INFORMATION TO USERS

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

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

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

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

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

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

UMI[®]

QoS-Enabled E-Commerce Web Application

by

Ping Li

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

**Ottawa-Carleton Institute of Computer Science
(OCICS)**

**Carleton University
Ottawa, Ontario, Canada, K1S 5B6
December 2001**

©Copyright 2001, Ping Li

**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

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

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

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

Your file Votre référence

Our file Notre référence

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

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

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

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

0-612-66874-6

Canada

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

QoS-Enabled E-Commerce Web Application

submitted by **Ping Li**
in partial fulfillment of the requirements for
the degree of Master of Computer Science

Dr. Shikharesh Majumdar, Thesis Co-Supervisor

Dr. Eric Parsons, Thesis Co-Supervisor

Dr. Frank Dehne, Director of School of Computer Science

Carleton University
December 2001

Abstract

Performance sensitive distributed applications leveraging Object Request Broker (ORB) technology, especially in the context of e-commerce web services, are gaining more and more attention. However, most conventional commodity-off-the-shelf (COTS) middleware products and web servers do not have the ability to support applications with quality of service (QoS) requirements. This thesis explores the ability to control resource allocation for e-commerce applications that use such conventional CORBA compliant-ORBs for inter-component communications fronted by an Apache http web server. In existing systems, all incoming requests are treated equally from the web server and the CORBA ORB perspectives, which make it impossible to offer differentiated performance for different classes of customers. In many environments, it is desirable to ensure that preferred classes of customers be given improved service, especially when the load on a server becomes significant. The goal of the thesis is to provide insights into the way in which an e-commerce site can add certain real time features to e-commerce applications and provide QoS requirements for different customers. By applying three techniques – setting scheduling priority in web server tier, setting scheduling priority in the CORBA back end servers, and controlling priorities dynamically based on the shopping states of customers – it is found that QoS requirements can be preserved end to end.

Acknowledgements

I would like to express my sincere thanks to my supervisors, Professor Shikharesh Majumdar and Professor Eric Parsons, for their valuable advice, encouragement, patience and effort on my thesis. I feel very fortunate to have had their supervision and extremely grateful for the learning experience they have given me.

I would also like to extend my sincere thanks to the faculty members of the RADS lab group for their help and support.

I am grateful to my family for their understanding and support.

TABLE OF CONTENTS

List of Tables	v
List of Figures.....	vi
Glossary of Terms and Symbols	viii
Chapter 1 Introduction.....	1
1.1 Thesis Motivations	1
1.1.1 QoS in E-Commerce	1
1.1.2 Why CORBA in E-Commerce Applications	2
1.1.3 The Goals of this Thesis	4
1.2 Thesis Contributions.....	4
1.3 Thesis Organization.....	5
Chapter 2 Literature Overview	7
2.1 Common Middleware in E-Commerce Systems	7
2.2 CORBA	12
2.3 Real-Time CORBA	14
2.3.1 Real-Time CORBA Key Components	14
2.3.2 Discussion	17
2.4 Process Scheduling.....	18
2.5 Integration of Web and CORBA	21
2.5.1 Invoking CORBA Objects from Applets	21
2.5.2 Invoking CORBA Objects from Servlets.....	22
2.5.3 Invoking CORBA Objects from CGI Processes	23
Chapter 3 QoS-enabled CORBA-based E-Commerce Web Application	25
3.1 Application Descriptions	25
3.2 Sample E-Commerce Application.....	26
3.3 Implementation of Sample E-Commerce Application	29
3.4 Message Sequence Charts for CORBA Clients and Servers.....	34
3.5 CBMG with Random Behavior (CBMG-RB).....	37
3.6 CBMG with Random and Balk Behavior (CBMG-RBB).....	39
3.7 Techniques Used to Preserve QoS End to End.....	40
3.7.1 Preserving QoS Requirements in the Middle Tier – the CGI Application Servers.....	41
3.7.2 Preserving QoS Requirements in the Third Tier – the CORBA Back End Server	42
3.7.3 Controlling Priorities Dynamically Based on the Shopping State of a Customer	42
Chapter 4 CBMG-oriented Open Model Workload Generation Tool.....	44
4.1 The Existing Workload Generation Tools.....	44
4.1.1 Httpperf	44

4.1.2 Scalable URL Reference Generator (SURGE)	45
4.2 CBMG-based Open Model Workload Generation Tool.....	47
4.2.1 Closed Model vs. Open Model.....	47
4.2.2 Tool Descriptions	47
4.2.2.1 Main Thread.....	48
4.2.2.2 Request Thread	50
Chapter 5 Performance Prototype	54
5.1 System Architecture	54
5.2 Workload Parameters and System Parameters	58
5.2.1 Workload Parameters	58
5.2.2 System Parameters	60
5.3 Performance Metrics.....	62
5.4 Validations.....	65
5.4.1 Applying Little's Law	65
5.4.2 The Same Priority Setting vs. the Different Priority Setting.....	66
5.4.3 Effect of Arrival Rate on Experiment Time and Total Number of Arrivals During Experiment	68
5.4.4 Fixed CORBA Service Time vs. Exponential CORBA Service Time	70
5.4.5 Effect of Different Balk Time Settings on Total Number of Items Bought...	72
Chapter 6 Results of Experiments.....	74
6.1 Experiments Based on the CBMG-RB Model	74
6.1.1 Effect of Varying Client Arrival Rate on Request Time.....	75
6.1.2 Effect of Varying Client Arrival Rate on End-to-End Time.....	75
6.1.3 Effect of Different Priority Settings	76
6.1.4 Effect of Varying Go-Back Probability Setting	78
6.1.5 Operation-by-Operation Performance Comparisons.....	80
6.1.6 Comparisons Between CGI-nice-on and CGI-nice-off.....	83
6.2 Experiments Based on the CBMG-RBB Model.....	85
6.2.1 Effect of State-Based-On/Off on Total Number of Items Bought	86
6.2.2 Effect of State-Based-On/Off on Number of Items Bought by Different Classes.....	87
6.2.3 Class-by-Class Comparisons Between State-Based-On/Off on Number of Items Bought	89
6.2.4 Effect of State-Based-On/Off on Total Lost Rate.....	92
Chapter 7 Conclusions.....	94
7.1 Summary.....	94
7.2 Conclusions	96
7.3 Future Research	98
Appendix.....	101
References	103

List of Tables

Table 3-1: States of the E-Commerce Application Prototype.....	43
Table 5-1: Apache Server Configuration Parameters	55
Table 5-2: Levels of Factors.....	61
Table 5-3: The Default Parameter Values in CBMG-RB	62
Table 5-4: The Default Parameter Values in CBMG-RBB	62
Table 5-5: Difference Percentages	66
Table 6-1: Priority Setting in Different States	86

List of Figures

Figure 1-1: The Three Tier E-Commerce System Architecture.....	4
Figure 2-1: The CORBA 2.0 Inter-ORB Architecture.....	8
Figure 2-2: The Key Components in the CORBA Architecture	12
Figure 2-3: The Key Components in Real-Time CORBA Architecture.....	15
Figure 2-4: Servlet in Web Server.....	23
Figure 3-1: Login State	26
Figure 3-2: Select State	27
Figure 3-3: Confirm State	27
Figure 3-4: Pay State	28
Figure 3-5: Thanks State	28
Figure 3-6: Components in the CORBA E-Commerce Application.....	29
Figure 3-7: UCM upon Clicking Submit Button in the Login State	30
Figure 3-8: UCM upon Clicking Submit Button in the Select State.....	31
Figure 3-9: UCM upon Clicking Submit Button in the Pay State.....	31
Figure 3-10: Authentication Message Sequence Chart.....	34
Figure 3-11: Add to Shopping Cart Message Sequence Chart.....	35
Figure 3-12: Message Sequence Chart for Clicking Submit Button in the Pay State.....	36
Figure 3-13: CBMG with Random Behavior.....	38
Figure 3-14: CBMG with Random and Balk Behavior	40
Figure 4-1: Open Model Workload Generation Tool.....	49
Figure 4-2: CBMG-RB Request Thread Control Flow Diagram.....	52
Figure 4-3: CBMG-RBB Request Thread Control Flow Diagram	53
Figure 5-1: System Architecture	57
Figure 5-2: Request Time with the Same Priority Setting (10-10-10).....	67
Figure 5-3: The Same Priority Setting vs. the Different Priority Setting.....	68
Figure 5-4: Effects of Arrival Rate on Experiment Time	69
Figure 5-5: Effect of Arrival Rate on Total Number of Arrivals During Experiment	70
Figure 5-6: Fixed Service Time vs. Exponential Service Time	71
Figure 5-7: Effect of Balk Time on Total Number of Items Bought	72
Figure 6-1: Effect of Arrival Rate on Request Time.....	75
Figure 6-2: Effect of Arrival Rate on End-to-End Time.....	76
Figure 6-3: Discrimination Factors at Priority Setting 3-10-17	77
Figure 6-4: Discrimination Factors at Priority Setting 7-10-13	77
Figure 6-5: Effect of Varying Go-Back Probabilities with Arrival Rate 0.5	78
Figure 6-6: Effect of Varying Go-Back Probabilities with Arrival Rate 0.7	79
Figure 6-7: Effect of Varying Go-Back Probabilities on Gold Request Time.....	79
Figure 6-8: One Direct Call Operation Diagram.....	80

Figure 6-9: Response Time of One Direct Call.....	81
Figure 6-10: Single Nested Call Operation Diagram	81
Figure 6-11: Response Time of Single Nested Call	82
Figure 6-12: Double Nested Call Operation Diagram	82
Figure 6-13: Response Time of Double Nested Call	83
Figure 6-14: Discrimination Factors with CGI-nice-off	84
Figure 6-15: Discrimination Factors with CGI-nice-on.....	84
Figure 6-16: Effect of State-based-on/off on Total Number of Items Bought.....	87
Figure 6-17: Number of Items Bought by Different Classes with State-based-off.....	88
Figure 6-18: Number of Items Bought by Different Classes with State-based-on	88
Figure 6-19: Effect of State-based-on/off on the Number of Items Bought by Gold	90
Figure 6-20: Effect of State-based-on/off on the Number of Items Bought by Silver.....	91
Figure 6-21: Effect of State-based-on/off on the Number of Items Bought by Bronze....	91
Figure 6-22: Effect of State-based-on and State-based-off on Total Lost Rate	92
 Figure A-1: Pseudo Code of CBMG-RB-based Workload Generation Tool.....	 101
Figure A-2: Pseudo Code of CBMG-RBB-based Workload Generation Tool.....	102

Glossary of Terms and Symbols

API	Application Programming Interface
BOA	Basic Object Adapter
CBMG	Customer Behavior Model Graph
CBMG-RB	Customer Behavior Model Graph with Random Behavior
CBMG-RBB	Customer Behavior Model Graph with Random and Balk Behavior
CGI	Common Gateway Interface
COTS	Commodity-Of-The-Shelf
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DBMS	DataBase Management System
DCOM	Distributed Component Object Model
FIFO	First In – First Out
GIOP	General Inter-ORB Protocol
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IP	Internet Protocol
ISO	International Standards Organization
IT	Information Technology
JDBC	Java DataBase Connectivity
MDAC	Microsoft Data Access Component
MOM	Message-Oriented Middleware
OA	Object Adapter
ODBC	Open DataBase Connectivity
OLE	Object Linking and Embedding
OMG	Object Management Group
ORB	Object Request Broker
ORPC	Object Remote Procedure Call
POA	Portable Object Adapter
RPC	Remote Procedure Call
RR	Round Robin
RTOS	Real-time Operating Systems
SURGE	Scalable URL Reference GEnerator
TCP	Transmission Control Protocol
TP	Transaction Processing
URL	Uniform Resource Locator
WOSA	Windows Open Services Architecture (Microsoft).

Chapter 1 Introduction

1.1 Thesis Motivations

1.1.1 QoS in E-Commerce

Almost all business and commerce sectors have been affected by the Internet and they are facing many new kinds of competition. Every company has had to reconsider how it conducts business in the age of e-commerce. In recent years, hundreds of thousands of e-business companies were formed and the competition among them has been extremely intense. Only those with the most flexibility and the quickest response time will be the ones that succeed.

Quality of service (QoS) of e-business sites has become an important factor for the success of the company. But the general-purpose operating systems and web servers were not designed with QoS in mind. The research area of QoS in e-businesses has been toward reallocating resources such as CPU, disk, and network bandwidth based on the site's existing resources. For example, QoS requirements can be controlled by dynamically tuning the configuration parameters such as maximum number of connections, number of threads, and cache sizes through tracking the site's workload and the value of QoS metrics [19]. Resource management strategies based on business-oriented policies, using priorities that are dynamically assigned as a function of the state a customer is in and as a function of the amount of money in his/her shopping cart have been also investigated [17], but not in the context of multiple customer classes.

Both of the approaches described in the previous paragraph demonstrate usefulness, but both of them are complex and neither of them makes any difference between customer classes. For example, an e-business site may want to distinguish customers who have high probabilities to purchase items based on past behavior.

Since e-businesses are widespread throughout the world, more and more players, such as businesses, individuals, governments and organizations participate in the electronic market. E-businesses can be placed into five categories [16]: 1) Business-to-Business (B2B). 2) Business-to-Customer (B2C). 3) Customer-to-Customer. 4) Government-to-Business. 5) Government-to-Citizens. Just as e-businesses are categorized in different ways, so are their customers. Customers can be categorized as heavy buyers or light buyers, frequent partners or infrequent partners, and registered customers or unregistered customers. Due to the increasing demands for faster response time and higher availability from e-business sites, online companies must continuously offer good performance to avoid losing sales and customers. Failure to provide high-quality services results in lost revenue. In any e-business site, whether online bookstores or brokerage services, it is necessary to guarantee the quality of service for customers, and especially for important customers (i.e., heavy buyers, frequent partners, or registered customers).

The next section presents the motivations of using CORBA middleware in e-commerce applications.

1.1.2 Why CORBA in E-Commerce Applications

E-Commerce applications are by definition distributed systems since they allow customers to buy products over the Internet. Internally, there are many challenges faced

by e-business sites. First, an e-commerce application may need to share information among distributed inventory, manufacturing, or shipping systems to determine the exact availability of the product requested [42]. Second, there must be an object-based development approach to manage the cumbersome process of developing e-commerce systems [33]. Third, it should have abilities to grow across different platforms as needs change.

The Object Management Group (OMG) devised the CORBA specification, which is an object-oriented middleware specification facilitating object communication in a heterogeneous distributed paradigm. The power of CORBA lies in enabling the building of distributed and object-oriented applications and facilitating interoperability between software objects regardless of their implementation details and locations [33]. By using a CORBA middleware, organizations can focus on building objects that represent business functions and then combine these business functions with others through CORBA interfaces.

Therefore, the challenges faced by e-business sites can be gracefully solved by using CORBA middleware. In this thesis, CORBA servers are used as the back-end servers of the web-based e-commerce application. Figure 1-1 illustrates a multi-tiered architecture: at the front end, (e.g., browsers) customers can access the e-commerce application web pages, which in turn access the operations of CORBA servers at the back end.

Figure 1-1: The Three Tier E-Commerce System Architecture

1.1.3 The Goals of this Thesis

One purpose of this thesis is to investigate how to provide QoS requirements end to end based on the classes of customers for an e-commerce site. It is achieved by managing resources in the middle tier – the Apache web server and the last tier – the CORBA back end servers. The other purpose of this thesis is to investigate how to improve revenue for an e-commerce site. It is achieved by controlling resources dynamically based on the shopping state of a customer.

1.2 Thesis Contributions

We have built a prototype and investigated its performance for various workload parameters. The prototyping and measurement approach enables the capturing of the effect of system overheads on system behavior and performance that are difficult to capture accurately in simulation or analytic model-based investigations.

The important contributions that concern the preservation of end to end QoS requirements based on a CORBA-based e-commerce web applications are briefly summarized:

- The thesis describes a design and implementation of a prototype consisting of an Apache web server and an omniORB-based e-commerce sample application.
- A workload generation tool based on an open queuing model is also developed. This workload generation tool uses a state transition graph called Customer Behavior Model Graph (CBMG) to simulate a customer session. Two different CBMG models are examined. One is Customer Behavior Model Graph with Random Behavior (CBMG-RB) model, and the other is Customer Behavior Model Graph with Random and Balk Behavior (CBMG-RBB) model.
- It devises three techniques for preserving QoS requirements end to end:
 1. Setting scheduling priorities in the middle tier – the CGI application servers;
 2. Setting scheduling priorities in the last tier – the CORBA back-end servers;
 3. Controlling priorities dynamically based on the shopping state of a customer.
- A significant performance benefit for favored customers is obtained by setting priorities in CGI application servers and back-end servers.
- Customer state-based priority control technique demonstrates a significant revenue benefit for the electronic shop.

1.3 Thesis Organization

Chapter 2 first presents the common middleware used in e-commerce systems. Then specific topics such as CORBA, Real-time CORBA, and process scheduling are

presented in greater detail. Finally research concerning the techniques of integrating web-based system and CORBA is discussed.

The detailed implementation of the CORBA-based and QoS-enabled e-commerce web application is described in Chapter 3. The web interfaces of the application and the message sequence charts (MSC) for the implementation are presented. At the end, the concept of the Customer Behavior Model Graph (CBMG) is introduced.

Chapter 4 describes the workload generation tool. First a brief description of the differences between open model and closed model is given. Then the detailed description of the implementation of the tool is presented.

The system architecture, workload parameters, and system parameters are presented in Chapter 5. Also, the results of validation of the performance measurement model are provided by using various approaches, including applying Little's Law, comparing results between common priority setting and different priority settings, and investigating some performance metrics of the system.

Chapter 6 presents the results of the experiment. The results concern the three strategies: the CGI servers' priority setting, the back-end servers' priority setting, and customer state-based dynamic priority control. A detailed explanation of the experimental results is also offered here.

Chapter 7 gives the conclusions of the thesis and suggests direction for future research.

Chapter 2 Literature Overview

2.1 Common Middleware in E-Commerce Systems

Numerous definitions exist for middleware. The IT (Information Technology) industry has failed to standardize on specific functionality that characterizes middleware. It has been noted that “the only definition that the industry tends to agree upon is that middleware is anything that resides between the client (user) and server (database and application resources)” [33]. This thesis follows that definition.

We categorize the common middleware used in e-commerce systems as five types: 1) middleware framework (e.g. CORBA – Common Object Request Broker Architecture, COM – Component Object Model), 2) transaction processing middleware (e.g. TP monitor – Transaction Processing monitor), 3) communication middleware (e.g. MOM – Message-Oriented Middleware, DCOM – Distributed Component Object Model), 4) application middleware (e.g. CGI – Common Gateway Interface), 5) database middleware (e.g. ODBC – Open DataBase Connectivity, JDBC – Java DataBase Connectivity) [33]. This section reviews some middlewares and their capabilities. Additional information on middleware for e-commerce can be found in [32].

- **CORBA**

The Common Object Request Broker Architecture (CORBA) defined by the Object Management Group (OMG) is a distributed object computing middleware standard. It provides a standard, seamless, and transparent way to distribute objects across multiple

platforms and operating systems such as UNIX, Linux, and Macintosh by using the Internet Inter ORB Protocol (IIOP).

IIOP is a part of the CORBA specification. It specifies how CORBA-defined messages, such as GIOP (General Inter-ORB Protocol), are exchanged over TCP/IP network. GIOP is also a part of the CORBA specification, specifying the set of message formats and common data representations for communications between Object Request Brokers (ORBs). GIOP maps CORBA ORB requests to different network transports and IIOP maps GIOP messages to TCP/IP transport (see Figure 2-1 [28]).

Figure 2-1: The CORBA 2.0 Inter-ORB Architecture

Since CORBA allows distributed objects to behave exactly like local objects, CORBA is the obvious choice for distributed object-oriented applications. In Section 2.2, we will give a detailed overview of CORBA.

- **(COM)**

Microsoft's COM grew from its earlier standard Object Link Embedding (OLE). OLE was used to make Windows applications communicate effectively by building software

components. For example, the cut-and-paste and the drag-and-drop operations in Window environment are based on these components [33] [43]. COM is a middleware specification for how components interact with their clients. It allows applications and systems to be built by components that are supplied by different software vendors. Since COM is a binary standard, it is language transparent. With COM and Distributed COM (DCOM), component interactions can happen among all applications and among all different machines in a network. Microsoft has implemented COM for Win32 and for Apple MacOS while COMPAQ has implemented COM for the Tru64 UNIX platform.

- **Transaction Processing (TP) monitor technology**

TP monitor technology grew from Atlantic Power and Light Corp. It allows transaction applications in distributed client/server environment to be efficiently developed, run and managed [33]. TP monitor is mainly used in data management, network access, security systems, delivery order processing, airline reservations, and customer service [38]. With the emergence of e-commerce computing, TP monitor technology has evolved to handle e-commerce applications.

- **Message-Oriented Middleware (MOM)**

MOM is a communication middleware. It supports general-purpose messages such as account-opening requests to interact in a distributed application environment. Since data is exchanged by message passing, MOM supports both synchronous and asynchronous interactions between distributed computing processes. Messages under a MOM system are placed into a queue and retrieved whenever the server requests them. In addition, since messages can be retrieved off the queue in any order, MOM can also facilitate retrieving of messages based on priority or load-balancing schemes. OMG has introduced

an object-oriented CORBA messaging standard based on MOM to facilitate the development of object-oriented e-commerce applications [33].

- **DCOM**

Microsoft's Distributed Component Object Model (DCOM) is an extension of the Component Object Model (COM). It supports interactions among components located on different computers by using a protocol called the Object Remote Procedure Call (ORPC). This ORPC is layered on top of the Open Group Distributed Computing Environment (DCE) Remote Procedure Call (RPC) mechanism on a network communications protocol developed by IBM, Sun Microsystems, Hewlett-Packard, and Digital Equipment Corporation. ORPC interacts with COM's run-time services [21] [31]. Since the COM specification is at the binary level, it allows DCOM server components to be written in diverse programming languages such as C++, Java, Visual Basic and even COBOL. As long as a platform supports COM services, DCOM can be used on that platform. DCOM is heavily used on the Window platforms.

- **CGI**

The Common Gateway Interface (CGI) is a standard for building bridges between external applications and HTTP web servers. A plain HTML document retrieved by the web server is static, containing no dynamic content. A CGI program, on the other hand, is executed at run time. It can output dynamic information according to the form information entered by a user [4]. A CGI program can be written in any language that allows it to be executed on the system, such as C/C++, Fortran, Perl, TCL, Unix shell scripts, Visual Basic and AppleScript, depending on what is available on the system.

- **Open DataBase Connectivity (ODBC)**

ODBC is an Application Programming Interface (API) developed by Microsoft. It allows a programmer to access data in a heterogeneous environment of relational and non-relational database management systems (DBMSs). With ODBC, applications can concurrently access, view, and modify data from multiple, diverse databases. When programming to interact with a database via ODBC, it is only necessary to interact with the ODBC API and SQL itself. The ODBC Manager will decide how to interact with the type of database that is being targeted [35].

ODBC is a core component of Microsoft Windows Open Services Architecture (WOSA). ODBC is merged into MDAC (Microsoft Data Access Component), which is a freely available package that can be installed on a Windows machine. Many databases, such as Oracle, Informix, DB2, and SQL server are supported by one or more ODBC drivers.

- **Java DataBase Connectivity (JDBC)**

Similar to ODBC, JDBC is an open specification for database isolation, except that it is oriented toward Java applications. It allows a smooth transition between the world of the relational database and the world of a Java application. For example, the return results obtained from JDBC are transferred as Java variables, and access errors are thrown as Java exceptions [34]. JDBC is implemented by a set of Java interfaces and different software vendors may provide different Java interfaces. By installing a JDBC driver, which is a set of Java classes supporting for a particular database engine, it is possible to access the back-end databases. Many vendors (e.g. Sybase, OpenLink, IBM etc.) have

developed JDBC drivers to provide access to various databases such as Sybase, Oracle, and DB2.

2.2 CORBA

CORBA is a distributed object computing middleware standard defined by the Object Management Group (OMG). “CORBA Object Request Brokers (ORBs) allow clients to invoke operations on distributed objects without concerning for object location, programming language, OS platform, communication protocols and interconnects, and hardware” [9]. Figure 2-2 illustrates the primary components in the OMG Reference Model architecture [25]:

Figure 2-2: The Key Components in the CORBA Architecture

- **Client**

Client and server are relative concepts. This means that a client may be a server and a server may be a client of another object. Whenever an object obtains another object's reference and invokes operations on that object, it becomes a client. There is no difference for a client to access a remote object or a local object in the context of CORBA.

- **Servant**

This component provides the implementation of operations defined in IDL interface. In object-oriented (OO) languages, such as C++ and Java, classes are used to implement the servant. In non-OO languages such as C, functions and structs are used to implement the servant. A client can invoke operations on a servant through an object reference.

- **IDL compiler**

An IDL compiler is a CORBA application pre-compiler. It automatically translates OMG IDL codes into a client stub and a server skeleton that are mapped in an application programming language, such as C++ or Java. The IDL compiler cooperating with IDL provides language transparency for CORBA applications.

- **IDL Stub and Skeleton**

IDL stub and skeleton are generated by the IDL compiler. The IDL stub is used by the client. It implements the Proxy pattern [8] to marshal remote call parameters into common message-level representation and to send them to a server. The IDL skeleton is used by the servant. It implements the Adapter pattern [8] and coordinates invocations with the Object Adapter (OA) to demarshal the message-level representation back into typed parameters that are meaningful to an application.

- **Object Adapter (OA)**

The CORBA Object Adapter sits on top of the ORB and accepts requests. It provides the run-time environment for creating and interpreting object references, demultiplexing incoming requests to servants, and dispatching the appropriate operation to a servant through IDL skeletons. Basic Object Adapter (BOA) and Portable Object Adapter (POA) are two standard object adapters.

- **Object Request Broker (ORB)**

A CORBA Object Request Broker provides a transparent way for locating objects [37]. This means that it can find objects that may exist in the same process, different processes, or different machines across the network. Most services provided by the ORB are implemented by coordinating with the object adapter, stubs, skeleton, or dynamic invocation. The ORB itself is usually implemented as a run-time library that is used to connect client and server applications.

The next section presents the key components of Real Time CORBA. It also summarizes the inter-relationship between the goals of Real Time CORBA and the QoS-enabled e-commerce system based on a general CORBA compliant ORB that this thesis focuses on.

2.3 Real-Time CORBA

2.3.1 Real-Time CORBA Key Components

Real-time CORBA is an extension to CORBA 2.3 and CORBA Messaging Specification. It allows applications to schedule, and control key resources such as CPU,

memory, and networking to ensure end-to-end quality of service. Figure 2-3 illustrates the primary components in the OMG Reference Model architecture [26].

- **Real-Time ORB**

The Real-time ORB is an extension to the normal CORBA ORB interface. A Real-time ORB initialization method is used to pass user priorities to the Real-time ORB. The Real-time ORB then maps these priorities into CORBA native priorities, which are in the range 0 to 32767, and which provide a common representation of priority across different Real-Time Operating Systems (RTOS).

Figure 2-3: The Key Components in Real-Time CORBA Architecture

- **Real-Time CORBA Priority Mapping**

The Real-time CORBA priority mapping provides a mapping between CORBA priorities and native priorities to overcome the heterogeneity of Real-time Operating Systems (RTOS). There are two priority models for handling Real-time CORBA

priorities during invocations. One is a Client Propagated Model and the other is a Server Declared Model.

The Client Propagated Priority Model is the default setting in Real-time CORBA. In this model, the CORBA priority is assigned in the client side. The CORBA priority is provided in the service context list and is sent to the ORB with an invocation request. The server side RTORB and the RTOS then process the request at that priority.

In the Server Declared Priority Model, the Real-time CORBA priority is not passed with invocations. The CORBA priority is assigned a priori by a server based on the value of an attribute. The Client ORB knows the server priority since the priority is encoded in an object reference. In this case, the client's Real-time CORBA priority is not passed with the invocation via the service context list. Therefore, the RTORB and RTOS handle the request at the priority assigned by the server.

- **Real-Time CORBA Thread Pools**

Real-time CORBA thread pools manage threads of execution on the server-side of the ORB. A thread pool's parameters, such as the default priority of threads in the pool, the maximum buffer size of the pool, the default number of static threads that are created initially, and the maximum number of threads that can be created on demand, can be specified when the thread pool is created [36].

The default thread priority of a pool may be changed dynamically in accordance with the priority models (e.g. Client Propagated Priority Model and Server Declared Priority Model) discussed earlier.

The maximum buffer size associated with a pool can be optionally pre-configured. If the maximum buffer size is pre-configured, the request will be buffered until a thread is

available to process it. If the buffer space is not available, the ORB will raise a TRANSIENT (temporary resource shortage) exception [29].

The static threads are pre-allocated and they will consume system resources even if they do not serve requests. One thread is used to serve one request. When these pre-allocated threads are all activated, the subsequent requests may be served by creating new threads. However, if the total number of the threads spawned reaches the maximum number of threads set during initialization, no additional thread will be spawned.

2.3.2 Discussion

Real-time CORBA currently only supports fixed priority scheduling technique. The dynamically scheduling [24] techniques such as deadline-based [40] and value-based scheduling [11] are being standardized. Implementing Real-time CORBA is relatively challenging and most Commodity-Off-The-Shelf (COTS) CORBA ORBs including omniORB have not yet provided the QoS features.

Relationships between the Real-time CORBA system and our QoS-enabled e-commerce system are summarized:

1) Real-time CORBA relies on static real time priorities whereas our QoS enabled e-commerce systems uses the default time-sharing scheduling found in Linux.

2) Real-time CORBA supports “object level” priority setting. The server priority can be assigned at the time of object reference creation or servant activation. Our QoS-enabled e-commerce system supports “operation level” priority setting. The server priority can be assigned at the time of object operation activation.

2.4 Process Scheduling

CPU resource management is commonly known as scheduling. The default Linux scheduling is based on the time-sharing technique [3]. The CPU time is divided into time slices. The time slice duration depends on the time quantum durations of all runnable processes. The time quantum value is the CPU time assigned to a process in a time slice. Different processes may have different time quantum. At the beginning of each time slice, the value of time quantum of each runnable process is computed and assigned. If all runnable processes exhaust their time quantum, the CPU time changes to a new time slice. If the currently running process is not finished when its time slice or time quantum expires, a context switch may take place.

Context switching can significantly affect performance. This is because whenever the CPU is multiplexed, two pairs of context switches occur. In the first, the original running process has its context saved and the dispatcher's context is loaded, then the dispatcher can select the next running process from the ready queue. The second pair is for the dispatcher to be removed and the selected process to be loaded onto the CPU.

"The scheduler of Linux is quite efficient while there are few tens of processes running at a time" [3]. But if the number of existing processes is very large, it is inefficient to recompute all dynamic priorities at once. As a result, the overhead of recomputation is increased. Also, since priorities are recomputed only when all runnable processes have exhausted their time quantum, the frequency of recomputation is decreased as the system load increases, and therefore, system responsiveness decreases with increasing load.

The aspect of performance related to how long a process has to wait once it becomes ready is determined by the scheduling policy. Scheduling policy is a set of rules to determine when it is time for a process to be removed from the CPU, and which ready process should be allocated the CPU next. It ranks processes according to their priority.

There are two kinds of priorities in the Linux 6.2, kernel 2.2.16-3: static priority and dynamic priority. Static priorities range from 1 to 99 and can be assigned by superusers. Processes with this range of static priority (1-99) are real time processes. Dynamic priority is only applied to normal processes, whose static priority is assigned to 0. The dynamic priority is based on time quantum and the number of ticks of CPU time left to the process before its quantum expires in the current time slice.

The Linux scheduler decides which runnable process will be executed by the CPU next. There are three scheduling policies supported by the scheduler: First In – First Out (FIFO) scheduling, Round Robin (RR) scheduling, and standard time-sharing scheduling. FIFO and RR are used for real time applications and the standard time-sharing scheduling is used for normal processes and it is also the default scheduling policy in the Linux 6.2, kernel 2.2.16-3 operating system [13].

Next, the semantics of the three policies are discussed.

- **First In-First out (FIFO) Scheduling**

FIFO scheduling can be used with static priorities in the range 1 to 99 and it is a simple scheduling algorithm without time slicing. FIFO processes are real-time processes. When a FIFO process becomes ready, it will always preempt immediately any currently running standard time-sharing process. If no other higher priority real-time process is ready, the process will continue to use CPU until it finishes, even if another

equal priority real-time processes become ready. A FIFO process that has been preempted by another process with higher static priority will remain at the head of its static priority list and will resume execution as soon as all higher priority processes are blocked again. When a FIFO process becomes ready, it will be inserted at the end of the its static priority queue [13].

- **Round Robin (RR) Scheduling**

RR scheduling also can be used with static priorities in the range 1 to 99 and it is a simple enhancement of FIFO scheduling. Everything described above for FIFO scheduling also applies to RR scheduling, except that each process is only allowed to run for a maximum time quantum. If a RR process has been running for a time period equal to the time quantum, it will be put at the end of the its priority queue. This ensures a fair usage of CPU time to all RR processes that have the same priority. A higher priority process can not preempt a RR process that has a lower priority immediately. The lower priority RR process will continue to run until its CPU time reaches its round robin time quantum [13].

- **Standard Time-sharing Scheduling**

Standard time-sharing scheduling can only be used at static priority 0. It is used for all normal processes that do not need real-time services. All the normal processes are put in the static priority 0 list. Choosing the next process to run is based on the dynamic priority of the runnable process that is determined only inside this static priority 0 queue. If a normal process is ready to run, its dynamic priority is dynamically increased after each time quantum that the runnable process has waited, and dynamically decreased by the

scheduler if it exhausts its time quantum in a CPU time slice. This also ensures fair assignment of CPU time among all standard time-sharing scheduling processes [13].

Two system calls, denoted as `nice()` and `setpriority()`, allow processes to change their base priority, which is the value of time quantum assigned by scheduler at the beginning of each CPU time slice. Both of the system calls can change a normal process's dynamic priority. The `nice()` system call can affect only the process that invokes it, but the `setpriority()` system call will affect the base priority of all processes in a given group [3]. The details of these system calls are covered later (see Section 6.1 and Section 6.1.6).

2.5 Integration of Web and CORBA

There are many techniques already in existence for integrating the web with CORBA services. From our investigations, three techniques are possible: 1) by using Applets, 2) by using Servlets, 3) by using the CGI protocol. The following sections will address the three techniques.

2.5.1 Invoking CORBA Objects from Applets

Applets run at the web server client side. In order to call a CORBA object from an Applet, it is necessary to obtain the reference of the CORBA object in the `init()` method of an Applet. If the CORBA object is not found by the Applet, additional requirements for the browser are needed (e.g. the browser must have an ORB or it must be compatible with JavaTM 2, Standard Edition v1.3.). It is possible to include the Applet in an HTML page, thus allowing Java-enabled browsers to execute the Applet. By embedding Applets

in an HTML page, CORBA can be used to interact with the web server by clicking on any of the Applet-embedded components without switching out of the page's context to obtain each response. In this way, it is not necessary to dynamically generate an HTML page for every response from a CORBA object.

2.5.2 Invoking CORBA Objects from Servlets

Servlets are modules of Java code that run at web server side to serve client requests. They retrieve HTML form data, generates dynamic content and manages state information on top of the stateless HTTP. The client class must extend from "HttpServlet" class. When a client sends the first Servlet request, the request is sent to the web server, which loads Servlet class, creates a Servlet instance for the client, and then sends the response to the client. The subsequent requests from this client are handled using the same Servlet instance (see Figure 2-4). This is the main advantage of Servlet over CGI since each CGI request is handled by a new process spawned by the web server.

Similar to invoking a CORBA Object from an Applet, a CORBA object reference is also instantiated in the `init()` method of a client Servlet class. A CORBA method can be called either from the `doPost()` method or from the `doGet()` method of the client Servlet class.

Figure 2-4: Servlet in Web Server

2.5.3 Invoking CORBA Objects from CGI Processes

The CGI protocol can be used to build the bridge between a web server and CORBA services in two different ways. The first option is to make the CGI application a CORBA client application. This means that the CGI applications should initiate a CORBA object reference and invoke operations on that object directly. Programming languages such as C or C++, which are supported by CORBA and can be a CGI gateway language, can be used to implement this kind of CORBA client CGI application. Another option is to use an intermediate CGI script as a CGI gateway language, and make the CGI script execute a CORBA client application, which has to get a CORBA object reference and invoke operations on that object. Perl or Visual Basic can be used for such CGI scripts.

This thesis takes advantage of the first option of invoking CORBA objects from CGI processes. This is because C++ can be a CGI gateway language and be used to develop CORBA servers and clients easily. C++, together with a C++ library for CGI programming [6] are chosen to implement the interactions between a web page and a

CORBA object in this thesis. An example of this type of implementation can be found in [15].

The CorbaWeb [20] environment developed at the University de Lille is a way of using CGI as a bridge between the Web browser and the CORBA object. It is a generic gateway between CORBA and the web. In this environment, users can navigate over and invoke the CORBA object as easily as users navigate over a web browser. It uses a new CGI script language – CorbaScript to achieve its objectives. CorbaScript can greatly simplify the tasks of invoking a CORBA object and generating an HTML page from the results obtained. The CorbaWeb (CorbaScript) interpreter translates CGI parameters into CorbaScript variables, and the scripts can invoke any world-wide CORBA object through IIOP. The current version of CorbaScript is 1.3.3. It can be compiled on omniORB3. It does not support the Apache server, but uses its own http server.

Chapter 3 QoS-enabled CORBA-based E-Commerce Web Application

3.1 Application Descriptions

E-Commerce applications are virtual online stores. By using a computer web browser, a customer can conveniently log onto a virtual on-line store and order products 24 hours a day, 7 days a week. E-commerce web applications vary from business to business. However, they share several common characteristics [33]. For example, most e-commerce sites usually have the following services: 1) Providing product information (e.g. product catalog, and search engine); 2) Managing an electronic shopping cart; 3) Ordering items; 4) Identifying a customer for payment and shipping; 5) Checking out.

Based on these common characteristics, a QoS-enabled CORBA-based e-commerce web application prototype is developed. It provides basic on-line store services such as login, add/remove to/from a shopping cart, and purchase processing. QoS-enabled means that this application has the ability to provide prioritized services based on different user service levels. The user service levels may be based on past usage history or membership level of a customer. Obtaining these levels is outside the scope of this thesis. In this study, we assume that user information such as username, password, user service level, and address is established in advance and stored in a corresponding database. (User name should be a unique identifier to serve as a primary key in the database.)

For the purpose of this study, we assume we only have three different user service levels: *level-1*, *level-2*, and *level-3*. The different quality of services should be provided based on these three different user service levels by this application. *Level-1* customers should be given the best services, *level-2* customers should be given the next best service, and *level-3* customers are prioritized last. This application is only applicable for registered users. The application could be broadened to include another service level for guest users.

3.2 Sample E-Commerce Application

Before giving detailed descriptions of how to provide quality of services in the CORBA-based e-commerce web application according to the three user service levels just listed in Section 3.1, we present what the web-based application resembles from a web browser perspective. The following figures represent the key web-based interfaces of this prototype. These interfaces provide customers with the means for logging in, accessing a list of products, selecting products from the list for purchase, and completing a purchase.

Figure 3-1: Login State

In the Login State (see Figure 3-1), a customer should provide his/her user name and password in order to access the product list. Based on customers' account information, a corresponding product list is displayed.

Figure 3-2: Select State

After login, a customer's product list, amount of discount, and full name are shown in the Select State (see Figure 3-2). From this state, a customer can add/remove products to/from the shopping cart by checking/unchecking products in the list.

Figure 3-3: Confirm State

The Confirm State shown in Figure 3-3 displays the customer's shopping cart, and asks the customer to confirm the purchase. After clicking the submit button, the customer moves to the Pay State (see Figure 3-4).

A screenshot of a web form titled "Please enter your credit number:". Below the title is a text input field labeled "Credit Card Number". At the bottom of the form, there are two buttons: "Submit" and "Cancel". The form has a simple, functional design with a light background and dark text.

Figure 3-4: Pay State

In the Pay State, the customer inputs their credit card number to checkout (see Figure 3-4). Note that a simple page as shown in Figure 3-4 that ignores additional details such as date of expiry is adequate for the performance experiments described in the thesis.

A screenshot of a web form titled "Your purchase has been successfully saved!". Below the title is a horizontal line. Underneath the line, the text reads "Thank you very much Ping Li" and "We will mail all your purchases in 10 days". The form has a simple, functional design with a light background and dark text.

Figure 3-5: Thanks State

After the customer's credit card number is validated, the service ends with the Thanks State in which delivery information is presented (see Figure 3-5).

3.3 Implementation of Sample E-Commerce Application

Most of these services are provided by six CORBA objects and three CGI programs. The six CORBA objects are Authentication, Account, Product, ShoppingCart, Purchase, and Validation, respectively. The three CGI processes are CGI-Authentication, CGI-ShoppingCart, and CGI-Validation (see Figure 3-6). Next, the interactions among these components are presented. After that, the detailed descriptions of each component are provided.

Figure 3-6: Components in the CORBA E-Commerce Application

The interactions among these nine components are activated by clicking the submit buttons in the web interfaces. The consecutive interactions can be shown by Use Case Map (UCM) diagrams. The UCM diagrams shown in Figure 3-7, Figure 3-8, and Figure 3-9 are associated with the submit button click in the Login State (Figure 3-1), the Select State (Figure 3-2), and Pay State (Figure 3-3). The three UCM diagrams all involve

CORBA objects. Those interactions that do not involve CORBA objects are omitted for clarify.

Figure 3-7: UCM upon Clicking Submit Button in the Login State

After a user clicks the “Login” button in the Login State, the web server activates the CGI-Authentication process. The underlying interactions between the CGI process and CORBA objects are shown in Figure 3-7. The CGI process first contacts the Authentication CORBA object in order to check if the user name matches its password. If it does not match, the login is denied and the user returns to the Login page. If it does match, the CGI process queries the Account CORBA object to obtain the user’s profile information such as the user’s full name, and address. The CGI process then accesses the Product CORBA object to obtain available products for the user, and the Product object in turn calls the Account CORBA object to get discount for that product for that user. After all this done, the corresponding product list is shown in the Select State (see Figure 3-2).

Figure 3-8: UCM upon Clicking Submit Button in the Select State

The interactions shown in Figure 3-8 are activated by clicking the “Purchase” button in the Select State (see Figure 3-2). It contains a double nested CORBA call, where the CGI-ShoppingCart object calls the ShoppingCart CORBA object, which in turn calls the Product CORBA object, and which further makes a call to the Account CORBA object. The detailed transactions are not necessary to describe here since they are beyond the essence of our discussion.

Figure 3-9: UCM upon Clicking Submit Button in the Pay State

The interactions in Figure 3-9 are activated by clicking the “Submit” button in Pay State (see Figure 3-4). As you see in the figure, there are four CORBA objects involved in the successive invocations. Also, the detail transaction description is omitted here, as once again, they beyond the essence of our discussion.

Next the detailed descriptions of each component are provided.

- **Authentication**

In the web world, authentication usually is the process that users may need to follow before being permitted to access a web site. A user must know which web sites he/she wants to access, and the web site must have already stored a user’s profile. The authentication process in our application is implemented by the Authentication CORBA object. For simplicity, it simply checks if a specified user name matches its password.

- **Account**

The Account object provides profile information about a customer. In our example, this includes full name, user service level, address, and amount of discount.

- **Product**

The Product CORBA object has the ability to retrieve a list of products in a particular category according to the account information. For example, different user service level may have different product selections available to the user.

- **ShoppingCart**

The ShoppingCart CORBA object is used for adding/removing product items to/from shopping cart, which is valid for the duration of the session.

- **Purchase**

The Purchase CORBA object is used for performing the purchase, assuming the credit card validation succeeds.

- **Validation**

The Validation CORBA object is used for checking if the credit card number provided by the customer is correct.

- **CGI-Authentication**

The CGI-Authentication process is executed by the web server after a user clicks the “Login” button in Login State. It generates one of two web interfaces depending on the authentication result. If the authentication fails, a new Login web page is presented, and if the authentication succeeds, a web page including the product list and the other information (e.g. full name, discount percentage) are provided (see Figure 3-2).

- **CGI-ShoppingCart**

The CGI-ShoppingCart process is activated by clicking the “Purchase” button in the Select State (see Figure 3-2). It saves the content of the shopping cart, and generates a web page, which displays the content of the user’s shopping cart, gives the total price, and asks the user for confirmation of the purchase.

- **CGI-Validation**

The CGI-Validation process is activated by clicking the “Submit” button in the Pay State after a user input his/her credit card number (see Figure 3-4). The process calls the Validation CORBA object to validate the credit card number. If the validation succeeds, a web page with shipping information is presented. Otherwise, a web page containing the

wrong credit card number information is shown, and requests that the users return and input their credit card number again.

3.4 Message Sequence Charts for CORBA Clients and Servers

Message Sequence Chart (MSC) is an ISO standard formal description technique for showing simple message flow in time between concurrent communicating processes [39]. The following three charts: Figure 3-10, Figure 3-11 and Figure 3-12 show all the message interactions between the three CGI CORBA clients and the six CORBA servers. From these, one also can see the QoS parameter involved in each CORBA object call.

Figure 3-10: Authentication Message Sequence Chart

The message sequence chart shown in Figure 3-10 is built upon the use case map shown in Figure 3-7. In this chart, the CORBA method names, input parameters, and the output parameters are provided. Note that message sequences that belong to false branches are not shown in Figure 3-10 and Figure 3-11. The reason is that they do not involve CORBA object calls and our experiments never go to these branches. Figure 3-10 also shows a One Direct Call and a Single Nested Call that will be discussed in Chapter 6. The One Direct Call means that a client calls a single CORBA object and then the CORBA object sends back a response to the CORBA client. A Single Nested Call means that a client calls a CORBA object, which in turn calls another CORBA object, and then the second CORBA object sends its response back to the first CORBA object, which in turn sends its response back to the CORBA client.

Figure 3-11: Add to Shopping Cart Message Sequence Chart

The message sequence chart shown in Figure 3-11 is built upon the use case map shown in Figure 3-8. It clearly shows a Double Nested Call. A Double Nested Call means that a client calls a CORBA object, which in turn calls another CORBA object, which also in turn calls the third CORBA object.

Figure 3-12: Message Sequence Chart for Clicking Submit Button in the Pay State

The message sequence chart shown in Figure 3-12 is built upon the use case map shown in Figure 3-9. Although we have investigated the nested calls involved in this chart, we do not show their results in Chapter 6 because they have the similar results as the One Direct Call and Double Nested Call described previously.

3.5 CBMG with Random Behavior (CBMG-RB)

When a customer navigates a web site, he/she typically issues some consecutive requests. A standard definition of a session is a sequence of consecutive requests issued by the same customer during a single visit to an e-commerce site [16]. Within a session, a customer can send different requests, and different customers may present different surfing patterns through the web site. Since customers' behavior has an important impact on the revenue of e-businesses, it is necessary to characterize the various behaviors of customers.

In our e-commerce web site, we can identify different sequences. For example, a customer may go from Login State to Select State, from Select State to Confirm State, from Confirm State to Pay State, and from Pay State to Thanks State. Another customer may go from Login State to Select State and leave the site without buying anything.

A Customer Behavior Model Graph (CBMG) [16] [17] [18] reflects the navigational pattern of a customer during a visit to an e-commerce site. Such a model has been used in evaluating the performance of an e-commerce system subjected to the TPC-W benchmark. A CBMG has n states, and state 0 is always the Entry State and state n is always the Exit State. Among these states, transitions may happen with certain probabilities. For instance, a transition may occur if a customer issues a request from state j to state $j-1$, or to state $j+1$, or if a customer leaves the web site from any state (except the Entry State) to the Exit State.

After adapting the CBMG model to our e-commerce web application, the CBMG model graph shown in Figure 3-13 is established. Note that five states are adequate for describing the customer behavior in our system. A higher number of states may be required for other systems (see [17] for example). The nodes of the CBMG, represented by squares, depict the states a customer is in during a visit to the e-commerce site. Arrows connecting states indicate possible transitions between them. The think time of a state is denoted by a half circle with an arrow. The Entry State is a special state that immediately precedes a customer's entry to the online store. This state is a part of the CBMG as modeling conveniences and does not correspond to any action started by the customer. The Exit State refers customers leaving the site from any state. There is a transition from all states, except the Entry state, to the Exit state [16].

Figure 3-13: CBMG with Random Behavior

P_i are transition probabilities between states, where $i \in \{0,1,2,3,4,5,6,7,8,9\}$. The approach for determining these transition probabilities from HTTP logs is discussed in [16]. In this study, the transition probabilities are workload parameters input. For simplicity, a uniform probability distribution is used to control transitions between states by comparing them with transition probability provided. The CBMG developed here, therefore, gets the name - CBMG with random behavior (CBMG-RB).

3.6 CBMG with Random and Balk Behavior (CBMG-RBB)

CBMG scenario discussed in Section 3.5 captures the characteristics of a group of customers who exit the system at various points but for no specific reason. We do not always know why they leave but it occurs in the real world while customers are surfing the Internet. However, it is also true that some customers may exit the system if the server response time is large. The CBMG-RB model discussed in Section 3.5 is not suited for this type of customer behavior, and a new CBMG model must be investigated.

We thus adapt the model to have a user specifically exit when the system response time becomes large. The resulting CBMG model is shown in Figure 3-14. The time duration is called Balk Time. If the response time is larger than the Balk Time, the customer always exits the system; otherwise the customer exits the system with a given probability. For example, in the Login State (see Figure 3-14) if R is less than B , the customer proceeds to the Select State with a probability P_0 and exits the system with a probability P_6 ($P_6 = 1 - P_0$). If R is larger than B , the customer always exits the system. Since customers' behavior is influenced by both Balk Time and random behavior, we name the new CBMG model CBMG with Random and Balk behavior (CBMG-RBB).

There is an “eight-second rule” that e-business sites try to follow. The “eight-second rule” is an unsubstantiated but widely held belief that after eight seconds of waiting for a web page to be downloaded, a customer becomes impatient and will likely abandon the site [16]. Therefore, the default Balk Time value is set to 8 seconds in our experiments.

Figure 3-14: CBMG with Random and Balk Behavior

In Chapter 4, we will discuss how our workload generation tool is built upon the two models – CBMG-RB and CBMG-RBB.

3.7 Techniques Used to Preserve QoS End to End

Providing quality of service for distributed applications end to end is the major concern of this thesis. But how to pass a QoS parameter, which maps one of the three different user service levels (*level-1*, *level-2*, and *level-3*) from clients to servers is a key

issue. Based on the three user service levels, we also have three different QoS parameters: Gold, Silver, and Bronze. Next, the detailed QoS parameter passing implementation is described.

3.7.1 Preserving QoS Requirements in the Middle Tier – the CGI Application Servers

The e-commerce application prototype discussed above is a three-tiered architecture. In order to provide customer QoS requirements end to end, it is essential to provide the QoS in all server tiers. The Apache Web server used in this thesis is in the middle tier. It does not support QoS requirements. Because it is difficult to reengineer the web server to satisfy QoS requirements, we must find another way to provide QoS support.

As we know, many HTML pages that are returned to clients are dynamically generated by application servers and the dynamic pages are usually expensive to generate. Therefore, it is important to enable QoS in these application servers. In our --commerce prototype, CGI processes such as CGI-Authentication, CGI-ShoppingCart, and CGI-Validation (see Section 3.3), which are activated by the web server, are used to generate three dynamic pages. Since the three CGI processes also belong to the middle tier – the web server (web server processes and CGI processes are running on the same system), the middle tier QoS can be provided through these CGI processes.

In order to provide different services for different classes of customers in the CGI processes, the key task is to provide priority-based policy based on QoS parameters, which in turn map to the user service levels.

3.7.2 Preserving QoS Requirements in the Third Tier – the CORBA Back End Server

Since omniORB 3.0 is not a Real-time CORBA compliant ORB, it does not have the ability to provide QoS natively. In our e-commerce web application prototype, CORBA back-end servers provide most of the functionality of performing a purchase. It is important to add the QoS feature into these CORBA back end servers. Similar to the first technique, this is achieved by adjusting the priority of the threads invoking methods, using the QoS parameters obtained from the user service levels passed in initially by the CGI process and propagated with each method invocation.

3.7.3 Controlling Priorities Dynamically Based on the Shopping State of a Customer

The two techniques discussed in Section 3.7.1 and 3.7.2 only pay attention to the types of customers. In some cases, it is worth adjusting priorities according to a user's state other than his/her service level. For example, one might consider increasing priority, when a user is in some important shopping states. This is particularly useful in the CBMG-RBB model discussed in Section 3.6. In the CBMG-RBB model, a customer's decision to leave or stay may heavily depend on the response time. In some key states such as Confirm State (see Table 3-1 for rationale for choosing it as a key state), if the performance is particularly poor, a customer has a high chance of leaving, and therefore the e-commerce site may lose money. Based on this point, the technique of controlling priority dynamically is developed.

Choosing a key state for increased prioritization for a site is very important. Sometimes, negative effects may occur if one chooses a wrong state as a key state. From

Figure 3-14, we observe that the previous page's response time may influence a customer's next behavior. For example, if the response time for requesting the Login page is larger than the Balk Time, the customer will not go to the Select page, and if the response time for requesting the Pay page is larger than the Balk Time, the customer will not go to the Thanks page. Since there is no Balk Time associated with the Thanks page, requesting the Thanks page will not be blocked if a customer passes the first four pages. Therefore, the Thanks State does not qualify as a key state. Since the time for retrieving static pages is much faster than for dynamic pages, it is less likely that these exhibit significant delays. Therefore static pages such as the Login page and the Pay page do not need to be key states. Since the Confirm State has the longer response times compared to the Select State, it has higher probability to exceed the balk threshold, so the Confirm State is chosen as the key state (see Table 3-1). Selecting additional states in which priorities can be changed dynamically is possible. However a large number of such states implies a large time required for system configuration and tuning.

Table 3-1: States of the E-Commerce Application Prototype

	Login	Select	Confirm	Pay	Thanks
Balk Time Association	Yes	Yes	Yes	Yes	No
Page Status	Static	Dynamic	Dynamic	Static	Dynamic
Response Time	Low	Middle	High	Low	High
Balk Possibility	Low	Middle	High	Low	N/A
Key State	No	No	Yes	No	No

In the next chapter, we will discuss how a workload generation tool can be implemented to work with the two CBMG models (CBMG-RB, and CBMG-RBB) discussed in this chapter.

Chapter 4 CBMG-oriented Open Model Workload Generation Tool

4.1 The Existing Workload Generation Tools

Synthetic workloads, whose characteristics are similar to those of the real workload and can be applied repeatedly in a controlled manner, are developed and used for this thesis. The main reason for using a synthetic workload is that it is a representation or model of the real workload. The other reasons for using a synthetic workload are that no real-world data files, which may be large and contain sensitive data, are required; the workload can be easily modified without affecting operation; it can be easily ported to different systems due to its small size; and it may have built-in measurement capabilities [10]. In the next two sections, two existing synthetic workload tools are introduced. After that, a specialized workload generation tool that is oriented to our CORBA-based e-commerce web application is presented.

4.1.1 Httpperf

Httpperf is a tool for measuring web performance. It provides a flexible facility for measuring web server performance. It supports both HTTP/1.0 and HTTP/1.1 [7] protocols and offers a variety of workload generators. While running, it keeps track of a number of performance metrics that are summarized in the form of statistics and printed at the end of a test run. The most basic operation of httpperf is to generate a fixed number

of HTTP GET requests, and to measure how many replies (responses) come back from the server and at what rate the responses arrive [22].

The current version of `httperf` supports two kinds of workload generators: one is a request generator that creates sessions deterministically and at a fixed rate, and the other is a URL generator that simply generates the specified sequence of URLs repeatedly. The `httperf` tool is suitable for measuring web server performance. It does not work well for capturing the performance metrics related to our e-commerce application (e.g. the number of items bought by customers). Also, we cannot use `httperf` directly to demonstrate our customers' behavior based on a CBMG (Customer Behavior Model Graph) model. Therefore, we cannot use `httperf` directly.

4.1.2 Scalable URL Reference Generator (SURGE)

The Scalable URL Reference Generator (SURGE) from Boston University is a WWW workload generator [2]. It generates web traffic and simulates the behavior of real web users. The requests generated by SURGE have six statistical properties:

- **Server File Size** – The variability in file sizes in a file system.
- **Request Sizes** – The distribution of file sizes that are requested for transfer.
- **Popularity** – The distribution of requests on a per file basis, having a strong effect on the behavior of caches. It typically follows Zipf's Law [44].
- **Embedded Object References** – The characterization of the number of embedded object references in web files. This parameter affects the distribution of Active OFF periods. An Active OFF period corresponds to the time between the transfer of embedded files of a single Web object [2].

- **Temporal Locality** – The likelihood that once requested, a file will be requested in near future. The distribution of the request stack distance is used as a measure of the temporal locality.
- **Off Times** – The distribution of inactive OFF time, which reflects the bursty nature of an individual Web user's requests and active OFF time, which reflects the transfer of Web objects.

SURGE generates a sequence of URL requests, which matches these representative distributions. It provides a sample database of documents from which to generate requests. Therefore, it is not possible to specify URLs for our CORBA-based e-commerce application directly. Similar to httpperf, SURGE has no ability to capture the performance metrics related to our e-commerce application and has no ability to handle customer requests based on the CBMG model used in our system. So we cannot directly use it either.

A number of other workload generation tools such as SPECWeb, and WebStone follow similar paradigms, but none of them supports the ability to measure e-commerce-oriented performance metrics or allows for a CBMG model. Thus, a special workload generation tool is needed. Also, the workload generation tool must provide support for both CBMG-RB and CBMG-RBB models. So the CBMG-oriented workload Generation tool, which has two different versions, has been developed for this thesis. Next, we will discuss the detailed implementation of the tool.

4.2 CBMG-based Open Model Workload Generation Tool

4.2.1 Closed Model vs. Open Model

The load on an online e-commerce site may be defined by the intensity of the request arrival rates or by the number of simultaneous customers. There are two classes of model representations for systems. One is a closed model and the other is an open model.

Closed models are characterized by having a fixed number of clients. In most online stores, there is a limitation on the maximum number of simultaneous customers logged into the system. Usually, this is due to the maximum number of TCP connections that the system can accept. Imagine an online store that, during peak hours, operates under a very heavy load in which the number of customers simultaneously logged in is constantly near the maximum. These situations can be adequately represented by a closed model.

However, open models allow requests to arrive, go through the various resources (e.g., web server, application server and database server in an e-commerce site), and leave the system. For example, consider an e-commerce site with no limit on the number of customers who typically arrive, obtain services, and leave the system.

An open model would represent our e-commerce system nicely since the number of total participants in the closed model is not to be known a priori and can vary from time to time. So, the open model is used in our workload generation tool.

4.2.2 Tool Descriptions

A CBMG-based open model workload generation tool has been developed for this thesis. The tool uses the HTTP/1.1 protocol to offer two different workloads based on the

CBMG with Random Behavior (CBMG-RB), and the CBMG with Random and Balk Behavior (CBMG-RBB). It keeps track of a number of performance metrics such as Request Time, End-to-End Time, and Total Lost Rate (see Chapter 5) to get insight on our system. A number of parameters such as transition probabilities, Balk Time, and Think Time (see Chapter 5) are used to control the behavior of customers.

A multithreaded Java environment is chosen to implement the CBMG-based open model workload generation tool. By starting a number of independent threads, a number of real users accessing the system concurrently can be simulated. Figure 4-1 illustrates this design.

4.2.2.1 Main Thread

The main thread shown in Figure 4-1 is used to control issuing client requests. The tasks of the main thread are: 1) starting a request thread at a time, 2) marking a flag for a thread as necessary, 3) computing and outputting average measured time, 4) outputting experiment time and total arrivals.

While the main thread is dispatching a thread, it sets a flag for each thread when the system is in a steady state. Since threads dispatched at the beginning may run on a lower system load than those dispatched later, they may generate unreliable results and so we exclude them from our results [10]. We refer those threads that are included as flagged threads.

Figure 4-1: Open Model Workload Generation Tool

Figure 4-1 illustrates how the main thread defines the threads that are flagged for measurement. The threads dispatched at the beginning and at the end by the main thread are not flagged threads and do not contribute to the results. The number of unflagged threads dispatched at the beginning is set to a fixed number according to system load. If the system load is higher, the number should be bigger. (In all our experiments, the number is set to 60 to satisfy different system loads). The number of unflagged threads dispatched at the end varies with the number of flagged threads and the system load. The number of flagged threads is an input parameter of this tool. The default value of number of flagged threads is 3000. After the unflagged threads are dispatched at the beginning, the following consecutive threads must be dispatched as flagged. The setting flag procedure will be stopped when the 3000 flagged threads have been dispatched. Then the subsequent threads are unflagged thread again. Dispatching of unflagged threads continues until all 3000 flagged threads have exited the system. If the system load is

higher, the time needed to complete the experiment increases, and the number of unflagged threads dispatched at the end increases as well. This expectation is validated by Figure 5-5.

4.2.2.2 Request Thread

The request threads shown in Figure 4-1 are created by the main thread. Each request thread corresponds to a customer. Since different customers have different navigational behavior while accessing the web site (discussed in Chapter 3), our customer behavior models such as CBMG-RB and CBMG-RBB must be built in the request thread. Two different versions of request threads are developed here, one is for CMBG-RB and the other is for CBMG-RBB, specified at system startup. Our experiments are separated into two groups based on CBMG-RB and CBMG-RBB (see Chapter 6). Figure 4-2 shows the control flow diagram of request thread for CBMG-RB, and Figure 4-3 gives the control flow diagram of request thread for CBMG-RBB. The appendix presents the pseudo codes of the main thread and the request threads that are based on both of the CBMG models.

The control flows shown in Figure 4-2 corresponds to the implementation of the CBMG with the Random Behavior based on Figure 3-13. Note that a thread may go back to a prior state, or may exit the system entirely. Random numbers generated in Request Thread are uniformly distributed within $(0, 1]$. They are used to map to the transition probabilities: $P_0, P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8,$ and P_9 , which are provided by the input parameter list of the tool.

The control flows in Figure 4-3 is the implementation of the CBMG with the Random and Balk Behavior based on Figure 3-14. It is similar to CBMG-RB model and the

random behavior is applicable here. Based on CBMG-RB, one more condition is added to a probability number and a random number comparison. The condition is if a page request time is large than the balk time, as specified for the experiment, the thread exits the system. If not, the thread continues to open another connection.

Figure 4-2: CBMG-RB Request Thread Control Flow Diagram

Figure 4-3: CBMG-RBB Request Thread Control Flow Diagram

Chapter 5 Performance Prototype

5.1 System Architecture

The performance prototype that was developed in the study is based on the CORBA compliant omniORB middleware and the Apache web server. First, a brief introduction of omniORB and the Apache web server is given. Then the system architecture is presented.

OmniORB is a robust, high-performance CORBA 2.3 ORB, developed by AT&T Laboratories, Cambridge. It provides a fairly complete multithreaded implementation of the CORBA standards and of the C++ language mapping, and performs well, particularly under Linux [30]. At any time, at most one call can be served in each communication channel between two address spaces [27]. Each channel is served by a dedicated thread. Thus by setting thread priorities differently one can provide different levels of quality of service easily without making any changes to the underlying operating system. The “setpriority” or “nice” system calls have been used in this thesis to set the priorities of threads and processes respectively. The prototype uses the COS Naming Service to get the address of an object.

Apache was originally based on code and ideas found in the most popular HTTP server, NCSA (the National Center for Supercomputing Applications). Apache has been the most popular web server on the Internet since April of 1996. The December 2000 Netcraft web server survey found that over 60% of the web sites on the Internet are using

Apache [12]. Because of its wide usage, Apache (version 1.3.12) was chosen as the web server for our prototype. The following Table 5-1 presents the Apache web server configuration parameters used in this study.

Table 5-1: Apache Server Configuration Parameters

Configuration Parameters	Default Value	Setting Value
StartServers	5	20
MinSpareServers	5	20
MaxSpareServers	10	25
MaxClients	150	8000
MaxKeepAliveRequests	100	0 (unlimited)
KeepAliveTimeOut	15 seconds	150 seconds

The rationale for the chosen levels of Apache server parameters shown in Table 5-1 is presented here.

- **StartServers, MinSpareServers and MaxSpareServers**

StartServers is the number of server processes that are started at the beginning. MinSpareServers and MaxSpareServers form the lower and upper limit on the number of spare server processes. The number of processes waiting for requests is compared with these values within the web server to ensure that enough server processes are available for handling the current load. Since increasing these values will increase the overhead within the server, the performance will be different upon different settings [23]. In order to ensure that all our experiments can be handled under the same configuration setting, we increase their values over the defaults included with Apache.

- **MaxClients**

MaxClients limits the total number of server processes running, (i.e., the number of clients that can simultaneously connect to the servers). Clients will be locked out if more

servers than this are needed. It is intended solely to keep the server from crashing when too many clients are accessing the system simultaneously [1]. Since the default value of 150 is easily reached in our experiments, the value of MaxClient was changed to 8000 in order to prevent requests from being refused by the web server.

- **MaxKeepAliveRequests**

MaxKeepAliveRequests denotes the maximum number of requests allowed during a persistent connection. It is set to 0 in our experiments to allow an unlimited number of requests. It is recommended that the number be set high for maximum performance [1].

- **KeepAliveTimeout**

KeepAliveTimeout is the number of seconds the server waits for the next request from the same client on the same connection. If the value is too small, an error is encountered with the current experimental setup. It was increased to 150 seconds to avoid the error, but it is not expected that the results would change significantly with lower values.

The workload generation tool developed sends URL requests to the web server at a certain rate by using HTTP/1.1 GET method. There are three URL requests that result in the web server executing a corresponding CGI, which in turn calls the CORBA objects, as illustrated in Figure 5-1. Note that the web server and application server (CGI processes) are separate processes running concurrently in the system.

The workload generation tool, CORBA servers, and web server all run under Red Hat Linux 6.2, kernel 2.2.16-3 on 266MHz Pentium II PCs with 64MB of RAMs. The PC's are inter-connected by a 100 MB Ethernet LAN. The PCs and LAN form the "quiet

network” in the Real Time and Distributed System Lab where experiments run without interference from other users.

Figure 5-1: System Architecture

Finally, we need to consider the scheduling policy. As discussed in Section 2.4, the Linux 6.2, kernel 2.2.16-3 supports three scheduling policies, two for real-time processes and one for normal processes. Although our e-commerce application provides quality of service, we still do not need to use the two real-time scheduling policies. The reason is that real-time scheduling policies are fixed priority, which would cause lower-priority customers to be starved at higher loads. The standard time-sharing scheduling policy described in Section 2.4 is suitable for the e-commerce application discussed in this

thesis. The dynamic priorities of standard time-sharing processes can be set according to the group to which a customer belongs.

5.2 Workload Parameters and System Parameters

The open model workload generation tool developed in this study provides a synthetic workload. The synthetic workload's characteristics are similar to those of the real workload and can be applied repeatedly in a controlled manner [10]. A number of experiments were conducted on the system. Each experiment is characterized by a set of workload and system parameters. The various parameters used in the experiments provide insight into system performance. A list of the workload and system parameters is presented next.

5.2.1 Workload Parameters

- **Mean Arrival Rate (requests/second)**

We assume that customer arrival follows a Poisson distribution. The Mean Arrival Rate in our experiments may vary from 0.3 requests/second to 1.2 requests/second.

- **Number of Flagged Jobs**

Only the flagged requests are measured in experiments. Flagged jobs do not include arrivals that are dispatched at the beginning and at the end of an experiment. This ensures that measured jobs always come from a system in a steady state. The number of Flagged Jobs in most experiments is set to 3000. For more detail on Flagged Jobs, see Section 4.2.2.1.

- **Mean Think Time**

Customers may spend time to think, input data, or click a button in a state of a site before they go to the next state. The think time is exponentially distributed. The default mean think time is 5 seconds in each state and does not change in any of the experiments. The exact value for mean think time chosen is unlikely to affect the relative performances of the different customer classes.

- **P_i**

They are the transition probabilities from a state to another state, where $i \in \{0,1,2,3,4,5\}$ (see Figure 3-13). Note that it is not necessary to set the values of P_6 , P_7 , P_8 , and P_9 since they can be computed automatically after providing the values of P_0 , P_1 , P_2 , P_3 , P_4 and P_5 . Two sets of P_i values have been used:

$$P_0 = 0.5, P_1 = 0.5; P_2 = 0.5; P_3 = 0; P_4 = 0; P_5 = 0.5$$

$$P_0 = 0.9, P_1 = 0.9; P_2 = 0.8; P_3 = 0.1; P_4 = 0.1; P_5 = 0.8$$

The first set assumes that customers do not go back to the previous states, and they either continue to the next state, or exit the system with equal probability. In the second set, there is a non-zero probability of a customer going back to the previous states. The graphs capturing the result of each experiment are labeled with the priority set used. The relative performances of the different customer classes are unlikely to depend on the exact P_i values chosen.

- **Balk Time**

It is the maximum time that a customer will wait to receive a response from a web server. We assume that customers leave the system if the server response time is larger than the Balk Time. The “eight-second rule” that was discussed in Section 3.6 is followed

here. Therefore, we set the default Balk Time to 8 seconds. For the sake of simplicity, a fixed balk time is used in this thesis. Modelling variabilities in balk time and studying the impact of such variabilities warrant investigation.

5.2.2 System Parameters

- **CGI-nice-switch (on/off)**

This parameter is used in the application server (CGI processes). If the CGI-nice-switch is turned on, the different priorities of the CGI processes can be set to different levels to satisfy the customers' QoS requirements. If CGI-nice-switch is turned off, there are no differences among CGI processes and all CGI processes have the default priority 0.

- **State-based-switch (yes/no)**

It is used to control priority setting dynamically during a session. If the State-based-switch is turned on, the CORBA server's priority setting not only depends upon which group a customer belongs, but also on the state the customer is in as described in Section 3.7.3. This flexibility provides more control on setting priorities.

- **Priority Setting**

Several sets of priority values are used in our experiments to investigate the impact of specific priority values on performance. The smaller the priority value, the higher is the priority. For priority setting 3-10-17, for example, the Gold customers are given the highest priority 3, the Silver customers are given 10, and the Bronze customers are given

17. For priority setting 7-10-13, the Gold customers are given the highest priority 7, the Silver customers are given 10, and the Bronze customers are given 13.

▪ **Mean Service Time**

Mean Service Time is synthetically added to each CORBA method to simulate complex processing in a CORBA back-end server. Service Time can be a fixed value, or can follow the exponential distribution in this study. The default mean service time is set to 200 milliseconds. Observing the behavior of the system at high utilization is important. This chosen mean service time allowed the system to saturate at a reasonable arrival rate.

Table 5-2 presents the levels of parameters used in the experiments. The default parameter values used in the experiments that are conducted for systems with CBMG-RB model and CBMG-RBB model are shown in Table 5-3 and Table 5-4 respectively.

Table 5-2: Levels of Factors

Parameter	Levels
Arrival Rate (requests/second)	0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.2
Balk Time (seconds)	4, 8, 12, 16
Priority Setting	3-10-17, 7-10-13, 10-10-10
P_0	1.0, 0.5, 0.9
P_1	Same as P_0
P_2	1.0, 0.5, 0.8
P_3	0.0, 0.25, 0.1
P_4	Same as P_3
P_5	Same as P_2
CGI-nice-switch	On, Off
State-based-switch	On, Off
Service Time (milliseconds)	Fixed, Exponential

Table 5-3: The Default Parameter Values in CBMG-RB

Parameter	Default Value
Number of Flagged Jobs	3000
Priority Setting	3-10-17
CGI-nice-switch	Off
State-based-switch	Off
Balk Time	Infinite
P_0, P_1, P_2, P_5	0.5
P_3, P_4	0.0
Mean Think Time	5 seconds
Mean Service Time	Fixed

Table 5-4: The Default Parameter Values in CBMG-RBB

Parameter	Default Value
Mean Arrival Rate	0.7 requests/second
Number of Flagged Jobs	3000
Priority Setting	3-10-17
CGI-nice-switch	Off
State-based-switch	Off
Balk Time	8 seconds
$P_0, P_1,$	0.9
P_2, P_5	0.8
P_3, P_4	0.1
Mean Think Time	5 seconds
Mean Service Time	Fixed

5.3 Performance Metrics

For each performance study, a number of performance criteria or metrics must be chosen. The following are the performance metrics that are used in this thesis.

- **Request Time**

It is measured usually as the time interval between the end of a URL request submission and the end of the corresponding responses from the system. In this study, there may be several URL requests associated to a customer, so the Request Time here is the sum of all URL Request Times of the customer.

- **End-to-End Time**

It is the time interval between the entry of a customer and his/her exit from the system. It includes all the Think Times in all the states the customer was in.

- **Total Number of Items Bought**

It is the total number of items bought by all customers including Gold customers, Silver customers, and Bronze customers.

- **Number of Items Bought by Gold**

It is the total number of items bought by all Gold customers in an experiment.

- **Number of Items Bought by Silver**

It is the total number of items bought by all Silver customers in an experiment.

- **Number of Items Bought by Bronze**

It is the total number of items bought by all Bronze customers in an experiment.

- **Total Lost Rate**

It is the ratio of total number of lost items to total number of clicked items. The total number of lost items is equal to the total number of clicked items minus the Total Number of Bought Items. The total number of clicked items is the number of the products that were put into shopping cart by a customer. These items may or may not be bought by the customer. We assume that if a customer clicks the “Submit” button in the Select State, two items are put in his/her shopping cart. So we call these items clicked items.

- **Discrimination Factor**

It is used to denote the differences between the End-to-End Times separated by customers in different classes. We use the following three formulas for computation of Discrimination Factors.

$$D_{\text{Gold/Gold}} = \frac{\text{Mean End to End Time of Gold jobs}}{\text{Mean End to End Time of Gold jobs}}$$

$$D_{\text{Gold/Silver}} = \frac{\text{Mean End to End Time of Silver jobs}}{\text{Mean End to End Time of Gold jobs}}$$

$$D_{\text{Gold/Bronze}} = \frac{\text{Mean End to End Time of Bronze jobs}}{\text{Mean End to End Time of Gold jobs}}$$

Note that $D_{\text{Gold/Gold}} = 1.0$.

- **Total Number of Arrivals During Experiment¹**

The Total Number of Arrivals During Experiment is the sum of number of unflagged jobs that are dispatched at the beginning, the number of flagged jobs and the number of unflagged jobs that are dispatched at the end. The unflagged jobs that are dispatched at the end are used for maintaining a consistent system load until all flagged jobs finish.

- **Experiment Time²**

Experiment Time is the time duration between the beginning of dispatching the first job and the completion of the last flagged job.

^{1, 2} Note: the Total Number of Arrivals During Experiment and the Experiment Time are not real performance metrics. They are put here because they are measured to validate our system (see Section 5.4.3).

End-to-End Time measurements reported have a confidence interval less than +/- 5% of the mean at a confidence level of 95% for most of the data points. The only exceptions are the measurements made for arrival rates of 1.1 and 1.2 requests per second.

5.4 Validations

5.4.1 Applying Little's Law

The correctness of the experimental system was verified using Little's Law. As we know, there are two ways to get the average number of jobs in the system. One way is that the average number of jobs in the system (N_0) equals sum of the residence times for all jobs divided by the experiment time. Another way is to apply Little's Law: $N_1 = \lambda W$, where N_1 is the average number of jobs in the system, λ is average measured Arrival Rate, and W is the average residence time of jobs in the system. In a correctly functioning experiment, N_0 and N_1 will be very close to one another. The difference between N_0 and N_1 is obtained using the following formula.

$$DP_i = \frac{\lambda_i * W_i - T_i / E}{T_i / E} * 100\%$$

Where $i = 0$ is for all job classes, $i = 1$ is for Gold jobs, $i = 2$ is for Silver jobs, and $i = 3$ is for Bronze jobs.

W_i : average residence time or average End-to-End Time

T_i : total residence time or total End-to-End Time

E : Experiment Time

λ_i : Arrival Rate. It is the number of marked jobs over experiment time.

$\lambda_i * W_i$: the number of jobs in the system by applying Little's Law.

DP_i : difference percentage.

This validation was performed for two sample experiments, leading to the results shown in Table 5-5.

Table 5-5: Difference Percentages

Difference Percentage	Mean Arrival Rate = 0.3	Mean Arrival Rate = 0.9
DP_0	-0.016%	-0.85%
DP_1	-0.033%	-1.51%
DP_2	-0.004%	-0.47%
DP_3	-0.005%	-0.74%

Note that the Arrival Rate λ_i is based on measured Arrival Rate that is slightly different from the input parameter Mean Arrival Rate.

5.4.2 The Same Priority Setting vs. the Different Priority Setting

The comparison between the same priority setting and the different priority setting demonstrates that the differentiated services can be provided by setting different priorities based on the classes of customers. If all customers are given to the same priority, which means that Gold, Silver and Bronze all have the same priority of 10 (see Figure 5-2), Gold, Silver, and Bronze jobs are treated with no differences. As expected the mean Request Times for all the classes shown in Figure 5-2 are close to one another.

Figure 5-2: Request Time with the Same Priority Setting (10-10-10)

Also, as expected, the mean Request Times for different classes of customers are different if the priority setting is different (3-10-17) (see Figure 5-3). Gold jobs demonstrate the best in Figure 5-3. This is because Gold jobs are always given the highest dynamic priority, which is “3” in this experiment. From this figure we also observe that the Request Time of Gold jobs ranges from 150 milliseconds to 400 milliseconds. The performance of Silver jobs in the different priority setting is close to that obtained when all classes have a priority of 10. Bronze jobs have the lowest priority setting, and demonstrate the worst performance. Figure 5-3 demonstrates that the relative performances of Gold, Silver, and Bronze are in line with our expectations.

Figure 5-3: The Same Priority Setting vs. the Different Priority Setting

5.4.3 Effect of Arrival Rate on Experiment Time and Total Number of Arrivals

During Experiment

Figure 5-4 presents the effect of Arrival Rate on Experiment Time. The Experiment Time that is needed to finish all the 3000 flagged jobs reaches the lowest value at Arrival Rate = 0.9 requests/second. As the Arrival Rate decreases, the Experiment Time increases. This relationship is true before the knee (Arrival Rate = 0.9 requests/second) of observed in Figure 5-4. The reason is that before the occurrence of the knee, the system does not completely reach saturation and the Experiment Time decreases as the Arrival Rate increase. After the knee Arrival Rate, the experiment time increases as the Arrival Rate increases. This is because the system reaches saturation at these higher Arrival Rate and some requests have to remain longer in the system. As a result the Experiment Time starts increase after reaching the Arrival Rate of 0.9 requests/second.

Figure 5-4: Effects of Arrival Rate on Experiment Time

Figure 5-5 demonstrates that the effects of Arrival Rate on the total number of arrivals during the experiment. At low loads (less than 0.9 requests/second), the Total Number of Arrivals During Experiment remains relatively flat (see Figure 5-5). After this point, the Total Number of Arrivals During Experiment increases dramatically. The reason is that if the load on the system rises to a high value (e.g. Arrival Rate > 0.9 requests/second), the system approaches saturation and residence time for each job is increased in a non-linear fashion. Therefore, the total residence time for all flagged jobs is increased. During this time more unflagged jobs are generated so that a consistent system load is maintained. As a result there is a non-linear growth in the total number of jobs that arrived during the experiment following the same trend as the mean residence time for jobs. The results shown in Figure 5-5 are also what we expected. It provides further validation for our system.

Note that if the client Arrival Rate is higher than 1.2 request/second, a “COMM_FAILURE” exception will be raised by omniORB3. The reason is that the connection is shutdown automatically in omniORB3 if a remote call has not completed within a defined period of time. (The default time at client sides is 60 seconds.) Even when the Arrival Rate is less than 1.2 requests/second, sometimes a “COMM_FAILURE” exception may be captured. This error may come from the Apache web server. When a great number of clients are present in the system, some customers may experience extremely long delays and may even get disconnected [1].

Figure 5-5: Effect of Arrival Rate on Total Number of Arrivals During Experiment

5.4.4 Fixed CORBA Service Time vs. Exponential CORBA Service Time

Since our experiments are primarily based on fixed CORBA service times, we need to investigate whether our system demonstrates the same performance trend for exponential CORBA service times. Figure 5-6 shows the effect of Arrival Rate on the average

Request Times for both fixed CORBA service time and exponential CORBA service time. The average Request Time is the average of the three Request Times - Gold, Silver, and Bronze. From Figure 5-6, we observe that both average Request Times of fixed CORBA service time and exponential CORBA service time are increased as the system load increases. At the low system load (e.g. Arrival Rate = 0.1 request/second), their average Request Times are almost the same. The reason is that the jobs can be served immediately after they arrive when the system load is low, and no queuing occurs. Therefore, they demonstrate the same result. But as the system load increases, the difference of the average Request Times between the fixed CORBA service time and exponential CORBA service time increases. This is because the job queue length for exponential CORBA service time increases more than for fixed CORBA service time due to increased variability in service times.

Figure 5-6: Fixed Service Time vs. Exponential Service Time

5.4.5 Effect of Different Balk Time Settings on Total Number of Items Bought

If all customers' behaviors are influenced only by the balk time, the effect of different balk time settings on the Total Number of Bought Items (see Figure 5-7) can be clearly shown. In this experiment, we assume that all customers leave the system only when the response time of a page is larger than the Balk Time. So the probabilities of P_0 , P_1 , P_2 , and P_5 are set to 1 (see Figure 3-14), and the go-back probabilities of P_3 , P_4 are set to 0.

Figure 5-7: Effect of Balk Time on Total Number of Items Bought

Balk Time is the maximum time duration that a customer can wait to get a response from the web server. The longer Balk Time, the more customers in the system. When the Balk Time is less than 12000 milliseconds, the Total Number of Items Bought increases as the Balk Time increases. This is because the system does not reach its saturation state before the knee value of 12000 milliseconds. But if the Balk Time is larger than 12000 milliseconds, the Total Number of Items Bought will decrease as the Balk Time increases. This is because the system reaches its saturation state and each customer waits

longer to complete a purchase. Although more and more customers choose to stay in the system, the Total Number of Items Bought still decreases. Since the last step that may be balked contains six CORBA method calls (see Figure 3-11), it is the longest Request Time among the first four³ URL requests. Many more customers may finish the first three steps at a longer Balk Time, but many more of them may quit at this step. This is the reason that the Total Number of Items Bought decreases at Balk Time 16000 milliseconds. Therefore, Figure 5-7 demonstrates further validation on our system.

³ Note: As discussed in Section 3.7.3, the fifth URL request, which dynamically generates the Thanks page, does not have a Balk Time associated with it. For instance, if a customer clicks the submit button in the Pay state after he/she enter his/her credit card number, the customer will eventually get the Thanks page no matter how big the Thanks page response time is.

Chapter 6 Results of Experiments

This chapter presents the results of our experiments. The experiments are divided into two groups: one is based on the CBMG with Random Behavior (CBMG-RB) model, and the other is based on the CBMG with Random and Balk Behavior (CBMG-RBB) model. In the next two sections, the experiment results from the two groups are discussed respectively.

6.1 Experiments Based on the CBMG-RB Model

The experiments from Section 6.1.1 to Section 6.1.5 are purely based on the approach discussed in Section 3.7.2, where the priority of threads are adjusted in the CORBA back end server. The QoS parameters of customers described in Section 3.7.2 are obtained from user service levels. They must be passed to CORBA back end servers to provide quality of service. In this implementation, a QoS parameter is passed as an “in” parameter of a CORBA method since each CORBA method implemented in this study has an additional argument for the QoS. In the CORBA server side, the first task of each method is to set a specific thread priority according to the QoS passed in. In omniORB, each CORBA request is served by a dedicated thread, so each thread’s dynamic default priority which is zero must be reset according to the value of the QoS. System call “setpriority” is used to reset current thread priority.

6.1.1 Effect of Varying Client Arrival Rate on Request Time

Figure 6-1 shows the effect of client Arrival Rate on Request Time. From Figure 6-1, we observe that the Gold customer Request Time is always less than 10 seconds, even at a higher Arrival Rate. But the Bronze customer Request Time is six times as long as the Gold Request Time at the highest Arrival Rate. Also, when Arrival Rate is higher, the Bronze customer Request Times is sharply increased, but the Gold customer Request Time is not. This clearly demonstrates that different QoS requirements can be supported by our system.

Figure 6-1: Effect of Arrival Rate on Request Time

6.1.2 Effect of Varying Client Arrival Rate on End-to-End Time

Figure 6-2 shows the effect of client Arrival Rate on End-to-End Time. The End-to-End Time is approximately 10 seconds larger than the Request Time (see Figure 6-1 and

Figure 6-2). The reason is that the End-to-End Time includes Think Times, which are associated with the first four states in this e-commerce application. The average Think Time in a session is about 10 seconds. This is because 1) the default Think Time in each state is 5 seconds, 2) there are four states associated with the Think Time, 3) the average number of states visited by a customer is about two. As expected, the End-to-End Time should follow the same trend as the Request Time (see Figure 6-2).

Figure 6-2: Effect of Arrival Rate on End-to-End Time

6.1.3 Effect of Different Priority Settings

This section investigates the effect of different Priority Settings on End-to-End Time. The different Priority Settings, which are based on Priority Setting 3-10-17 and Priority Setting 7-10-13 respectively, may provide different QoS that can be captured by using the discrimination factors. From the discrimination factors shown in Figure 6-3 and Figure 6-4, we observe that if the priorities of the three different classes are set to be closer to each other, the discrimination factors are closer to each other as well. Closer

discrimination factors mean that the system treats different user service level's customers (e.g. Gold, Silver, and Bronze) with less difference, and the wider discrimination factors mean that the system treats different classes of customers with bigger differences. So the priority setting value can be tuned to satisfy the requirements of a specific e-commerce site.

Figure 6-3: Discrimination Factors at Priority Setting 3-10-17

Figure 6-4: Discrimination Factors at Priority Setting 7-10-13

6.1.4 Effect of Varying Go-Back Probability Setting

The Go-Back Probabilities (P_3 and P_4) discussed in Figure 3-13 are set to zero in most of our experiments since these transition probabilities do not influence providing different services based on customers' QoS requirements.

Figure 6-5 and Figure 6-6 show the same behavior as seen so far for non-zero go-back probabilities. This means that the differentiated services are given to different classes of customers for both go-back probability settings, at both low load (Arrival Rate=0.5 requests/second) and at high load (Arrival Rate=0.7 requests/second).

From Figure 6-7, we observe that the Request Time of Gold customers is much higher with go-back probabilities set to 0.25 than those set to zero. The reason is that when the go-back probabilities are set to non-zero value (0.25), more pages' Request Time is added to the total Request Time, therefore, the average Request Time is increased. Also, the system load is increased with these non-zero probability settings.

Figure 6-5: Effect of Varying Go-Back Probabilities with Arrival Rate 0.5

Figure 6-6: Effect of Varying Go-Back Probabilities with Arrival Rate 0.7

Figure 6-7: Effect of Varying Go-Back Probabilities on Gold Request Time

6.1.5 Operation-by-Operation Performance Comparisons

An operation-based response time corresponds to a CORBA method response time. It is the time duration between the end of a CORBA method request submission and the end of the corresponding response from the CORBA server. From the message sequence charts discussed in Section 3.4, we know that the number of calls and the level of nesting are different within each operation. Now we would like to investigate the CORBA response times of different classes of customers for these three operations: One Direct Call, Single Nested Call, and Double Nested Call.

The diagram for the One Direct Call is given in Figure 6-8. The operation's CORBA response times of different classes of customers are presented in Figure 6-9.

Different response times (see Figure 6-9) are provided for different classes of customers, and this is held from the lowest Arrival Rate of 0.3 requests/second to the highest Arrival Rate of 0.9 requests/second. So, the QoS requirements in the e-commerce system are supported as described for the One Direct Call operation with increasing discrimination for increasing load.

Figure 6-8: One Direct Call Operation Diagram

Figure 6-9: Response Time of One Direct Call

The response times observed in Figure 6-11 are different than those in Figure 6-9. The same response time is observed for different classes of customers at the lower Arrival Rates of 0.3 requests/second and of 0.5 requests/second, while service differentiation occurs at the higher Arrival Rates of 0.7 requests/second and of 0.9 requests/second, where greater resource contention occurs.

Figure 6-10: Single Nested Call Operation Diagram

Figure 6-11: Response Time of Single Nested Call

The operation diagram of Double Nested Call is given in Figure 6-12, and the CORBA response times for different classes of customers are presented in Figure 6-13. The response times of the Double Nested Call follow the same trend as those of the One Direct Call.

Figure 6-12: Double Nested Call Operation Diagram

Figure 6-13: Response Time of Double Nested Call

6.1.6 Comparisons Between CGI-nice-on and CGI-nice-off

The CGI-nice-off experiment is also only based on the second technique discussed in Section 3.7.2. But the CGI-nice-on experiment is based on the two techniques discussed in Section 3.7.1 and Section 3.7.2 respectively. The first technique involves setting the priority of the CGI process, based on the QoS parameter associated with the customers using the nice() system call. CGI-nice-on means that the middle tier is providing QoS now, and CGI-nice-off means that there is no priority enhancement technique applying in the middle tier.

Again the performance metric Discrimination Factor is chosen to observe the differences between CGI-nice-off and CGI-nice-on.

Figure 6-14: Discrimination Factors with CGI-nice-off

Figure 6-15: Discrimination Factors with CGI-nice-on

From the discrimination factors provided in Figure 6-14 and Figure 6-15, we can see the differences between CGI-nice-on and CGI-nice-off. $D_{Gold/gold}$, $D_{Silver/Gold}$ and

$D_{\text{Bronze/Gold}}$ in Figure 6-14 are much closer to each other than those in Figure 6-15. This means that if CGI-nice-switch is turned on, the differentiated services can be enhanced even in the middle tier – the application server. This is a technique to further differentiate the end to end QoS requirements in the e-commerce system. For details on the technique see Section 3.7.1.

Note that the system call “nice” is used to set the priorities of the CGI processes, and this makes performance of Gold, Silver, and Bronze with CGI-nice-on worse than that achieved with CGI-nice-off. Since there are web server processes and background processes running with these CGI processes on the same machine, and the priority set by using system call “nice” is lower than all other processes’ priorities, the End-to-End Time with CGI-nice-on is worse than that with CGI-nice-off.

6.2 Experiments Based on the CBMG-RBB Model

This section discusses the results of experiments from the CBMG-RBB Model. The third technique proposed in this thesis is studied here. Based on e-business-oriented performance metrics such as Number of Items Bought and Total Lost Rate, the differences between State-based-off and Stated-based-on are presented.

The third technique provides the ability to control priority setting dynamically according to the shopping state of a customer. If a customer is in a key state, better service should be provided no matter which user service level he/she belongs to. The key state in this application is the Confirm State (see Section 3.7.3 for choosing a key state). The “Submit” button clicking in the key state corresponds to activating a CGI process. The QoS parameter that was set before is based on user service level and must be reset to

support the third technique. The CGI process passes the reset QoS parameter into the CORBA back end servers. Then the subsequent CORBA requests are based on the new QoS value. In this study, the new QoS value is reset to the highest priority (Gold priority) in the key state (see Table 6-1). This technique is intended to prevent revenue from being lost from our e-commerce site (see the experiment results in the following sections); it is particularly useful when the load on the system is moderate.

Table 6-1: Priority Setting in Different States

State Class	State 1 Login	State 2 Select	State 3 Confirm	State 4 Pay	State 5 Thanks
Gold	3	3	3	3	3
Silver	10	10	3	10	10
Bronze	17	17	3	17	17

6.2.1 Effect of State-Based-On/Off on Total Number of Items Bought

When the State-based-switch is turned on, the Total Number of Items Bought shown in Figure 6-16 demonstrates that the Total Number of Items Bought does not decrease. This means that the third technique does not have the negative effect on the revenue of the e-commerce site. Sometimes, it may help to increase the revenue for the e-commerce site.

From Figure 6-16, we observed that the Total Number of Items Bought increases by nearly 240 items (9.8% improvement) at an Arrival Rate of 0.3 requests/second and the Total Number of Items Bought increases by nearly 100 items (8.3% improvement) at an Arrival Rate of 0.5 requests/second when the State-based-switch is turned on. However there is a smaller increase in the Total Number of Items Bought at higher Arrival Rates

(e.g. Arrival Rate = 0.7 requests/sec and Arrival Rate = 0.9 requests/second) and at lower Arrival Rate (e.g. Arrival Rate = 0.1 requests/second).

Figure 6-16: Effect of State-based-on/off on Total Number of Items Bought

6.2.2 Effect of State-Based-On/Off on Number of Items Bought by Different Classes

From Figure 6-17 and Figure 6-18, we observe that the differences among the Number of Items Bought by different classes of customers is slightly closer to each other with State-based-on than those with State-based-off. This is because whenever a Silver customer or a Bronze customer is in the Confirm State, his/her QoS parameter is reset to the highest priority 3. The result is that more chances are given to Silver and Bronze customers to finish their purchases and more Gold customers are forced out from the purchase list. Therefore, the differences among the values of items bought by different classes of customers are decreased when the State-based-switch is turned on.

Figure 6-17: Number of Items Bought by Different Classes with State-based-off

Figure 6-18: Number of Items Bought by Different Classes with State-based-on

Note that at the lowest system load (e.g. Arrival Rate = 0.1 requests/second) in Figure 6-17 and Figure 6-18, the three values of the Number of Items Bought by different classes of customers are almost equal. The reason is that there is little contention for resources and little balking takes place. Since Gold, Silver and Bronze customers are evenly distributed in the system and the behaviors of all different classes of customers are same, the items bought by different classes of customers tend to be the same. Even when the State-based-switch is turned on, these items will not increase since almost no customer is balked. This means that State-based-on mechanism has no influence on Number of Items Bought when system load is very low.

At the higher system load (e.g. Arrival Rate = 0.9 requests/second), the three values of the Number of Items Bought by different classes of customers all decrease because of poor performance and the large number of balked customers. The result is that the three numbers are close to each other again. Also, when the State-based-switch is turned on, smaller benefit is achieved in comparison with State-based-off since the group of non-balked customers is very small. So, State-base-on results in lesser revenue improvement when system load is very high. A more detailed comparison between State-Based-on and State-Based-off is presented next.

6.2.3 Class-by-Class Comparisons Between State-Based-On/Off on Number of Items Bought

This section compares the differences between the Number of Items Bought by Gold, by Silver and by Bronze with State-based-on and the Number of Items Bought by Gold, by Silver and by Bronze with State-based-off respectively.

Figure 6-19: Effect of State-based-on/off on the Number of Items Bought by Gold

The Number of Items Bought by Gold customers in Figure 6-19 is observed to decrease as the Arrival Rate increases. The reason is that the number of customers in the system increases as the Arrival Rate increases and this makes more customers be balked before finish purchases. But the Number of Items Bought by Gold decreases at any system load as the State-base switch is turned on from off. The reason for the negative effect on Gold customers is that if the State-based-switch is turned on, more CPU time is given to Bronze and Silver customers in the key state (see Table 6-1) than when the State-based-switch is turned off. As the Number of Items Bought by Gold decreases, the Number of Items Bought by Silver and the Number of Items Bought by Bronze increases (see Figure 6-20 and Figure 6-21) at all load levels.

Figure 6-20: Effect of State-based-on/off on the Number of Items Bought by Silver

Figure 6-21: Effect of State-based-on/off on the Number of Items Bought by Bronze

6.2.4 Effect of State-Based-On/Off on Total Lost Rate

The Total Lost Rate shown in Figure 6-22 gives us more clear conclusions about the dynamic priority setting technique. For example, a significant decrease in the Total Lost Rate is obtained at moderate system load (e.g. Arrival Rate = 0.3 requests/second and Arrival Rate = 0.5 requests/second) when the State-based-switch is turned on. The smaller the Lost Rate, the smaller the revenue lost. Choosing different dynamic state priority settings on a key state can also help tune the lost rate of different classes of customers. However, it is always true that the total lost rate is reduced if all three classes of customers have the same buying probability.

Figure 6-22: Effect of State-based-on and State-based-off on Total Lost Rate

Only two data points are obtained for the system that does not use customer balking. At higher arrival rates for the given set of workload parameters, the system ran out of file descriptors before the completion of experiments.

Note that for the two data points, the Total Lost Rate does not change (nearly 40% lost) when we set the Balk Time to unlimited (see Without balk in Figure 6-22). From comparisons between balk behavior (based on the “8 second rule”, Balk Time = 8 seconds) and without balk behavior (Balk Time = unlimited), we observe that both revenues and customers are lost with customer balking because a high percentage of frustrated customers abandon the site due to the poor performance. Again, the good performance of an e-commerce site can really help to attract customers and increase revenue.

Chapter 7 Conclusions

Section 7.1 provides a summary of the work undertaken in this thesis. Section 7.2 gives the conclusions that are drawn from the experimental results. Finally, Section 7.3 offers directions for future research.

7.1 Summary

In order to maximize the revenue generated by a site and support its business goals, this thesis provides three techniques to preserve end-to-end QoS and enhance system performance for a CORBA-based e-commerce application. The three techniques are:

- Preserving priority in the middle tier - the web server, achieved by setting priority in the CGI processes.
- Preserving thread priority in the third tier – the CORBA back end servers, achieved by setting priority in the CORBA servers.
- Controlling priority dynamically in the third tier according to the state that a customer is in.

The performance prototype described in the thesis is based on the popular open source Apache web server and OmniORB, running under Red Hat Linux 6.2, kernel 2.2.16-3 on 266MHz Pentium II PC's with 64MB of RAMs. The PC's are interconnected by a 100 MB Ethernet LAN. We have used a prototype CORBA-based E-commerce Web Application, the Apache web server, and a synthetic open model

workload generation tool. By using this performance prototype, this thesis has analyzed the effect of different techniques for preserving end-to-end QoS on system performance. This prototyping and measurement based approach is selected as it captures the effect of system overheads that are difficult to capture accurately in analytic or pure simulation models.

The CORBA-based E-commerce Web Application prototype implemented in this thesis provides functions such as login, display product information, add/remove products to/from shopping carts, and credit card checking. C++ is chosen to implement the application because C++ can be used as a CGI gateway language and can be used to develop CORBA servers and clients easily. The back-end servers consist of six CORBA objects and run on a Linux Pentium II machine. Each method in CORBA objects has an “in” parameter to pass the QoS information, so the different priority values can be set based on this parameter. In the dynamic priority setting technique, the QoS information passed to CORBA objects is not solely obtained based on the QoS database. As a customer progresses through the site, the priority is changed dynamically according to the customer’s state.

A workload generation tool, which is also developed in this thesis, is used to simulate the workload produced by a client browser. It uses the HTTP 1.1 GET method to request a page of the CORBA-based E-commerce Web Application and it measures Request Time, End-To-End Time, Number of Items Bought and Total Lost Rate. The workload generation tool is implemented based on an open model, since the open model has the ability to better capture the characteristics of an e-commerce application as compared to a closed model. Our experiments are based on two different models: CBMG with Random

Behavior model and CBMG with Random and Balk Behavior model. Response time measurements have a confidence interval less than $\pm 5\%$ of the mean response time at a confidence level of 95% for all but a few measurements at very high load (see Section 5.3 for details).

Through comparisons between different priority settings, between CGI-nice-on and CGI-nice-off, and between State-based-on and State-based-off, we have obtained valuable insights into system behavior and performance.

The next section summarizes the insights derived from the results of the experiments presented in Chapter 6 of the thesis.

7.2 Conclusions

A complete end-to-end e-commerce QoS solution cannot be achieved without application support on the end hosts. Simulation results shown in Chapter 6 demonstrate that the three techniques are effective in attracting customers and improving revenue for a three-tiered e-commerce web site. Any one of or a combination of these techniques can be applied to an e-commerce site according to the site's needs. The insights gained into the effect of these techniques on system behavior and performance are summarized.

- If a site does not need to support QoS requirements, the dynamic priority controlling technique may really help increase the revenue since good service is always given to customers who have higher probability of buying. But identifying a key state needs to be done carefully since a negative effect may occur if a wrong key state is selected.

- Usually, Gold customers are the heaviest buyers, Silver customers are next, and Bronze customers are the least in an e-commerce site. The middle tier QoS preserving technique and/or the third tier QoS preserving technique may be the best choice for these kind of sites. But if marketing efforts are such that Silver customers and Bronze customers are expected to become heavier buyers, the dynamic QoS controlling technique can be added to improve revenues in the face of balking.
- Both QoS preserving techniques in the middle tier and in the third tier can achieve the same purpose – supporting QoS requirements. But sometimes, choosing to implement it may be more useful in one tier than the other. For example, if the processes in the middle tier are heavy consumers of CPU resource, then choosing the middle tier QoS setting technique may obtain more significant results for improving QoS. Similarly, if the processes in the last tier are heavy consumers of CPU resource, then choosing the last tier QoS setting technique may be more desirable.
- If customers in a site vary from very heavy buyers (e.g. companies) to light buyers (e.g. individuals), enhanced QoS preserving techniques should be provided. This means that both the middle tier QoS preserving technique and the third tier QoS preserving technique should be applied at the same time.
- The different priority values (e.g. 3, 10 and 17) are set based on QoS parameters. Different value setting may generate different QoS. For example, closer priority values (e.g. 7, 10 and 13) generate lesser differences among different classes of customers and more widely different priority values (e.g. 3, 10 and 17) generate bigger difference among customers. These values can be tuned from time to time with the changes of characteristics of an e-commerce site.

Three interesting performance metrics, namely one QoS-oriented performance metric – Discrimination Factor, and two e-business-oriented performance metrics – Number of Items Bought and Total Lost Rate, which are explored in this thesis, can be applied to any e-business site to demonstrate its QoS and performance.

Although the QoS provided in this prototype are only based on three levels (Gold, Silver and Bronze), more than three levels of QoS can be easily implemented by adapting the idea of the three levels of QoS.

Also, the insights resulting from this thesis can be extended to other types of QoS-enabled applications. They can be implemented without changing the underlying systems such as the Apache web server, and the omniORB. Moreover, differentiated QoS can be provided without using a real time operating system or a real-time ORB.

CORBA has already been used by many organizations, and thousands of CORBA applications are currently in use. Netscape has made CORBA a cornerstone technology [41]. So the techniques implemented in our thesis have a promising future.

The next section suggests directions for future research.

7.3 Future Research

Future research falls in three directions: 1) the performance prototype developed in this thesis, 2) the next generation of Apache web server, 3) the Real-time CORBA.

The mean Arrival Rate for different classes of customers is the same in all our experiments. In the real world, it may not be true. For example, sometimes, the Gold customers may visit the site more frequently than other customers. So the effect of

different Arrival Rates based on Gold, Silver, and Bronze customers on our experiment results needs to be investigated further.

The third technique discussed in this thesis is based on a shopping state of a customer. This technique demonstrates revenue improvement for our e-commerce site. A similar technique, which is based on the number of items in a customer's shopping cart, needs to be investigated since it may improve the revenue also. For example, a customer's priority may dynamically be set as the number of items in his/her shopping cart changes.

At the Apache web server processes level, we do not preserve all web server's priorities based on the QoS parameters of customers. We only set priorities of the CGI processes spawned by the Apache server. If there is a way that the QoS parameters can be passed into web server processes (including web reverse proxy [5] [12] server processes and web back end server processes), we can enhance the QoS end to end. Also, the next generation of Apache web server, Apache 2.0, is undergoing testing and will be available in the future. The most visible and noteworthy enhancement beyond Apache 1.3 that has been used in this thesis is the ability to run a web server in a hybrid thread/process model on Linux systems. The hybrid solution provides the reliability that comes with using multiple machines and processes. This is combined with the scalability that threads provide. Investigating the performance of a prototype-based on such a system is an important direction for future research.

Using priorities based on class may lead to "starvation" of certain customers. Using aging techniques that slowly increase the priority of customers waiting for a long time are important and need further research. The impact of different priority settings on End-to-

End Time has been analyzed. How different priority settings affect the revenue generated by the different customer classes seems to be a interesting question.

As captured in Figure 6-2 and Figure 6-3, the performance for Bronze customers start deteriorating much earlier in comparison to the Silver and Gold customers. Using an appropriate admission control strategy that prevents too many Bronze customers from entering the system at high load can lead to a better utilization of system resources and more uniformity in discrimination factors attained by the different classes.

The AT&T laboratory is enhancing omniORB3 that has been used in this thesis to include some notion of “quality of service” [14]. How to adapt a QoS-enabled application to the Real-time CORBA compliant omniORB would be also an important direction for further research.

Strategies for assigning a customer to a class need investigation. Using past history of existing customers and providing frequent buyers with a higher QoS as well as having effective techniques for encouraging new customers to use the system are important in this context.

Appendix

The pseudo code of CBMG-RB-based workload generation tool is shown in Figure

A-1.

Figure A-1: Pseudo Code of CBMG-RB-based Workload Generation Tool

The pseudo code of CBMG-RBB-based workload generation tool is shown in Figure A-2.

Figure A-2: Pseudo Code of CBMG-RBB-based Workload Generation Tool

References

- [1] Apache web server documentation. Available from: <http://www.apache.org>.
- [2] P. Barford and M. Crovella Computer Science Department Boston University "Generating Representative Web Workloads for Network and Server Performance Evaluation". In *Proceedings of the ACM SIGMETRIC International Conference on Measurement and Modeling of Computer Systems*, Madison, U.S.A., June 1998, pp.69-79.
- [3] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, O'Reilly and Associates, 2001.
- [4] Common Gateway Interface Documentation web site. Available from: <http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>.
- [5] R. S. Engelschall, "Load Balancing Your Web Site", *Web Techniques Magazine*, Vol.3, Issue 5, May 1998. Also available from: <http://www.webtechniques.com/archives/1998/05/engelschall/>.
- [6] GNU cgicc – A C++ library for CGI programming. Available from: <http://www.gnu.org/software/cgicc/cgicc-doc.html>
- [7] R. Fielding, J. Gettys, J.Mogul, H.Frystyk, T.Berners-Lee, et al., "RFC 2068: Hypertext transfer protocol – HTTP/1.1," January 1997.
- [8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [9] M. Henning and S. Vinoski, *Advanced CORBA Programming With C++*. Addison-Wesley Longman, 1999.
- [10] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, 1991.
- [11] E. D. Jensen, "Eliminating the Hard/Soft Real-Time Dichotomy," *Embedded Systems Programming*, Vol.7, No.10, Oct. 1994, pp.17-25.
- [12] B. Laurie, and P. Laurie, *Apache- The Definitive Guide*, O'Reilly & Associates, 1999.
- [13] Linux Programmer's Manual. Available from the man page of Linux 6.2, kernel 2.2.16-3.

- [14] S. Lo and S. Pope, "The Implementation of a High Performance ORB over Multiple Network Transports", in *Proceedings of IFIP Middleware '98*, The Lake District, England, September 1998, pp.157-172.
- [15] J. Mehta, "A CORBA Based Web Server Distributed Computing Meets the Web", *Web Techniques Magazine*, Vol.2, Issue 1, January 1997. Also available from: <http://www.webtechniques.com/archives/1997/01/mehta/>.
- [16] D. A. Menasce and V. A. F. Almeida, *Scaling for E-Business Technologies, Models, Performance, and Capacity Planning*, Prentice Hall, 2000.
- [17] D. A. Menasce, V. A. F. Almeida, R. C. Fonseca, and M. A. Mendes, "Resource Management Policies for E-commerce Servers" in *Proceedings of the 2nd workshop on Internet Server Performance*, Atlanta, May 1999.
- [18] D. A. Menasce, V. A. F. Almeida, R. C. Fonseca, and M. A. Mendes, "A Methodology for Workload Characterization for E-commerce Servers," in *Proceeding 1990 ACM Conference in Electronic Commerce*, Denver, U.S.A., November 1999, pp.119-128.
- [19] D. A. Menasce and R. Dodge, "Preserving QoS of E-commerce Sites Through Self-Tuning: A Performance Model Approach," in *Proceedings of the 3rd ACM Conference in Electronic Commerce*, Tampa, U.S.A., October 2001, pp.224-234.
- [20] P. Merle, C. Gransart, and J. Geib, "CorbaWeb: A Generic Object Navigator", Journal reference in *Computer Networks and ISDN Systems*, Vol.28, No.7-11, May 1996, pp.1269-1281.
- [21] Microsoft Corporation, "DCOM Technical Overview", Technical Report, November 1996.
- [22] D. Mosberger and T. Jin, "httpperf – A Tool to Measuring Web Server Performance", in *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, Monterey, U.S.A, December 1997, pp.59-67.
- [23] S. Nadimpalli, "High Performance Web Servers", M. Eng. Thesis, Department of System and Computer Engineer, Carleton University, Ottawa, Canada, 2000.
- [24] Object Management Group, *Dynamic Scheduling*, OMG Document orbos/99-03-32 ed., March 1999.
- [25] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.3 ed., June 1999.

- [26] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, 2.4.2 ed., February 2001.
- [27] OmniORB document. Available from:
<http://www.uk.research.att.com/omniORB/doc/3.0/omniORB/omniORB001.html#toc1>
- [28] R. Orfali and D. Harkey, *Client/Server Programming with JAVA and CORBA*, Wiley Computer Publishing, 1998.
- [29] C. O’Ryan and D. C. Schmidt “Evaluating Policies and Mechanisms to Support Distributed Real-time Applications with CORBA”, *The Journal of Theory and Practice of Object Systems, Special Issue on Distributed Objects and Applications*, Wiley and Sons, Vol.3, No.2, February 2001.
- [30] L. Pautet, T. Quinot and S. Tardieu, “CORBA and CORBA services for DSA”, in *Proceedings of the Annual International Conference of ACM SIGAda’99*, Paris, France, October 1999, pp.31 – 38.
- [31] G. S. Raj, “A Detailed Comparison of CORBA, DCOM and Java/RMI (with specific code examples)”, in *Microsoft visual J++ Informant*, September 1999.
- [32] G. S. Raj, “Emerging Technologies for eCommerce”, Available from:
<http://gsraj.tripod.com/etorgfit.html>
- [33] W. E. Rajput, *E-Commerce Systems Architecture and Applications*, Artech House, 2000.
- [34] G. Reese, *Database Programming with JDBC and Java*, O’Reilly & Associates, 1997.
- [35] Roth Consulting web page. Available from: <http://www.roth.net/perl/odbc/faq/>
- [36] D. C. Schmidt and F. Kuhns, “An Overview of the Real-time CORBA Specification”, in *Proceeding of the IEEE Computer special issue on Object-Oriented Real-time Distributed Computing*, Vol.33, No.6, June 2000, pp.56-63.
- [37] D.C. Schmide, D.L. Levine, and S. Mungsee, “The Design and Performance of Real-time Object Request Brokers”, *Computer Communications*, Vol.21, No.4, April 1998, pp.294-324.
- [38] Transaction Processing Monitor Technology - Software Technology Review, Available from: http://www.sei.cmu.edu/str/descriptions/tpmt_body.html

- [39] University of Bielefeld, Faculty of technology web page. Available from: <http://www.rvs.uni-bielefeld.de/research/MFGfor/>.
- [40] S. Wang, Y. C. Wang, and K. J. Lin, "A General Scheduling Framework for Real-Time Systems," in *Proceeding of IEEE Real-Time Technology and Applications Symposium*, IEEE, Phoenix, U.S.A., June 1999, pp.246-255.
- [41] White paper, "CORBA Catching the Next Wave". Available from: <http://developer.netscape.com/docs/wpapers/corba/>
- [42] White paper, "E-business, Success on a Sound Software Architecture". Available from: <http://www.omg.org/news/about/marketing.htm>.
- [43] S. Williams and C. Kindel, "The Component Object Model: A Technical Overview", adapted from an article appearing in *Dr. Dobbs's Journal, the Newsstand Special Edition*, December 1994. Also available from: http://msdn.microsoft.com/library/en-us/dncomg/html/msdn_comppr.asp.
- [44] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, 1949.