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

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

Load Balancing Strategies for CORBA-Based Systems

by

Changshuang Lin

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 and Computer Engineering

Faculty of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

August 2003

© Copyright 2003, Changshuang Lin

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-88883-5

Our file Notre référence

ISBN: 0-612-88883-5

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

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

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

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

Canada

Abstract

Load balancing has proven to be an effective way for achieving good performance and high scalability in distributed systems, and a great deal of research has been conducted in this domain in the past. In addition to distributed systems in general, load balancing is also used in heterogeneous distributed environments that use the Common Object Request Broker Architecture (CORBA) based middleware for achieving interoperability. So far, most of the research on load balancing has been concerned with wired systems, and little work has focused on wireless systems, in which certain inherent characteristics, such as low communication bandwidth, high delay and high packet drop rate, call for careful examination.

This thesis investigates the performances of load balancing strategies for CORBA-based systems focusing particularly on wireless environments. Each strategy reflects a distinct combination of a request mapping policy and an interaction pattern among the three components in the load balancing system: client, server and balancer. Using the commercially available CORBA-compliant software “Orbix E2A Application Server Platform 5.1”, a performance prototype of a load balancing system is designed and implemented. A wireless network is emulated with a well-known network emulation package, NIST Net. Based on extensive performance measurements, the effects of different workload factors on the performances of the different load balancing strategies are described, and derived insights are also discussed. In particular, a comparison of the effects of wired and wireless environments on the performances of various load balancing strategies is presented.

Acknowledgments

I would like to express my sincere thanks to my supervisor, Professor Shikharesh Majumdar, for his invaluable guidance and continuous support over the past two years.

I would also like to thank John Knox for his great help as a lab technologist. The support and friendship from our RADS lab group members, especially Ming, Umar and Yinjing, are highly acknowledged as well.

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

Financial support for this research was provided through the Ontario Graduate Scholarship and the Ontario Graduate Scholarships in Science and Technology. Thanks are due to IONA for providing a free evaluation copy of the middleware product used in this research.

Table of Contents

Chapter 1	Introduction.....	1
1.1	Background	1
1.2	Motivation	2
1.3	Goals and Contribution	4
1.4	Thesis Outline	5
Chapter 2	Overview of CORBA	6
2.1	Distributed systems	6
2.2	Distributed Object Computing	7
2.3	Common Object Request Broker Architecture (CORBA)	8
2.3.1	CORBA Architecture and Structure	8
2.3.2	Portable Object Adaptor	13
2.3.3	CORBA Services	15
2.3.4	CORBA Request Invocation	16
2.4	Wireless CORBA	18
2.5	Minimum CORBA	20
Chapter 3	Performance Prototype of a Load Balancing System.....	22
3.1	Load Balancing Overview	22
3.2	Related Work	24
3.2.1	Existing Work on CORBA-based Load Balancing	25
3.3	Load Balancing Strategy	26
3.4	Design of the Performance Prototype	32
3.5	Implementation Overview	35
3.5.1	IDL for the System Prototype	36
3.5.2	System Execution	40

Chapter 4 Performance Measurement and Instrumentation	44
4.1 Performance Metrics	44
4.2 Workload.....	45
4.3 Experimental Environment	49
4.4 Measurement Accuracy.....	50
 Chapter 5 Results of Experiments.....	 52
5.1 Introduction	52
5.2 Relative Performances of the Load Balancing Strategies for the Base Test.....	54
5.3 Effect of Message Size.....	62
5.4 Effect of Bandwidth	68
5.5 Effect of Inter-node Delay.....	73
5.6 Effect of Drop Rate	78
5.7 Effect of Server Demand.....	85
5.8 Effect of Coefficient of Variation of Think Time	90
 Chapter 6 Conclusions	 94
6.1 Summary	94
6.2 Derived Insights	95
6.3 Future Research.....	97
 References	 99
 Appendix A Supplementary Results for the Base Test	 105
Appendix B Supplementary Results for the Experiment Investigating the Effect of Message Size	106
Appendix C Supplementary Results for the Experiment Investigating the Effect of Bandwidth	107
Appendix D Supplementary Results for the Experiment Investigating the Effect of Inter- node Delay	108

Appendix E	Supplementary Results for the Experiment Investigating the Effect of Drop Rate	109
Appendix F	Supplementary Results for the Experiment Investigating the Effect of Server Demand	111
Appendix G	Supplementary Results for the Experiment Investigating the Effect of Variability in Think Time	112

List of Figures

Figure 2-1: CORBA Object Management Architecture: Marketing Reference Model (from [SIE-98])	9
Figure 2-2: Structure of CORBA Reference Model (from [SCH-98])	10
Figure 2-3: Architecture for Terminal Mobility in CORBA	19
Figure 3-1: FB Load Balancing Strategy	28
Figure 3-2: FS Load Balancing Strategy	29
Figure 3-3: HD Load Balancing Strategy	29
Figure 3-4: CBD1 Load Balancing Strategy	30
Figure 3-5: CBD2 Load Balancing Strategy	31
Figure 3-6: IDL Source Code of the System Interface	37
Figure 3-7: FBA Execution Scenario	41
Figure 4-1: Experimental Network Environment	50
Figure 5-1: Mean Response Time Measurement for the Base Test	55
Figure 5-2: Request Forwarding When NC=4	57
Figure 5-3: Client-Based Dispatching with Two Handles (CBD2) When NC=4	58
Figure 5-4: Response Time Difference between FBA and FB	59
Figure 5-5: Response Time Difference between FBA and CBD1	61
Figure 5-6: Response Time Degradation of CBD2 from CBD1	61
Figure 5-7: Response Time Measurement for the Test with MS of 5000 Bytes	64
Figure 5-8: Response Time Measurement for Each Strategy with Different Message Sizes When NC=12	67
Figure 5-9: Response Time Measurement for FBA with Different Message Sizes and Numbers of Clients	68
Figure 5-10: Response Time Measurement for the Test with BW of 100 Mbps	70
Figure 5-11: Mean Response Time Increase with BW of 1 Mbps over BW of 100 Mbps	72
Figure 5-12: Response Time Measurement for the Test with ND of 0 ms	75

Figure 5-13: Response Time Measurement for Each Strategy with Different Inter-node Delays When NC=12	77
Figure 5-14: Response Time Measurement for FBA with Different Inter-node Delays and Numbers of Clients.....	78
Figure 5-15: Response Time Measurement for Test with DR of 2%.....	80
Figure 5-16: Mean Response Time Increase of the Test with DR of 2% over 0%.....	81
Figure 5-17: Response Time Measurement for the Test with DR of 5%.....	83
Figure 5-18: Response Time Measurement for Each Strategy with Different Drop Rates When NC=12	84
Figure 5-19: Response Time Measurement for FBA with Different Drop Rates and Numbers of Clients.....	85
Figure 5-20: Response Time Measurement for the Test with SD of 50 ms.....	87
Figure 5-21: Response Time Measurement for Each Strategy with Different Server Demands When NC=12	89
Figure 5-22: Response Time Measurement for FBA with Different Server Demands and Numbers of Clients.....	90
Figure 5-23: Response Time Measurement for the Test with CV of 1.0	92
Figure A-1: Throughput Measurement for the Base Test	105
Figure B-1: Throughput Measurement for the Test with MS of 5000 Bytes	106
Figure C-1: Throughput Measurement for the Test with BW of 100 Mbps	107
Figure D-1: Throughput Measurement for the Test with ND of 0 ms	108
Figure E-1: Throughput Measurement for the Test with DR of 2%	109
Figure E-2: Throughput Measurement (request/sec) for the Test with DR of 5%.....	110
Figure F-1: Throughput Measurement for the Test with SD of 50 ms	111
Figure G-1: Throughput Measurement for the Test with CV of 1.0	112
Figure G-2: Response Time Measurement for the Test with CV of 2.0	113
Figure G-3: Throughput Measurement for the Test with CV of 2.0	114

List of Tables

Table 5-1: Workload Parameters.....	53
Table 5-2: Workload Parameters of the Base Test.....	54
Table 5-3: Response Time Measurement (ms) for the Base Test	55
Table 5-4: Average Server Utilizations in the Base Test When NC=16.....	58
Table 5-5: Parameters of the Experiment Investigating the Effect of Message Size	63
Table 5-6: Response Time Measurement (ms) for the Test with MS of 5000 Bytes.....	64
Table 5-7: Mean Response Time (ms) with MS of 100 and 5000 Bytes When NC=1.....	65
Table 5-8: Average Server Utilizations of Test with MS of 5000 Bytes When NC=16...	65
Table 5-9: Response Time Measurement (ms) for the Test with MS of 7500 Bytes.....	66
Table 5-10: Parameters of the Experiment Investigating the Effect of Bandwidth	69
Table 5-11: Response Time Measurement (ms) for the Test with BW of 100 Mbps.....	70
Table 5-12: Parameters of the Experiment Investigating the Effect of Inter-node Delay.	74
Table 5-13: Response Time Measurement (ms) for the Test with ND of 0 ms	75
Table 5-14: Response Time Measurement (ms) for the Test with ND of 30 ms	76
Table 5-15: Parameters of the Experiment Investigating the Effect of Drop Rate	79
Table 5-16: Response Time Measurement (ms) for Test with DR of 2%.....	80
Table 5-17: Response Time Measurement (ms) for the Test with DR of 5%.....	83
Table 5-18: Parameters of the Experiment Investigating the Effect of Server Demand...	86
Table 5-19: Response Time Measurement (ms) for the Test with SD of 50 ms.....	87
Table 5-20: Response Time Measurement (ms) for the Test with SD of 150 ms.....	88
Table 5-21: Parameters of the Experiment Investigating the Effect of Variability in Think Time	91
Table 5-22: Response Time Measurement (ms) for the Test with CV of 1.0	92
Table 5-23: Difference in Mean Response Time (percentage) between the Test with CV of 1.0 and the Test with CV of 1.5	92
Table A-1: Throughput Measurement (request/second) for the Base Test	105
Table B-1: Throughput Measurement (request/sec) for the Test with MS of 5000	106
Table C-1: Throughput Measurement (request/sec) for the Test with BW of 100	107

Table E-1: Throughput Measurement (request/sec) for the Test with DR of 2%	109
Table E-2: Throughput Measurement (request/sec) for the Test with DR of 5%	110
Table F-1: Throughput Measurement for the Test with SD of 50 ms	111
Table G-1: Throughput Measurement (request/sec) for the Test with CV of 1.0	112
Table G-2: Response Time Measurement (ms) for the Test with CV of 2.0	113
Table G-3: Throughput Measurement (request/sec) for the Test with CV of 2.0	114

Glossary of Terms and Symbols

ASP	Application Server Platform
BOA	Basic Object Adapter
BW	Bandwidth
CBD	Client Based Dispatching
CBD1	Client Based Dispatching with One Handle
CBD2	Client Based Dispatching with Two Handles
CORBA	Common Object Request Broker Architecture
CV	Coefficient of Variation of Think Time
DII	Dynamic Invocation Interface
DNS	Domain Name Server
DR	Drop Rate
DSI	Dynamic Skeleton Interface
FB	Forwarding with reply coming from Balancer
FS	Forwarding with reply coming from Server
GIOP	General Inter-ORB
HD	Handle Driven
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IOR	Interoperable Object Reference
MS	Message Size
NC	Number of Clients
ND	Inter-node Communication Delay
OMG	Object Management Group
ORB	Object Request Broker
POA	Potable Object Adapter
RADS	Real -Time and Distributed Systems
RPC	Remote Procedure Call
SD	Server Demand

TCP	Transmission Communication Protocol
TT	Think Time

Chapter 1 Introduction

1.1 Background

Middleware technologies for heterogeneous distributed systems have been built and used successfully in the industry for the past decade. Among others, the Common Object Request Broker Architecture (CORBA) is a widely used standard for middleware, providing interoperability and various services in distributed systems, either wired or wireless. Using a CORBA middleware, it is possible for a client to communicate with a server even though they are implemented in different programming languages and are running on top of different operating systems [OMG-02e].

The rapid growth of wireless technology over the past few years has resulted in an increased use of mobile and wireless devices in the context of mobile computing. The number of users of devices such as mobile phones, laptops, and personal digital assistants requiring wireless data services has been increasing, with growth in many fields, including telecommunication, e-commerce, and on-line banking. Coupled with the advance of the Internet, wireless technology enables people to access their personal information, public as well as enterprise resources, anytime and anywhere.

Wireless distributed systems in mobile computing environments consist of fixed host computers and mobile client computers [MUR-97]. Fixed hosts are usually connected via a wired high-speed network. The mobile clients are able to connect to the wired network via a wireless link. As in the traditional wired distributed systems, the development of distributed applications in wireless systems usually requires help from middleware

technology in order for the developer to be able to deal with problems related to distribution, such as heterogeneity, scalability, and resource sharing.

The demand for wireless distributed systems with good performance and high scalability has been increasing over the past decade, with growth in many fields, as mentioned earlier. Load balancing has proven to be effective with regard to good performance and high scalability, in both wired and wireless distributed systems. By distributing client workload equally among back-end servers, overall system performance and scalability can be improved.

Load balancing can be static or dynamic. In the case of static load balancing, the server run-time load state is not required at the time of the initial request placement on a server, and requests from a client will continue to be served on the same server. In dynamic load balancing, however, the server run time load state plays an important role during the initial placement phase and during subsequent phases. Processes may be migrated from one server to another if necessary.

1.2 Motivation

Load balancing in CORBA-based systems has gained attention in recent years, especially in wired network environments. Studies of load balancing strategies could provide application and system designers with insights and experience regarding how to improve performance in a particular setting. For example, Rackl has conducted research on various load distribution strategies for CORBA-based applications [RAC-97]. In her Master's thesis, Tao carried out a study which focuses on the relative performances of different load balancing strategies for CORBA applications, and provides certain load

balancing guidelines at the application level [TAO-02]. More recently, Othman et al. conducted a study on the design and performance of an adaptive middleware load balancing service in CORBA environments [OTH-03].

However, most of the research conducted in the past has focused on wired systems, and little attention has been paid to wireless systems, in which certain inherent characteristics such as limited communication bandwidth, high delay, and packet drop call for careful investigation when developing a CORBA-based load balancing strategy.

An earlier study conducted in the Real-Time and Distributed Systems (RADS) lab at Carleton University has focused on the relative performances of different CORBA-based client server interaction architectures [ABD-02]. However, it does not address load balancing on multiple back-end servers. More recently, a study conducted at RADS lab has focused on the performance of mobile wireless systems, and presented a semi-durable subscriptions technique for publish/subscribe systems in mobile wireless environments to achieve high performance [FAR-03].

In contrast to the research mentioned above, the research in this thesis focuses on the relative performances of different load balancing strategies for CORBA-based applications in wireless environments. Such strategies originate from different client server interaction patterns and request mapping policies. To the best of our knowledge, this kind of research has not been carried out by other researchers. It is believed that the performance results and insights gained from this research will provide system and application developers with valuable assistance when working on CORBA-based load balancing systems in general and wireless environments in particular.

1.3 Goals and Contribution

The primary goal of this thesis is to investigate a CORBA-based load balancing system. Different load balancing strategies are proposed and their performances are evaluated. Particular attention has been paid to the performances of the strategies in wireless environments. These strategies are: Handle Driven (HD), Forwarding with reply coming from Balancer (FB), Forwarding with reply coming from Server (FS), Forwarding with reply coming from Balancer and using the number of Active requests in server as the load index (FBA), and Client Based Dispatching (CBD). Each strategy reflects a distinct combination of a request mapping policy and an interaction pattern among the three components in the load balancing system: client, server and balancer.

A performance prototype of a load balancing system is designed and implemented with the commercially available software Orbix E2A Application Server Platform from IONA [ION-02a]. This system provides a testing tool for different strategies. The wireless network environment is emulated with a well-known network emulation package, NIST Net [NAT-02].

The main contributions of this thesis include:

- The design and implementation of the performance prototype of the load balancing system using the Orbix E2A Application Server Platform.
- The proposal of a number of different load balancing strategies.
- An investigation of the relative performances of the load balancing strategies for CORBA-based systems focusing primarily on wireless environments.
- A presentation of the impacts of various workload and system parameters on the performances of the load balancing strategies.

1.4 Thesis Outline

This thesis consists of six chapters. In Chapter 2, we present the basic concepts of distributed systems focusing on distributed object computing, and provide an overview of CORBA, as well as its related specifications used in mobile computing. This includes Wireless CORBA and Minimum CORBA. Chapter 3 presents the design and implementation of the performance prototype of a load balancing system. It also presents related work. Chapter 4 describes the performance metrics, the workload and the experimental environment used in the investigation. Discussion of the performances of different load balancing strategies is presented in Chapter 5. Finally, Chapter 6 presents the concluding remarks and further research issues.

Chapter 2 Overview of CORBA

2.1 Distributed systems

A distributed system can be defined as a collection of autonomous computers linked by a network to provide an integrated computing facility. Such a system might consist of a dozen hosts in a local area network environment, or might consist of thousands of hosts in an Internet setting.

The decline in the price performance ratio of computer systems, the increasing availability of high speed networks, and the advance of distributed computing technology have been the most important factors for the increasing popularity of distributed systems. Distributed systems exhibit many advantages, such as hardware and software resource-sharing, concurrency in terms of multi-programming and multi-processing, scalability, fault tolerance, and transparency with regard to location and access.

Distributed systems include both wired and wireless or mobile systems. As explained in [MAS-02], the differences between the two can be better understood from the following three concepts: the type of device, the type of network and the type of execution context. In a wired distributed system, devices are usually fixed and vary from home PCs, to UNIX workstations, to IBM mainframes. And fixed hosts are usually permanently connected to the network through continuous high-bandwidth links. In addition, the application execution context is more or less static: location almost never changes, and the frequency of hosts being added or deleted is low. While in a wireless or mobile system, devices vary from personal digital assistants, to mobile phones, to smart cards. And such devices are usually connected to a network through a wireless link with

relatively low bandwidth. Disconnections may happen unpredictably. Moreover, in some systems, the application execution context is extremely dynamic in the sense that mobile hosts may come and leave generally more rapidly.

2.2 Distributed Object Computing

Software programs written in object-oriented languages, such as C++ and Java, are made up of cooperating objects. An object is simply a group of related data, with methods available for queuing or altering data, or for taking action based on the data. With the introduction of distributed object technology, objects can be distributed across the network. The power of such distribution is that the requestors of services (clients) can request a service from the providers of services (servers) that probably reside on other hosts through a well-defined encapsulation interface.

Computing in such a distributed object system is called distributed object computing. It is concerned with the following important issues [SCH-95].

- 1) Providing enhancements to Remote Procedure Call (RPC) toolkits with object-oriented language features such as encapsulation, interface inheritance and object-based exception handling.
- 2) Enabling inter-working between applications at higher levels of abstraction.
- 3) Providing a foundation for building higher-level mechanisms to facilitate the collaboration among services in a distributed system.

2.3 Common Object Request Broker Architecture (CORBA)

CORBA is an open standard for middleware in distributed object computing environments [OMG-02e]. It was first published in 1991 by the Object Management Group (OMG), which is an international organization formed in 1989 to promote the theory and practice of object oriented technology, by accelerating the introduction of standardized object software, such as CORBA, UML, XMI. Since then, CORBA has evolved and grown into a well-established software standard, and has gained a phenomenal industry acceptance.

With the ongoing change in technology, and the requirements of systems for various and robust services, distributed systems are typically heterogeneous. That is, a system may consist of multiple computing platforms in terms of software and hardware technology. CORBA is meant to be used in such heterogeneous systems for providing interoperability among different hosts.

2.3.1 CORBA Architecture and Structure

CORBA is a specification, and not an implementation. It provides an object-oriented framework that allows applications to interoperate in a distributed and heterogeneous environment. The overall CORBA architecture is depicted in Figure 2-1.

The Object Request Broker (ORB) is the central piece of the architecture which serves as a “software bus,” connecting different objects over the network. CORBA Services are the services essential for implementing objects, and they provide basic functionalities that are useful in many applications. A few examples of CORBA Services, such as the Event Service and Naming Service, will be presented later. CORBA Domains

specify various application areas, such as health care and financial services. On the other hand, CORBA Facilities are useful for the construction of applications across a wide range of application domains, and they are divided into Horizontal Facilities, which are user-oriented, and Vertical Facilities, which support specific application domains [SIE-98].

Figure 2-1: CORBA Object Management Architecture: Marketing Reference Model
(from [SIE-98])

CORBA is structured to allow the integration of a wide variety of object systems. Objects on different hosts can be implemented in different languages, such as C++ and Java, and they can run on top of different operating systems, such as Windows and Linux. Figure 2-2 shows the structure of a CORBA reference model. CORBA Objects in different hosts can interoperate through the use of Interface Definition Language, which is language-independent and described in the CORBA specification. Each component in that structure is briefly described. The descriptions are based on the latest version of the CORBA specification [OMG-02e].

ORB CORE: This component works as a software bus, accepting requests from clients and activating object implementation if it is not running; delivering requests; and returning responses back to clients. It is the communication infrastructure of any CORBA implementation. ORBs between different hosts communicate via the Internet Inter-ORB Protocol (IIOP) that is an implementation of the General Inter-ORB (GIOP) protocol running on top of the Transmission Communication Protocol (TCP).

Figure 2-2: Structure of CORBA Reference Model (from [SCH-98])

Client: A client is usually considered to be a process that has access to an object reference for an object and invokes operations on it. An object reference is the information needed to specify an object within an ORB. Occasionally a client is

recognized to be relative to a particular object. For instance, the implementation of one object may be a client of other objects.

Object Implementation: An object implementation provides the semantics of the object through the definition of data for the object instance and the code for the object's methods. It often uses other objects or additional software in the implementation of the object behavior.

Interface Definition Language (IDL): IDL defines the types of objects by specifying their interfaces, each of which consists of a set of named operations and the corresponding parameters. It provides the means of making a particular object implementation available to its potential clients and the way in which it is to be invoked. It also makes it possible to map CORBA objects to particular programming languages or object systems.

IDL Stub and Skeleton: These two components are generated by the IDL compiler on the client and server side, respectively. A stub provides a mechanism that creates and issues requests on behalf of a client, whereas a skeleton provides a mechanism that delivers requests to the CORBA object implementation on the server side.

Dynamic Invocation Interface (DII): DII is an interface that allows for the dynamic construction of object invocation in a client program. That is, instead of calling a stub routine that is specific to a particular operation on a particular object, a client may specify the object to be invoked, the operation to be performed, and the set of parameters for the operation through a call or a sequence of calls at run time.

Dynamic Skeleton Interface (DSI): DSI is an interface that allows for the dynamic handling of object invocation on the server side. An object implementation is reached

through an interface providing access to the operation name and parameters in a similar manner with client side's DII at run time, rather than being accessed through a skeleton specific to a particular operation.

Object Adapter (OA): An object adapter provides the means by which an object implementation accesses services provided by the ORB. Such services by the ORB through the Object Adapter include object reference generation and interpretation, method invocation, object and implementation activation and deactivation, and so on. Up to version 2.1 of the CORBA specification, only the Basic Object Adapter (BOA) was specified. It has been replaced with the Potable Object Adapter (POA), which provides richer functionality and more portability over the BOA. Since POA is important for the construction of an application system, such as the performance prototype of load balancing system in this thesis, it will be discussed in more detail in section 2.3.2.

ORB Interface: This interface contains operations that are available to both clients and servers. Such operations include the initialization or shutdown of the ORB and conversion between the object references and strings, among others.

Interface Repository: The interface repository provides persistent objects representing the IDL information in a form available at run-time. In other words, it provides for the storage, distribution, and management of a collection of related objects' interface definitions. With the information stored in this repository, a program is able to determine the operations of an object, and perform an invocation on it at run-time.

Implementation Repository: The interface repository provides the ORB with information on locating and activating the implementation of objects. It is also often used for storing additional information associated with the implementation of ORB objects.

2.3.2 Portable Object Adaptor

The Portable Object Adaptor provides fundamental services such as object creation, object mapping, and request dispatching. The services provided allow developers to write scalable, high performance server applications, giving them proper control at the application level over the resources required for implementing CORBA objects. Due to the limitation in CPU power, network connections and memory available for server applications, it is essential that POA be understood thoroughly in order to provide the best possible service to every client, and create high-performance server applications [VIN-99].

The POA provides a wide range of interfaces that offers many of the features missing in the original BOA specification [PUD-99].

- It supports the transparent activation of objects. Servers can export object references for not-yet-active servants, which will be incarnated on demand.
- It allows a single servant to support many object identities.
- It allows for many POAs in a single server, each governed by its own policy.
- It can either delegate requests for nonexistent servants to a default servant or ask a servant manager for an appropriate servant.

A servant provides the implementation for one or more object. In C++ language mapping, a servant is an instance of a C++ class that inherits from the special class called *ServantBase*. The mapping of a servant with an object id is called activation and is performed using POA methods. An object id is an opaque sequence of octets generated by the system or specified by the application. An object reference is just an object id with opaque, POA-added “internal” information.

Each POA maintains an active object map that maps activated objects in the POA to a servant. For each incoming request, the POA tries to find the responsible servant based on the following request processing policies:

- **USE_ACTIVE_OBJECT_MAP_ONLY**

The object reference is looked up in the active object map only. If none is found, the request is rejected.

- **USE_DEFAULT_SERVANT**

The object reference is looked up in the active object map first. If not found, the request is delegated to a default servant.

- **USE_SERVANT_MANAGER**

The object reference is looked up in the active object map first. If not found, a servant manager is invoked to locate or incarnate an appropriate servant. There are two types of servant manager: a servant activator and a servant locator. A servant activator activates a new servant, which will be retained in the POA's active object map to serve further requests on the same object. In contrast, a servant locator is used to locate a servant for a single invocation only. Since active servants take up CPU memory, and activating of a new servant is usually operation-expensive, as a result, the proper use of the servant manager may improve server performance [VIN-99].

In addition to the request processing policy, the thread policy in a POA can determine whether invocations are scheduled by the ORB simultaneously or in sequence. In the former case, an application can service each new request in a separate, newly created thread. Depending on the various CORBA implementations, such multithreading support can be provided in one of three models: a thread pool, a thread-per-request, or a thread-

per-object model. The use of multiple threads to service multiple requests concurrently usually results in an improvement in client response time and system throughput over the sequential scheduling of request to a single thread.

2.3.3 CORBA Services

To allow standard communication for application objects, CORBA provides a set of services on a low-level interface. The services which are commonly used in practical application development include the following:

- Naming Service

The Naming Service provides the principle mechanism through which most clients of an ORB-based application locate and invoke objects. It uses the association of name-to-object to locate an object based on a human-readable name. Such an association is also called a name binding, and is always defined relative to a naming context, which is an object containing a set of name bindings. Under a same naming context, each name is unique.

Of the many operations provided in the Naming Service, two are the most fundamental. The first is the resolving of a name, which determines the object associated with the name in a given context. By contrast, the second is the binding of the name, which creates a name binding in a given context [OMG-02d].

- Event Service

Event Service provides a model of decoupled communication between objects. The standard synchronous execution of an operation by an object requests that both the client and the server be available. Event Service decouples this communication by

defining two roles for objects: the supplier and the consumer. Suppliers produce event data that consumers expect to process. The communication of the event data is performed using standard CORBA requests. There is an event channel between the suppliers and consumers that allows multiple suppliers to communicate with multiple consumers asynchronously.

There are two approaches to initiating event communication in the Event Service: the push model and the pull model. In the push model, the event suppliers initiate the transfer of the event data to consumers, whereas in the pull model, the consumers take the initiative to request the event data from a supplier [OMG-02a].

- **Trading Object Service**

The Trading Object Service provides another type of communication, which facilitates the offering and discovery of instances of services of particular types. The Trading Object Service in a distributed environment is supported by a trader, an object through which other objects can advertise their capabilities and match their needs against advertised capabilities.

With the Trading Object Service, an object can export a service by giving the trader a description of a service and the location of an interface where the service is available. When an object wants to import a certain type of service, it asks the trader for it, including a description of certain characteristics [OMG-02b].

2.3.4 CORBA Request Invocation

In CORBA, when a client performs an operation on a server in a remote host, the following steps are executed in a typical POA-based request invocation.

- The client makes a call to the object using the object reference.
- A corresponding client stub is called to perform message marshaling, whose purpose is to transfer the request-related application data in the client's native language format onto a common packet level representation.
- The marshaled message is then transmitted to the server application over a newly-created or existing connection between the client and the server.
- Upon receiving the marshaled message on the server side, the server ORB un-marshals the message, which contains the object key.
- Since the object key uniquely identifies the target object in the server application, the ORB dispatches the request to the POA which hosts the target object.
- The POA then uses a portion of the object key, called the object ID, to find the responsible servant for the target object based on the request-processing policy mentioned earlier.
- The servant processes the request, and delivers the results back to the POA, which sends the results the server ORB.
- The server skeleton then marshals the results and transmits them to the client side, where another un-marshaling is required for obtaining the results.

The entire request invocation mechanism is completely transparent to the client, to which a remote object looks like an ordinary method invocation. In particular, it exhibits the following important characteristics of the CORBA specification [VIN-99]:

- Location transparency

The client does not know or care whether the target object is local to its own address space, in a different process in the same host, or even in another host. The operation is invoked in the same way in each case.

- **Server transparency**

The client does not need to know which server implements which object.

- **Language transparency**

The client does not need to care about the language used by the server. A C++ client, for example, may call a Java implementation on the server side.

- **Operating system transparency**

The client does not care what operating system is used by the server, and this transparency is provided by the middleware that takes care of byte ordering and data alignment.

2.4 Wireless CORBA

The recent rapid development of wireless technology has brought about the growth of mobile computing. The development of applications in mobile and wireless environments calls for a need for a language-neutral distributed object-oriented middleware platform to manage and use different mobile devices, and to provide services on top of it. CORBA, which is widely used in wired networks, has been adopted to fit the requirement. Recently, OMG published a formal Wireless CORBA Specification [OMG-03] in an effort to promote the use of Wireless CORBA. An example implementation of Wireless CORBA is MIWCO, which is a collection of extensions for MICO to add support for wireless access and terminal mobility [KAN-02].

The basic design principles in Wireless CORBA are the client-side ORB transparency and simplicity. The target of transparency is the interoperability between mobile ORBs and non-mobile ORBs. The overall architecture of Wireless CORBA is depicted in Figure 2-3, where three different domains and various components are identified [OMG-03].

Figure 2-3: Architecture for Terminal Mobility in CORBA

The Home Domain is the domain hosting the Home Domain Agent of the terminal, which keeps track of the Access Bridge with which a mobile terminal is associated and provides operations to update and query the terminal location. The Access Bridge on the visited domain is the network side end-point of the GIOP tunnel, encapsulating the GIOP messages to the terminal Bridge and de-encapsulating these messages from the terminal bridge. The Visited Domain hosts one or more access bridges, through which it provides ORB access to some mobile terminals.

The Terminal Bridge is the terminal side end-point of the GIOP tunnel, which provides the means of transmitting GIOP messages between the Terminal Bridge and the Access Bridge. The generic GIOP Tunneling Protocol defines how the GIOP messages are transmitted. The Terminal Domain consists of a terminal device that hosts an ORB and a Terminal.

In contrast to regular CORBA, Wireless CORBA introduces the concept of a Mobile IOR to identify mobile objects and to provide a solution to the mobility problem in wireless systems. An IOR refers to an interoperable object reference that is used to identify objects in a CORBA-based system. Each object has a unique IOR used by the client for invoking a method on the object. A Mobile IOR is a special interoperable object reference. It hides the mobility of a terminal from the client application that invokes operations on target objects located on the terminal. That is, it provides mobility transparency in the mobile computing environment.

2.5 Minimum CORBA

As we know, some devices have limited resources, such as a smaller CPU power, smaller memory, and a lower transmission capacity. Those devices include mobile terminals, such as personal digital assistants, laptops, and cell phones. CORBA is too large to meet the memory size and performance requirements of the applications in such devices. Therefore, a cut-down version of CORBA is required, and this is called Minimum CORBA [OMG-02c].

Minimum CORBA is meant to provide a middleware standard that is used in systems with limited resources. Its specification supports all of OMG IDL, as defined in CORBA

specifications to allow maximum compatibility between Minimum CORBA and the full CORBA specification.

Aiming at providing a trade-off between usability and conserving resources, Minimum CORBA omits some features of the general CORBA. For example, Dynamic Invocation Interface and Dynamic Skeleton Interface are omitted altogether from Minimum CORBA. Therefore, CORBA has a greater degree of user-friendliness, whereas Minimum CORBA is better for systems with limited resources.

Chapter 3 Performance Prototype of a Load Balancing System

In this chapter, we will present an overview of load balancing in Section 3.1. The related work will be described in Section 3.2. Section 3.3 proposes six load balancing strategies for a CORBA-based system. The relative performances of these strategies are investigated in this thesis. Section 3.4 presents the design of the performance prototype for the load balancing system, while Section 3.5 gives a high-level overview of the implementation.

3.1 Load Balancing Overview

Distributed systems often employ multiple back-end servers to process service requests coming from a large number of clients. The use of multiple back-end servers in such systems protects initial hardware investment and provides an organization with a cost effective method of handling the growth resulting from the increase in the number of clients, by simply providing more servers on demand.

Load balancing has been one of the main advantages of distributed systems over stand-alone systems. It aims to distribute the workload among the back-end servers equally, in order to improve system performance. For example, in a web server system, many requests can be processed concurrently with multiple back-end web servers, so that client response time can be minimized.

In a distributed system, there are four different layers at which load balancing can be implemented: the network, the operating system, the middleware and the application

[OTH-03]. In certain systems, a Domain Name Server (DNS) performs network-based load balancing and a client requesting a hostname contacts a dynamically assigned server for services. Distributed operating systems provide OS-based load balancing through clustering and processor scheduling. On the other hand, middleware provides load balancing in terms of its component service. For example, Orbix 2000 provides a load balancing service through server clustering [ION-01].

The last layer of implementation is the application level, where application developers design and implement their own load balancing systems by exploring application level techniques to suit the need of a particular application in a certain environment. One such example is a distributed cooperative web server load balancing system, presented in [BAK-99].

Regardless of the implementation layers, load balancing strategies fall broadly into either the static or dynamic category. In a static load balancing system, the distribution of the workload does not depend on the system state or the current server load. While in a dynamic load balancing system, load is distributed according to the server load condition. For instance, a request from a client can be routed to the server with the least load so as to ensure a minimum response time.

Dynamic load balancing strategies adapt to changes in the system state and are believed to have better performance than static strategies. Therefore, most research conducted in the past has focused on dynamic load balancing [HUI-99]. However, major concerns with regard to dynamic load balancing systems include the complexity related to the implementation of dynamic policies, the overhead caused in communicating and monitoring server loads, and the choice of load index. At the same time, the simplicity

and lower incurred overhead has made static load balancing very attractive. Moreover, research has shown that if the overheads in dynamic load balancing are not negligibly small, static load balancing can offer better performance [ZHA-97].

3.2 Related Work

Many studies on CORBA performance were conducted in the past. For example, high overheads of excessive data copying and convention in CORBA implementation were analyzed [GOK-96]. A perfect hashing technique was proposed as the message demultiplexing strategy for CORBA implementation [GOK-97]. Different from these approaches, a technique called Flyover that explored limited heterogeneity in systems in order to achieve high performance was developed [WU-02]. With such a technique, a separate path between the client and its server was employed to avoid a number of CORBA overheads when the client and server were implemented using the same programming language and operating platform. Other examples of existing research on CORBA performance were described in Section 1.2.

Performance of wireless systems has started receiving attention from researchers recently. Yuan presented a technique for balancing the numbers of subscribers among multiple Mobile Switching Center (MSC) elements in a wireless network [YUA-97]. This study showed that the mobility of subscribers was a significant factor in resource allocation. By effectively load balancing the number of subscribers among MSCs, network capacity was shown to increase.

A study focusing on performance comparison of load balancing methods in multiple carrier CDMA systems has been conducted by He [HE-00]. These load balancing

methods include mobile hashing, round-robin, Walsh-Code (WC) usage and power usage. The simulation results show that WC usage-based load balancing provides the best overall performance.

3.2.1 Existing Work on CORBA-based Load Balancing

As a standardized middleware in distributed object computing, CORBA has been well used in building applications in distribution systems in the fields of telecommunication, finance and other sectors. As a result, CORBA-based load balancing has also received much attention in recent years. A representative set of works is described in this section.

By extending the naming service in CORBA, a resource management system called WINNER was built to provide load distribution to the CORBA applications [BAR-98]. It provides an approach for integrating the load distribution mechanism into the naming service, and features load information collection and placement decision ability.

An interceptor-based load balancing system called LODIN was developed by Kim et al. [KIM-01]. This system features a transparent load distribution mechanism that has the advantages of the interceptor and efficient communication with the event service.

In both of the above-mentioned systems, load balancing is dynamic and centralized in the sense that the centralized balancer collects run-time load information on the back-end servers and makes decisions regarding load distribution. Two primary concerns arise with these centralized strategies. The first is reliability, as a failure of the balancer or scheduler can cause the entire system to fail. The second is that the balancer may become a performance bottleneck as the system scales. In contrast, in decentralized policies, each node can broadcast its load state to other nodes, or send a request-for-bid message to

collect the load information of the other nodes. Each node makes its own load balancing decision [KAM-97]. There are three types of policies in decentralized load balancing: sender-initialized, received-initialized, and symmetrically-initialized. An implementation of a CORBA-based system to study these policies can be found in [RAC-97].

More recently, Othman et al. presented the design and implementation of an adaptive CORBA load balancing and monitoring service [OTH-03]. They also pointed out the deficiencies in the common load balancing techniques, and described the design challenges in building an adaptive load balancing service for the real-time CORBA TAO.

3.3 Load Balancing Strategy

With regard to static load balancing, since there is no job migration or transfer during the processing of a client request, and no exchange of load information between each back-end server and the balancer, the design and implementation of load distribution become much simpler compared to the dynamic load balancing system. However, careful consideration of the following two issues is very important for overall system performance.

The first issue is the *request mapping* policy. It refers to a particular algorithm that is used by the balancer to map each incoming request to a chosen server replica. Examples of such algorithms include round-robin and randomized. Choice of the request mapping policy is expected to have a significant impact on the system performance.

The second issue is the *interaction pattern* among the three components of the system: the client, the balancer and the server. In [ADL-95], two broker-based models are presented: forwarding and handle driven. In the forwarding model, the broker forwards

the request from the client to the server and forwards the reply from the server to the client. In the handle driven case, on the other hand, a client gets an IOR also called a “handle” from the balancer, and makes a request to the server using the handle. Abdul-Fatah has conducted a thorough study on the performances of CORBA-based client server interaction architectures, and presented many insights on the advantages and disadvantages of the various architectures [ABD-97].

The interaction pattern has an influence on the client binding granularity, which reflects the request-binding period, i.e. how long a client is bound to a server. In static load balancing, there are two types of binding: per-request and per-session. With the interaction pattern in which each request goes through the balancer, the client binding granularity is said to be per-request based. In this case, each request may be served by a potentially different server replica. While in the per-session case, a client makes requests to the same server replica during a session without going through the balancer. A session is defined as the lifetime of the client. Relatively speaking, per-session approach has an advantage over the per-request one, in that a client does not need to obtain a handle from the balancer for each request, or each request from clients does not need to go through the balancer. However, per-request gives the balancer a better opportunity to balance the load, since it takes the whole system into account when making a request mapping decision.

Taking these two issues into consideration, we propose six load balancing strategies and investigate their relative performances. The system includes multiple instances of servers, each of which is capable of satisfying the client request.

With all the strategies, except the strategy called FBA, a round-robin request mapping policy has been adopted in the selection of a server for the balancer or clients. This policy

is straightforward in the sense that for each client request each server is selected sequentially from a list in a round-robin order. It is simple to implement, and it can provide significant performance improvements over the randomized policy, in which a server is selected randomly.

Load Balancing Strategies:

(1) Forwarding with reply coming from Balancer (FB)

As shown in Figure 3-1, the client makes a request to the balancer, which forwards the request to the chosen server. Upon finishing the processing of the request, the server sends the reply to the balancer, which sends the reply to the client that originated the request.

Figure 3-1: FB Load Balancing Strategy

The main characteristics of FB are threefold. First, there are four message exchanges among the client, balancer, and the server. Second, client and server become decoupled, due to the role of the balancer. Third, it is a per-request approach but avoids the overhead of getting a handle before making a request.

(2) Forwarding with reply coming from Server (FS)

FS is similar to FB but differs with regard to the return of the reply from the server. As depicted in Figure 3-2, the reply is sent directly to the client by the server.

Figure 3-2: FS Load Balancing Strategy

In this strategy, there are only three message exchanges for each request. Meanwhile, because the server must contact the client, the client's handle must be included with the request. Furthermore, each call is in the form of a one-way asynchronous invocation.

(3) Handle Driven (HD)

The HD strategy is shown in Figure 3-3. For each request, the client must ask the balancer for a handle first. The balancer selects a specific server instance and returns its handle to the client, which then invokes operations on the server using the given handle.

Figure 3-3: HD Load Balancing Strategy

In this case, there are a total of four message exchanges for each request, and there is no direct communication between the server and the balancer during the service of the request.

(4) Client-Based Dispatching with One Handle (CBD1)

In the above-mentioned strategies, the balancer alone makes the decision regarding the request mapping, while the client can only take the decision when making a request. In the case of client-based dispatching, the client is given one or more handles from the balancer at the beginning. If the number of the handles is more than one, then the client can make its own decision regarding the choice of a server with the available handles. In CBD1, each client has only one handle, and keeps using this handle for services during the entire session. Figure 3-4 below depicts this strategy.

Figure 3-4: CBD1 Load Balancing Strategy

Here, the client requests one handle for a server only at the beginning of its session. Once it has obtained a handle from the balancer, it invokes operations on the corresponding server for the entire lifetime of the client. Note that the difference between CBD1 and HD is expected to diminish as the session length becomes smaller and smaller.

(5) Client-Based Dispatching with Two Handles (CBD2)

This strategy is similar to CBD1 but each client is given two handles from the balancer before making any requests for service from the server. Each handle corresponds to a specific server instance of the same server. The client then uses the two

handles in a round-robin order to make each request to the corresponding server during its lifetime. The motivation for using CBD2 is as an investigation into the benefits of static localized load balancing, as opposed to centralized load balancing. Figure 3-5 depicts the strategy, where there is one client and two servers.

After obtaining two handles from the balancer, the client makes the first request to server A, which sends the reply to the client. The client then makes a second request to server B, and receives the reply. Throughout the session, the client makes requests to the two servers in a round-round fashion.

Figure 3-5: CBD2 Load Balancing Strategy

(6) Forwarding with reply coming from Balancer and using the number of Active requests in server as the load index (FBA)

Hui et al. proposed a dynamic load balancing strategy in which the number of active running jobs allocated to the server is used as one of the factors in making a scheduling decision [HUI-99]. Through a close examination of the interaction pattern in FB, we can take advantage of such a dynamic load balancing strategy in a static context. As we know, the balancer knows the number of active requests being processed in each server, since

each request and reply goes through the balancer. Thus, instead of forwarding requests to servers in a round-round fashion, the balancer forwards the request to the server with the least number of pending requests being processed. In cases where there is more than one server with the lowest number of active requests, one is chosen at random.

3.4 Design of the Performance Prototype

In order to carry out performance measurements and compare the load balancing strategies, a performance prototype of a CORBA-based load balancing system was designed and implemented. The prototype-based approach is believed to have many advantages over simulation or mathematical analysis, in the sense that it reflects the real environment in a more practical way and results in more convincing performance comparisons among the different strategies.

There are three components to the load balancing system: the client, the balancer, and the server. Each client is responsible for making requests. After making a request and receiving a reply from the server, the client sleeps for certain duration before making another request. This duration of time is termed *think time*, and it controls the request arrival rate.

The server is the service provider. It processes requests coming from the client and communicates the responses to the client. The balancer plays the most important role in the system, processing requests for handles from the clients, in the case of HD and others. It forwards the service request to the chosen server in FS, FB, and FBA, and also forwards the reply from the server to the client in the cases of FB and FBA. Note that the

balancer is implemented as a separate process and is not a part of the middleware product used in this thesis.

In order to develop a system in which different load balancing strategies can be executed and a fair performance measurement can be obtained, several design issues had to be addressed.

- Client Representation

As we know, in practice, there is usually one client running in one host. Given the numbers of clients in the experiments and the available number of hosts in the lab, it is impossible to allocate a computer for each client. Therefore, we can only provide one process or one thread for each client. The process-based approach is simple in that we can just run an identical program in a terminal to represent a client. However, due to the overhead associated with context switching and memory resources, it is more efficient to use one thread for each client. The context switching overhead and memory usage are much lower with a thread-based approach in comparison to a process-based approach. Moreover, control and synchronization among the clients becomes easier as all threads share the same address space in the process that creates them. In this way, we avoid unnecessary contention on the client side and achieve measurement efficiency and correctness.

- Client Synchronization

The duration of an experiment is based on the time required for achieving a good accuracy of measurement. For example, if a fifteen-minute running time is selected, then all the clients must be able to start and end simultaneously. We could have used the number of calls to be executed by each client to determine the end of an

experiment. However, using a time-based approach provides a fairer comparison of the different strategies, since all the clients run in the same load condition, whereas in the call number-based approach, some clients may finish earlier than others, and therefore the clients that finish their runs later run in a less loaded environment.

- Forwarding Mechanism

In the forwarding cases, i.e. FS, FB and FBA, the balancer needs to forward requests to the chosen server. Moreover, in the latter two cases, it must also forward responses back to the clients. For this system, we took advantage of the Portable Object Adapter, as explained in Section 2.3.2. A dummy server object is deployed in the balancer as a representative for each set of server replicas. Such a server object is called dummy in the sense that it implements the *ServerObject* interface but it does not provide real service as done by the *ServerObject* implementation on the server side. Instead, it uses `USE_SERVANT_MANAGER` as the request processing policy in the POA on the balancer side to intercept the requests from the clients for the server object it represents, and deliver them to a server based on the specific request mapping policy. This facilitates the implementation of the request mapping functionality on the balancer side due to the one-to-one relationship between the set of server replica and the representative.

In this thesis, we use the servant locator for the implementation of this representative. As mentioned in Section 2.3.2, when the request arrives at the balancer, the POA delegates the task of locating a suitable servant for the servant locator, rather than using the servant lookup mechanism in the POA's active object

map [VIN-99]. The locator then provides a forwarding mechanism by making a request operation on a chosen server, and returns the response if necessary.

- **Multithreading**

Multithreading allows operations to execute simultaneously, without impeding the progress of other operations. The use of multithreading in a CORBA-based system can improve end-to-end throughput and latency, as well as the perceived response time for interactive client applications [SCH-98]. The use of multithreading in the performance prototype is discussed further in the beginning of the following section.

3.5 Implementation Overview

Orbix 2000 from IONA is one of the popular CORBA platforms. It supports the CORBA 2.4 standard and provides various services in addition to the traditional ones, such as the notification service, transaction and persistent state service. More importantly, it is one of the few CORBA implementations to offer multithreading support. It also replaces the Basic Object Adapter with the more flexible and performance-oriented Portable Object Adapter, as described in Section 2.3.2.

Orbix E2A Application Server Platform (ASP) provides Orbix 2000 for the support of CORBA. Orbix 2000 uses a thread pool model to support multithreading in an ORB instance running on a host [ION-02a]. Each thread is responsible for reading inbound requests from the network and placing them on an internal request queue. The minimum, the maximum, and the initial numbers of threads are configurable in the application. With the default policy, the initial number of threads is five, and there is no limit on the maximum number of threads allowed in the pool. In addition, idle threads are not

terminated by default. In our experiments, we take advantage of the multithreading support provided by the default policy in Orbix 2000 and make both the balancer and the servers multithreaded.

The system is developed in Java with the standard edition of IONA Orbix E2A ASP 5.1 [ION-02a]. ASP 5.1 provides a single multi-technology middleware platform for CORBA, J2EE, Web Services, and Mainframe technologies.

3.5.1 IDL for the System Prototype

Defining interfaces using the Interface Definition Language (IDL) is usually an important step for every CORBA application. IDL was briefly explained in Section 2.3.1, and IDL source code will provide us with a high-level overview of system implementation. The prototype used in the investigation uses a server with a single object containing a method called *foo*. The IDL contains a sequence definition and three interfaces, as shown in Figure 3-6.

The sequence *message_type* is defined as a one-dimensional array of octets. The length of the array varies according to its actual number of elements.

The *ServerObject* interface defines an object running on the server application. It has the following four operations:

- *foo()*

This operation represents the service request. It takes the message, the number of iterations in a for-loop, and the number of bytes to be returned as arguments, and returns a reply of type *message_type*. The number of iterations is used to control the CPU processing time requested by a client at the application level. The number of

bytes to be returned reflects the message size of the reply. These parameters are discussed in more detail in Section 4.2.

```
#ifndef _SERVER_LBS_IDL_
#define _SERVER_LBS_IDL_

module LBS
{
    typedef sequence<octet> message_type;
    interface ClientObject;

    interface ServerObject {
        message_type foo(in message_type message, in long long loop_count,
in long return_bytes);
        oneway void one_way_foo(in ClientObject client, in message_type
message, in long long loop_count, in long return_bytes);
        void start();
        double stop();
    };

    interface ClientObject {
        oneway void foo_done(in message_type message);
        void setParameters(in ClientObject client, in ServerObject
forward_server );
        void stop();
    };

    interface Balancer {
        typedef sequence<ServerObject> Servers;

        ServerObject get_forward_locator();
        Servers get_servers(in short desired, out short returned);
        void register_client(in string client_name, in ClientObject
client);
        void report_results_from_client(in string client_name, in long
calls, in long long total_time);
        void report_test_case_from_client(in string test_case);
    };
};

#endif
```

Figure 3-6: IDL Source Code of the System Interface

- *one_way_foo()*

This is a one-way call counterpart of the *foo* operation, to be used for the FS load balancing strategy (see Section 3.3). It returns void and contains a handle to the client,

so that the serving server can make contact with the client and return a response with the call *foo_done*, as defined in the *ClientObject* interface.

- *start()*

This operation is simply called by the balancer to inform the server of the beginning of request processing, so that it can start measuring CPU utilization.

- *stop()*

This is called by the balancer to inform the server of the end of request processing, so that the server can finish measuring CPU utilization.

The *ClientObject* interface defines an object running on the client application on behalf of the user. It has the following three operations:

- *foo_done()*

This is a one-way operation to be called by the servers for the FS load balancing strategy, indicating the completion of a *foo* request. It returns a reply of type *message_type* from the server.

- *setParameters()*

This is just a helper operation used by the client application to set the related parameters.

- *stop()*

This operation is called by the balancer to tell the client to stop making requests. This call signals the end of a testing run.

The *Balancer* interface defines an object running on the balancer application. It contains the following operations used to communicate with clients. It also defines a sequence of *Servers* that is an array of server handles.

- *get_forward_locator()*

This is called by clients to get the handle of the dummy server object on the balancer application for the FBA, FB and FS load balancing strategies (see Section 3.3).

- *get_servers()*

This operation is called by clients to get server handles for the client-based dispatching load balancing strategies.

- *register_client()*

This is called by clients to register themselves with the balancer so that the balancer can inform them of the end of the testing period.

- *report_results_from_client()*

This operation is called by clients to report their response time measurement results. Once a testing run is finished, all the results are collected by the balancer and written to a log file.

- *report_test_case_from_client()*

This is called by a client in the client application to report the test case parameters to the balancer. These parameters are used to construct file name for the log file that stores the measurement results. The file name also contains the timestamp of the file being created in order to make sure that each log file has a unique name. For each testing run, only one client reports the testing parameters to the balancer.

3.5.2 System Execution

The execution of the system involves many steps. First, we need to create a configuration domain in order to determine the type of configuration for the running of load balancing system being deployed; the ports used by the core services, such as the event and naming services; and the initial services to be deployed in the domain [ION-02b]. The host running the domain services is called domain server.

For this system, the domain services include the naming service and the event service. The configuration also creates a configuration file, which includes the host and the ports the other host can contact in order to gain access to the deployed services. Once this file is available to a host, it can then communicate with the domain server for different services. For instance, in this load balancing system prototype, the client application accesses the naming service using the configuration file in order to obtain the object reference for the balancer.

For the purpose of illustrating the execution of this load balancing system prototype, we present the execution scenario of the FBA strategy, as depicted in Figure 3-7. A dashed line with an arrow head represents a request invocation. All the clients reside on a single PC. Each of the domain server, the balancer, the NIST Net emulator, and the four servers is executed on an independent PC. Passing of a dashed line through a component indicates that the corresponding request goes through that component. For instance, the request labeled 1 corresponds to the balancer registration that goes through the NIST Net emulator, and arrives at the domain server. Note that each client, server, the balancer and the domain server interact with each other through Orbix 2000, the CORBA middleware.

The essential steps of the FBA execution scenario are explained.

Figure 3-7: FBA Execution Scenario

Step 1: After initialization, the balancer application creates a servant that is an implementation of the *Balancer* and registers it with the naming service in the domain server.

Step 2: The server programs are then started manually after the balancer. They contact the domain server to register themselves with the event service by providing the name and the handle of the server for each program. The event service will notify the balancer of the server registration through the event channel. A separate thread has been created in the balancer application to deal with this event notification.

Step 3: The client application is then started manually after the server programs. It contacts the domain server for the handle of the balancer. It also creates a thread of the *ControlThread*, which is responsible for creating a number of threads of the *CallingThread*. Each *CallingThread* represents a client making requests. The

ControlThead also controls the setting of various workload parameters, such as think time, bandwidth, message size, and so on.

Step 4: The client application then contacts the balancer to obtain a handle of the servant for a dummy server object (see Section 3.4), using the call *get_forward_locator* mentioned in the last section. The servant is created at the balancer application and is responsible for the forwarding of requests to servers and responses to clients. We call it the forwarding servant.

Step 5: Each client then requests a service by invoking the *foo* operation on the returned handle of the forwarding servant.

Step 6: The forwarding servant in the balancer application forwards the request to a chosen server, based on the number of allocated pending requests on each server, as explained in Section 3.3.

Step 7: After processing the request, the server returns the result to the balancer in response to the call in Step 6.

Step 8: The balancer forwards the result to the client in response to the call in Step 4.

The system execution of other strategies is similar to this FBA scenario; only the way in which requests are called and responses are returned differs. A combination of the above execution scenario and the explanation of the various load balancing strategies in Section 3.3 can provide a clear view of the system execution of other strategies.

Testing:

The system prototype has been tested thoroughly, and the testing results show that each strategy works properly. The process of testing involved creating log files, checking

the trace of request mapping for short runs manually, and verifying measurement results using the *Little's Law* that is described in Section 4.1.

Chapter 4 Performance Measurement and Instrumentation

Chapter 3 presented a discussion of the different load balancing strategies, along with the design and implementation of the performance prototype of the load balancing system. In this chapter, we present the performance metrics, the workload, the experimental environment, and the measurement accuracy.

4.1 Performance Metrics

For the performance measurement, we use a closed system model, in which the number of clients is fixed at each testing run and each client makes a request after receiving the response to the previous request. There is a think time between receiving a response and making another request, as mentioned in Section 3.4.

The following three performance metrics are used in the experiments.

- **Mean Response Time**

The client response time is the elapsed time measured in milliseconds (ms), from the moment when a client submits a request to the moment the client receives a corresponding response. In the HD (Handle Driven) case, the client response time includes the response time for getting a handle and the response time for the request processing. The mean response time is the average client response time, which is equal to the total response time divided by the total number of requests from all clients.

- **System Throughput**

System throughput is the number of requests processed in one second by the whole system. Throughput is used to measure the capacity of the system. It can be obtained from the mean response time using the *Little's Law*:

$$T = (NC * 1000) / (TT + R)$$

Where,

T: the system throughput in number of requests per second

NC: the number of clients

TT: client think time in millisecond (ms)

R: mean response time millisecond (ms)

- CPU Utilization

For each testing run, the server and the balancer utilizations were measured in order to gain more insights into system performance. For each test, the ORB in each host was multithreaded. Due to the multithreaded nature of the ORB, this utilization was measured for a host instead of a particular process, and it reflected the proportion of time for which the CPU of a server or the balancer was busy.

4.2 Workload

Workload reflects the characteristics of the system, as well as of the application. Different workload parameters have different degrees of impact on the overall system performance. In this thesis, we present the following workload and system parameters, which not only helped us in investigating the relative performances of the various load balancing strategies in wireless environments, but also allowed us to gain insight into the

fundamental differences in performance between the wired and the wireless environments for the different strategies.

Workload Parameters:

(1) Number of Clients (NC)

This workload parameter is used to control the number of clients making requests on the system. A client is implemented as a thread in the client application. The choice of this parameter influences the system load, and varies from one to the number that saturates the system, so that we can gain an overall picture of the system performance under different system load conditions.

(2) Mean Server Demand (SD)

This parameter is a mean value and refers to the CPU processing time of a server program for a request at the application level. It does not include processing time at the ORB and operating system levels, such as marshaling or un-marshaling time.

The use of exponential distribution reflects different types of services with different CPU processing times requested by clients. An increase in server demand makes the system more computation-bound. The use of this parameter is helpful in gaining insight into the impact of the request demand on system performance.

The server demand was controlled by the number of iterations in a for-loop, and followed an exponential distribution. The use of exponential distribution reflects different types of services with different CPU processing times requested by clients. The server demand for a specific invocation was generated in the following way. The number of iterations required to consume 25 ms of CPU time was determined offline. Using this information, the number of iterations required to generate a given SD was determined.

An exponentially distributed random variable was then generated with the computed number of iterations as the mean. The resulting output was cast into an integer and used as the number of iterations for the for-loop to be used for the given server invocation.

(3) Coefficient of Variation of Think Time (CV)

As mentioned in Section 3.4, after receiving a reply from the server, the client “thinks” for a certain period of time in order to model the user activity after receiving the reply from the server. Think time is assumed to be either exponentially or hyper-exponentially distributed. Given a mean think time, the coefficient of variation reflects the variability of think time, and as a consequence, the burstiness of the request arrival on the servers. The bigger the CV, the more variable are the think times. For the purposes of this thesis, we fixed the mean think time (TT) and changed the CV in order to investigate the impact of the arrival pattern of requests on system performance.

(4) Message Size (MS)

This refers to the size in bytes of the message that the client sends to the server in each request, as well as the size of the reply message from the server. It is a fixed value in each experiment. The message is an array of bytes, which is included in each request and reply. A substantial increase in message size is expected to make the system more communication-bound, as opposed to computation-bound, even though an increase in the size of the message also results in an increase in CPU computation time. This becomes more apparent in a wireless environment, where bandwidth is relatively low and the resulting increase of communication time becomes more significant than CPU time.

System Parameters:

(5) Mean Inter-node Communication Delay (ND)

The client nodes can be several nodes away from the server, and this is usually true in mobile and wireless environments, where intermediate nodes, such as the base station and gateway node, take part in the communication between clients and servers. In this thesis, we assume that the servers and the balancer reside on the same wired local area network, and ND represents the mean inter-node communication delay between the client and the balancer, or between the client and the servers. Such inter-node delay follows an exponential distribution in our experiments. There is no manually set delay between a server and the balancer. The impact of inter-node delay on performance is investigated in this thesis.

(6) Bandwidth (BW)

Bandwidth refers to the data transmission rate in a network system. Although it is increasing in wireless systems as technology advances, bandwidth for wireless devices is still relatively low compared to desktop systems. This lower bandwidth represents a fundamental difference between the wired and wireless systems.

(7) Drop Rate (DR)

Besides the low bandwidth, wireless systems also experience a loss of packets over the wireless links, due to transient random errors caused by channel fading, multi-path fading, interference or handoffs, or in the hosts where packets are dropped due to congestion. The drop rate reflects the degree of such loss. Changes in the drop rate in the experiments helped us to investigate its impact on performance.

4.3 Experimental Environment

Experiments were carried out in a network located at the Real-Time and Distributed Systems (RADS) lab in Carleton University. The network consists of eight PCs, as depicted in Figure 4-1. All the PCs are Intel Pentiums and run under Red Hat Linux, version 7.2. They are connected by a 100 Mbps Ethernet LAN. The two PCs, where the clients and the domain server resided, are Pentium IVs with a CPU speed of 2.40 GHz, and the other PCs are Pentium IIs, with a CPU speed of 265 MHz. All the clients ran on a single host, with each client modeled by a thread.

The network was configured so as to model different network environments. All packets coming from the client or the domain server host had to go through the wireless emulator host before reaching the balancer or the servers. Likewise, all packets coming from the balancer or the servers had to go through the network emulator host before reaching the client or the domain server host. The client host and domain server could communicate directly without going through the wireless emulator, and the same was true for the communication between the servers and the balancer. Note that the communications between the clients and the domain server and between the servers and the domain server do not have any influence on the relative performances of the load balancing strategies. This is because the domain server is used by the clients to get the balancer handle through the naming service, and by the servers to contact the balancer through the event service in the initialization phase only. The performance results reported are based on measurement made after this phase of operation.

The network emulator host ran the NIST Net network emulation package [NAT-02] and acted as a router to emulate a wide variety of network conditions. With NIST Net,

bandwidth, packet delay, drop rate and other parameters between clients and the balancer or servers could be set, and the various wired or wireless network environments could be emulated for the purposes of our testing.

Figure 4-1: Experimental Network Environment

4.4 Measurement Accuracy

Even though the network was dedicated to running our testing and there was little interference from other machines, due to the random nature of the simulated workload parameters, such as exponentially distributed think time, server demand, and randomness provided by the network emulator, we found that the testing results still varied from one

testing run to another, by different degrees. Therefore, we repeated each experiment with a given set of parameters a number of times, and statistics were gathered to compute confidence intervals for performance results. For the performance result of every experiment of interest, we achieved an interval that was less than $\pm 5\%$ of the mean, at a confidence level of 95%.

The overhead of processing a packet introduced by the network emulator NIST Net was observed to be in the range of 10 to 30 microseconds. Since the response times are usually more than 100 milliseconds, such an overhead has little impact on the accuracy of the response time measurements made in the experiments.

Chapter 5 Results of Experiments

5.1 Introduction

Several sets of experiments were run in order to investigate the effects of various workload parameters on the relative performances of different load balancing strategies. As described in Section 4.2, the workload parameters include message size, bandwidth, inter-node delay, drop rate, server demand, and the coefficient of variation of think time. A factor-at-a-time approach was adopted for each experiment, which was aimed at investigating the impact of the underlying factor on system performance. That is, one of the factors was varied, while others were held constant.

The different levels of the factors used to generate the workload in the experiments are listed in Table 5-1. The number and the choice of the various levels for each factor reflect a compromise between representativeness and an acceptable run time for the experiments. Except in the experiment focusing on the effect of bandwidth on performance, where a message size of 5000 bytes was taken as the default level, the underlined levels for different factors in this table represent the default values that were held constant during an experiment that focused on the effect of another factor on performance. Note that the mean think time TT was fixed at 300 ms in all the experiments.

When considering the relative performances of load balancing strategies, we focus mainly on the two components of a particular load balancing strategy: the request mapping policy and the interaction pattern (see Section 3.3), which influence overall system performance. There are three different interaction patterns involved in the six load

balancing strategies, namely: forwarding, handle driven and client-based dispatching. Two types of request mapping policies are used. The FBA strategy uses the number of active (or pending) requests in server as the load index in mapping each client request to a server, whereas all other five strategies use the round-robin algorithm as a request mapping policy.

Table 5-1: Workload Parameters

Factor	Levels
MS (bytes)	<u>100</u> , 5000, 7500
BW (Mbps)	1, 100
ND (ms)	0, <u>15</u> , 30
DR (%)	<u>0</u> , 2, 5
SD (ms)	50, <u>100</u> , 150
CV	1.0, <u>1.5</u> , 2.0
TT (ms)	<u>300</u>

In each test, we measured the utilizations of the balancer and each server, and calculated the average server utilization. These measurements not only helped us understand the overall system load condition, but also helped us gain more insight into the causes of performance changes. For example, with the same number of clients, a higher value of average server utilization reflects a higher performance.

For each experiment, we divide our discussions into two parts, based on the system load conditions. Unless indicated otherwise, we consider the system to be in low load conditions when the number of clients is less than or equal to 4, and in medium to high load conditions when it exceeds 4.

The test using all default levels, shown in Table 5-1, provides a base case for us to consider the effects of most workload factors, and is referred to as “base test”. The

relative performances of the six load balancing strategies in the base test will be discussed in Section 5.2. Sections 5.3 to 5.7 will contain analyses of the effects of different workload factors on the relative performances of the various load balancing strategies.

5.2 Relative Performances of the Load Balancing Strategies for the Base Test

The workload parameters for the base test are listed in Table 5-2. All clients shared a single wireless link, which had a bandwidth of 1 Mega bits per second (Mbps), and provided a communication channel between the clients and the balancer or servers. The balancer and the servers were assumed to be in a wired local area network with bandwidth of 100 Mbps. The bandwidths of both the wired and wireless networks were fixed throughout the experiment. The number of clients varied from 1 to 24 in this test, and the number of servers was fixed at 4.

Table 5-2: Workload Parameters of the Base Test

Factor	Levels
MS (bytes)	100
BW (Mbps)	1
ND (ms)	15
DR (%)	0
SD(ms)	100
CV	1.5

The message sizes for service requests and replies were the same, and were fixed at 100 bytes. The drop rate was set to zero, whereas the mean value for exponentially

distributed inter-node delay between the clients and the balancer or servers was set to 15 milliseconds in the both directions. The think time is hyper-exponentially distributed with a mean of 300 milliseconds and a coefficient of variation of 1.5.

The mean response times for the different load balancing strategies for the base test are shown in Table 5-3 and Figure 5-1.

Table 5-3: Response Time Measurement (ms) for the Base Test

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	190	210	264	349	454	580	735
FS	189	208	268	355	464	580	720
HD	216	224	275	364	466	587	722
CBD2	184	213	282	376	498	625	766
CBD1	184	187	246	318	401	506	629
FBA	189	200	231	286	361	466	618

Figure 5-1: Response Time Measurement for the Base Test

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

The observations under different load conditions are discussed next.

Low Load Conditions:

As we can see from Figure 5-1, when the number of clients (NC) is equal to 1, all the strategies but HD have similar response times. HD exhibits the worst performance, resulting from its interaction pattern. That is, as described in Section 3.3, a request in HD needs to traverse the wireless link twice. As a result, the client response time increases, due to an extra inter-node communication delay.

Each of the other five strategies performs comparably when NC is equal to 1. In particular, we noticed that the mean response times in forwarding-based strategies, namely FB, FS and FBA, are approximately 5 milliseconds higher than that in client-based dispatching strategies. This is due to the additional processing of requests in the balancer for FB, FS and FBA.

As NC reaches 4, FBA demonstrates a slightly better performance than FB or FS due to its more effective request mapping policy in comparison to round-robin. There are 4 clients and 4 servers in the system. Thus with FBA, at most one request on a server can be allocated at any time, since the number of requests being and to be processed at any point in time is 4 at the most. In contrast, with FS or FB, due to the use of a round-robin policy and the difference in server demands and think times from one request to another, a request may be dispatched to a busy server, which is currently processing a request, instead of an idle server in the system. As a result, one server may be processing two requests while another server is idle. For instance, in Figure 5-2, Server 1 may be processing two requests while Server 3 is idle. This load imbalance contributes to an

increase in mean response time of approximately 5% with regard to FBA strategy, when NC is equal to 4 in this base test.

Figure 5-2: Request Forwarding When NC=4

When NC equals 4, while HD remains the worst performer due to an extra round trip on the wireless link, CBD1 performs the best. In addition, the response time for CBD1 remains almost the same as in the case when NC equals 1. This is because each client is allocated a separate server, and one server only processes one client's requests. Therefore, as in FBA, a server processes at most one request at any time. In this case, the CBD1's shorter response time over FBA results purely from the fact that the clients invoke the servers directly, instead of going through the balancer.

Another interesting observation is that CBD2 performs much worse than CBD1. The main reason for this degradation in performance is explained in Figure 5-3, in which each client is allocated two handles when NC is equal to 4. Due to the randomness of server demand, think time and inter-node delay, a server may be processing two requests from two clients while another server is totally idle. For instance, as shown in Figure 5-3, Server 1 may be processing two requests from Client 1 and Client 3, while Server 2 is idle. Such a load imbalance accounts for an increase in response time of about 12%, compared with CBD1 when NC equals 4 (see Table 5-3). We also observe that CBD2 has a higher mean response time than FB or FS at NC of 4.

Figure 5-3: Client-Based Dispatching with Two Handles (CBD2) When NC=4

Medium to High Load Conditions:

As the load increases and NC exceeds 4, we observe that FBA performs the best, and CBD1 performs the second best. The next three strategies, FB, FS and HD, performed similarly. CBD2 performs the worst of all.

The most dominant factor influencing performance is believed to be the degree of effectiveness of load balancing among the servers exhibited in each strategy in this test. More balanced load among different servers produces faster client response times, which in turn result in a higher rate of request arrival on servers. Thus, a higher average server utilization indicates the effectiveness of a load balancing strategy. Table 5-4 shows an example of the average server utilizations in the base test, when NC is equal to 16.

Table 5-4: Average Server Utilizations in the Base Test When NC=16

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Strategy	FB	FS	HD	CBD2	CBD1	FBA
Utilization	0.81	0.80	0.79	0.76	0.87	0.90

In this table, FBA has the highest server utilization of 0.90. It takes each server into account when mapping each request to a server based on the number of requests being processed in each server. Although it works as a static load balancing algorithm, in the

sense that there is no explicit server load information exchange between the balancer and servers, it uses the advantage of the forwarding interaction pattern to gain server load information by maintaining a list that contains the number of pending requests for each server at run time.

The benefit of a request mapping policy that is based on the number of pending requests in servers in a CORBA-based load balancing system can be illustrated through a direct comparison of FB and FBA. The percentage improvement of FBA over FB in terms of mean response time is plotted along the Y-axis, while the number of clients is plotted along the X-axis of Figure 5-4.

Figure 5-4: Response Time Difference between FBA and FB

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

This figure shows that FBA achieves a maximum improvement of 20.5% when NC is equal to 16. With the increase of NC from 1 to 16, the improvement is increased, due to its effective load balancing policy. On the other hand, with the increase of NC from 16 to 24, improvement decreases as the system goes near saturation.

Note that in this test, the numbers of clients other than 1 are multiples of the number of the servers. In CBD1, the load is fairly balanced since each server processes requests from the same number of clients when NC is equal to or greater than 4. However, in comparison to FBA, once a client is allocated a server, it can only make requests on that particular server throughout the session. At some points, when a client begins to invoke an operation, the allocated server may be busy serving other clients, while other servers may be under-loaded or idle, but it can only invoke that server anyway. Therefore, although it has an advantage in that it can make invocations directly on the server without having to go through the balancer, it lacks the flexibility of being able to look at all the servers and make request mapping decisions dynamically, as in FBA.

As shown in Figure 5-5, the response time improvement of FBA over CBD1 reaches as high as 10.1 % when NC equals 12. As NC increases from 12 to 24, this improvement decreases. For instance, when NC is equal to 24, improvement is only 1.7%. The average server utilizations in CBD1 and FBA are 0.97 and 0.99, respectively, with an NC of 24. Such high server utilizations reveal that the system is saturated at the servers for both strategies. As a result, the performance difference among the strategies diminishes.

As stated above, CBD2 allocates two servers to each client, and each client makes requests on the allocated two servers in a round-robin fashion. When the number of clients increases, the load becomes less and less balanced, with a greater possibility that more and more clients can make requests on the same server while other servers may well be under-loaded. This reverse impact on performance due to load imbalance can be seen in Figure 5-1, as well as Figure 5-6, which shows a direct comparison between CBD2 and CBD1.

Figure 5-5: Response Time Difference between FBA and CBD1

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Figure 5-6: Response Time Degradation of CBD2 from CBD1

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

As is evident from this figure, the performance degradation is as high as 24.2% when NC reaches 16, as a result of less effective load balancing strategy in terms of mean response time. It also shows that the degree of degradation increases with the increase of NC from 1 to 16, and decreases slightly with the increase of NC from 16 to 24.

FB, FS, and HD perform similarly in medium to high conditions, since they share the same request mapping mechanism in the balancer. That is, the balancer uses the same

static round-robin algorithm to dispatch requests in these three strategies. This indicates that the effect of different interaction patterns used by these strategies have a smaller impact on performance at medium to high load.

The corresponding throughput measurement for this test is shown in Appendix A. As expected, the ranking of throughputs for the different load balancing strategies follows the same order as that of mean response times.

From the discussions concerning the base test under various load conditions, we understand that the load balancing strategies have a profound impact on performance. The effects of workload factors on performance are presented in the next few sections in this chapter.

5.3 Effect of Message Size

The size of the message corresponding to the client request and the server response is expected to vary with the type of request. In this experiment, we intended to investigate the effect of message size, MS, on the relative performances of the six load balancing strategies.

The various parameters used in this experiment are shown in Table 5-5. The message sizes are 100, 5000, and 7500 bytes. All other parameters are the same as in the base test. The wireless bandwidth was set at 1 Mbps, and the drop rate was set to zero.

The choice of message size is due to two reasons. On one hand, given the amounts of the server demand and the round trip delays in a request invocation path, a small difference in message size is not expected to create a significant difference in client

response time. On the other hand, as the number of clients increases, an overly large message could make network bandwidth a performance bottleneck.

Table 5-5: Parameters of the Experiment Investigating the Effect of Message Size

Factor	Levels
MS (bytes)	100, 5000, 7500
BW (Mbps)	1
ND (ms)	15
DR (%)	0
SD (ms)	100
CV	1.5

The mean response time for the test with a message size of 5000 bytes is shown in Table 5-6 and Figure 5-7. In comparison with the results obtained from the base test discussed in Section 5.2, we present our observations and discussions as the message size increases to 5000 bytes from 100 bytes under various load conditions.

Low Load Conditions:

With NC equal to 1, a comparison of response time measurements is shown in Table 5-7 between tests with message sizes of 100 bytes and 5000 bytes.

As shown in this table, with an MS of 5000 bytes, FB and FBA exhibit the highest increase in response time. The interaction pattern is believed to be the primary cause of the differing degrees of increase shown in this table. FB and FBA experience the highest response time increase because they share the same interaction pattern, in which all the requests and replies need to go through the balancer and the request demand on the balancer becomes higher when the MS increases from 100 to 5000 bytes. Thus, an

increase in the request demands on the balancer in FB and FBA leads to a higher increase in response time, compared to other strategies with different interaction patterns.

Table 5-6: Response Time Measurement (ms) for the Test with MS of 5000 Bytes

BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	338	331	370	448	561	685	852
FS	257	259	334	423	557	706	873
HD	318	349	402	517	650	805	970
CBD2	293	341	416	497	617	739	901
CBD1	295	295	359	444	560	674	834
FBA	328	328	357	406	509	624	786

Figure 5-7: Response Time Measurement for the Test with MS of 5000 Bytes

BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Looking at the measurement results shown in Figure 5-7, it is evident that FS is the best performer under low load conditions. As discussed in Section 3.3, the communications in FS are all one-way invocations. Research to evaluate the relative performances of one-way and two-way calls has been conducted at University of Virginia

[VIR-99], and is also described in Tao's thesis [TAO-02]. The results of the research show that a one-way call is much faster than a two-way call. For example, in Orbix 3.0, a one-way call is more than four times faster than a two-way call when the operation contains no parameter and a void returning result.

Table 5-7: Mean Response Time (ms) with MS of 100 and 5000 Bytes When NC=1
BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Strategy	FB	FS	HD	CBD2	CBD1	FBA
MS=100	190	189	216	184	184	189
MS=5000	338	257	318	286	295	328
Increase over MS=100	78%	36%	47%	55%	60%	74%

Medium to High Load Conditions:

When NC exceeds 8, we observe that FBA performs the best in all strategies. HD performs the worst in this test instead of CBD2 that is the worst performer in the base test with MS of 100 bytes (see Figure 5-1). FBA performs favorably as a result of the best-balanced load in the servers, which is reflected in an example of average server utilizations shown in Table 5-8, when NC is equal to 16.

Table 5-8: Average Server Utilizations of Test with MS of 5000 Bytes When NC=16
BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Strategy	FB	FS	HD	CBD2	CBD1	FBA
Utilization	0.70	0.71	0.63	0.65	0.71	0.75

At higher system loads, the benefit of load balancing in servers achieved by FBA seems to offset the extra request demand in the balancer, and FBA even outperforms CBD1, which is the second best in the medium to high load conditions.

The degraded performance in HD is caused primarily by its request interaction pattern described in Section 3.3. One extra round-trip over the wireless link results not only in extra inter-node communication delay, but also in a significant message transmission time over the wireless link. This is because as the message size and the number of clients increase, all clients are competing against each other for the limited bandwidth on the wireless link. As a result, performance degrades.

We also observe that the relative performance of FS degrades as the number of clients increases. This is because as the system load increases, the impact of the request mapping policy has more effect on the overall response time than the fast one-way call. As a result, FBA that uses a more sophisticated mapping policy performs better than FS.

The corresponding throughput measurement for this test is shown in Appendix B. As expected, the ranking of throughputs for the different load balancing strategies follows the same order as that of mean response times.

The results of the test with an MS of 7500 bytes are shown in Table 5-9. The small difference in mean response time among the various strategies with NC of 24 is a result of the fact that wireless bandwidth forms a performance bottleneck. Observations on NIST Net indicated that utilized active bandwidth in the test was close to the available bandwidth of 1 Mbps.

Table 5-9: Response Time Measurement (ms) for the Test with MS of 7500 Bytes

BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

NC	1	12	24
FB	367	593	1328
FS	343	623	1370
HD	346	770	1462
CBD2	315	664	1273
CBD1	312	626	1259
FBA	368	562	1304

Figure 5 – 8 shows the response time measurement of different strategies at NC of 12 with three message sizes. It is evident that FBA remains to be the best performer under the deferent message sizes, while HD performs the worst when the message size is high (5000 or 7500 bytes).

Figure 5-8: Response Time Measurement for Each Strategy with Different Message Sizes When NC=12

BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Figure 5-9 shows the response time measurement for FBA, with different message sizes and NC of 1, 12 and 24. Note that degree of increase in response time as a result of message size increase from 5000 to 7500 bytes increases with the increase in NC. For instance, as the message size increases from 5000 to 7500 bytes at NC of 24, performance becomes much worse in comparison to the case with NC of 12. A similar observation is made with the other load balancing strategies.

Figure 5-9: Response Time Measurement for FBA with Different Message Sizes and Numbers of Clients

BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

5.4 Effect of Bandwidth

As stated in Section 4.2, bandwidth is one of the most fundamental factors distinguishing wireless systems from wired systems. This experiment is designed to investigate the effect of bandwidth on the relative performances of various strategies focusing in particular on the performance difference between wired and wireless environments. The parameters used in this experiment are shown in Table 5-10.

Since bandwidth is expected to have a small effect on response time when the message is of a small size, we used a message size of 5000 bytes rather than 100 bytes in order to understand the effect of bandwidth on performance.

With a BW of 100 Mbps, the system represents a wired local area network environment. The test results with a BW of 1 Mbps were presented and discussed in

Section 5.3, and the test results with a BW of 100 Mbps are shown in Table 5-11 and Figure 5-10. Based on these two sets of test results, the effect of bandwidth on the performances of different strategies is discussed.

Table 5-10: Parameters of the Experiment Investigating the Effect of Bandwidth

Factor	Levels
MS (bytes)	5000
BW (Mbps)	1, 100
ND (ms)	15
DR (%)	0
SD (ms)	100
CV	1.5

Low Load Conditions:

In Figure 5 – 10, the relative performances of the strategies are similar to those captured in Figure 5-7. Due to its faster one-way call, FS remains the best in both cases. Under low load conditions, the performance of CBD2 degrades when the number of clients increases from 1 to 4 in comparison to CBD1, because the load is less balanced on the four servers.

The main reason for the similarities in relative performance is that, under low load conditions, there is a minimum degree of competition for bandwidth between the clients and other system components, such as the balancer and the servers. A change of bandwidth thus does not have much impact on the response time, given that the transmission time over the link is relatively small compared to the other constituents of client response time, such as inter-node delay and server demand.

Medium to High Load Conditions:

As the load increases from low to medium and high, among others, two interesting changes can be observed, as noted in Figure 5-10 and Figure 5-7. First, in medium to high load conditions, CBD1 performs the best in the wired environment, whereas FBA performs the best in the wireless environment. Second, FB performs the worst in the wired environment, whereas HD performs the worst in the wireless environment.

Table 5-11: Response Time Measurement (ms) for the Test with BW of 100 Mbps

MS=5000 bytes, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	318	327	352	424	529	649	815
FS	240	253	301	380	489	616	759
HD	312	315	350	424	514	616	729
CBD2	280	307	360	425	515	631	790
CBD1	271	277	310	369	446	537	663
FBA	318	319	335	379	467	597	749

Figure 5-10: Response Time Measurement for the Test with BW of 100 Mbps

MS=5000 bytes, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Note that due to a larger number of messages used per request in FBA, and because of the potential heavy load at its balancer that forwards all the requests and replies, FBA demonstrates an inferior performance in comparison to CBD1 in a wired environment (BW=1000 Mbps). However, in a wireless environment (BW=1 Mbps), the servers in CBD1 spend more time in communication in comparison to a wired environment. Note that this does not occur in FBA since the balancer and servers are connected by a high speed wired link. As a result, the performance of FBA is higher than that of CBD1.

An explanation for the second change in observation is presented. Note that both FB and HD use 4 messages per request. However, the balancer in FB is involved in all message transmissions and can experience a high load as NC increases. As a result, FB performs worse than HD in a wired environment. Although the number of messages used in both strategies is the same, the numbers of messages transmitted in HD and FB on slower links in a wireless environment are 4 and 2, respectively. The additional message transmission delay offsets the performance advantage of HD described earlier, and it demonstrates an inferior performance.

The overall effect of bandwidth on performance is further shown in Figure 5-11, where percentages of response time increase in the wireless environment over the wired are plotted along the Y-axis.

For the three per-request round-robin-based load balancing strategies, FB, FS and HD, the changes in the wireless environment vary to a high degree under medium to high load conditions. HD exhibits the largest increase, and FS comes in second, while FB shows the smallest degree of increase.

Figure 5-11: Mean Response Time Increase with BW of 1 Mbps over BW of 100 Mbps

MS=5000 bytes, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

As explained in Section 3.3, each request in HD requires one extra round-trip over the wireless link. This extra trip results not only in an extra inter-node communication delay, but also more message transmission time over the wireless link where clients are competing for limited bandwidth. The impact of such interaction pattern on mean response time in HD is expected to be more profound as message size increases.

Next, we discuss the interaction patterns of FS and FB. These two patterns only differ with regard to the way in which replies are returned. In FS, the servers send the replies directly to the clients. Although the replies are sent as one-way invocations, the operating system and the low-level software system are still responsible for the delivery of replies to clients, and this delivery ties up the underlying CPU and other resources. With the

increase on system load and server utilization, it is expected that there will be some deterioration on the overall response time.

In contrast, in FB, servers are only responsible for sending replies to the balancer located in the same local network. Replies are delivered by the balancer to the clients over the wireless link. The balancer takes some load from the servers in FB compared to FS. This sharing of the load by the balancer is believed to produce a better performance than FS. Therefore, as is evident in Figure 5-11, there is an increase of around 5% in FB, but an increase of approximately 10-15% in FS, when NC reaches 8 and above.

In the client-based dispatching strategies, each server is responsible for the delivery of replies back to clients. As a result, they also share high degree of increase in response time as FS. In particular, CBD2 shows very similar degree of increase as FS, as shown in Figure 5 – 11.

As the bandwidth decreases, the message transmission overhead placed on the underlying host increases. The unique interaction pattern exhibited in FB and FBA releases the servers from carrying out this heavy message transmission over the low bandwidth link. As a result, the servers become less bottlenecked. FBA shares similar degree of increase as FB when NC reaches 8 and above.

The corresponding throughput measurement for this wired test is shown in Appendix C. As expected, the ranking of throughputs for the different load balancing strategies follows the same order as that of mean response times.

5.5 Effect of Inter-node Delay

The client, server and balancer may be separated by several intermediate nodes. As mentioned in Section 4.2, we assume that the servers and the balancer are located in the

same local area network, while the clients are connected through wireless links to the balancer and the servers. Inter-node delay refers to the delay on the intermediate nodes experienced by a client request from the client to a server or the balancer, or experienced by a server response from the server to a client. This delay does not include the transmission time over the wireless link, and it follows exponential distribution to reflect the randomness in a network environment.

The parameters used in this experiment are shown in Table 5-12. The inter-node delays are distributed exponentially with mean values of 0, 15, and 30 milliseconds.

Table 5-12: Parameters of the Experiment Investigating the Effect of Inter-node Delay

Factor	Levels
MS (bytes)	100
BW (Mbps)	1
ND (ms)	0, 15, 30
DR (%)	0
SD (ms)	100
CV	1.5

As mentioned above, the test using an ND (inter-node delay) of 15 milliseconds is referred to as the base test and has been presented in Section 5.2. To investigate the effect of inter-node delay under various load conditions, we present the complete measurement results of the test with an ND of 0 in Table 5-13 and Figure 5-12.

Low Load Conditions:

Figure 5-12 shows an interesting change under the low load conditions compared to the response time results shown in Figure 5-1 that corresponds to the base test. That is, HD is no longer the worst performer as in the base test with ND of 15 ms at NC of 1 and

4. Note that each request experiences 4 inter-node delays in HD while it experiences only 2 inter-node delays in other strategies. As ND decreases from 15 ms in the base test to 0 ms in this test, such a performance disadvantage in HD diminishes. As a result, ND is no longer the worst performer at NC of 1 and 4 with ND of 0 ms.

Table 5-13: Response Time Measurement (ms) for the Test with ND of 0 ms

MS=100 bytes, BW=1 Mbps, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	156	168	221	302	426	576	729
FS	159	166	223	310	423	574	716
HD	157	165	222	312	409	561	709
CBD2	155	209	278	362	479	612	749
CBD1	151	151	209	283	378	499	615
FBA	154	158	182	244	348	471	600

Figure 5-12: Response Time Measurement for the Test with ND of 0 ms

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

We also observe that, at NC of 1, all strategies perform comparably. CBD2 performs the worst at NC of 4. One of the possible reasons is that in CBD2, two requests can be directed to the same servers when another server is idle.

Medium to High Load Conditions:

When NC exceeds 4, the relative performances of the six load balancing strategies remain similar to the base test with an ND of 15 milliseconds.

The corresponding throughput measurement for this test is shown in Appendix D. As expected, the ranking of throughputs for the different load balancing strategies follows the same order as that of mean response times.

In order to better understand the effect of inter-node delay, we performed another test using an ND of 30 milliseconds. The results are shown in Table 5-14. We can observe that FBA performs the best at NC of 12 and 24.

Table 5-14: Response Time Measurement (ms) for the Test with ND of 30 ms

MS=100 bytes, BW=1 Mbps, DR=0%, SD=100 ms, CV=1.5

NC	1	12	24
FB	215	396	732
FS	216	406	752
HD	279	442	755
CBD2	218	406	786
CBD1	214	349	646
FBA	217	340	627

In addition, Figure 5 – 13 shows the mean response times for each strategy with different inter-node delays at NC of 12. It shows that FBA and CBD1 perform the best and the second best, respectively, with different inter-node delays.

Figure 5-13: Response Time Measurement for Each Strategy with Different Inter-Node Delays When NC=12

MS=100 bytes, BW=1 Mbps, DR=0%, SD=100 ms, CV=1.5

Figure 5 – 14 shows the response time measurement for FBA, with different inter-node delays and NC of 1, 12 and 24. Note that the mean response time increases with the increase in the inter-node delay. The degree of increase at NC of 24 is less than that at NC of 1 and 12. A possible reason for this is that the servers are more heavily utilized at NC of 24, and ND has less impact on the mean response time. A similar observation is made with the other load balancing strategies.

Figure 5-14: Response Time Measurement for FBA with Different Inter-node Delays and Numbers of Clients

MS=100 bytes, BW=1 Mbps, DR=0%, SD=100 ms, CV=1.5

5.6 Effect of Drop Rate

In a wired network environment, loss of packets occurs mainly due to network congestion, and the packet drop rate is extremely low. In contrast, as stated in Section 4.2, due to transient random errors over wireless links, the packet drop rate can be relatively higher in a wireless network environment. This experiment is designed to investigate the relative performances of the six load balancing strategies under different packet drop rates, in order to understand the effect of drop rate.

The parameters used in this experiment are shown in Table 5-15. The range of the drop rate, from 0% to 5%, is consistent with the values used in the research conducted by

Wang et al. [WAN-02]. The drop rate of 5% also reflects the existence of lossy wireless links in some real mobile telecommunication systems.

With an approach similar to that taken in the last few sections, we look into the effects of packet drop rate on system performance, and discuss the results of tests with DR of 0% and 2%. The former test is in fact the base test, the results of which have been presented and discussed in Section 5.2. The response time measurement results for the later test are shown in Table 5-16 and Figure 5-15.

Table 5-15: Parameters of the Experiment Investigating the Effect of Drop Rate

Factor	Levels
MS (bytes)	100
BW (Mbps)	1
ND (ms)	15
DR (%)	0, 2, 5
SD (ms)	100
CV	1.5

Loss of packets causes an increase in the client response time, since retransmitting the lost packets takes additional time. In addition, such an increase in transmission time is believed to be much higher in wireless networks than in wired networks, when Transport Control Protocol (TCP) is used as the transport layer protocol. TCP was designed for wired network environments and it performs well over such networks by adapting end-to-end delays and packet losses caused by congestion. However, in wireless networks, a packet drop due to random transmission errors is treated the same way as network congestion in TCP. This results in performance being degraded, since the source host has to wait for an extended timeout period to retransmit the lost packet, while it should have

done it immediately. Recently a great deal of research has focused on such issues, and various solutions have been proposed [TSA-02].

Table 5-16: Response Time Measurement (ms) for Test with DR of 2%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	197	228	291	377	483	600	723
FS	199	232	301	389	490	607	758
HD	236	261	326	408	509	620	750
CBD2	192	227	301	399	502	626	774
CBD1	194	193	257	323	405	514	632
FBA	197	222	265	316	394	508	648

Figure Error! No text of specified style in document.-1

Figure 5-15: Response Time Measurement for Test with DR of 2%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

A response time comparison of the results from the tests with drop rates of 0% and 2% is depicted in Figure 5-16.

Figure 5-16: Mean Response Time Increase of the Test with DR of 2% over 0%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

This figure shows that even though the drop rate is increased to 2% only, its effect on the performances of some strategies can be significant. For example, a 2% increase in the drop rate in the wireless environment results in an increase in response time of about 18% for HD when NC equals 8. A discussion of the results under different load conditions is presented next.

Low Load Conditions:

When NC equals 1, there is an increase of 5% in response time for all strategies except HD, which shows an increase of about 10%. This larger increase in HD results from its interaction pattern. We know that it requires one more round trip over the wireless link than the other strategies, and as a result, the probability of losing packets associated with each request becomes relatively higher. Therefore, it results in a higher increase in the overall mean response time.

With the increase of NC from 1 to 4, we observe that the increase in client-based dispatching strategies is lower than that of the per-request based strategies. This is due to their differing interaction patterns. In FBA and FB, the balancer is responsible for the transmission of all the replies coming from the servers to all the clients over the wireless link. The loss of one packet may inevitably delay the transmission of other packets waiting in the queue to potentially different clients. However, in CBD1 or CBD2, each server is responsible for transmitting its own reply to its corresponding clients over the wireless link, and the loss of one packet is expected to delay fewer packets waiting in the queue, since there are four servers in total. As a result, packet loss has a lower impact on performance in CBD1 or CBD2 than that in FB or FBA.

Medium to High Load Conditions:

As the number of clients increases from 4 to 8, the increase in response time in CBD1 and CBD2 remains around 5%, while the others achieve the highest degree of increase in response time. In particular, as mentioned earlier, HD shows an increase of about 18%, while FBA demonstrates an increase of 15%. However, as the number of clients continues to increase, the increase in mean response time tends to become lower and lower. With the increase in the system load, the servers become more saturated, and the drop rate seems to have a lower impact on performance.

Figure 5-16 also shows a significant difference between CBD1 and FBA, in terms of the increase in response time. When NC is between 12 and 24, CBD1 shows an increase of lower than 2%, but FBA shows an increase of around 10%. As a result, CBD1 performs comparably with FBA. As explained earlier in this section, this is due to the lower impact of packet loss on performance of CBD1 than that of FBA.

To further illustrate the effect of drop rate, we also ran another test in this experiment with a drop rate of 5%. The results of the response time measurement are shown in Table 5-17 and Figure 5-17. A greater degree of degradation in FB and FBA can be observed. Meanwhile, notice also that CBD1 even outperforms FBA under various load conditions when the drop rate increases to 5%.

Table 5-17: Response Time Measurement (ms) for the Test with DR of 5%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	214	266	344	428	519	638	773
FS	221	290	389	485	583	687	811
HD	275	318	400	496	592	694	811
CBD2	212	258	324	414	535	655	783
CBD1	212	214	278	347	442	542	663
FBA	212	264	319	384	461	562	678

Figure 5-17: Response Time Measurement for the Test with DR of 5%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

The corresponding throughput measurement in this experiment is shown in Appendix E. As expected, the ranking of throughputs for the different load balancing strategies follows the same order as that of mean response times.

In addition, Figure 5 – 18 shows mean response times for each strategy with different drop rates when NC equals 12. FBA performs the best with a DR of 0%, and performs comparably with CBD2 with a DR of 2%. But with a DR of 5%, FBA is outperformed by CBD1 and becomes the second best performer. The change in the ranking of FBA and CBD1 is believed to be caused by the higher impact of drop rate on performance in FBA than in CBD1. High increases in response time for FS and HD can also be observed when DR increases from 2 to 5%.

Figure 5-18: Response Time Measurement for Each Strategy with Different Drop Rates When NC=12

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

Figure 5 – 19 shows the response time measurement for FBA, with different drop rates and NC of 1, 12 and 24. Note that the mean response time increases with the increase of the drop rate. In particular, the degree of increase at NC of 12 in response time is higher than that at NC of 1 or 24. A similar observation is made with the other load balancing strategies.

Figure 5-19: Response Time Measurement for FBA with Different Drop Rates and Numbers of Clients

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

5.7 Effect of Server Demand

Server demand refers to the computational demand of the client's request on a server. An increase in server demand makes client requests more computation-bound, as opposed to communication-bound. This experiment is designed to investigate the effect of server

demand on the relative performances of various load balancing strategies in a wireless environment. The main parameters used in this experiment are listed in Table 5 – 18.

The mean server demands (SD) are 50, 100 and 150 milliseconds, and the server demands are exponentially distributed. The test using an SD of 100 milliseconds is the base test, and was discussed in Section 5.2. The response time measurement results of the test using an SD of 50 milliseconds are presented in Table 5-19 and Figure 5-20. Note that a different set of NC was used in this test in order to fully depict the system performance in various conditions (see Table 5-19). The system is considered to be in low load conditions when NC was less than or equal to 8, and in medium to high conditions when NC exceeded 8. Observations and discussions follow.

Table 5-18: Parameters of the Experiment Investigating the Effect of Server Demand

Factor	Levels
Message Size (bytes)	100
Bandwidth (Mbps)	1
Inter-node Delay (ms)	15
Drop Rate (%)	0
Server Demand (ms)	50, 100, 150
Coefficient of Variation	1.5

Low Load Conditions:

By comparing Figure 5-1 and Figure 5-20, we observe that, compared to other strategies, HD's performance degrades more under the low load conditions when the server demand is decreased from 100 to 50 milliseconds. This is because each request in HD needs to traverse the wireless link twice, and increase of ND can result in relatively larger increase in the overall response time.

CBD1 remains the best performer, and the relative performances of the other strategies remain similar under low load conditions.

Table 5-19: Response Time Measurement (ms) for the Test with SD of 50 ms

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, CV=1.5

NC	1	8	16	24	32	40
FB	110	150	196	279	390	535
FS	111	156	209	283	405	548
HD	143	170	229	310	402	535
CBD2	108	142	200	299	410	543
CBD1	107	128	176	245	348	482
FBA	110	143	174	233	330	465

Figure 5-20: Response Time Measurement for the Test with SD of 50 ms

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, CV=1.5

Medium to High Load Conditions:

As the load increases to medium and high, we observe that FBA remains the best performer due to its superior load balancing mechanism. However, the difference

between CBD1 and FBA decreases as server demand decreases. For instance, NC=20 and NC=32 in Figure 5-1 and Figure 5-20 respectively give rise to similar average server utilizations. The response time of CBD1 is 8.6% higher than that of FBA with an SD of 100 milliseconds (NC=20). The response time difference is only 5.5% with an SD of 50 milliseconds (NC=32). This is because as the server demand decreases, the influence of the request mapping policy on performance becomes less significant. As a result, the interaction pattern plays a more important role in determining system performance.

We also observe that, due to the increased impact of inter-node delay on the mean response time, HD continues to be the worst performer in this test. CBD2 performs slightly better than HD under medium load conditions and about the same as HD under high load conditions. The change from Figure 5-1 to Figure 5-20 is a result of the lower impact of the load balancing mechanism with the decrease of server demand.

In both tests, FB and FS perform comparably well. The corresponding throughput for the test with an SD of 50 milliseconds is shown in Appendix F.

To better investigate the effect of server demand, we also performed the test with an SD of 150 ms and NC of 1, 12 and 24. The test results are shown in Table 5-20.

Table 5-20: Response Time Measurement (ms) for the Test with SD of 150 ms

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, CV=1.5

NC	1	12	24
FB	257	563	1172
FS	261	570	1199
HD	292	590	1167
CBD2	259	651	1284
CBD1	261	519	1068
FBA	268	448	1070

From this table, we observe that, when NC equals 24, the response time differences among the various strategies diminish as the system becomes saturated.

In addition, Figure 5 – 21 shows mean response times for each strategy with different server demands when NC equals 12. It is evident that FBA and CBD1 perform remarkably well with different server demands in comparison with other strategies. In particular, as SD increases from 50 to 100 and 150 ms, FBA outperforms CBD1 and is the best performer at NC of 12.

Figure 5-21: Response Time Measurement for Each Strategy with Different Server Demands When NC=12

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, CV=1.5

Figure 5 – 22 shows the response time measurement for FBA, with different server demands and NC of 1, 12 and 24. As expected, the mean response time increases with the increase of the server demand. In particular, the degree of increase at NC of 24 is higher

than that at NC of 1 and 12. A similar observation is made with the other load balancing strategies.

Figure 5-22: Response Time Measurement for FBA with Different Server Demands and Numbers of Clients

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, CV=1.5

5.8 Effect of Coefficient of Variation of Think Time

The coefficient of variation of think time reflects the variability in think time and the resulting burstiness of the request arrivals on the servers. In a dynamic load balancing system, the sudden arrival of a large number of requests at the system may greatly upset the state of global balance and cause poor performance [HUI-00]. This phenomenon is called a “thundering herd”, and can occur if the load balancer forwards all requests to a server during a transient period, when the server load is not reported quickly enough [OTH-01].

In a static load balancing system, where a round-robin request mapping policy is used, this phenomenon cannot occur since requests will be dispatched to the servers in the list in a round-robin fashion. This experiment is designed to investigate the effect of coefficient of variation of think time on the relative performances of different load balancing strategies. The main parameters used in the experiment are listed in Table 5 – 21.

The mean think time was set to 300 milliseconds. When CV is equal to 1.0, the think time distribution is essentially exponential. The test carried out with a CV of 1.5 is the base test, and its results were discussed in Section 5.2.

Table 5-21: Parameters of the Experiment Investigating the Effect of Variability in Think Time

Factor	Levels
MS (bytes)	100, 5000, 7500
BW (Mbps)	1
ND (ms)	15
DR (%)	0
SD (ms)	100
CV	1.0, 1.5, 2.0

The response time measurement for the test with a CV of 1.0 is shown in Table 5-22 and Figure 5-23.

Table 5-23 shows a comparison between the test performed with a CV of 1.0 and the base test with a CV of 1.5. The table shows the percentage of response time increases for the test with a CV of 1.0 over the test with a CV of 1.5 for the various load balancing strategies.

Table 5-22: Response Time Measurement (ms) for the Test with CV of 1.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms

NC	1	4	8	12	16	20	24
FB	183	205	262	345	448	587	717
FS	191	206	271	363	451	584	722
HD	218	228	281	357	457	577	731
CBD2	180	206	280	375	489	624	772
CBD1	184	184	238	308	395	509	631
FBA	187	203	232	285	364	477	601

Figure 5-23: Response Time Measurement for the Test with CV of 1.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms

Table 5-23: Difference in Mean Response Time (percentage) between the Test with CV of 1.0 and the Test with CV of 1.5

NC	1	4	8	12	16	20	24
FB	-4	-2	-1	-1	-1	1	-2
FS	1	-1	1	2	-3	1	0
HD	1	2	2	-2	-2	-2	1
CBD2	-2	-2	-1	0	-2	0	1
CBD1	0	-2	-3	-3	-1	1	0
FBA	-1	2	0	0	1	2	-3

As we can see from this table, the differences are less than or equal to 4%. Given the nature of the results of the experiments, we can conclude that there is no significant difference in the two test response time results, and the coefficient of variation of think time has little effect on the relative performances of different strategies given a fixed mean think time, and an exponential or hyper-exponential distribution. This conclusion is also confirmed by the measurement results of the test carried out with a CV of 2.0, which also shows a maximum difference of 4% in comparison to the base test for each strategy.

The corresponding throughput measurement for the test with a CV of 1.5 is shown in Appendix G. As expected, the ranking of throughputs for the different load balancing strategies follows the same order as that of mean response times. Appendix G also includes all the measurement results of the test carried out with a CV of 2.0.

The set of tests described in this section shows that, for the CV values used in the experiment, there is no “thundering herd” phenomenon in the FBA strategy, even though FBA is not round-robin based. The load record of a server is updated every time a response for a request is returned through the balancer from the server to a client. In this way, requests will be dispatched based on the number of active requests on each server and the thundering herd phenomenon will not occur.

Chapter 6 Conclusions

6.1 Summary

A performance prototype of a CORBA-based load balancing system was designed and implemented to investigate the relative performances of various load balancing strategies, with a focus on wireless environments. These strategies include FB, FS, HD, CBD1, CBD2 and FBA (see Section 3.3). The wireless network connecting the clients with the balancer and the servers was emulated with the help of NIST Net. Various wireless bandwidths and message drop rates were experimented with.

A number of experiments were carried out in order to understand the performance impacts of different workload parameters, including message size, bandwidth, inter-node delay, server demand, drop rate and coefficient of variation of think time.

Different strategies emerged as the best performer under various workload conditions. For a broad range of workload parameters, the performance of FBA was seen to be the best for CORBA-based systems in wireless environments as long as the message drop rate is small. In this strategy, the balancer makes a load distribution decision based on the number of active jobs being processed on each sever, and this strategy shows a significant performance improvement over the round-robin-based strategies. For example, the improvement in mean client response time could be as high as 20% in comparison to FB (see Section 5.2).

This thesis investigated the impact of the system environment: wired or wireless on the performances of the load balancing strategies. For example, as shown in Section 5.3, a test has been carried out in a wired network environment in order to understand the

performance differences between the wired and wireless environments. The results of the test show that, with the given set of workload parameters, while FBA is the best performer in wireless environments with a message drop rate of 0%, CBD1 is the best in wired environments (see Figure 5 – 10 for example). Although FBA is the second best performer in the wired environment, its response time is only 4.7 % higher than that of CBD1 with NC of 16.

6.2 Derived Insights

Based on the experimental results and analyses presented in Chapter 5, we derive the following insights that can be useful in the design and development of CORBA-based load balancing systems in a wireless environment:

- (1) In general, HD is not a good load balancing strategy. It performs relatively poorly, especially when the system exhibits one or more of following characteristics: low system load, large message size, high inter-node delay, and relatively small server demand.
- (2) FS performs the best for large message sizes under low load conditions (see Section 5.3).
- (3) CBD1 exhibits a good performance for a broad range of workload conditions in comparison with the other strategies. The performance of CBD2 is much worse than CBD1. This performance degradation can be as high as 24% (see Section 5.2 for example).
- (4) A novel strategy called FBA proposed in this thesis takes advantage of the benefits of dynamic load balancing, while avoiding the overheads caused by the nature of

dynamic load balancing. Its request mapping policy is shown to be effective under various combinations of workload parameters. The results of the experiments carried out in this thesis have shown that, in many situations, it is a good choice for medium to high loads as long as the message drop rate on the wireless network is small.

- (5) Given a mean think time with an exponential or hyper-exponential distribution, the results of the experiments show that the coefficient of variation of think time has little impact on the relative performances of the different strategies.

Wired Environment:

Observations (1), (2) and (3) remain valid for wired environment (see Figure 5-10). However, as shown in this figure, CBD1 performs the best in the experiment with a wired environment. FBA emerges as the second best performer for medium to high loads.

Since the relative performances of FBA and CBD1 changes with the environment types, the strategies are discussed further in this section. Both the interaction pattern and the request mapping policy determine the performance of a strategy. Due to a larger number of messages used in FBA, and because the potential heavy load at its balancer that forwards both requests and replies, FBA performs interior to CBD1 in a wired environment. With a wireless environment characterized by a small drop rate and load bandwidth connection between clients and servers, the servers in CBD1 spends more time in communication in comparison to a wired environment. Since the balancer to server communication is through a wired (high bandwidth) connection, this additional server overhead is not incurred in FBA. As a result, FBA performs better than CBD1. At higher message drop rates in a wireless environment, the service time of clients, servers and balancer increases, due to the message retransmission occurring in the system. The

message queue at the balancer in FBA is expected to be longer than that of individual servers (in CBD1), each of which communicates with only with a subset of the total number of clients (NC). As a result, the load at the balancer increases at a higher message drop rate and FBA demonstrates an inferior performance in comparison to CBD1.

Of the six strategies investigated, both FBA and CBD1 are reasonable choices in a wired or wireless environment. They tend to perform better than the other strategies investigated in this thesis. In particular, CBD1 demonstrates a better performance in a wired environment or in a wireless environment with a high message drop rate, whereas FBA is superior to CBD1 in a wireless environment with a small message drop rate.

6.3 Future Research

The research in this thesis leads to a few future research topics, which are presented next.

- Mobility and temporary disconnection are also unique characteristics exhibited in a wireless environment. The impact of these two factors on the relative performances of various load balancing strategies is worthy of further investigation.
- In an enterprise setting, a load balancing system may contain multiple balancers (one per region for example) in order to provide high scalability, good performance and fault tolerance. A performance study of the various load balancing strategies in such a multiple balancer environment forms an interesting topic for future research.
- In the experiments described in this thesis, the servers have the same CPU speed. An investigation of the relative performances of the load balancing strategies in a system where the servers have different CPU speeds is worthy of future work.

- The clients are of the same type in terms of their service requests in the experiments reported in this thesis. Investigation of a system with multiple types of clients each characterized by a set of different service requirements seems interesting.
- The experimental results on the performances of the six strategies investigated in this thesis show that no single strategy performs the best under all conditions. Therefore, a CORBA-based load balancing system can potentially execute different strategies based on the changing workload and system conditions at run time, in order to achieve high performance. A study of such an adaptive load balancing system in a wireless environment forms an important direction for future research.

References

- [ADL-95] R. Adler, "Distributed Coordination Models for Client/Server Computing", *IEEE Computer*, Vol. 28, No. 4, April, 1995, pp. 14-22.
- [ABD-97] I.N. Abdul-Fatah, "Performance of CORBA-Based Client-Server Architectures", M. Eng. Thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, May 1997.
- [ABD-02] I.N. Abdul-Fatah and S. Majumdar, "Performance of CORBA-Based Client-Server Architectures", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, No. 2, 2002, pp.111-127.
- [BAK-99] S. M. Baker and B. Moon, "Distributed Cooperative Web Servers", *Proc. 8th International WWW Conference*, Toronto, Canada, May 1999, pp. 1215-1229.
- [BAR-98] T. Barth, G. Flender, B. Freisleben and F. Thilo, "Load Distribution in a CORBA Environment", *Proc. International Symposium on Distributed Objects and Applications*, Edinburgh, UK, September 1999, pp 158-166.
- [FAR-03] U. Farooq, S. Majumdar and E. Parsons, "Semi-Durable Subscriptions: A Technique to Achieve High Performance in Mobile Wireless Publish/Subscribe Systems", Technical Report SCE-03-17, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, July 2003.

- [GOK-96] A. Gokhale and D. C. Schmidt, "Measuring the Performance of Communication Middleware on High-Speed Networks", *Proc. SIGCOMM '96*, Stanford, USA, August 1996, pp. 306-317.
- [GOK-97] A. Gokhale and D. C. Schmidt, "Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks," *Proc. 17th International Conference on Distributed Computing Systems (ICDCS 97)*, Baltimore, USA, May 1997, pp. 401-410.
- [HE-00] A. He, "Performance Comparison of Load Balancing Methods in Multiple Carrier CDMA Systems", *Proc. 11th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2000)*, Vol. 1, London, UK, September 2000, pp. 113-118.
- [HUI-99] C. Hui and S.T. Chanson, "Improved Strategies for Dynamic Load Balancing", *IEEE Concurrency*, Vol. 7, No. 3, July-Sept. 1999, pp. 58-67.
- [ION-01] IONA, *Orbix 2000 White Paper*, April 2001.
Available from:
<http://www.proinfo.com.cn/chinese/menu2/orbix2000whitepaper.pdf>
- [ION-02a] IONA, *Orbix E2A Application Server Platform CORBA Programmer Guide Java*, ver 5.1, April 2003.
Available from:
http://www.iona.com/support/docs/e2a/asp/5.1/corba/pguide_java/pguide_java.pdf
- [ION-02b] IONA, *Orbix E2A Application Server Platform Administrator's Guide*, ver 5.1, April 2002.

Available from:

<http://www.iona.com/support/docs/e2a/asp/5.1/platform/admin/admin.pdf>

[KAM-97] H. Kameda, J. Li, C. Kim and Y. Zhang, *Optimal Balancing in Distributed Computer Systems*, Springer, 1997.

[KAN-02] J. Kangasharju, "MIWCO-Wireless CORBA Extensions for MICO", Department of Computer Science, University of Helsinki, Helsinki, Finland, November 2002.

Available from: <http://www.cs.helsinki.fi/u/jkangash/miwco/miwco.pdf>

[KIM-01] J.-H. Kim, R.S. Ramakrishna and Y.-S. Kim, "LODIN: Load Distribution Mechanism in CORBA using Interceptor", *Proc. IEEE Region 10 International Conference on Electrical and Electronic Technology*, Vol. 1, Pkuket Island and Lankawi Island, Singapore, August 2001, pp. 61-64.

[MAS-02] C. Mascolo, L. Capra and W. Emmerich, "Mobile Computing Middleware", Department of Computer Science, University College London, London, UK, 2002.

Available from:

<http://citeseer.nj.nec.com/cache/papers/cs/29222/http:zSzzSzwww.cs.ucl.ac.ukzSzstaffzSzc.mascolozSzwwwzSzntw02.pdf/mobile-computing-middleware.pdf>

[MUR-97] V.K. Murthy, "Transactional Workflow Paradigm for Mobile Computing", *Proc. 13th ACM Conference on Applied Computing*, Atlanta, USA, February 27-March 1, 1998, pp. 424-432.

- [NAT-02] National Institute of Standards and Technology, *NIST Net*, ver 2.0.12, June 2002.
- Available from: <http://snad.ncsl.nist.gov/itg/nistnet/>
- [OMG-02a] OMG, *Event Service Specification*, ver 1.1, March 2001.
- Available from: <http://www.omg.org/docs/formal/01-03-01.pdf>
- [OMG-02b] OMG, *Trading Object Service Specification*, ver 1.0, May 2000.
- Available from: <http://www.omg.org/docs/formal/00-06-27.pdf>
- [OMG-02c] OMG, *Minimum CORBA Specification*, ver 1.0, August 2002.
- Available from: <http://www.omg.org/docs/formal/02-08-01.pdf>
- [OMG-02d] OMG, *Naming Service Specification*, ver 1.2, September 2001.
- Available from: <http://www.omg.org/docs/formal/02-09-0.pdf>
- [OMG-02e] OMG, *Common Object Request Broker Architecture: Core Specification*, ver 3.0.2, December 2002.
- Available from: <http://www.omg.org/docs/formal/02-12-02.pdf>
- [OMG-03] OMG, *Wireless Access and Terminal Mobility in CORBA*, March 2003.
- Available from: <http://www.omg.org/docs/formal/03-03-64.pdf>
- [OTH-01] O. Othman, C. O’Ryan and D.C. Schmidt, “The Design and Performance of an Adaptive CORBA Load Balancing Service”, Department of Electrical and Computer Engineering, University of California, Irvine, USA, 2001.
- Available from:
- http://www.cs.wustl.edu/~schmidt/PDF/load_balancing.pdf

- [OTH-03] O. Othman, J. Balasubramanian and D.C. Schmidt, "The Design and Performance of an Adaptive Middleware Load Balancing and Monitoring Service", *Proc. 3rd International Workshop on Self-Adaptive Software*, Arlington, USA, June 2003.
- [PUD-99] A. Puder and K. Romer, *MICO: An Open CORBA Implementation*, Morgan Kaufmann, 1999.
- [RAC-97] G. Rackl, "Load Distribution for CORBA Environments", Diploma Thesis, Department of Informatics, University of Munich, Munich, Germany, 1997.
- [SCH-95] D.C. Schmidt and S. Vinoski, "Object Interconnections", *C++ report*, January 1995.
- [SCH-98] D.C. Schmidt, "Evaluating architectures for Multithreaded Object Request Brokers", *Communications of the ACM*, Vol. 41, No. 10, October 1998, pp. 54-61.
- [SIE-98] J. Siegel, "OMG Overview CORBA and the OMA in Enterprise Computing", *Communications of the ACM*, Vol. 41, No. 10, October 1998, pp. 37-43.
- [TAO-02] W. TAO, "Application Level Guidelines for Building High Performance CORBA-Based System", M.C.S. Thesis, School of Computer Science, Carleton University, Ottawa, Canada, May 2002.
- [TSA-02] V. Tsaoussidis and I. Matta, "Open issues on TCP for Mobile Computing", *The Journal of Wireless Communications and Mobile Computing*, Vol. 2, No. 1, February 2002, pp. 3-20.

- [YUA-97] W. Yuan, S. Djoko, S. Subramanian, S. Madhavapeddy and P. Maveddat,
 "Load Balancing in Wireless Network" , *Proc. IEEE Global Telecommunications Conference, 1997(GLOBECOM '97)*, Phoenix, USA, Vol. 3, November 1997, pp. 1616-1620.
- [ZHA-97] Y. Zhang, H. Kameda and S.-L. Hung, "Comparison of Dynamic and Static Load-balancing Strategies in Heterogeneous Distributed Systems", *IEEE Proceedings on Computers and Digital Techniques*, Vol. 144, No. 2, March 1997, pp. 100-106.
- [VIN-99] S. Vinoski and M. Henning, *Advanced CORBA Programming with C++*, Addison Wesley, 1999.
- [VIR-99] University of Virginia, "Benchmarks Results", 1997.
 Available from: <http://www.beust.com/virginie/Benchmarks>
- [WAN-02] R. Wang, M. Valla, M.Y. Sanadidi, B. Ng and M. Gerla,
 "Efficiency/Friendliness Tradeoffs in TCP Westwood", *IEEE Symposium on Computers and Communications*, Taormina, Italy, July 2000, pp. 304-311.
- [WU-02] W. Wu and S. Majumdar, "Engineering CORBA-based Systems for High Performance", *Proc. 2002 International Conference on Parallel Processing (ICPP'02)*, Vancouver, Canada, August 2002, pp. 473-482.

Appendix A Supplementary Results for the Base Test

Table A-1: Throughput Measurement (request/second) for the Base Test

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	2.04	7.84	14.18	18.49	21.22	22.73	23.19
FS	2.04	7.87	14.08	18.32	20.94	22.73	23.53
HD	1.94	7.63	13.91	18.07	20.89	22.55	23.48
CBD2	2.07	7.80	13.75	17.75	20.05	21.62	22.51
CBD1	2.07	8.21	14.65	19.42	22.82	24.81	25.83
FBA	2.04	8.00	15.07	20.48	24.21	26.11	26.14

Figure A-1: Throughput Measurement for the Base Test

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Appendix B Supplementary Results for the Experiment Investigating the Effect of Message Size

Table B-1: Throughput Measurement (request/sec) for the Test with MS of 5000 Bytes

BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	1.57	6.34	11.94	16.04	18.58	20.30	20.83
FS	1.80	7.16	12.62	16.60	18.67	19.88	20.46
HD	1.62	6.16	11.40	14.69	16.84	18.10	18.90
CBD2	1.71	6.24	11.17	15.06	17.45	19.25	19.98
CBD1	1.68	6.72	12.14	16.13	18.60	20.53	21.16
FBA	1.59	6.37	12.18	17.00	19.78	21.65	22.10

Figure B-1: Throughput Measurement for the Test with MS of 5000 Bytes

BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Appendix C Supplementary Results for the Experiment Investigating the Effect of Bandwidth

Table C-1: Throughput Measurement (request/sec) for the Test with BW of 100 Mbps

MS = 5000 bytes, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	1.62	6.38	12.27	16.57	19.30	21.07	21.52
FS	1.85	7.23	13.31	17.65	20.28	21.83	22.66
HD	1.63	6.50	12.31	16.57	19.66	21.83	23.32
CBD2	1.72	6.59	12.12	16.55	19.63	21.48	22.02
CBD1	1.75	6.93	13.11	17.94	21.45	23.89	24.92
FBA	1.62	6.46	12.60	17.67	20.86	22.30	22.88

Figure C-1: Throughput Measurement for the Test with BW of 100 Mbps

MS=5000 bytes, ND=15 ms, DR=0%, SD=100 ms, CV=1.5

Appendix D Supplementary Results for the Experiment Investigating the Effect of Inter-node Delay

Table D-1: Throughput Measurement (request/sec) for the Test with ND of 0 ms

MS=100 bytes, BW=1 Mbps, DR=0%, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	2.19	8.56	15.37	19.93	22.05	22.83	23.32
FS	2.18	8.59	15.31	19.67	22.15	22.88	23.62
HD	2.19	8.60	15.34	19.61	22.57	23.23	23.79
CBD2	2.20	7.86	13.84	18.13	20.54	21.93	22.88
CBD1	2.22	8.87	15.72	20.58	23.62	25.05	26.23
FBA	2.20	8.74	16.60	22.06	24.69	25.94	26.68

Figure D-1: Throughput Measurement for the Test with ND of 0 ms

MS=100 bytes, BW=1 Mbps, DR=0%, SD=100 ms, CV=1.5

Appendix E Supplementary Results for the Experiment Investigating the Effect of Drop Rate

Table E-1: Throughput Measurement (request/sec) for the Test with DR of 2%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	2.01	7.58	13.54	17.73	20.43	22.22	23.46
FS	2.00	7.52	13.32	17.42	20.27	22.05	22.68
HD	1.87	7.13	12.79	16.95	19.78	21.75	22.86
CBD2	2.03	7.59	13.31	17.17	19.95	21.59	22.35
CBD1	2.02	8.11	14.36	19.28	22.70	24.57	25.76
FBA	2.01	7.67	14.16	19.48	23.05	24.75	25.33

Figure E- 1: Throughput Measurement for the Test with DR of 2%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

Table E-2: Throughput Measurement (request/sec) for the Test with DR of 5%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

NC	1	4	8	12	16	20	24
FB	1.95	7.07	12.42	16.48	19.54	21.33	22.37
FS	1.92	6.78	11.62	15.29	18.13	20.26	21.60
HD	1.74	6.47	11.44	15.08	17.94	20.12	21.60
CBD2	1.96	7.17	12.82	16.81	19.16	20.95	22.17
CBD1	1.95	7.78	13.85	18.55	21.56	23.75	24.92
FBA	1.95	7.10	12.92	17.54	21.02	23.20	24.54

Figure E-2: Throughput Measurement (request/sec) for the Test with DR of 5%

MS=100 bytes, BW=1 Mbps, ND=15 ms, SD=100 ms, CV=1.5

Appendix F Supplementary Results for the Experiment Investigating the Effect of Server Demand

Table F-1: Throughput Measurement for the Test with SD of 50 ms

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, CV=1.5

NC	1	8	16	24	32	40
FB	2.44	17.78	32.26	41.45	46.38	47.90
FS	2.43	17.54	31.43	41.17	45.39	47.17
HD	2.26	17.02	30.25	39.34	45.58	47.90
CBD2	2.45	18.10	32.00	40.07	45.07	47.45
CBD1	2.46	18.69	33.61	44.04	49.38	51.15
FBA	2.44	18.06	33.76	45.03	50.79	52.29

Figure F-1: Throughput Measurement for the Test with SD of 50 ms

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, CV=1.5

Appendix G Supplementary Results for the Experiment Investigating the Effect of Variability in Think Time

Table G-1: Throughput Measurement (request/sec) for the Test with CV of 1.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms

NC	1	4	8	12	16	20	24
FB	2.07	7.92	14.23	18.60	21.39	22.55	23.60
FS	2.04	7.91	14.01	18.10	21.30	22.62	23.48
HD	1.93	7.58	13.77	18.26	21.14	22.81	23.28
CBD2	2.08	7.91	13.79	17.78	20.28	21.65	22.39
CBD1	2.07	8.26	14.87	19.74	23.02	24.72	25.78
FBA	2.05	7.95	15.04	20.51	24.10	25.74	26.64

Figure G-1: Throughput Measurement for the Test with CV of 1.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms

Table G-2: Response Time Measurement (ms) for the Test with CV of 2.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms

NC	1	4	8	12	16	20	24
FB	184	209	262	343	448	582	718
FS	186	210	264	354	447	568	720
HD	217	231	274	360	473	586	726
CBD2	182	208	291	377	493	620	747
CBD1	182	185	236	311	403	521	645
FBA	186	197	232	282	371	482	611

Figure G-2: Response Time Measurement for the Test with CV of 2.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms

Table G-3: Throughput Measurement (request/sec) for the Test with CV of 2.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms

NC	1	4	8	12	16	20	24
FB	2.07	7.86	14.23	18.66	21.39	22.68	23.58
FS	2.06	7.84	14.18	18.35	21.42	23.04	23.53
HD	1.93	7.53	13.94	18.18	20.70	22.57	23.39
CBD2	2.07	7.87	13.54	17.73	20.18	21.74	22.92
CBD1	2.07	8.25	14.93	19.64	22.76	24.36	25.40
FBA	2.06	8.05	15.04	20.62	23.85	25.58	26.34

Figure G-3: Throughput Measurement for the Test with CV of 2.0

MS=100 bytes, BW=1 Mbps, ND=15 ms, DR=0%, SD=100 ms