OL-2LL_3	3300	3000 (5% below EngCap)
1OL-2LL_4	3300	2833 (10% below EngCap)

95th POD

The overall 95th POD for different load configurations using all the redirection policies are summarized in Figure 5-10. The average origination delays also show the same ranking of redirection policies for this scenario. The details of these results are presented in Appendix B. The first row in the table of Figure 5-10 shows 95th POD of the system without using any load sharing technique. If we compare the overall 95th POD achieved with all redirection policies at different load configurations, Adaptive #2 is found to be consistently performing better than the other policies (see Figure 5-10).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

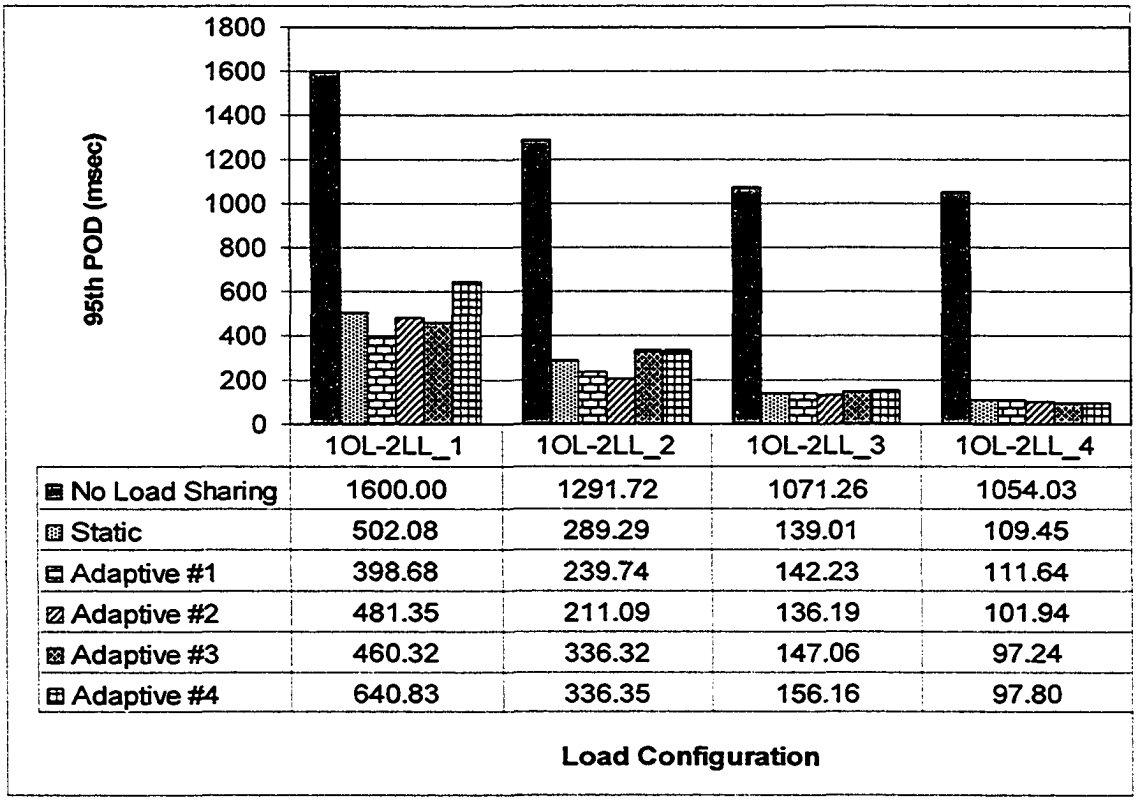


Figure 5-10: Comparison of the 95th PODs achieved using different redirection policies for the variable lightly loaded Call Servers scenario

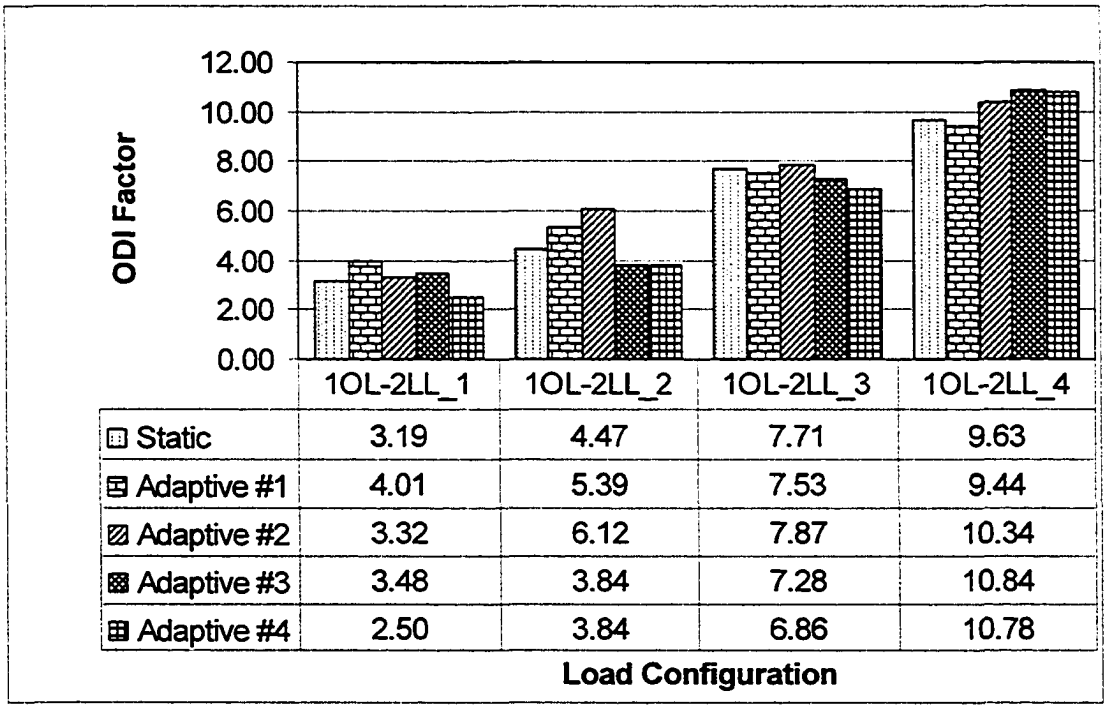
Figure 5-10 shows the variation of the 95th PODs at the different load configurations in the form of a bar graph. The first solid bar represents the 95th POD without load sharing. For the first two load configurations (1OL-2LL_1 and 1OL-2LL_2), the lightly loaded Call Servers are operating at a load intensity equal to the EngCap and 2.5% below the EngCap respectively. So both the lightly loaded CSs have very little spare capacity to absorb the extra load of the overloaded CS. This is the reason for the high values of 95th PODs observed for these two load configurations. For the first load configuration (1OL-2LL_1), Adaptive #1 shows a slightly better performance than all other policies. For the second load configuration (1OL-2LL_2), Adaptive #2 shows better performance than the other policies. The 95th POD of the overloaded CS (CS1) with Adaptive #1 (239.74

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

msec) is also closer to the value collected with Adaptive #2 (211.09 msec), but the 95th PODs obtained for the other policies are higher.

At the load configurations 1OL-2LL_3 and 1OL-2LL_4, the load intensities of the lightly loaded CSs are 5% and 10% below the EngCap respectively, so the performances of all the redirection polices are significantly improved. Adaptive #2 is the winner for the load configuration 1OL-2LL_3. Adaptive #3 and Adaptive #4 show better performance than others at load configuration 1OL-2LL_4. For Adaptive #3 and Adaptive #4, the redirection factor, δ, is proportional to the spare capacity of the receiver CS. So the two lightly loaded CSs operating 10% below the EngCap, in case of 1OL-2LL_4, aid in producing the improved performances of Adaptive #3 and Adaptive #4.

ODI Factor



	1OL-2LL_1	1OL-2LL_2	1OL-2LL_3	1OL-2LL_4
Static	3.19	4.47	7.71	9.63
Adaptive #1	4.01	5.39	7.53	9.44
Adaptive #2	3.32	6.12	7.87	10.34
Adaptive #3	3.48	3.84	7.28	10.84
Adaptive #4	2.50	3.84	6.86	10.78

Figure 5-11: Comparison of the ODI factors achieved using different redirection policies for the variable lightly loaded Call Servers scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

We observe the same ranking of redirection strategies in the improvement factor of the 95th POD as seen for the overall 95th POD of the cluster system. The ODI factors at the different load configurations for all redirection policies are shown in Figure 5-11. For the second (1OL-2LL_2) and the third configuration (1OL-2LL_3), Adaptive #2 performs better than the other policies. For the last load configuration (1OL-2LL_4), Adaptive #3 shows slightly better results than Adaptive #4.

5.5.3 Special Scenario

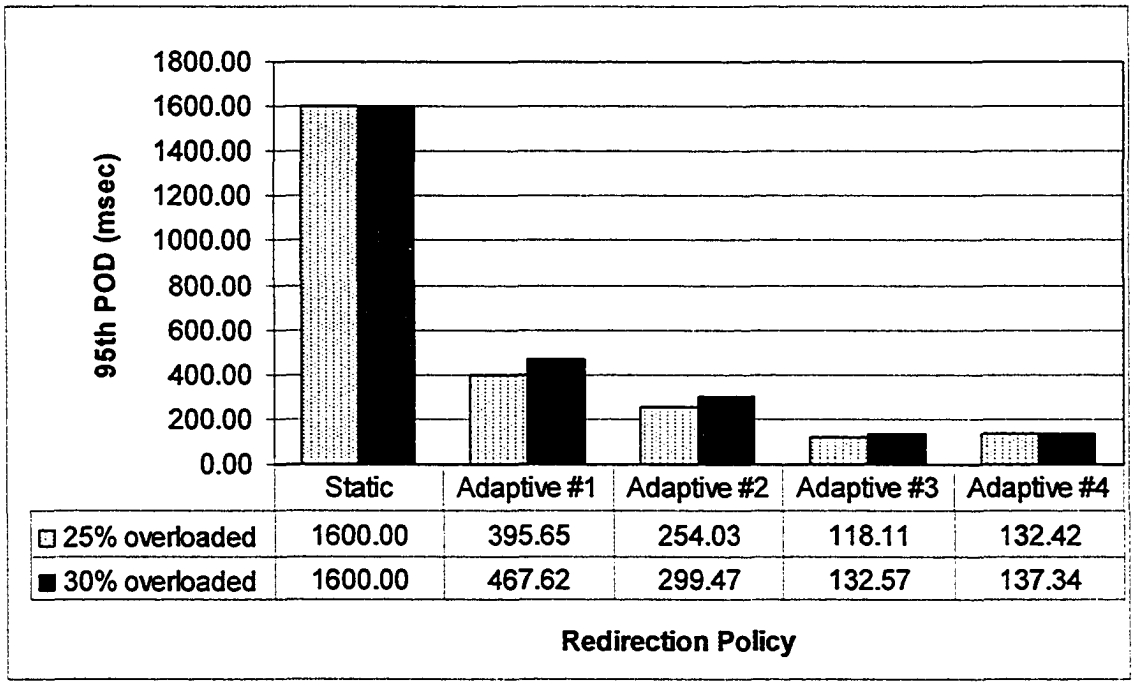
To investigate the effectiveness of the redirection policies for higher levels of overloading, we performed a few additional experiments. In these experiments, we assume one Call Server (CS1) is overloaded up to 25% and 30% above the EngCap and the two lightly loaded Call Servers (CS2 and CS3) are operating 50% below the EngCap. We used two different levels of overloading (25% and 30%) to see which policy demonstrates a more consistent performance. The results of all the redirection policies for these two levels of overloading are presented in Figure 5-12.

The overall 95th PODs for all redirection policies are shown in Figure 5-12. The static policy demonstrates a poor performance for both levels of overloading. The fixed value of the redirection factor (δ = 15%) is not enough to absorb the extra load of the overloaded CS. The performances of Adaptive #1 and Adaptive #2 are better than the static policy, but their 95th PODs are higher than the required value of 140 msec (T_{ec}). However, Adaptive #2 performs better than Adaptive #1.

Adaptive #3 and Adaptive #4 demonstrate good performance compared to the other redirection policies. The overall 95th PODs produced by both policies are under T_{ec}. The results obtained for these two policies are consistent at both levels (25% and 30%) of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

overloading. If we compare the results of these two policies, Adaptive #3 shows a consistently better result at both levels of overloading.



	Static	Adaptive #1	Adaptive #2	Adaptive #3	Adaptive #4
25% overloaded	1600.00	395.65	254.03	118.11	132.42
30% overloaded	1600.00	467.62	299.47	132.57	137.34

Figure 5-12: Comparison of the 95th PODs of the cluster using different redirection policies at 25% and 30% overloading levels

5.6 Effect of Cluster Configuration

In this set of experiments, we investigate performance of the cluster system for different sizes and compositions of the cluster. For this analysis, we carried out experiments with three different configurations. These configurations are described next.

2CS-1OL-1LL: This simplest configuration has only two Call Servers in the cluster. One Call Server (CS1) is operating at a load intensity of 5% above the EngCap and the second Call Server (CS2) is operating at a load intensity of 5% below the EngCap.

3CS-1OL-2LL: This configuration is composed of three Call Servers. One Call Server (CS1) is overloaded 5% above the EngCap and the other two Call Servers (CS2 and CS3) are operating 5% below the EngCap.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3CS-2OL-1LL: This configuration has three Call Servers. Two Call Servers (CS1 and CS2) are overloaded 5% above the EngCap and one Call Server (CS3) is operating 5% below the EngCap.

The default values of the other input parameters used for these experiments are listed in Table 5-1. The results for these three cluster configurations are shown in Figure 5-13 (without load sharing) and Figure 5-14 (with load sharing).

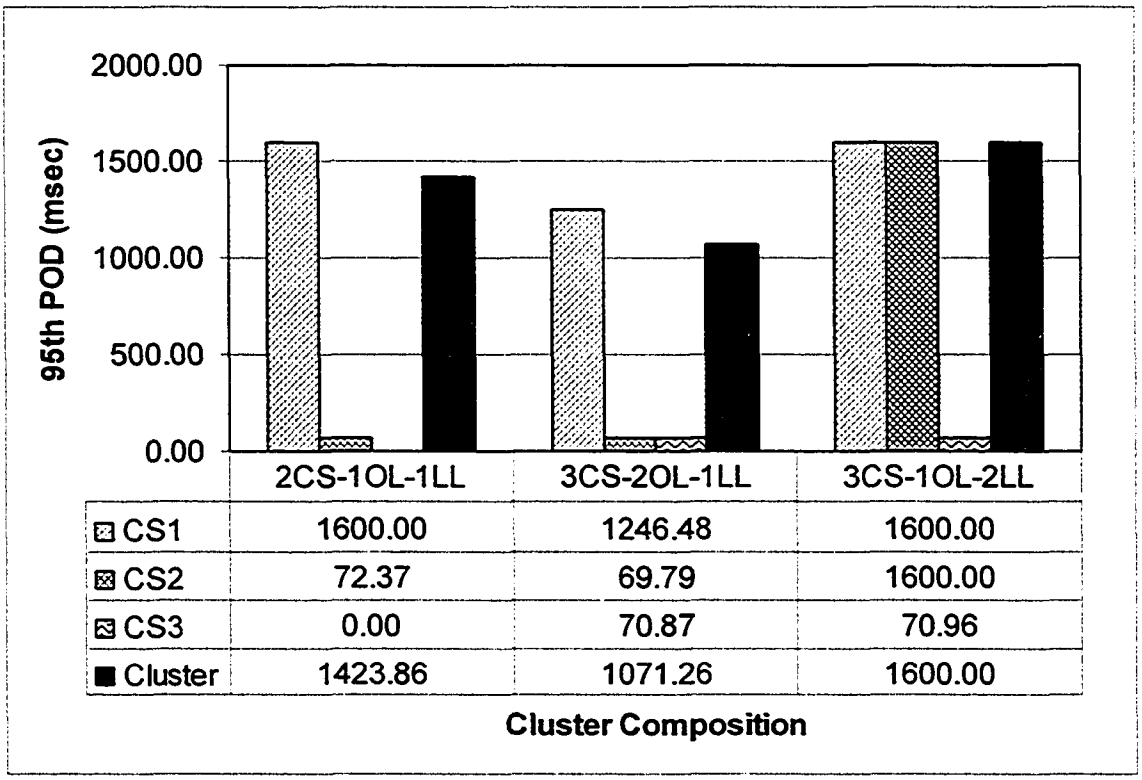
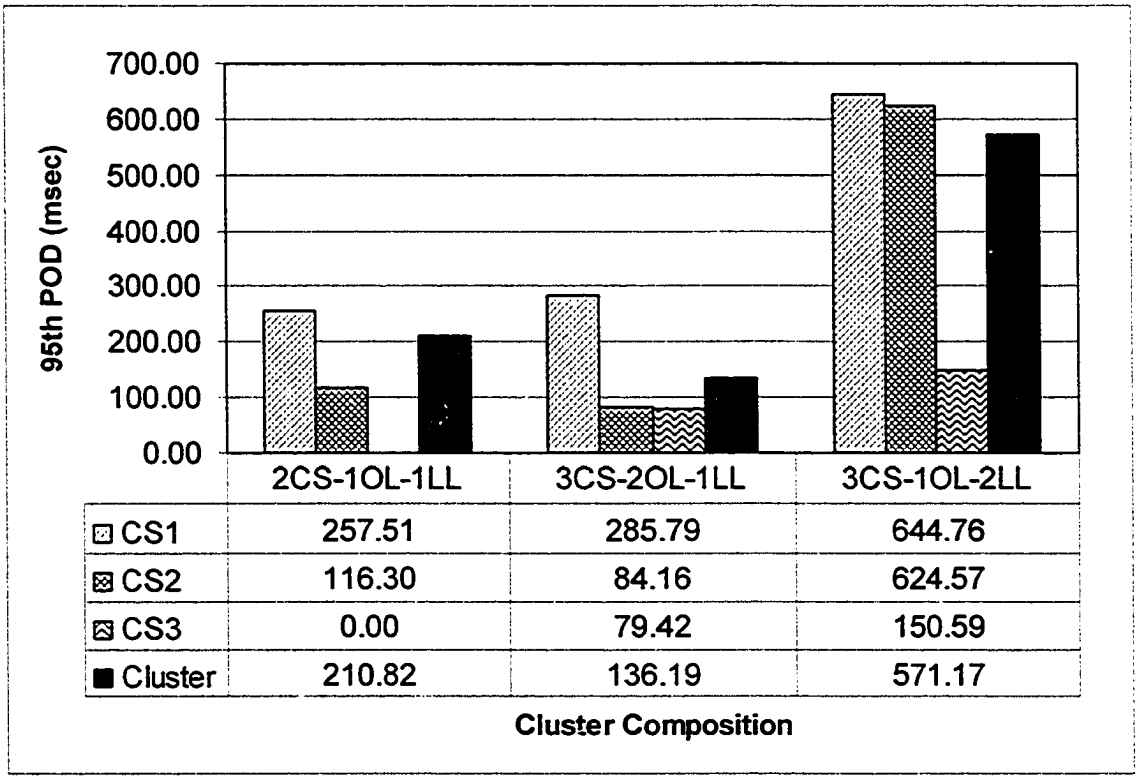


Figure 5-13 : Effect of cluster composition/size on the 95th PODs of the Call Server cluster when load sharing is not used

For load configurations 2CS-1OL-1LL and 3CS-2OL-1LL, the 95th PODs of the overloaded CS (CS1) and of the cluster system are very high as compared to the results achieved for the 3CS-1OL-2LL configuration. This is because of the lesser number of lightly loaded CS(s) for those cluster configurations. In addition to its own load intensity,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the 95th POD of a CS is also affected by the number of remote Call Servers and the load intensity on those remote Call Servers. Higher load intensity and more number of remote CSs cause higher number of terminator data sharing requests to a local Call Server. This is the reason for the degradation of results for 2CS-1OL-1LL and 3CS-2OL-1LL configurations.



	2CS-1OL-1LL	3CS-2OL-1LL	3CS-1OL-2LL
CS1	257.51	285.79	644.76
CS2	116.30	84.16	624.57
CS3	0.00	79.42	150.59
Cluster	210.82	136.19	571.17

Figure 5-14: Effect of cluster composition/size on the 95th PODs of the Call Server cluster when load sharing is used

Figure 5-14 shows that load sharing has improved system performance for all cluster configurations. But the improvement in the performance is more for those configurations that have a higher ratio of the number of lightly loaded Call Servers to overloaded Call Servers. The cluster configuration 3CS-1OL-2LL has two lightly loaded CSs, so we get the best overall 95th POD for this configuration. The configuration, 2CS-1OL-1LL, with

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

one lightly loaded CS and one overloaded CS has also shown better performance in comparison to the cluster configuration 3CS-2OL-1LL in which number of overloaded CSs is higher than the number of lightly loaded CSs. The results achieved for the 3CS-2OL-1LL configuration have shown that our load sharing technique does not try to overload the lightly loaded CS (CS3). This is observed from the value of 95th POD of CS3 in Figure 5-14. The 95th POD of CS3 is increased only up to 150.59 msec, although the overloaded CSs (CS1 and CS2) are still operating in the overloaded state.

We could not perform experimentation for cluster sizes of four or more Call Servers because of the following reason. It takes a lot of experimental time (greater than 20 hours for a 3 CS cluster) on an Intel Pentium 4 based PC equipped with 1 GB of memory to complete a simulation of eight minutes of call processing. This time increases with the size of the Call Server cluster.

5.7 Effect of Data Sharing Policies

In the first phase of this research [Maj04], Majumdar et al. proposed and investigated three data sharing policies to access the terminator state data within the Call Server cluster. These data sharing policies are remote access, replication and migration (see Section 2.2.4 for details). Experiments are carried out to study the effectiveness of the load sharing technique for all the three data sharing policies. As described in Section 3.1.5, the same data sharing policy is used for accessing both the originator and the terminator state data. One of the Call Servers (CS1) is operating at a load intensity that is 5% above the EngCap and the other two Call Servers (CS2 and CS3) are operating at load intensities 5% below the EngCap. For all other parameters, we use the same default

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

values listed in Table 5-1. The results for these three policies are summarized in Figure 5-15 (without load sharing) and Figure 5-16 (with load sharing).

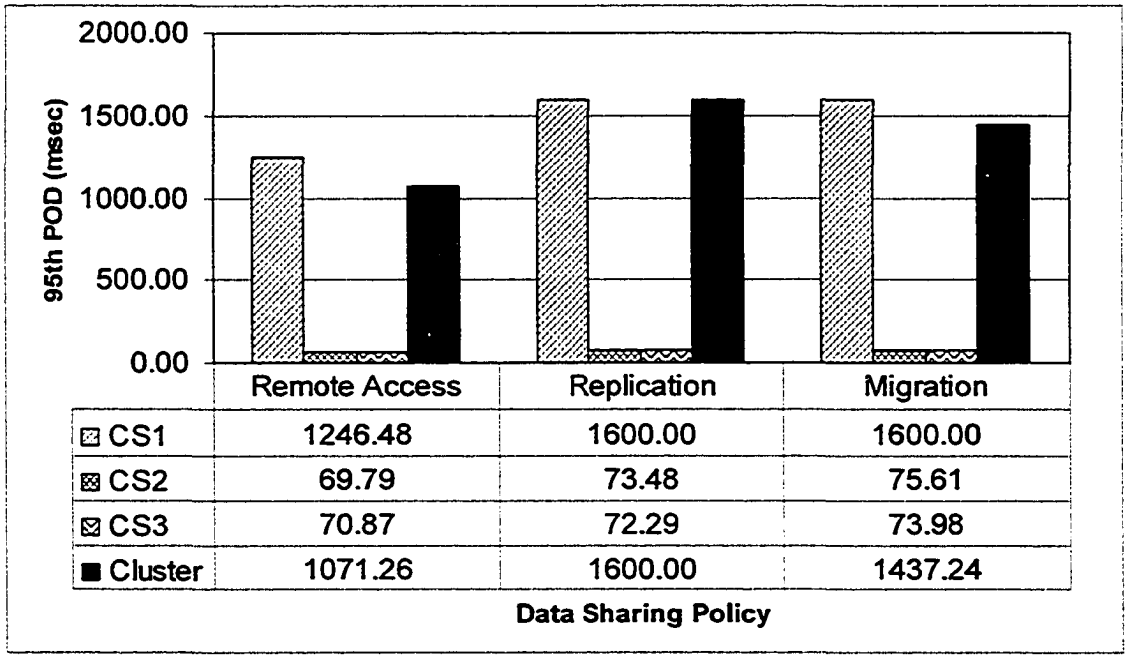


Figure 5-15: Effect of data sharing policies on the 95th PODs of the Call Server cluster when load sharing is not used

It was concluded in [Maj04] that, for a single read and a single write operation on the terminator state data, the engineered capacity achieved with the remote access policy is marginally better than the other two policies. The migration policy shows a slightly better performance as compared to the replication policy. The results shown in Figure 5-15 show the same ranking of the data sharing policies as observed for the engineered capacity in [Maj04]. The overall 95th POD of the cluster is the best for the remote access policy.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

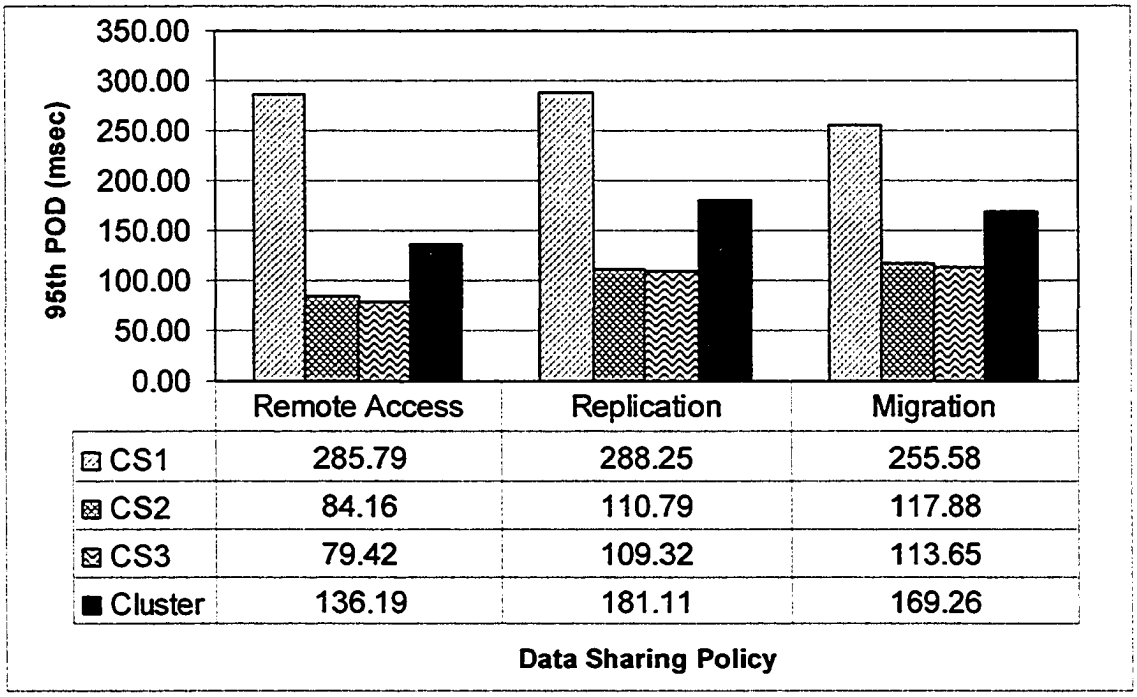


Figure 5-16: Effect of data sharing policies on the 95th PODs of the Call Server cluster when load sharing is used

There are two important conclusions which we can make from the results achieved with load sharing and shown in Figure 5-16. First, load sharing has improved the overall performance of the cluster system no matter which data sharing policy is applied. Secondly, the relative performance of the CS Selection strategies is the same as found in [Maj04] for a single read and a single write operation on the terminator state data. The remote access policy is better than the other two policies, but the migration policy is marginally better than the replication policy. The 95th PODs (see Figure 5-16) of lightly loaded Call Servers (CS2 and CS3) are slightly higher for the migration policy. This is because of the extra overhead required to migrate the data from a remote Call Server and the additional service time needed when the data is migrated back to the original CS.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.8 Analysis of the Duration of Call Events

There are four call events that depend on the response of subscribers. These four events are dialing a number, answering a call, talking on the phone and hanging up the phone. The durations of these call events vary from subscriber to subscriber and call to call. We simulated these events by using truncated exponential distributions. The time duration of these events vary from seconds to minutes depending on the subscribers. Ideally, the duration of these events should be simulated based on the real life data, but it has been observed that using such real life values of duration can cause the duration of the simulation experiments to become very large. For achieving a reasonable simulation time, a truncated exponential distribution with mean of 5 sec is used for all the call events. This is referred to as Scheme 1 in Figure 5-17. In Scheme 2, different mean values are used for the durations of the different call events. This scheme uses a mean of 3 sec for dialing a number, a mean of 5 sec for answering a call, a mean of 10 sec for talking and a mean of 2 sec for hanging up. Different mean values in Scheme 2 reflect the fact such as dialing a number takes lesser time as compared to talking on the phone. Similarly, hanging up the phone is the least time taking activity. Note that the overall mean duration for all the four events in Scheme 2 is 5 sec that is equal to the overall mean duration for the four events in Scheme 1. One of the Call Servers (CS1) is operating at a load intensity that is 5% above the EngCap and the other two Call Servers (CS2 and CS3) are operating at load intensities 5% below the EngCap. For all other parameters, we use the same default values listed in Table 5-1. The results for the two schemes with and without using load sharing (No LS) are shown in Figure 5-17.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

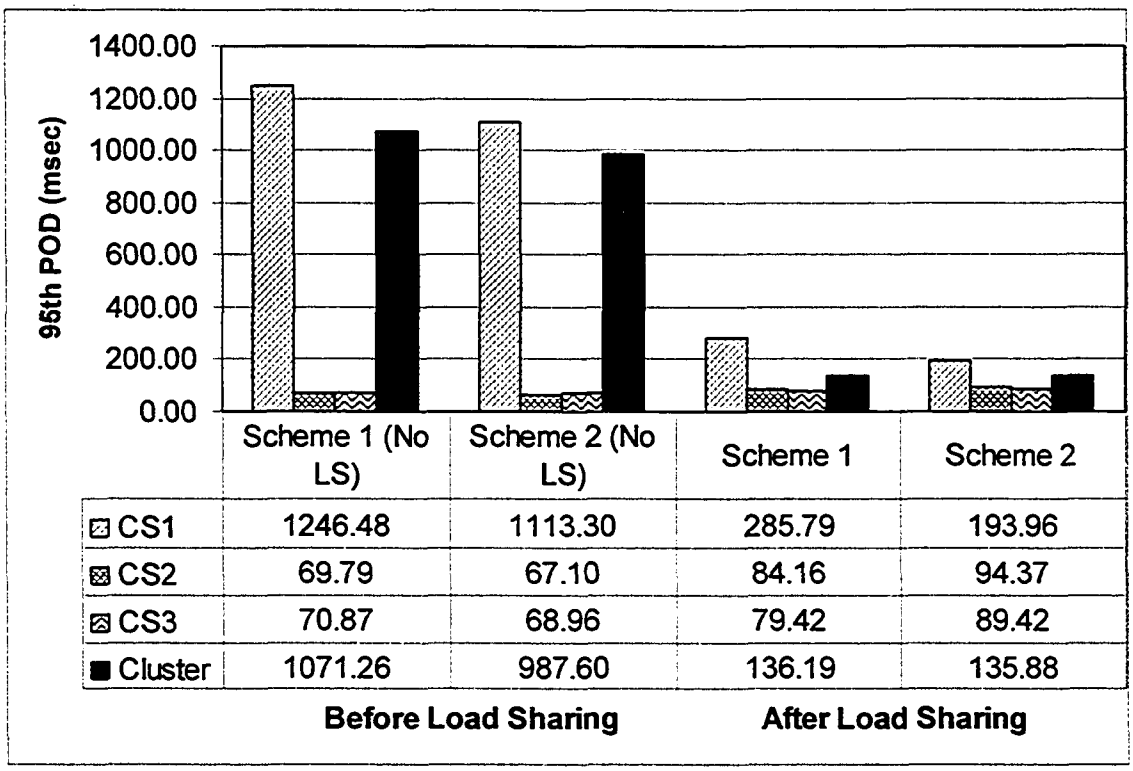


Figure 5-17: Effect of different call event durations on the 95th PODs of the Call Server cluster with and without load sharing

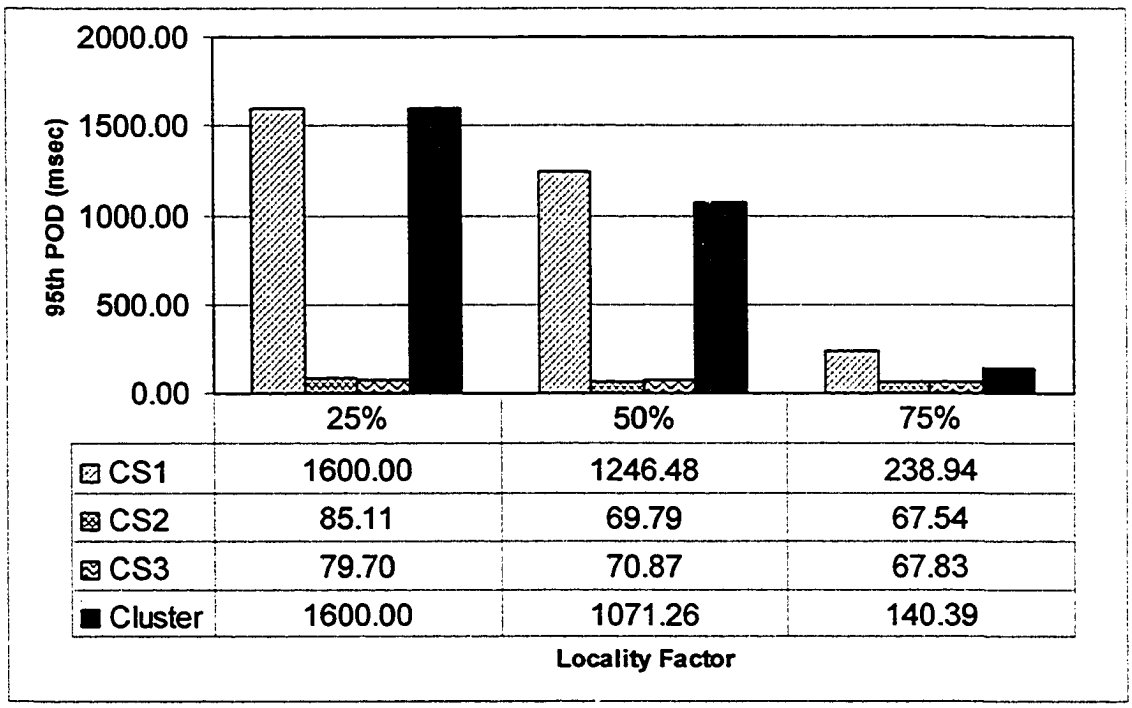
Without load sharing, Scheme 2 produces slightly better results than Scheme 1. The results shown in Figure 5-17 show that there is not a large difference (±7.84% without load sharing and ±0.2% with load sharing) in the overall 95th POD of the cluster achieved with Scheme 1 and Scheme 2, especially when load sharing is applied. The results indicate that system performance is not very sensitive to whether the same or different means are used for modeling the call events.

5.9 Effect of Locality Factor

Locality factor that depends on the location of the terminator state data has a direct effect on the performance of the cluster. If the terminator state data is local to the Call Server, that is handling the call, performance and capacity of the cluster system are improved

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[Maj04]. In this set of experiments, we investigate the effect of locality factor of a Call Server cluster with and without using load sharing technique. One of the Call Servers (CS1) is operating at a load intensity that is 5% above the EngCap and the other two Call Servers (CS2 and CS3) are operating at load intensities 5% below the EngCap. For all other parameters, we use the same default values listed in Table 5-1. The results are shown in Figure 5-18 (without load sharing) and Figure 5-19 (with load sharing).



	25%	50%	75%
CS1	1600.00	1246.48	238.94
CS2	85.11	69.79	67.54
CS3	79.70	70.87	67.83
Cluster	1600.00	1071.26	140.39

Figure 5-18: Effect of locality factor on the 95th PODs of the Call Server cluster when load sharing is not used

The three data sets shown in Figure 5-18 and Figure 5-19 are for three values of locality factor (25%, 50% and 75%). From Figure 5-18, we can see that the 95th PODs of individual CSs and of the cluster are high for lower values of the locality factor. By increasing the locality factor, the performance of the system is improved significantly.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For a locality factor of 75%, the overall 95th POD is reduced to 140.39 msec from a value of 1071 msec that was achieved with a locality factor of 50%.

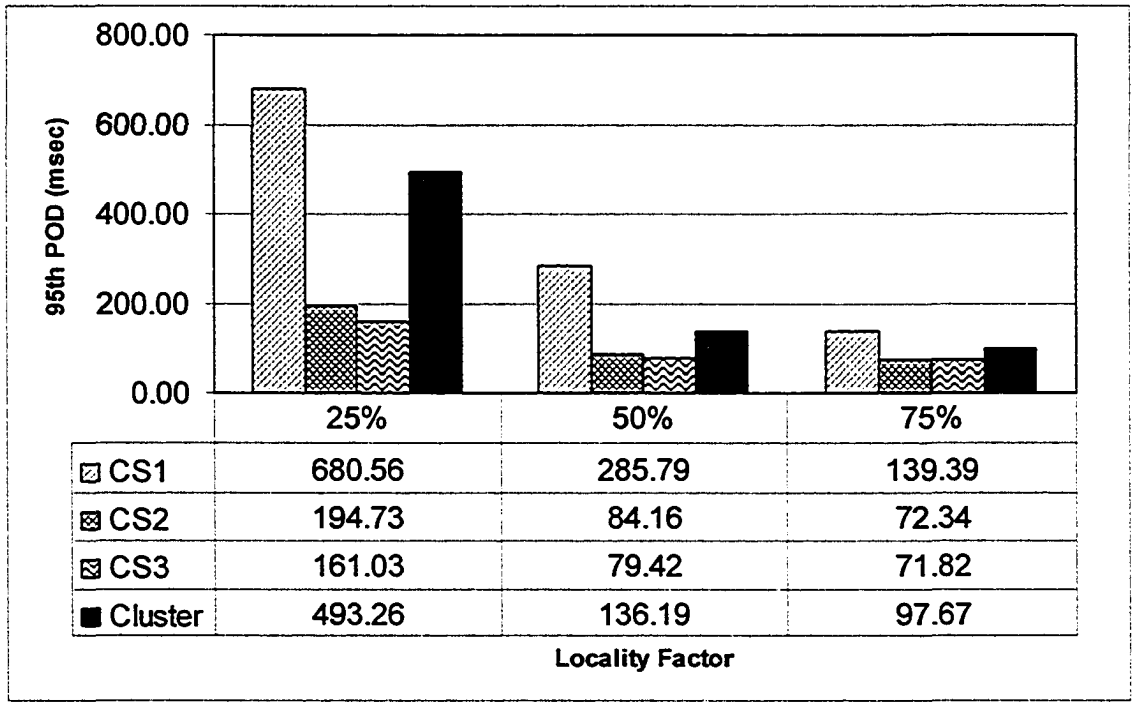


Figure 5-19: Effect of locality factor on the 95th PODs of the Call Server cluster when load sharing is used

It is difficult to accurately capture the impact of locality factor with load sharing techniques. If we transfer a call to a remote Call Server, then the terminator data that was originally remote can possibly become local and vice versa. From Figure 5-19, we observe that load sharing has improved results for all the three value of locality factor of a local Call Server. We also observe that higher the locality factor lower is the overall 95th POD for the cluster. Without load sharing, the higher probability of finding a terminator data on the same local CS improves the performance of the system. But because of load sharing, the probability of the terminator data on the remote CS for redirected calls can become high. So the overhead of handling these calls will be high

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

because both the originator and the terminator data now exist on the remote CS. For a locality factor of 25%, the overall 95th POD is improved from 1600 msec to 493.26 msec when load distribution is used. These results have shown that the load sharing always improves the performance of the system but locality factor is another important factor which affects system performance, with and without load sharing.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6 Conclusions

The experimental results using different work load parameters and different scenarios of the Call Server cluster are presented in Chapter 5. The investigation of load sharing techniques, redirection policies in particular, is very important in the design of Call Server clusters. Also, the thesis is useful for Nortel Networks in understanding the impact of different system and workload parameters on the performance of load sharing on a Call Server cluster. Based on the results of the simulations conducted, we first present our conclusions on the impact of different workload and system parameters. A comparison of the different redirection policies is presented next. Finally, we describe directions for future research.

6.1 Workload and System Parameters

The impact of different workload and system parameters is investigated with simulation experiments. A summary of the observations made from the simulation results are presented in the following sub-sections.

6.1.1 Load Index

Selecting an appropriate load index is important in load sharing. Our first choice of the load index was the size of the queue of new calls (origination queue). Since the call processing subsystem serves new calls in the form of batches and the progress queue has

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

more priority than the origination queue, it is difficult to find a linear relationship between the load on the Call Server and the size of the origination queue.

The origination delay is an alternative option of load index. Since the origination delay is a response of the system, so it is a performance index that reflects the system behavior. For this reason, we used 95th percentile value of the origination delay. Use of regular 95th percentile value was not good enough to achieve instantaneous values, so we use a moving 95th percentile origination delay which is calculated from the data stored for a fixed time period (Window Size) and from History Data. A short Window tends to make the load sharing system react prematurely to transient load spikes whereas a very large Window Size tends to make the system sluggish in reacting to load change. Thus, an intermediate Window Size is to be chosen. For the experiments described in this thesis, the effective value of the Window Size is observed to be 1.0 sec. It has also been seen that use of History Data helps to improve the performance of the system.

6.1.2 Thresholds

The thresholds used for the transfer policy and the location policy are in the range of the 95th POD achieved at the engineered capacity of a Call Server. The 95th POD should be T_{cc} (140 msec) at the engineered capacity of a Call Server. So we experimented with different values of thresholds slightly above and below T_{cc}.

For the experiments described in this thesis, the transfer policy threshold value which is almost 25% above T_{cc} is found to be more appropriate for designating a Call Server as a sender. By choosing a value above T_{cc} with a reasonable margin has the additional advantage that a CS does not become a sender because of minor transients.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For the location policy, the threshold value (105 msec) which is almost 25% lower than T_{ec} is observed to be effective for selecting a potential receiver CS. This is because the 95th POD rises sharply after the value of 105 msec due to its non-linear behavior. If we select a receiver CS operating very near the EngCap (T_{ec}), then such a Call Server can become overloaded after accepting a small number of calls from a remote CS. That is why using a threshold value that is smaller than T_{ec} by a reasonable margin is appropriate.

6.1.3 Data Sharing Policies

There are three data sharing policies for accessing the terminator state data. These are investigated in detail in the first phase of this research [Maj04]. As already mentioned in Section 2.2.4, we can apply the same three data sharing policies to access the originator state data as well. We compared the three policies using the same policy for both the originator and the terminator data in one experiment and found that the remote access policy is better than the other two policies, but migration policy is marginally better than the replication policy. The experiment was based on a system that performs a single read and a single write operation on the state data for both the originator and the terminator. In case of multiple read and write operations on the terminator state data, the migration policy is observed to be better than the other two policies on a system that does not use load sharing [Maj04]. A similar analysis on a system that uses load sharing is worthy of investigation.

6.1.4 Locality Factor

In the first phase of this project, Majumdar et al. [Maj04] have also shown that the location of the terminator state data has a significant impact on the capacity of the Call

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Server cluster. By increasing the locality factor, the capacity of the cluster system can be increased. We have seen that load sharing improves the performance of the system for all values of the locality factor. With load sharing, we also observed the same trend of improvement in performance by increasing the value of locality factor as was concluded in [Maj04].

6.2 Comparison of Redirection Policies

A number of redirection policies, static as well as adaptive, are investigated in this thesis. The important observations made for each policy are summarized in the following sub-sections. An effort is also made to find a redirection policy which can be reasonably effective for a wide range of the overloaded situations.

6.2.1 Static Policy

The static policy redirects a fixed proportion (δ) of new calls on every invocation of the broker, without considering the level of overloading of the sender CS or ascertaining how much below the threshold a receiver Call Server is operating at. The value of δ, however, is found to be dependent on the level of overloading of the overloaded node. Thus, an effective value of δ found for one particular level of overloading does not seem to be effective for the other levels of overloading.

6.2.2 Adaptive #1

Adaptive policies use one or more attributes of the load status of Call Servers to calculate the redirection factor, δ. Adaptive #1 is based on the limited knowledge of system load and uses the load status of the sender Call Server only. For the experiments described in this thesis, this policy shows satisfactory results only for the lower levels of

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

overloading (0-7.5% above the EngCap). It is found less effective for higher levels of overloading (see Section 5.5.3).

6.2.3 Adaptive #2

Adaptive #2 uses the load information of the sender CS but the maximum redirected constant, P, is selected from a set of values. Each value in the set of P corresponds to one availability level of the receiver CS. This policy shows a better performance in comparison to all other policies for lower levels of overloading (0-7.5% above the EngCap). This policy has also shown a significant performance improvement if the spare capacity on the receiver CS is high. For higher overloading levels, this policy demonstrates a satisfactory performance as well (see Section 5.5.3). The effectiveness of this policy stems from the use of the load status of both the sender and the receiver CSs.

6.2.4 Adaptive #3

Adaptive #3 is based on the load status of the receiver CS only. It shows satisfactory results for lower levels of overloading (0-7.5% above the EngCap) of an overloaded Call Server. But this policy has also shown a very effective and consistent performance for higher levels of overloading. We have seen from the results shown in Section 5.5.3 that this policy shows a consistent performance when overloading levels are changed in Call Server cluster. As already discussed, the receiver CS operating at a load intensity slightly below the EngCap is very sensitive to the number of calls transferred. So this policy uses lower values of δ (see Figure 3-6) to get improved performance in such situations.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2.5 Adaptive #4

Adaptive #4 is a uniform sender and receiver load based policy. It calculates the redirection factor, δ, based on the load status of the sender CS and the receiver CS independently. This policy has shown good performance only for higher levels of overloading on the overloaded Call Server. This policy produces poor results if receiver CS is operating slightly below the EngCap.

6.2.6 Discussion

It is observed that the adaptive policies are far better than the static policy. The adaptive policies mainly differ in the algorithm used for calculating the redirection factor, δ. The calculation of δ based only on the load status of the sender CS (Adaptive #1) does not produce satisfactory performance unless load status of the receiver CS is involved as well (Adaptive #2). Adaptive #4 does not perform well at lower levels of overloading because a large number of calls may get transferred to the receiver Call Server. Both Adaptive #2 and Adaptive #3 demonstrate good performance for a broad range of overloading. Adaptive #3, however, uses lesser state information than adaptive #2.

Most experiments described in this thesis correspond to a three-CS cluster. The performances of the redirection strategies depend on the type of information used in the computation of δ that is based on the load status of the two Call Servers: the sender and the receiver. We expect the conclusions regarding the relative performances of the redirection strategies to hold for a cluster containing more than three Call Servers. However, performing simulation experiments for large clusters and investigating the degree of performance improvement due to the deployment of load sharing is important.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.3 Overheads

There are two main sources of overheads associated with load sharing in Call Server clusters. The first accrues from the computation of load indices and the distribution of load information. The experimental results demonstrate the importance of using both current and recent past information in computation of the load index: whether simpler methods exist for the computation of the load index warrants investigation. The second overhead concerns the computation of δ performed by a redirection strategy. The static policy is expected to give rise to the smallest overhead. Adaptive #2 and Adaptive #4 that use information regarding both sender and receiver Call Servers are expected to give rise to higher overheads. The impact of these overheads on the relative performances of the redirection policies requires further investigation.

6.4 Future Research

A Call server cluster involves a number of issues. We have only covered the area of selection of Call Servers based on system load. Future work regarding this particular area of the Call Server clustering is summarized.

1. We used a fixed threshold based transfer policy in this research. To achieve further adaptability, a dynamic threshold based policy in which the threshold depends on the load information of the Call Servers in the cluster can be used. Such a dynamic threshold based policy warrants investigation.
2. We used a weighted random selection policy for the selection of a receiver. In this policy, we can only transfer to one receiver at a time. An enhanced selection policy that can select more than one receiver to transfer the load may lead to

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

higher system performance. Investigation of such a location policy is worthy of future research.

3. In the current versions of telephone switches, overloading is controlled by *throttling* the input traffic of new calls if load intensity increases beyond the level of the engineered capacity of a CS. During throttling, a Call Server directs its peripheral module to either slow down or stop the new calls. Use of throttling in Call Server clusters is an important direction for future research.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

References

[Bal04] J. Balasubramanian, D. C. Schmidt, L. Dowdy and O. Othman, "Evaluating the Performance of Middleware Load Balancing Strategies", in *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC'04)*, Monterey, CA, U.S.A., September 2004, pp. 135-146.

[Ben98] B. Beninger, "XA-Core: Guide to Multiprocessing", Nortel Networks, Ottawa, ON, Canada, 1998.

[Bry81] R. M. Bryant and R. A. Finkel, "A stable distributed scheduling algorithm", in *Proceedings of the 2nd International Conference on Distributed Computing Systems (ICDCS'81)*, Paris, France, April 1981, pp. 314-323.

[Buy99] R. Buyya, "High Performance Cluster Computing: Architectures and Systems (Volume 1)", Prentice Hall PTR, Paramus, NJ, U.S.A., 1999.

[Cha01] C. Chang, D. Lee and Y. Jou, "Load balanced Birkhoff-von Neumann switches", in *Proceedings of the 2001 IEEE Workshop on High Performance Switching and Routing (HPSR'01)*, Dallas, TX, U.S.A., May 2001, pp. 276-280.

[Cis01] Cisco Systems Inc., "Cisco CallManager System Guide (Release 3.1)", Corporate Headquarters, Cisco Systems Inc., CA, U.S.A., 2001.

[Dah99] M. Dahlin, "Interpreting Stale Load Information", in *Proceedings of the 10th IEEE International Conference on Distributed Computing Systems (ICDCS'99)*, Washington, DC, U.S.A., June 1999, pp. 285-296.

[Das97] S. K. Das, D. J. Harvey and R. Biswas, "Adaptive Load Balancing Algorithms Using Symmetric Broadcast Networks", *in Proceedings of the 26th International Conference on Parallel Processing (ICPP'97)*, Bloomingdale, IL, U.S.A., August 1997, pp. 360-367.

[Eag85] D. L. Eager, E. D. Lazowska and J. Zahorjan , "A comparison of receiver-initiated and sender-initiated adaptive load sharing", in *Proceedings of the 1985 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS'85)*, Austin, TX, U.S.A., August 1985, pp. 1-3.

[Eag86] D. L. Eager, E. D. Lazowska and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems", *IEEE Transactions on Software Engineering*, Vol.12, No. 5, May 1986, pp. 662-675.

[Fer86] D. Ferrari and S. Zhou, "A load index for dynamic load balancing", in *Proceedings of the 1986 ACM Fall Joint Computer Conference (COMPCON'86)*, Dallas, TX, U.S.A., November 1986, pp. 684-690.

[Gen00] Z. Genova and K. J. Christensen, "Challenges in URL Switching for Implementing Globally Distributed Web Sites", in *Proceedings of the 29th*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

International Conference on Parallel Processing (ICPP'00), Toronto, Canada, August 2000, pp. 89-94.

[Jai91] R. Jain, "The Art of Computer Systems Performance Analysis", John Wiley & Sons Inc., New York, NY, U.S.A., 1991.

[Jin01] L. Jin, F. Casati, M. Sayal and M. Shan, "Load balancing in distributed workflow management system", in *Proceedings of the 2001 ACM Symposium on Applied Computing (SAC'01)*, Las Vegas, NV, U.S.A., March 2001, pp. 522-530.

[Ju95] J. Ju, G. Xu and K. Yang, "An intelligent dynamic load balancer for workstation clusters", *ACM SIGOPS Operating Systems Review*, Vol. 29, No. 1, January 1995, pp. 7-16.

[Kei95] B. Keiser and E. Strange, "Digital Telephony and Network Integration", Van Nostrand Reinhold, New York, NY, U.S.A., 1995.

[Kop04.1] G. Kopec, *Personal Communication*, Nortel Networks, Ottawa, ON, Canada, November 2004.

[Kop04.2] G. Kopec, *Personal Communication*, Nortel Networks, Ottawa, ON, Canada, November 2004.

[Kop05] G. Kopec, *Personal Communication*, Nortel Networks, Ottawa, ON, Canada, February 2005.

[Kru88.1] P. E. Krueger, "Distributed scheduling for a changing environment", *Ph.D. Thesis (Technical Report 780)*, Department of Computer Science, University of Wisconsin, Madison, WI, U.S.A., June 1988.

[Kru88.2] P. Krueger and M. Livny, "A comparison of preemptive and non-preemptive load distributing", in *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS'88)*, San Jose, CA, U.S.A., June 1988, pp. 123-130.

[Kru91] P. Krueger and R. Chawla, "The Stealth distributed scheduler", in *Proceedings of the 11th International Conference on Distributed Computing Systems (ICDCS'91)*, Arlington, TX, U.S.A., May 1991, pp. 336-343.

[Lee96] G. Lee, H. Lee and J. Cho, "A prediction-based adaptive location policy for distributed load balancing", *Journal of Systems Architecture*, Vol. 42, No. 1, August 1996, pp. 1-18.

[Liv82] M. Livny and M. Melman, "Load balancing in homogeneous broadcast distributed systems", in *Proceedings of the Computer Network Performance Symposium*, College Park, MD, U.S.A., April 1982, pp. 47-55.

[Lit92] M. Litzkow and M. Solomon, "Supporting checkpointing and process migration outside the Unix kernel", in *Proceedings of the Usenix Conference*, San Francisco, CA, U.S.A., January 1992, pp. 283-290.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[Lit97] M. Litzkow, T. Tanenbaum, J. Basney and M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System", *Technical Report #1346*, University of Wisconsin-Madison Computer Sciences, Madison, WI, U.S.A., April 1997.

[Liv97] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for High Throughput Computing", *SPEEDUP Journal*, Vol. 11, No. 1, June 1997, pp. 36-40.

[Lon01] K. Long, Z. Zhang and S. Cheng, "Load balancing algorithms in MPLS traffic engineering", in *Proceedings of the 2001 IEEE Workshop on High Performance Switching and Routing (HPSR'01)*, Dallas, TX, U.S.A., May 2001, pp. 175-179.

[Maj04] S. Majumdar and B. Sun, "Call Server Clusters: Year 1 Report (Project: High Performance Telecommunication Servers SER03-104)", *Technical Report*, Nortel Networks, Ottawa, ON, Canada, September 2004.

[Mar02] S. Marei, A. Amri and R. E. Ahmed, "New job selection and location policies for load-distributing", *International Journal of Network Management*, Vol. 12, No. 3, May/June 2002, pp. 165-178.

[Nor01] Nortel Networks, "Feature Planning Guide: Local Access DMS Systems", Corporate Headquarters, Nortel Networks, Brampton, ON, Canada, 2001.

[Nor02] Nortel Networks, "OP4003: Introduction to Call Processing", *Nortel Networks Training Guide*, Corporate Headquarters, Nortel Networks, Brampton, ON, Canada, 2002.

[Opn02] OPNET Technologies Inc., "OPNET Modeler 8.0: Online Documentation", OPNET Headquarters, Bethesda, MD, U.S.A., 2002.

[Ram84] K. Ramamritham and J. A. Stankovic, "Dynamic task scheduling in distributed hard real-time systems", *IEEE Transactions on Software Engineering*, Vol. 1, No. 3, July 1984, pp. 65-75.

[Rob05] A. Robertson, "High Availability Linux Project", 2005, available from http://linux-ha.org/. (accessed in June, 2005).

[Rot90] H. G. Rotithor and S. S. Pyo, "Decentralized decision making in adaptive task sharing", in *Proceedings of the 2nd IEEE Symposium on Parallel and Distributed Processing (IPDPS'90)*, Dallas, TX, U.S.A., December 1990, pp. 34-41.

[She02] K. Shen, T. Yang and L. Chu, "Cluster load balancing for fine-grain network services", in *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, Ft. Lauderdale, FL, U.S.A., April 2002, pp. 51-58.

[Shi90] N.G. Shivaratri and P. Krueger, "Two adaptive Location Policies for Global Scheduling Algorithms", in *Proceedings of the 10th IEEE International Conference on Distributed Computing Systems (ICDCS'90)*, Paris, France, June 1990, pp. 502-509.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

[Sit04] H. Y. Sit, K. S. Ho, H. V. Leong and W. P. Reobert, "An adaptive clustering approach to dynamic load balancing", in *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)*, Hong Kong, May 2004, pp. 415-420.

[Sun04] B. Sun, "High Performance Telecommunication Server: Call Server Clustering", *M.A.Sc. Thesis (In Porgress)*, Department of Systems and Computer Engineering, Carleton University, Ottawa, ON, Canada, 2004.

[Uml04] Unified Modeling Language Specification, Version 2.0, October 2004, available from http://www.uml.org. (accessed in July, 2005).

[Wan93] C. Wang, P. Krueger and M. Liu, "Intelligent job selection for distributed scheduling", In *Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS'93)*, Pittsburg, PA, U.S.A., May 1993, pp. 517-524.

[Yos03] K. Yoshihara, M. Isomora and H. Horiuchi, "Dynamic load balancing for distributed network management", in *Proceedings of the Eighth IFIP/IEEE International Symposium on Integrated Network Management*, Colorado Springs, CO, U.S.A., March 2003, pp. 277-290.

[Zho92] S. Zhou, J. Wang, X. Zheng and P. Delisle, "Utopia – A load sharing facility for large, heterogeneous distributed computer systems", *Technical Report CSRI-257*, Computer Systems Research Institute, University of Toronto, Toronto, ON, Canada, April 1992.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A: Average Origination Delay for One Overloaded Call Server Scenario

Table A-1: Average origination delay achieved at different load configurations without load sharing for the one overloaded Call Server scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	54.06	36.19	35.89	42.05
1OL-2LL_2	71.38	35.70	36.45	47.84
1OL-2LL_3	598.39	35.63	36.36	223.46
1OL-2LL_4	8103.87	36.81	36.55	2725.74

Table A-2: Average origination delay achieved at different load configurations with the static policy for the one overloaded Call Server scenario

Load Configurations	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	49.45	36.94	36.90	41.10
1OL-2LL_2	52.43	37.36	38.71	42.84
1OL-2LL_3	72.82	44.02	43.08	53.31
1OL-2LL_4	119.24	58.86	61.56	79.89

Table A-3: Average origination delay achieved at different load configurations with Adaptive #1 for the one overloaded Call Server scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	49.67	38.12	37.53	41.77
1OL-2LL_2	56.31	38.38	39.46	44.72
1OL-2LL_3	69.10	45.70	46.69	53.83
1OL-2LL_4	106.30	54.38	54.19	71.62

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table A-4: Average origination delay achieved at different load configurations with Adaptive #2 for the one overloaded Call Server scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	46.61	37.74	37.01	40.45
1OL-2LL_2	53.77	37.43	38.12	43.11
1OL-2LL_3	69.47	44.69	42.99	52.38
1OL-2LL_4	128.55	54.02	52.44	78.34

Table A-5: Average origination delay achieved at different load configurations with Adaptive #3 for the one overloaded Call Server scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	50.96	41.08	40.94	44.33
1OL-2LL_2	52.92	43.86	44.91	47.23
1OL-2LL_3	71.99	48.68	50.15	56.94
1OL-2LL_4	163.16	57.65	54.38	91.73

Table A-6: Average origination delay achieved at different load configurations with Adaptive #4 for the one overloaded Call Server scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	46.92	37.48	37.99	40.80
1OL-2LL_2	55.63	39.60	42.80	46.01
1OL-2LL_3	79.23	51.50	52.12	60.95
1OL-2LL_4	115.83	65.33	68.82	83.33

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix B: Average Origination Delay for Variable Lightly Loaded Call Servers Scenario

Table B-1: Average origination delay achieved at different load configurations without load sharing for the two variable lightly loaded Call Servers scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	1141.05	59.21	56.41	418.89
1OL-2LL_2	775.93	45.42	45.60	288.99
1OL-2LL_3	598.39	35.63	36.36	223.46
1OL-2LL_4	574.22	30.02	29.86	211.37

Table B-2: Average origination delay achieved at different load configurations with the static policy for the two variable lightly loaded Call Servers scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	247.68	77.39	97.19	140.75
1OL-2LL_2	123.88	69.90	70.74	88.17
1OL-2LL_3	72.82	44.02	43.08	53.31
1OL-2LL_4	59.93	34.77	34.81	43.17

Table B-3: Average origination delay achieved at different load configurations with Adaptive #1 for the two variable lightly loaded Call Servers scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	176.56	88.74	93.09	119.46
1OL-2LL_2	99.60	62.98	67.07	76.55
1OL-2LL_3	69.10	45.70	46.69	53.83
1OL-2LL_4	64.46	33.37	33.50	43.78

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table B-4: Average origination delay achieved at different load configurations with Adaptive #2 for the two variable lightly loaded Call Servers scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	245.39	77.43	69.72	130.85
1OL-2LL_2	96.91	59.16	56.35	70.81
1OL-2LL_3	69.47	44.69	42.99	52.38
1OL-2LL_4	63.29	34.20	33.83	43.77

Table B-5: Average origination delay achieved at different load configurations with Adaptive #3 for the two variable lightly loaded Call Servers scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	274.69	65.94	79.79	140.14
1OL-2LL_2	157.95	82.69	70.15	103.60
1OL-2LL_3	71.99	48.68	50.15	56.94
1OL-2LL_4	60.73	37.84	38.16	41.58

Table B-6: Average origination delay achieved at different load configurations with Adaptive #4 for the two variable lightly loaded Call Servers scenario

Load Configuration	Average Origination Delay (msec)			
	CS1	CS2	CS3	Cluster
1OL-2LL_1	193.41	92.69	86.32	124.14
1OL-2LL_2	128.08	76.89	86.94	97.30
1OL-2LL_3	79.23	51.50	52.12	60.95
1OL-2LL_4	56.21	34.31	35.90	42.14

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix C: Results (95th POD and ODI Factor) for Two Variable Overloaded Call Servers and One Lightly Loaded Call Server Scenario

Table C-1: Load configurations for the two variable overloaded Call Servers and one lightly loaded Call Server scenario

Load Configuration	Load Intensity (Kcalls/hr)	
	Two Overloaded CSs (CS1 and CS2)	Lightly loaded CS (CS3)
2OL-1LL_1	3150 (EngCap)	3000
2OL-1LL_2	3220 (2.5% above EngCap)	3000
2OL-1LL_3	3300 (5% above EngCap)	3000
2OL-1LL_4	3400 (7.5% above EngCap)	3000

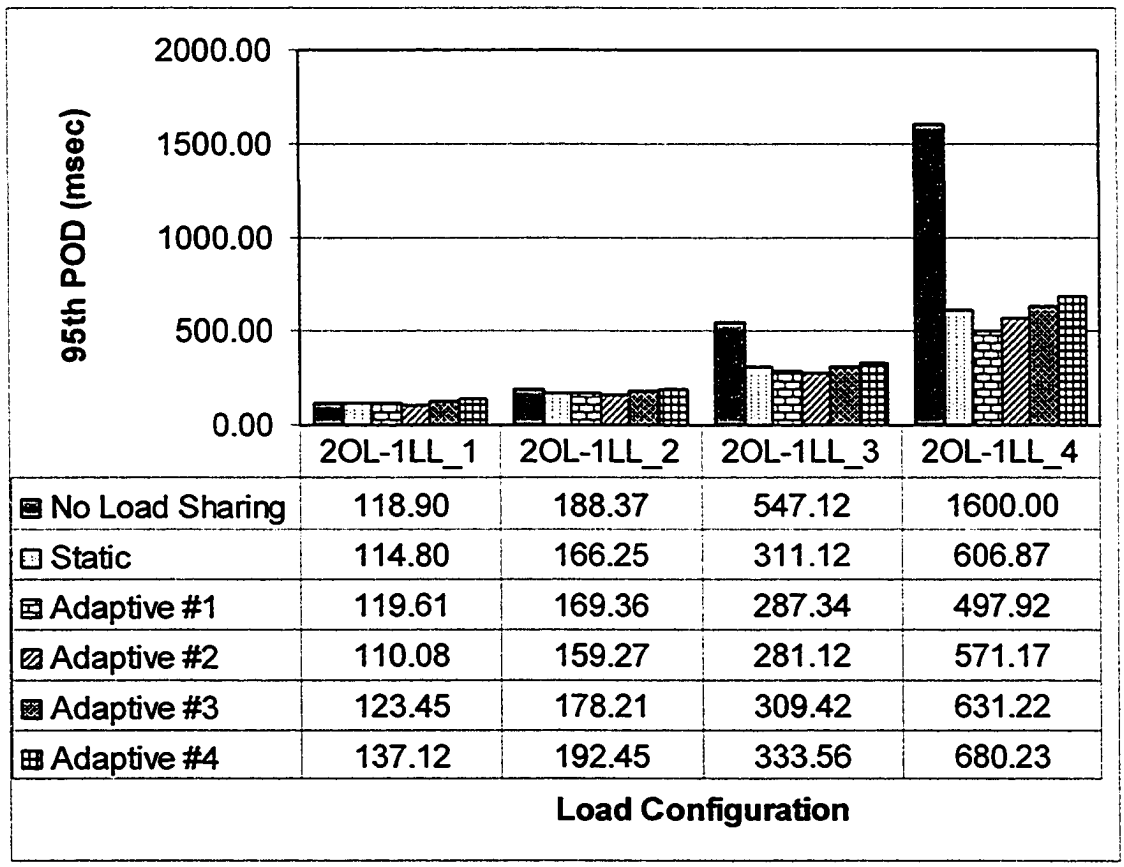


Figure C-1: Comparison of the 95th PODs achieved using different redirection policies for the two variable overloaded Call Servers and one lightly loaded Call Server scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

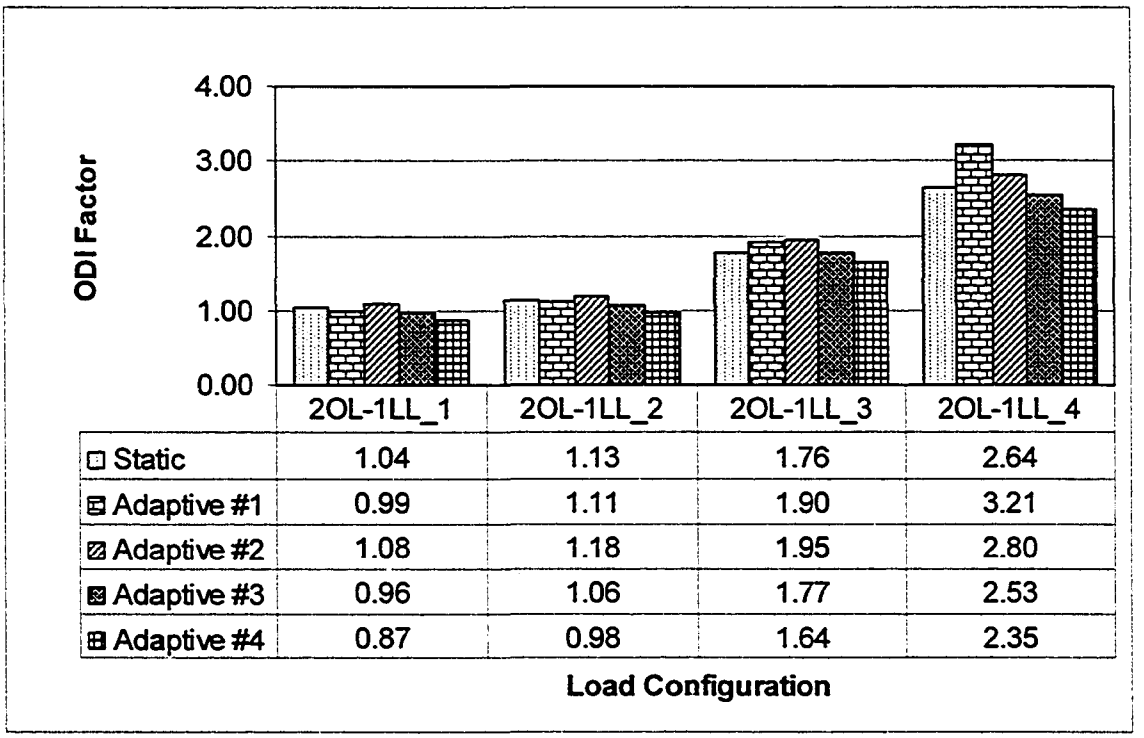


Figure C-2: Comparison of the ODI factors achieved using different redirection policies for the two variable overloaded Call Servers and one lightly loaded Call Server scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix D: Results (95th POD and ODI Factor) for Two Overloaded Call Servers and One Variable Lightly Loaded Call Server Scenario

Table D-1: Load configurations for the two overloaded Call Servers and one variable lightly loaded Call Server scenario

Load Configuration	Load Intensity (Kcalls/hr)	
	Two Overloaded CSs (CS1 and CS2)	Lightly loaded CS (CS3)
2OL-1LL_1	3300	3080 (2.5% below EngCap)
2OL-1LL_2	3300	3000 (5% below EngCap)
2OL-1LL_3	3300	2833 (10% below EngCap)

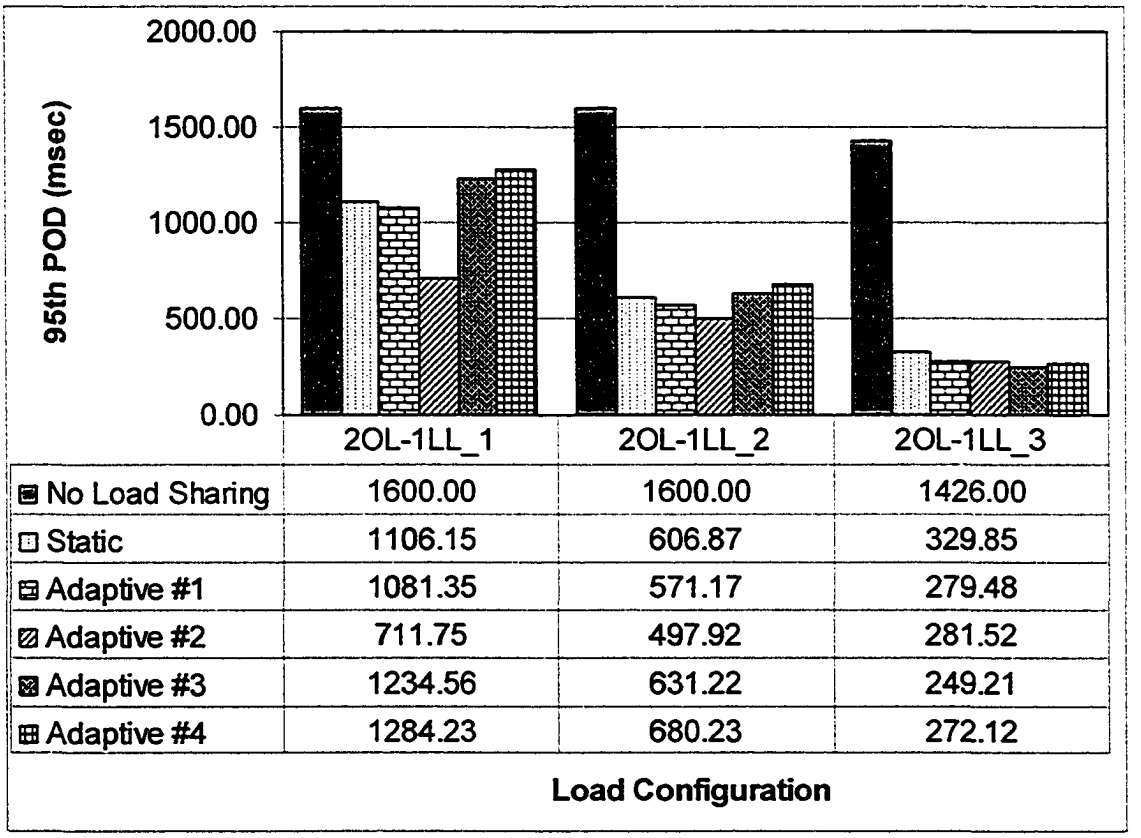


Figure D-1: Comparison of the 95th PODs achieved using different redirection policies for the two overloaded Call Servers and one variable lightly loaded Call Server scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

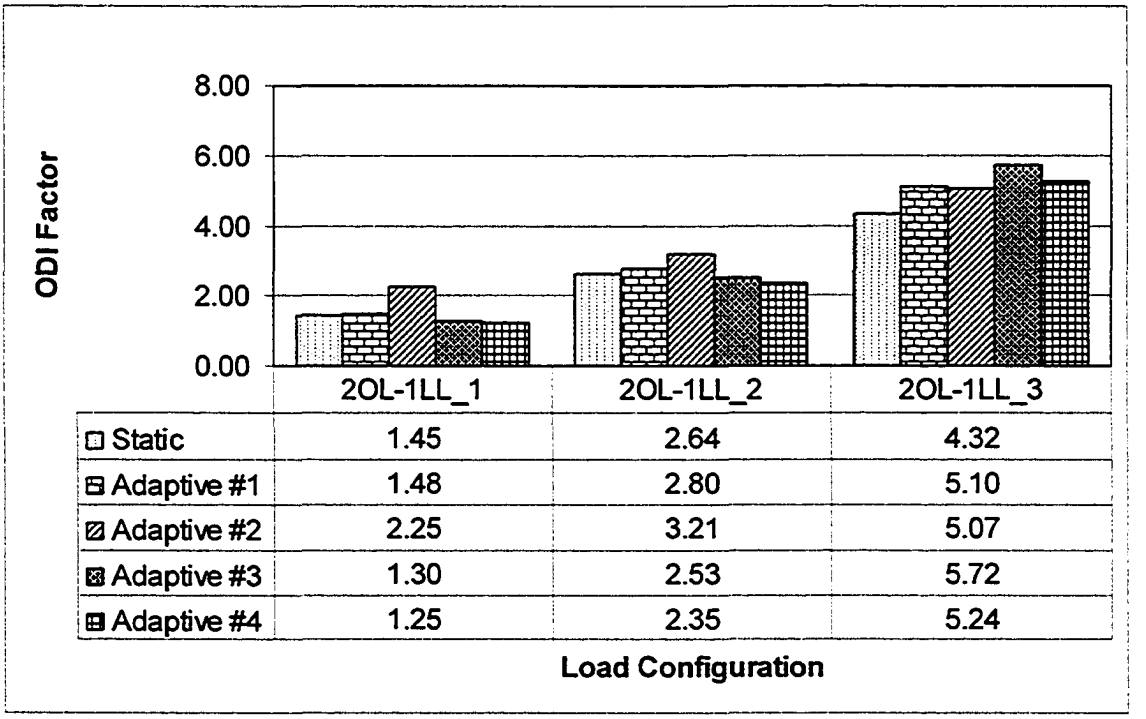


Figure D-2: Comparison of the ODI factors achieved using different redirection policies for the two overloaded Call Servers and one variable lightly loaded Call Server scenario

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.