PERFORMANCE MODELLING AND ANALYSIS OF E-COMMERCE SYSTEMS USING CLASS BASED PRIORITY SCHEDULING

I. T. NAFEA

PhD

June, 2012

I

Performance Modelling and Analysis of E-Commerce Systems Using Class Based Priority Scheduling

An investigation into the development of new class based priority scheduling mechanisms for e-commerce system combining different techniques

Ibtehal Talal Nafea

Submitted for the degree of

Doctor of Philosophy

Department of Computing

School of Computing, Informatics and Media

University of Bradford

2012

Abstract

Ibtehal Talal Nafea

Performance Modelling and Analysis of E-Commerce Systems Using Class Based Priority Scheduling

Recently, technological developments have affected most lifestyles, especially with the growth in Internet usage. Internet applications highlight the E-commerce capabilities and applications which are now available everywhere; they receive a great number of users on a 24-7 basis because online services are easy to use, faster and cheaper to acquire. Thus E-commerce web sites have become crucial for companies to increase their revenues. This importance has identified certain effective requirements needed from the performance of these applications. In particular, if the web server is overloaded, poor performance can result, due to either a huge rate of requests being generated which are beyond the server's capacity, or due to saturation of the communication links capacity which connects the web server to the network.

Recent researches consider the overload issue and explore different mechanisms for managing the performance of E-commerce applications under overload condition.

This thesis proposes a formal approach in order to investigate the effects of the extreme load and the number of dropped requests on the performance of E-

commerce web servers. The proposed approach is based on the class-based priority scheme that classifies E-commerce requests into different classes. Because no single technique can solve all aspects of overload problems, this research combines several techniques including: admission control mechanism, session-based admission control, service differentiation, request scheduling and queuing model-based approach.

Request classification is based on the premise that some requests (e.g. buy) are generally considered more important than others (e.g. browse or search). Moreover, this research considers the extended models from Priority Scheduling Mechanism (PSM). These models add a new parameter, such as a review model or modify the basic PSM to low priority fair model, after the discovery of ineffectiveness with low priority customers or to add new features such as portal models.

The proposed model is formally specified using the π -calculus in early stage of models design and a multi-actor simulation was developed to reflect the target models as accurately as possible and is implemented as a Java-based prototype system.

A formal specification that captures the essential PSM features while keeping the performance model sufficiently simple is presented. Furthermore, the simplicity of the UML bridges the gap between π-calculus and Java programming language.

There are many metrics for measuring the performance of E-commerce web servers. This research focuses on the performance of E-commerce web servers that refer to the throughput, utilisation, average response time, dropped requests and arrival rate. A number of experiments are conducted in order to test the performance management of the proposed approaches.

Keywords: E-commerce-Performance-Priority Scheduling Mechanism-Overload

Declaration

I hereby declare that this thesis has been genuinely carried out by myself and has not been used in any previous application for a degree. The invaluable participation of others in this thesis has been acknowledged where appropriate.

Ibtehal Talal Nafea

Dedication

To my Parents, my husband and my lovely children

Acknowledgements

In the name of Allah, the Most Gracious and the Most Merciful. Alhamdulillah, all praises to Allah, the Almighty, for the strengths and His blessing in completing this thesis.

I would like to express my deep gratitude and appreciation to a number of people for their contributions to this research and for their support and encouragement. Much credit for this research should go to my advisors, Prof. Irfan Awan and Dr. Rob Holton, for their continuous and generous support and guidance. Working under their supervision has been a most rewarding and wide experience for me because of their enthusiasm, helpfulness and encouragement especially Dr.Holton his strong encouragement as well as the confidence he placed in me, especially when things were looking down, is greatly acknowledged and appreciated. I am also deeply indebted to my external supervisor Dr. Muhammad Younas for his invaluable advice and careful reading. His timely and efficient contribution helped me shape this into its final form and I express my sincerest appreciation for his assistance in any way that I may have asked.

Many thanks go to Miss Rona Wilson from the Computing department for her co-operations.

I owe a special debt of gratitude to my beloved husband, Yousef Abdoh, and my wonderful children Ibrahim, Abdulrahman and Bushra, who have stood by me, supported me and patience throughout this long process. Without them, life would seem meaningless. Greatest thanks are given to my parents who have done so much for me. Though, thousands of kilometres away, they never stopped their love, prayers and support. And special thanks to my sisters and brothers for their generous support. To those who indirectly contributed in this research, your kindness means a lot to me. Thank you very much.

Finally, I give my greatest thanks to my colleagues and friends, Dr. Iman AlAnsari, Dr.Rasha Fares, Dr. Amina Alsawaai and Reem Darwesh for their supports and encouragements. Thanks for the friendship and memories.

Publications

1- Nafea, D.R.W. Holton, M. Younas and I. Awan. “Synthesis of Performance Management Mechanisms in Modern E-Commerce Services”, iiWAS2010, 8-10 November, 2010, Paris, France. Copyright 2010 ACM 978-1-4503-0421-4/10/11.

2- I. Nafea, D.R.W. Holton, M. Younas and I. Awan. “A Formal Approach to Investigate the Performance of Modern E-Commerce Services”, ASMTA 2010, LNCS 6148, pp. 233–246, 2010.

3- D.R.W. Holton, I. Nafea, M. Younas, and I. Awan. “A Class-based Scheme for Ecommerce Web Servers: Formal specification and Performance Evaluation”. Accepted by the Journal of Network and Computer Applications, Elsevier, Vol 32 (2), March 2009, 455-4602.

4- I. Nafea, D.R.W. Holton, I. Awan and M. Younas. “Priority Scheduling of requests to eCommerce Application”, The 9th Annual Postgraduate Symposium The Convergence of telecommunications, Networking and Broadcasting, PGNet Conference, Liverpool John Moores University, 23rd-24th June 2008.

Table of Content

List of Figures

List of Tables

LIST OF ABBREVIATIONS

CAC	Connection Admission Control
CBMG	Customer Behaviour Model Graph
cdf	cumulative distribution function
CoSAC	Coordinated Session Based Admission Control
CCS	Calculus of Communicating Systems
DAC	Dynamic Admission Control
DRA	Dynamic Resource Allocation
DOC	Distributed Object Computing
DTMC	Discrete Time Markov Chain
DWFS	Dynamic Weighted Fair Sharing
FCFS	First Come First Served
FIFO	First-In First-Out
GK	Gate Keeper

GPS	Generalised Processor Sharing
IP	Internet Protocol
OFDMA	Orthogonal Frequency Division Multiple Access
OS	Operating System
RDRP	Reward Driven Request Prioritisation
RTT	Round Trip Times
ROS	Random Order of Service
SBAC	Session Based Admission Control
SJF	Shortest Job First
SSL	Secure Socket Layer
SLA	Service Level Agreement
SLAP	Service Level Agreement Profits
SALSA	Simulated Annealing Load Spreading Algorithm
SRPT	Shortest Remaining Processing Time First
TCP	Transmission Control Protocol
TPC-W	Transactional Web e-Commerce benchmark

LAS	Least Attained Service
LCFS	Last Come First Served
MBAC	Measurement Based Admission Control
pdf	probability density function
PI	Proportional Integral
PSM	Priority Scheduling Mechanism
PS	Processor Sharing
PQ	Priority Queue
P-LCFS	Preemptive LCFS
QoS	Quality Of service
QNAP	Quality Network Appliance Provider
UML	Unified Modelling Language
URL	Uniform Resource Locator

Chapter 1: Introduction

1.1. Overview

The Internet is undergoing substantial changes from a communication infrastructure to a medium for conducting business and marketing a countless number of E-Commerce services. However, this dramatic increase in E-commerce is causing a rapid rise in the number of users due to its easy access from everywhere at any time. This increasing population of E-commerce users can lead to web servers' overload and consequently poor performance. Poor performance has negative effects on the image of businesses. For example, if a web site takes one minute to load, it is quite possible that the user will leave that site for an alternative, faster, one.

Several solutions have been proposed to alleviate web server overload problems, such as (i) clusters of multiple web servers which improve response time and minimise server overload [3] (ii) cache servers are used to improve the performance of web servers through caching information [4], and (iii) mechanisms to schedule requests have been proposed to improve the performance of web servers [5, 6, 7].

In the following sections this chapter first presents the motivation and defines the aims and objectives of this research. Moreover, in this chapter the main contributions are presented and thesis outline is described.

1.2. Motivation

Modern-day services are becoming increasingly popular as they are easy to use, faster and cheaper to acquire. There has been a significant increase in E-commerce-based spending in countries such as USA (with 24% increase) and China (47% increase) [14]. This dramatic increase in the E-commerce services is causing a rapid rise in the number of customers and the consequential overload on the E-commerce web servers. Overloaded web servers have to process large number of customers' requests which may go beyond their capacity. This can cause unacceptable response time or irregular behaviour or crashing such as servers crashing during black Friday [67] and cloud crash at Amazon's website [68].

Businesses employ various strategies in order to manage a surge in the E-commerce customers within their financial constraints — i.e. to increase profit by optimally using existing resources such as web servers, network, etc. Businesses may face severe financial consequences if they fail to properly manage the load on E-commerce web servers.

Recent research considering the overload issue thus explores different mechanisms for managing the performance of E-commerce applications under overload conditions.

In summary therefore, this thesis focuses on the extremely important challenges to today's E-commerce applications and the development of new mechanisms to control overload on the web servers. In addition, this thesis formally specifies the proposed approaches using the π -calculus in early stage of models design.

1.3. Research Rationale

This research takes into account web server overloading consideration in web server layer and how it can be improved by exploiting the class-based priority scheduling mechanism. Because the use of a single technique is not enough to solve all aspects of overload problems, this research combines several techniques including: admission control mechanism, session-based admission control, service differentiation, request scheduling and queuing model-based approach.

Generally, users can use different levels of web service, such as searching or browsing a web site for booking flights or buying on line. An analysis of the literature has revealed that the number of search and browse requests is significantly higher than payment requests. According to [1], the percentage of customers who buy items is significantly lower than those who usually use an E-commerce service to find information such as air fares or book prices, without

buying anything. Similarly, other research studies [3] report that the number of users (who buy items from the Internet) is 5% (see [3] for details). The large number of search and browse requests has performance consequences, as they severely affect the processing and response time of important requests, such as payments.

This research classifies E-commerce requests into high and low priority requests. Requests by paying customers should be favoured over others (e.g. search or browse). Moreover, this research considers the extended models from Priority Scheduling Mechanism (PSM). The extended models add a new parameter, such as a review model or modify the basic PSM to low priority fair model, after the discovery of ineffectiveness with low priority customers or to add new features such as portal models.

There are many metrics for measuring the performance of E-commerce web servers. This research focuses on the performance of E-commerce web servers that refer to the throughput, utilisation, average response time, dropped requests and arrival rate.

Experiments are conducted in an integrated environment which simulates real world E-commerce services by taking into account the clients, business web portals and the web servers deployed at the service provider sites. These experiments are conducted using multi-actor simulation.

1.4. Research Aims and Objectives

Performance improvement in web servers has become an increasingly important and challenging topic in the design of E-commerce applications. One aim of this research is to develop a new mechanism to meet high performance requirements. This thesis also aims at investigating different mechanisms for controlling server overload. It also aims to develop a new model that combines several mechanisms such as admission control mechanism, session-based admission control, service differentiation, request scheduling and queuing model-based approach.

In order to achieve the aims of this research, the objectives of the thesis are set as follows:

1. To review the development of different mechanisms that consider the web server overloading problem.
2. To investigate the impact of important metrics in web servers including average response time, percentage of dropped requests, number of handlers, throughput, arrival rates and utilisation using extensive simulations on revenue.
3. To evaluate the model by using several traffic loads such as burst traffic.
4. To test and validate the proposed models by comparing them with different properties such as different model basis or different buffer's size.

These objectives are achieved by developing the proposed model and extended models and validating them through a simulation model.

1.5. Contributions of the Research

The contributions of this research are summarised as follows:

1. Synthesise the performance management mechanisms of admission control, service differentiation, service degradation, queuing model and requests scheduling in a systematic manner in order to address the new and emerging issues such as performance of business web portals, classification of requests, and the effects of dropped (rejected) requests on the web servers used in the E-commerce services. Employing individual techniques such as admission control [5] or request scheduling [15] are inappropriate for addressing these issues as illustrated in chapter 2. The main proposed model is extended to give a chance to low priorities requests to be processed rather than reject them has demonstrated a further improvement in performance. In addition, a new parameter was added in review model to show that the proposed main approach is capable of assigning more than two types of priority to ecommerce requests. Therefore, review type is added because user reviews have become an important part of e-commerce because they influence customer purchasing behaviours. Finally, new features were added in portal model to provide a more effective way for managing the performance of modern e-commerce services that offer a flexible but complex setup involving multiple websites.

2. Formally specifying the proposed scheme: the specification process help to rapidly investigate a number of different protocols, as well as providing an architectural prototype for the implementation [8]. In addition, the architecture of the implementation can be generated automatically from the specification [9]. Moreover, a formal specification facilitates any behaviour construction such as changing priorities and adding client histories. Therefore, Priority Scheduling Mechanism (PSM) and extended models are more appropriate to modelling the complex nature of the modern E-commerce services.

3. Simulates realistic model is illustrated taking into account the clients, business web portals and the web servers deployed at the service provider sites. The multi-actor simulation was developed to reflect the target models as accurately as possible therefore, for all experiments; the proposed models were solved using multi-actor simulation. The structure of the simulation is based on the state of the proposed models which mimic e-commerce networks that have server and clients and it is seamlessly integrated by the Java programming language.

1.6. Outline of the thesis

The rest of the thesis is organized as follows:

Chapter 2 presents background in E-commerce application and the different types of architectures in which E-commerce applications can be implemented. Surveys related work for managing the performance of E-commerce

applications and the different mechanisms for improvement the performance is presented with examples from the literature.

Chapter 3 introduces the proposed approach of priority scheduling mechanism (PSM) combining several techniques and its characteristics. In addition, the chapter considers the architectures of the proposed models that have been developed from PSM, including their respective algorithms.

Chapter 4 explains the formal approach that investigates the performance of PSM and it contains the modify architecture of the extended proposed models. In addition, the chapter defines UML to understand the concepts and relationships between the model's components.

Chapter 5 gives a detailed explanation of the multi-actor simulation model which has been used as a basis for the rest of the simulation models throughout the thesis.

Chapter 6 presents the experimental results based on the implementation of the proposed models. The experiments cover a wide range of input parameterizations to demonstrate performance metrics**.**

Chapter 7 Concludes the thesis with a discussion on the limitations of the thesis and a proposal for future work to be carried out based on this research.

Chapter 2: Background and Related Work

2.1. Introduction

Recently, technological developments have affected most lifestyles, especially with the growth in Internet usage. Internet applications highlight the E-commerce capabilities and applications which are now available everywhere; they receive a great number of users on a 24-7 basis because the online services are easy to use, faster and cheaper to acquire. Thus E-commerce web sites have become crucial for companies to increase their revenues. This importance has identified certain effective requirements needed from the performance of these applications. In particular, if the web server is overloaded, poor performance can result, due to either a huge rate of requests being generated which are beyond the server's capacity.

There are different approaches that propose several solutions for server overload; this chapter explores the mechanisms for managing the performance of E-commerce applications and their related work. The chapter begins with an overview of the architecture of E-commerce applications. Then, in the second part, several categories of different proposals are presented in the literature for managing E-commerce application performance. Finally, the chapter concludes

with the justification of our proposed modelling approach for web server designs and its contributions.

2.2. Architecture of E-commerce Applications

The software and hardware are the major architectural components of any system; therefore, this section discusses the architecture of E-commerce systems to provide understanding of how the software can be divided into different parts.

In this part, generalised multi-tier E-commerce applications architectures will be presented. The aim is to facilitate the understanding of the working mechanism of E-commerce applications. In addition, it is important for companies to carefully build their E-commerce web architectures before they appear to their first customers. The system's software engineer is responsible for the architecture; they use their time and resource availability to critically develop the highest-level design of the E-commerce application.

In E-commerce, users interact with E-commerce web servers through sessions which contain related serial requests from the same user in order to acquire the required information or to buy products. For example, check personal email which involves sending the account information to the mail server, checking for new messages, and downloading the messages from the server. Once the messages have been downloaded, the session is complete. User requests are

sent to the web server which in turns passes the request onto the application server and then to the database server.

There are different types of architectures in which E-commerce applications can be implemented. These include for example: two-tier, three-tier and multi-tier architectures.

In two-tier architectures there is one client and one server, they interact using TCP/IP internet, as illustrated in Figure 2.1. In two-tier applications, the server responds directly to the request using its own resources; therefore, a client-server's architecture is flexible because it is capable of directly responding to all of the client server requests. The advantage of the two-tier architectures is in its simplicity; but, the simplicity comes with the cost of scalability. The newer three-tier architecture introduces a middle tier for the application logic.

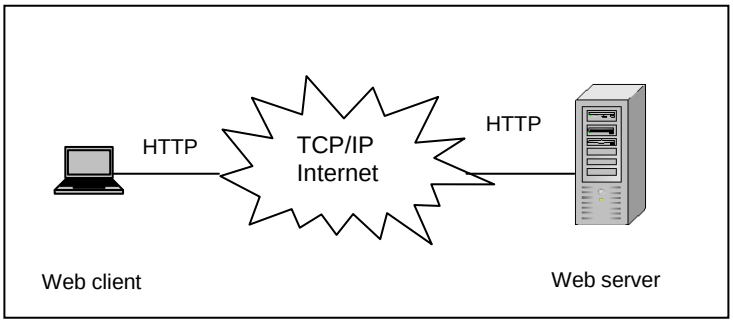

Figure 2.1: Two-tier architecture

The three-tier extends the two-tier architecture by adding a new tier to allow additional processing, for example, to architecture the adding of payment processing as illustrated in Figure 2.2. The advantage of a three-tier system

over a two-tier system is that when business logic changes, the change can be localised to the application server which is a software framework dedicated to the efficient execution of procedures such as programs, routines and scripts for supporting the construction of applications; in addition, this system provides for partitioning an application across a distributed architecture. As shown in Figure 2.3, E-commerce applications are generally implemented in multi-tier architectures that comprise of: presentation tier that includes client systems or user interface systems, usually web pages, networks, web servers, application servers in kernel levels or back-end data servers where data is stored to improve system functionality, performance, availability, scalability and reliability. Kernel levels provide the runtime environment and include: hardware, operating systems and database-specific web servers, typically they serve static contents such as HTML pages. Application servers (e.g. IBM WebSphere [63]) are commonly used to generate dynamic web content by running scripts which are written in a number of languages, such as: Active Server Pages (ASP) [64], Java Server Pages (JSP) [65] and Perl [66]. Scripts execute the necessary logic to process customer's requests by contacting various resources in order to retrieve, process and format the requested content into customer deliverable web pages. Some time there is a business tier which contains business objects and rules. Note that the most widespread example of multi-tier architecture is the three-tier architecture.

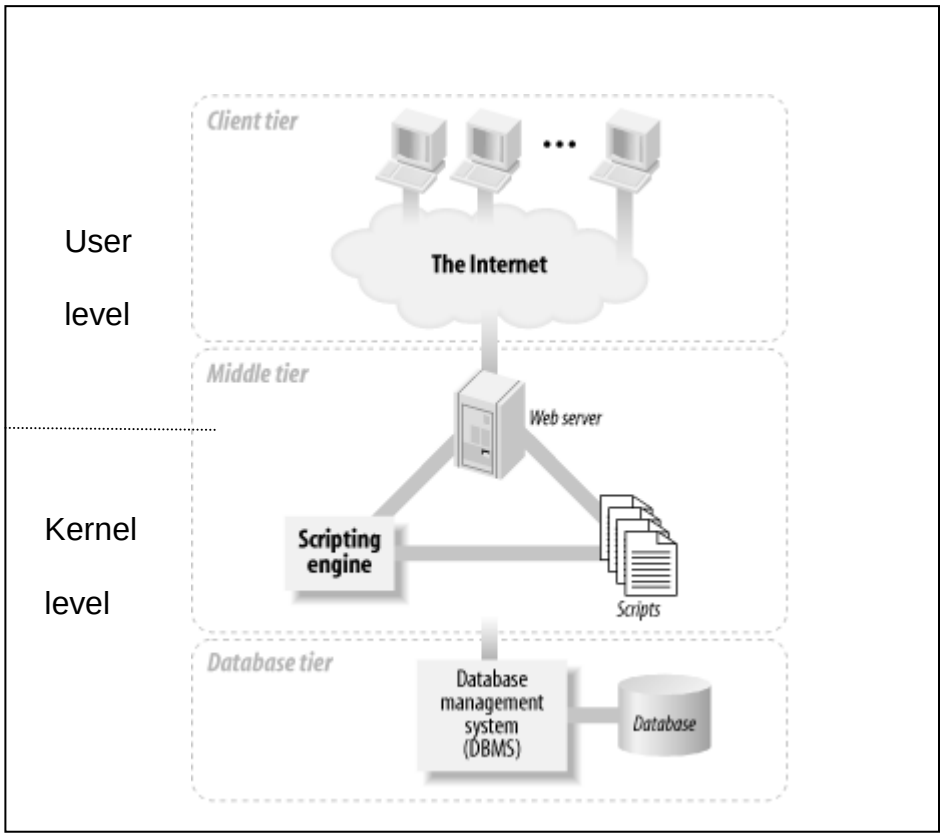

Figure 2.2: three-tier architecture

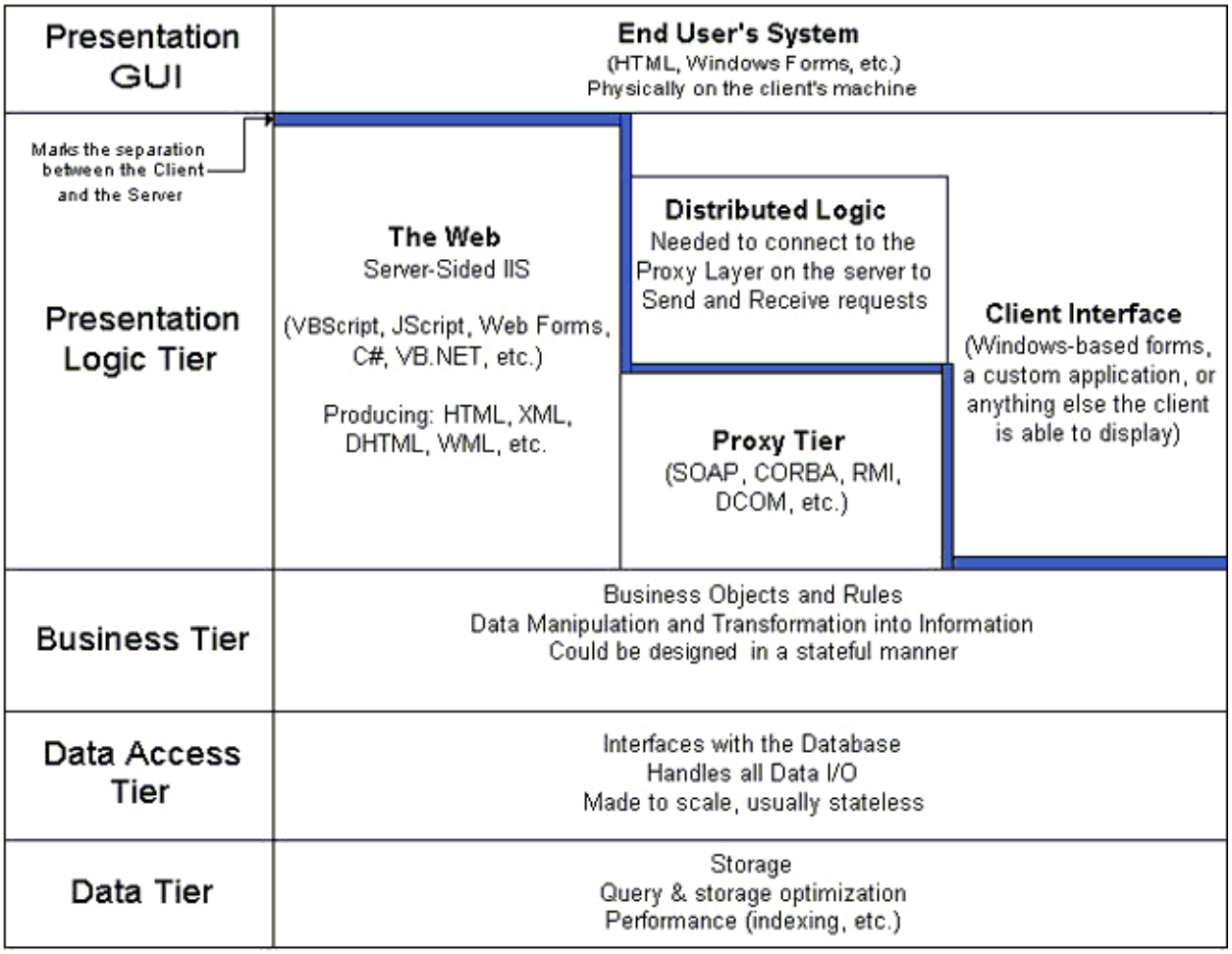

Figure 2.3: Multi-tier architecture [71]

2.3. Related Work

Over recent years, the design of E-commerce services has been crucial to the performance of web-based enterprises. Many techniques have been developed to improve the performance of E-commerce web services; each of them treats the problem from a different perspective or develops according to the progress of the Internet application. Therefore, classifying these techniques is a useful way to understand the progress of each in this field.

On one side, the techniques can be grouped depending on the actuation performed to handle the performance, including: request scheduling, admission control, service differentiation, dynamic resource management, service degradation and any combination of them. On the other side, the techniques can also be grouped depending on the mechanism used to make the performance management decision, including: queuing model-based approaches, control theoretical approaches, observation-based approaches and any combination of them.

The server's behaviour has a strong relation to high revenue; for example, if the server becomes overloaded the response time can grow to unacceptable levels that can lead to the user leaving the website. To maintain acceptable response time and minimise server overload, clusters of multiple web servers have been developed [3, 47]. Cache servers also help to improve the performance of web servers [4], by reducing the response time in real-world dynamic web applications.

A description of each group follows, with relevant studies from literature to support them. For this study, a combination of request scheduling, admission control, service differentiation and service degradation, from the first group of techniques, will be combined with queuing model-based approaches from the second group, to motivate this research proposed mechanism.

2.3.1. Scheduling Mechanisms

Mechanisms to schedule requests have been proposed to improve the performance of web servers [5, 6, 7]. Request scheduling refers to the order in which concurrent requests should be served. This section describes the existing techniques that are relevant to scheduling techniques for web-based E-commerce services at a user level, such as modifying the inclusion of a scheduler process in a web server to decide the order in which the requests should be handled to improve the mean response time. Kernel level such as controlling the order in which socket buffers are deployed to the network, or both (user and kernel levels). These techniques adopted non-traditional request ordering policies, such as: shortest remaining processing time first (SRPT). The main idea underlying these techniques is to classify the requests and schedule them in a given order, as a result of providing different quality of service (QoS) levels to each group, by assigning different priorities to the different requests.

For example, to implement policies based on SRPT, firstly scheduling to prioritise the service of short static web content requests is needed in front of the long requests [11, 12]. These studies conclude that SRPT scheduling provides a better response time to short requests at relatively low costs to the long requests. However, Crovella et al. [11] indicate that the application level scheduling does not provide fine enough control over the order in which packets enter the network. The other problem is that in traditional UNIX network stack

implementations, processing for all connections is handled in an aggregate manner. That is, outgoing packets are placed on the wire in response to the arrival of acknowledgements. This means that if many connections have data ready to send, and if the client and network are not the bottleneck, then data will be sent from the set of connections and acknowledgements will arrive, which is not under application control. Blater et al. [13] overcame this problem by implementing the scheduling at the kernel level; they reduced the low throughput and improved performance more than [11]. However, this also requires modification of the operating system (OS) kernel in order to integrate the scheduling policy.

Schroeder et al. [16] present an additional benefit from performing SRPT scheduling at the kernel level, it can determine which process is running next and how much time will be given for static content web requests by controlling the order in which the socket buffers, at the server, are depleted into the network. The authors show that SRPT scheduling can be used to mitigate the response time effects of transient overload conditions. They evaluated their work by generating a workload based on a one-day trace from the 1998 Soccer World Cup to a modified Apache web server. They conclude that 50% of files have a size of less than 1K bytes, and 90% of files have a size of less than 9.3K bytes.

Finally, Rawat et al. [12] extended the work of Balter et al. [13] by adding the size of request and the distance of the client from the server into the account to

consider the round-trip times (RTT) which is a measure of the distance of client from the server, as well, when making priority decisions of client requests. Rawat et al. [12] used the same workload, but they concluded that the new policy could improve the performance of large-sized files by 2.5-10%.

Previous studies indicate that static content of web requests work smoothly because they are directly work on one-tier architecture; however, they are not appropriate for the dynamic content web requests. These applications generally profit from considering the business value of the requests for ordering them. Consistent with this, the next studies propose scheduling policies depending on request priority.

Yue et al. [14] present a profit-aware admission control mechanism for overload protection in E-commerce websites. This approach classifies clients into two categories: premium customers (with previous purchase records) and basic customers (having no purchase records). Priority is given to the requests of premium customers on the basis that these customers are more likely to make purchases whenever they visit the website. This approach also employs hashing tables with full IP address and network ID prefix, in order to maintain records of the purchases of clients in a fine-grain and coarse-grain manner. Moreover, it differentiates premium customers from the basic customers based on the recorded hash tables. However, the proposed approach is not realistic due to recent dynamic IP addresses technology that allocates new IP address each time for customers.

Menasce et al. [1] consider the problem of resource scheduling in E-commerce sites with the aim of maximising revenue. The authors propose a characterisation approach to E-commerce workloads that takes into account the user model and its interaction with E-commerce sites. Their analysis methodology derives expressions for the performance and availability metrics of a site which are deemed important from a customer's perspective, namely: the session length and session availability. In the first step, the expressions are derived for each user group based on the navigational pattern of the group. The navigational pattern of a user group is represented by a customer behaviour model graph (CBMG), which is mapped to a discrete time Markov chain (DTMC) for analysis. In the second step, expressions for the session length and availability of the site are derived using the distribution of the customer groups and the expressions for each customer group. For example, when a customer starts navigating a web site, the web server can use the profile information which is stored in the log file and assign different priorities based on the user profile such as the navigation and buying patterns. However, it incurs processing overheads in constructing the CBMG using the log files that describe the customer's profiles. Another alternative is to use registration information to classify customers into 'occasional buyer' or 'heavy buyer'. However, this cannot always (guarantee) that registered customers will buy items each time they visit an e-business web site.

Mark et al. [17] provided detailed analyses of the CBMG. The authors proposed a method for transferring the CBMG graph to stable Markov chains in order to determine the length of an average visit of a client in an E-commerce website.

In particular, Almeida et al. [2] proposed priority-based request scheduling as a mechanism for providing differentiated QoS. Priorities to requests are assigned based on the customer to whom the requested file pertains and then maps the customer name into a priority value. They assume that the customer name is embedded in the uniform resource locator (URL), which determines the file requested. In the current web server there is no difference among requests in terms of priorities, the authors therefore implemented the priority-based scheduling at both the user and kernel levels to modify the web servers. In the user-level approach, the Apache web server was modified with the inclusion of a scheduler process responsible for deciding the order in which the requests should be managed. In the kernel-level approach, the Linux kernel was modified so that request priorities were mapped into the priorities of the HTTP processes managing them. The results showed up to 26% improvement for higher priority requests, with an accompanying 504% fall in the performance of lower ones, for the user-level approach. For the kernel-level approach, improvement was similar with a slowdown of around 208%. The web workload used in the experiments was generated using the WebStone benchmark [41].

Gupta et al. [10] analysed the mean response time under various scheduling policies, such as: processor sharing (PS), least attained service (LAS), random-

order-of-service (ROS),last-come first-served (LCFS) and preemptive LCFS with resume (P-LCFS), in the presence of correlated job sizes. The authors considered policies which know the generative correlation model, but not the actual realisations of the sizes (or the size-class) of the jobs. In most applications, including scheduling of CPU, IP flows, database queries, etc., the job sizes are often not known a priori; hence, size independent policies are more realistic. Furthermore, among the class of size-independent scheduling policies, there is no single scheduling policy that is optimal for all degrees of correlation between job sizes; thus, any optimal policy must learn the correlations. The authors evaluated their work by comparing the different scheduling policies analytically; they concluded that the P-LCFS and LCFS perform optimally with respect to response time among size-independent policies, with very high correlation.

Boone et al. [48] proposed reward-driven request prioritisation (RDRP) mechanisms, which maximise the profit (or any other application-specific reward) attained by an E-commerce service; they dynamically assign higher execution priorities to the requests of sessions which are likely to bring more profit (reward) to the service. The authors evaluated their work on the Transactional Web e-Commerce benchmark (TPC-W) application using CBMG-based web workloads. Experimental results show that RDRP techniques yield benefits in both load and overload situations, for both smooth and irregular client behaviour, against state-of-the-art alternatives, such as: session-based admission control and history-based session prioritisation approaches. In

addition, the results show that the history-based approach matched performance of RDRP mechanisms only if the correlation between the clients' past and future behaviours reached the mark of 75% for the profit attained and 50% for the request response times.

2.3.2. Service Differentiation and Admission Control

Service differentiation is based on differentiating classes of clients and then providing different priorities to each class. In fact, service differentiation can be implemented by means of request scheduling. Other mechanisms, including: refusing connection that comes from a given user's class, addressing the high priority level to other classes, or initiating different delays for other classes, are introduced by admission control.

Admission control is based on reducing overload on the servers by rejecting a percentage of connections. Service differentiation and admission control have been combined, in many works [20, 21], to prevent server overload and to provide different QoS to clients.

Bhatti et al. [20] propose that the architecture of web servers can provide QoS to differentiated clients. They used request classification, admission control and request scheduling to support distinct performance levels for different classes of clients. They classified the requests according to the clients' preference, the admission control of low priority requests was then triggered when thresholds in the number of requests were queued and the number of premium requests

queued were exceeded. The proposed architecture considers only static web content.

In order to achieve service differentiation, some studies accommodate the various bandwidth requirements of incoming flows that share the same departing link. For instance, Li et al. [22] propose a measurement-based admission-control algorithm to accurately control the proportional bandwidth that each client should receive. Incoming requests are rejected when it has both received more than its allocated bandwidth and the server is fully utilised. To ensure the different classes of requests are receiving a suitable share of the bandwidth, the authors considered the amount of delay in the processing of certain requests during these overloaded states.

In addition, Wang et al. [44] combine dynamic resource, admission control and service differentiation techniques. They proposed a downlink resource management framework for QoS scheduling in orthogonal frequency-division multiple access (OFDMA) based worldwide interoperability for microwave access (WiMAX) systems that is a series of wireless broadband standards that were authored by the IEEE 802.16 based technology [45]. OFDMA is a physical layer specification for IEEE 802.16 systems. The proposed framework consists of a dynamic resource allocation (DRA) module and a connection admission control (CAC) module. A two-level hierarchical scheduler is developed for the DRA module, it can provide more organised service differentiation among different service classes, and a measurement-based connection admission

control strategy is introduced for the CAC module. The results indicate that the framework can adapt efficiently with different kinds of traffic loads. In addition, QoS requirements for each service class can diverse with high spectral efficiency and low outage probability which defined as 3%.

2.3.2.1. Admission Control for Dynamic Web Content

The previous works on admission control are based on static web content; thus, they are not directly applicable to multi-tiered sites or the dynamic web content or web services that were presented earlier in multi-tier architecture. Therefore, some work focus on proposing admission control and service differentiation approaches for applications based on the dynamic web content. For example, Elnikety et al. [5] implemented a proxy server, called the gatekeeper, which is transparent to the database and application server. The gatekeeper enables admission control and provides the differentiated scheduling of requests to improve response time. Admission control is based on the principle that a maximum load should be maintained just below the capacity of an E-commerce system, preventing system overload and also achieving high throughput. This method employs preferential scheduling in the form of the shortest job first (SJF) that could be well approximated by the size of the file. Moreover, this method can make dramatic improvements to response time for dynamic web requests, while penalising large jobs only slightly. However, this work is dependent on

offline measurements to determine the capacity of the system rather than online measurements.

In contrast, Verma et al. [23] propose a service time-based online admission control methodology for maximising the profits of a service provider. The admission control of requests uses the shortest remaining job first (SRJF) policy which is restricted to a set of undecided requests, i.e. the requests which have neither been rejected nor serviced. The authors use an estimated service time and the prediction of arrivals of request, its QoS bounds and service times of request then come in the short-term future. Admission control rejects some of the requests that may maximise the profit of the service provider, so the remaining requests can be serviced within their QoS bound.

Caching Admission control and control theoretical, Kamra et al. [8] propose a control-theoretic approach for multi-tiered web applications. This approach aims to prevent overload and ensure high throughput while maintaining absolute response time. The authors use classical control theoretic techniques to design a proportional integral (PI) controller for admission control of the client HTTP requests. In addition, they present a processor-sharing model that is used to make the controller self-tuning, so that no parameter setting is required beyond a target response time.

Their proposed approach is implemented as a proxy system called Yaksha – which is claimed to be non-invasive and which avoids frequent operator intervention. The work concludes that Yaksha is able to bind the response times

for the requests and yet maintains a high throughput under overload. Moreover, Yaksha easily adapts to varying loads and workload characteristics because of the underlying self-tuning design.

Alonso et al. [15] propose a mechanism for providing different quality of service to the different client categories, by assigning different priorities to the threads attending the connections. After observing that Java thread priorities are only applied within the Java virtual machine (JVM), the authors propose to schedule threads using Linux real time priorities which ensure that higher priority processes are always executed before lower priority processes. These authors demonstrate the benefit of their approach which offers differentiated QoS to the client's type. In addition, their client differentiation technique can be combined with other admission control or request scheduling techniques.

2.3.2.2. Session-Based Admission Control for Dynamic Web Content

As mentioned before, E-commerce applications are session-based. A session is a sequence of independent requests from the same user. A higher number of completed sessions mean that a higher amount of revenue is likely to be generated.

Sessions have obvious characteristics, for instance session parameters can be stored in the application and then used to keeps users connected to the same

server even when a user is performing dynamic activities like accessing multiple pages within a browsing session or progressing to a payment state. Moreover, a performance measurement of web services, in terms of sessions, is more meaningful than on individual request measurement.

On the other hand, individual requests can complicate the overload control. When admission controls work with individual requests, they can produce more incomplete sessions during overloaded states because this limits the number of threads in the server or interrupts the active threads during the overload. A number of studies have focused on managing sessions to prevent overload in session-based applications.

To illustrate, Carlstrom et al. [24] propose an architecture for request scheduling at the user level and session-based admission control in web servers. Any new sessions which arrive when the maximum admitted session arrival rate has been achieved will be refused. If the first request of a session is admitted, all the following requests within the same session will also be admitted. There are different stages in which the session can reside, such as: establish, browsing, add to card and buying. The key idea is to breakdown the sessions into stages with specific service requirements and transition probabilities, and to make the web server aware of this structure. When the session is established, each request is classified with respect to the requested stage in the session, and entered in a stage-specific FIFO queue before receiving the service. Carlstrom et al. evaluated their architecture on an electronic store simulation and

concluded that when controlling the resource sharing between stages, an application-specific reward function is maximised. Moreover, their optimised GPS scheduler reached up to 8% higher reward rates than a server using the first-come-first server discipline.

Chen et al. [25] illustrate a commercial web server log analyser for deriving session-based dependency relationships among HTTP requests. They proposed a dynamic weighted fair sharing (DWFS) scheduling algorithm to control overload. DWFS is distinguished from other scheduling algorithms in its logic – it aims to avoid processing of requests that belong to sessions that is likely to be aborted in the near future. Requests of sessions that have a higher probability of being completed are scheduled first. They evaluated their proposal over an Apache web server using a modified version of WebStone2.5. They resulted that DWFS can improve server responsiveness by as high as 50% while providing QoS support using service differentiation for a class of application environments.

Muppala et al. [9] propose two new session-based admission control approaches for multi-tier Internet applications in order to improve the defined service throughput. Firstly, multi-tier measurement based admission control (MBAC) pro-actively accepts different session mixes based on the utilisation state of all tiers. Secondly, the coordinated session-based admission control approach (CoSAC) is based on a machine learning technique. They choose the session-based admission control (SBAC) strategy for performance comparison

between CoSAC with MBAC and a Blackbox approach because it is the prevalent approach used for session-based admission control in web servers. They evaluated their approaches using the TPC-W workload benchmark [43] in a typical three-tier e-commerce environment. They demonstrated the superior performance of CoSAC and found that it can improve the effective session throughput by about 50% compared to the Blackbox approach in most scenarios, while MBAC can improve that effective session throughput by about 20%.

2.3.3. Dynamic Resource Management

Dynamic resource management encapsulates the functions of allocating additional capacity to the application, configuring and monitoring resource instances sharing common attributes. For example, the server allocated to one web site can be reassigned to another site.

Dynamic resource management can help to provide performance, maintain acceptable response times and minimise server overload by allocating the resources to other service providers which avoids client degradation. Recent works [29, 30, 46] have considered the advantages of dynamic resource management techniques among hosted applications, based on the variations in workloads rather than statically over provisioning resources in a hosting platform.

Clusters of multiple web servers have been developed [3, 47] using the dynamic resource management technique. Cache servers also help to improve the performance of web servers [4].

Almeida et al. [46] propose joint resource allocation and admission control solutions that are designed to consider the provider's revenues, the cost of resource utilisation and the customers' QoS requirements which are specified in terms of the response time of the individual requests. They resolved the optimisation problem by means of an analytical queuing-based solution of a performance model. The effectiveness of the resource allocation and admission control policies identified by the optimisation model were tested using simulation in a number of different scenarios of interest. Results indicated that resource allocation and admission control policies satisfied QoS constraints. Furthermore, compared to state of-the-art resource management techniques, the proposed joint solution can yield a significant profit increase for the provider.

2.3.4. Service Degradation

Service degradation avoids refusing client connections by reducing the level of service offered to them under overload period as shown in Figure 2.4.

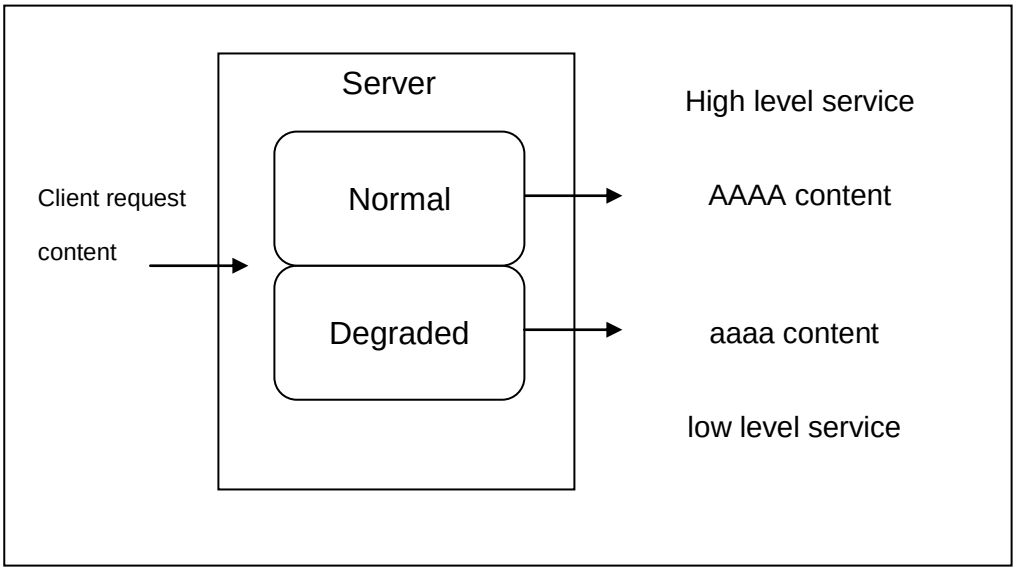

Figure 2.4: Service degradation technique

Urgaonkar et al. [26] implemented the QoS adaptive degradation approach. They considered that during overload conditions the performance of the admitted requests can be degraded within the limits established by the service level agreement (SLA). The same idea is applied by Abdelzaher et al. [27], but was named QoS adaptation. In this work, they used the mechanisms for content adaptation [28] to provide degraded services, within the values indicated in the SLA, when the resource utilisation exceeded a predefined threshold. The results indicated that control theoretic techniques offer a sound way of achieving desired performance in performance-critical Internet applications. In addition, the proposed QoS management solutions can be implemented either in

middleware, which is transparent to the server, or as a library called on by the server code.

2.3.5. Control Theoretical Approaches

Control theory has been generally used to modify the behaviour of dynamical systems. Several works present the use of control theory to: avoid overload, meet the individual response time and to guarantee throughput. The closed loop controls the parameters of the actuation technique using feedback information from the system, as represented in Figure 2.5, and it measures server utilisation. The admission control can be used as the actuator in the server because the admission control can perform deterministic control to accept or reject incoming connections; in other words, admission control can handle the load on the server thus it can improve utilisation by determine a safe utilisation level of servers. The sensor presented in Figure 2.5 is responsible for measuring the current utilisation consistently.

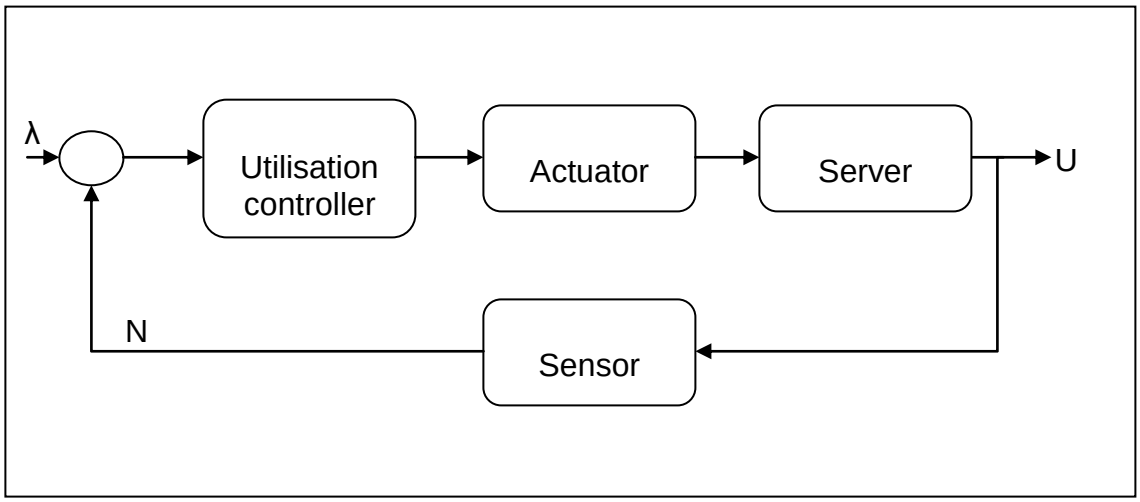

Figure 2.5: Architecture of a control theoretical technique

In particular, Abdelzaher et al. [27] describe performance control of a web server using classical feedback control theory. The authors use feedback control theory to achieve overload protection, performance guarantees and service differentiation, in the presence of load unpredictability. They demonstrate how a general web server may be modelled for purposes of performance control, to present the equivalents of sensors and actuators, and control loop that regulates the extent of degradation to satisfy a pre-specified utilisation bound. The work evaluates the efficacy of the scheme on an experimental test using an Apache server. The authors conclude that the control theoretic techniques offer performance improvement in Internet applications.

2.3.6. Queuing Model Based Approaches

Queuing theory has been used for modelling the behaviour of complex systems to improve their performance. Many studies address the queuing model in their proposal. For instance, Almeida et al. [31] propose an optimisation model that identifies the optimal resource allocation by maximising a provider's revenues while satisfying customers QoS constraints and minimising resource usage cost. They describe two tightly inter-related problems in autonomic computing, namely, a short-term resource allocation problem and a long-term capacity planning problem; they use queuing models to address the resource management problems in autonomic service-oriented architectures. Their work evaluates the algorithm execution time and runs experiments with various values of the model parameters. Experimental results showed that their work resolves reasonably large problem sizes, typically less than 15 seconds, which makes it practical for online implementation. Moreover, the results show that taking resource usage costs explicitly into account in the optimisation model can yield total cost savings for the provider of as much as 39%.

Liu et al. [32] present a methodology for maximising profits in a general class of E-commerce environments. They used a generalised processor sharing (GPS) closed queuing model with a multi-class queue for each server, and formulated the service level agreement (SLA) between service providers and their clients. This included the tail distributions of the per-class delays as delays experienced

by customers of a business can result in lost revenue for that business, in addition to more standard QoS metrics such as throughput and mean delays. Their work consisted of formulating the optimisation problem as a network flow model with a separable set of concave objective functions based on queuing-theoretic formulas. They solved the problem via a fixed-point iteration. Various experiments illustrate the benefits of their work; in addition, they compare the analytic and the simulation models. These findings provide important insight into the fundamental problem of maximising service level agreement profits (SLAP) in e-commerce environments.

Urgaonka et al. [33] propose a novel dynamic provisioning technique for multi-tier Internet applications that employ a flexible queuing model to determine how many resources to allocate to each tier of the application. Furthermore, they used a combination of predictive and reactive methods that determine when to provide these resources, both on large and small-time scales. These authors used G/G/1-based open queuing where short-term fluctuations caused the workload of a three-tier application to double. Flash crowd is one example of short-term fluctuations which is groups of users on a computer network that appear, then disappear, in a flash. They concluded that the proposed model maintained response time targets and reduced the overhead of switching servers across applications from several minutes to less than a second, while meeting the performance targets of residual sessions when a server was moved from one application to another.

Boone et al. [49] describe an adaptive load balancing strategy called SALSA (simulated annealing load spreading algorithm), which is able to guarantee for different customer priorities, such as default and premium customers, that the services are handled in a given time and without the need to adapt the servers executing the service logic themselves. They derived average waiting times through applying the standard queuing theory. They concluded that SALSA is able to dynamically adapt its load balancing strategy to handle dynamic request patterns without a priori over-dimensioning the web servers' resources in order to guarantee the SLAs to premium customers.

2.3.7. Works Combining Several Techniques

Some research [50, 51, 52] demonstrates that the most effective way to prevent Internet application overload and the best way to provide performance guarantee requires the combination of several techniques, instead of considering each one in isolation. In these works, the use of different techniques compensates for the limitations of the others.

Guitart et al. [50] propose an overload control strategy, for secure web applications, which brings together dynamic provisioning of platform resources and admission control based on secure-socket layer (SSL) connection differentiation. Dynamic provisioning enables additional resources to be allocated to an application on demand to handle increased workload, while the admission control mechanism avoids the server's performance degradation by

dynamically limiting the number of new SSL connections accepted and preferentially serving resumed SSL connections (to maximise performance on session-based environments), while additional resources are being provisioned. They demonstrated the benefit of their proposal for efficiently managing the resources and preventing server overload on a 4-way multiprocessor Linux hosting platform, especially when the hosting platform was fully overloaded.

Huang et al. [51] consider unpredictable response times for clients during web server overload. They combined admission control, scheduler and service differentiation techniques to present their model. Their work includes two admission control models to enable permits proportional delay differentiated service (PDDS) at the application level. Each proposed model predicts the total maximum arrival rate and maximum average waiting time of each priority task group for the next measurement period, according to the arrival rate of each class during the current and the last three measurement periods. They demonstrated their work among a series of simulations; the results indicate that the proposed models can effectively realise proportional delay differentiation services in multiclass web servers.

Finally, Kasigwa et al. [52] introduce the dynamic admission control (DAC) mechanism. The DAC allocates the network resource using the previous traffic pattern to each path. Dynamic means that the bandwidth broker allocates the resource to the path dynamically, and the amount of bandwidth allocated to the path is not fixed, but is variable upon the traffic flow's QoS requirements. These

authors performed extensive simulation experiments to ascertain the efficacy of the proposed solution. In addition, the proposed DAC mechanism was compared with static measurement-based admission control (MBAC). The results proved that the proposed DAC mechanism guarantees user QoS requirements and provides bandwidth efficiency.

2.4. Discussion

2.4.1. Overview of the proposed techniques

The increasing popularity of Internet applications has initiated big changes from simple static web content, HTML pages and images, to dynamic content, and from simple one-tier architectures to multi-tier complex E-commerce web pages. With these changes the Internet applications have improved the services for the customers by providing lower response time and higher availability; however, they also now involve a higher degree of complexity in their management. As shown in this chapter, many works deal with web servers during overload situations. Using a single technique is not enough to solve all of the overload problem situations. For example, the use of first come first served (FCFS) scheduling techniques can improve the response time for first come customer however, during overload situations it introduces unpredictable response times for other customers. Customers may therefore become frustrated by a long response time and end the network connection with the web server, prior to

finishing their transaction, thus, it will lead to a loss of revenue for the businesses. In the same way, dynamic resource management allows the release of resources to or from Internet services according to the varying load. However, dynamic resource oscillates in the presence of potentially concurrent dynamic resources. Admission control can increase the effective capacity, but during extreme overload a lot of requests will be rejected and this could lead to a loss in customers. The service degradation technique focuses on the level of service offered to clients, under overload conditions, and reduces it to avoid refusing them. However, service degradation is not applicable in many services due to their design: for example, an e-mail or chat or teleconferencing service cannot practically degrade service in response to overload because lower quality leads to misunderstanding. The provision of robust performance guarantee, in the presence of load, is the main idea of control theory, but it cannot be accurate with linear models and this can lead to response time issues. The queuing model can provide an accurate analysis of the steady state and, for this reason, it is useful for building predictive systems; however, it cannot handle complex systems with complex service time distributions.

Numerous techniques have been presented, the different examples of these related works are summarised in Table 2.1. The selected cells, in the table, indicate the works which used which type of technique.

Table 2.1: Summary of related works with used techniques

Work	Request scheduling	Admission control	Service differentiation	Control theory	Queuing model	Service degradation	Resource management
[1,2,3,6,7]	√						
[5,48]	√	√					
[8]				√			
[9]		√				√	√
[10,49]	√				√		
[11,12,13]	√						
[14]	√						
[15,16]	√		√				
[17]	√						
[18,23,24,25,51]	√	√	√				
[19]	√		√		√		
[20,21]	√	√	√				
[22]		√	√				
[26]		√	√		√	√	√
[27]			√	√		√	
[28,29,30]						√	
[31]			√				√
[44]	√	√	√				√
[32,33]			√		√		√
[46]		√	√		√		√
[50,52]		√	√				√

According to the issues presented from using a single technique, this research will therefore combine more than one technique to solve overload issues on web

servers. This work will encompass a mixture of several techniques, including: admission control mechanism, session-based admission control, service differentiation, request scheduling and queuing model-based approach. The remaining chapters of this thesis will thus address these issues in more detail.

2.4.2. Summary

This chapter presented the architecture of E-commerce applications. It is difficult to find a satisfactory solution for overload problems such as unacceptable response time level or decline throughput because an overloaded Internet web site is filled with huge web requests that are well beyond the system's capacity. This chapter characterises and classifies the different approaches suggested in the literature for managing the performance of Internet applications. The classification of these approaches is considered from two different viewpoints. On one side, the techniques can be grouped depending on the actuation performed to handle the performance that includes request scheduling, admission control, service differentiation, dynamic resource management, service degradation and any combination of them. On the other side, the techniques can also be grouped depending on the mechanism used to make the performance management decisions, this includes: queuing model-based approaches, control theoretical approaches, observation-based approaches or any combination of them.

In summary, the use of a single technique is not enough to solve all aspects of overload problems; thus, a number of works were presented which utilised more than one technique in their proposal [50, 51, 52]. The research in this thesis aims to develop their research further and focuses on joining techniques. This thesis will therefore demonstrate the most effective way to prevent Internet application overload and provide performance guarantees based on the combination of several techniques rather than considering each one in isolation. This approach will be complementary and can be combined with any of the described technique, such as: request scheduling, admission control, service differentiation, service degradation and queuing model.

To meet the needs aforementioned, this research will focus on service discipline which provides a different service for individual classes. A priority scheduling mechanism (PSM) will be developed in order to assign different priorities to different classes of requests that are introduced as a session. Therefore, in an online shopping scenario, the service provider might be interested in giving a higher execution priority to sessions that have placed something in their shopping cart, when compared to sessions that appear to be just browsing product catalogues; thus ensuring that the clients that buy something receive better QoS.

Scheduling mechanisms [2, 1, 14] can improve the performance of high priority requests, but the service quality of low priority requests is still somewhat degraded and can make a starvation situation. Therefore the model, presented

in this thesis, can improve the performance of high priority and low priority by giving more opportunities to process low priority requests, even when they are spending unacceptable times to avoid the customer from losing out. Similarly, this research will use service degradation techniques for low priority requests by giving them a delay, rather than just rejecting them.

Some web pages contain third-party sponsored advertisements or portal gates; the service provision may increase with more visits to these pages because it increases the chance of the client following the advertisement links. Consequently, the service provider may wish to give higher priority to the sessions that visit web pages with advertisements frequently. The model presented in this project gives attention to the portal web pages.

Unlike other studies [11, 13], the implementation of this research requires no changes to the source code, server software, application programs, or to the databases. The benefits of such an approach are clear: the use of unmodified commodity software components reduces development effort tremendously. As a result, the researcher is able to demonstrate the suggested approach using standard software components and workload generators. In other words, this approach does not require extensive modifications to the operating system or a complete re-write of the server.

It should be acknowledged that dynamic resource management has become a crucial technique for performance improvement. Furthermore in the future, this

will allow the adding of more than one CPU with associated priority scheduling mechanisms (PSM).

In order to apply control theory, this research should estimate the dynamic model based on online measurements that will be utilised in the future to model this work, they should be based on real-time rather than multi-actor simulations.

The following chapter will describe in detail the proposed model and the further extended models too.

Chapter 3: Priority Scheduling Mechanism

3.1. Introduction

The main contribution of this thesis is addressing overload on e-commerce web servers using a class-based priority scheme to process by available resources. As mentioned in chapter 2, a priority scheduling mechanism (PSM) is developed in order to assign different priorities to different classes of e-commerce requests and it combines more than one technique. In e-commerce, some requests, such as payments, are generally considered more important than others, like search or browse, due to their revenue raising capabilities. This project is not concerned with the performance aspects of the application server or the database server; instead it focuses on web server performance and how this can be improved by implementing a class-based scheduling scheme. By assigning class-based priorities at multiple service levels, e-commerce web servers can perform better and can improve the performance of high priority e-commerce requests, such as reducing the mean response time without introducing unfairness for other requests and reducing the rejected requests which have a higher priority as will see in chapter 6. PSM is a simple mechanism which can add to web servers without affecting the basic structure of these servers. Classification is necessary and financially beneficial to e-

commerce service providers because some requests are more valuable than others. For instance, the processing of ‘browse’ request should get less priority than ‘payment’ request as the latter is considered to be more valuable to the service provider since the customers issuing payment requests are making purchases. This will be described, in more detail, within the next chapters.

PSM is not the basic model, it is simple to adjust this model and adapted to different services therefore, it is easy to extend this model straightaway to various models such as review model, low priority fair model etc.

This chapter will initially provide an overview of the architecture of the proposed priority scheduling mechanisms (PSM). Then, the chapter will present characteristics of PSM and, in the second part, detail the combined techniques that produce PSM. The third part will consider the architectures of the proposed models that have developed from PSM, including their respective algorithms. These models were extended from the basic PSM and new parameter were added, such as a review model or modify the basic PSM to low priority fair model, after the discovery of ineffectiveness with low priority customers or to add new features such as portal models. Finally, the chapter will summarise the proposed models.

3.2. Architecture of Priority Scheduling Mechanism (PSM)

The architecture of the proposed mechanism efficiently specifies the basic concepts of this mechanism; Figure 3.1 illustrates how the scheme works.

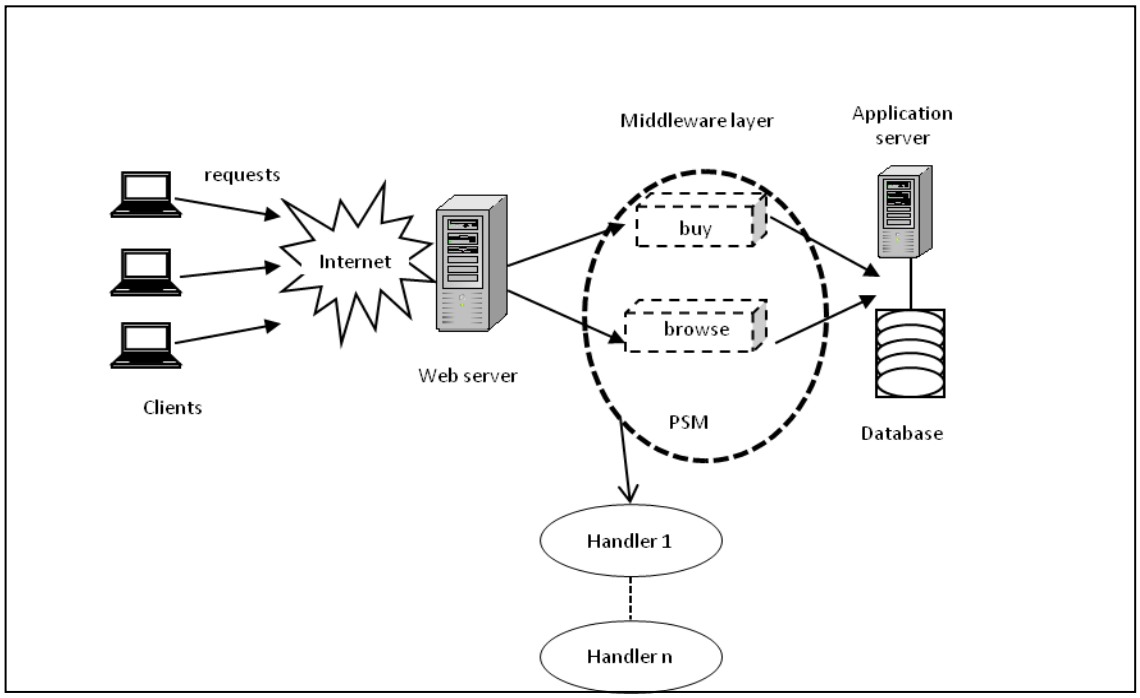

Figure 3.1: Priority scheduling mechanism (PSM)

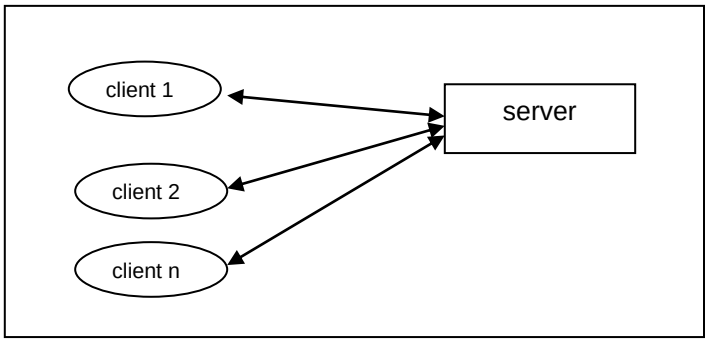

Figure 3.2: Traditional client-server approach

The PSM is produced within distributed object computing (DOC) instances of traditional client-server approaches (see Figure 3.2) to provide a robust system for client-server computing. DOC presents a middleware between the client and

the server and as such supports the interaction between the active object(s) that encapsulates a thread in a distributed computing environment. In addition, middleware focuses on the development of the application and ignores the peculiarities of a specific distributed environment. Many threads are created to access single shared resource so mutually exclusive access is enforced where appropriate for all created threads to ensure they can share resources safely. Synchronisation facility is used particularly in PSM and more detailed explanations will be presented within the simulation description in the fifth chapter.

With reference to Figure 3.1, each arriving request is assigned to a handler thread then classified into two types, namely, browse requests and payment requests, more details will present within chapter 4. Separate virtual buffers for each class are maintained to temporarily hold that type of request. Instead of a single n-place buffer, the approach uses ‘n’ active components, each storing one request. Payment requests are given higher priorities over the other requests, as follows:

- If the buffer of payment requests is not full then process the first buy request.
- If the buffer of payment requests is empty then process the first browse request.

Note that the process-requests command at Algorithm 3.1 means many threads completing for accessing to single shared resource, which is a request

processor. Unlike other existing mechanisms [1, 5, 13], this mechanism does not require pre-requisite information (e.g. registered users, server log files, or size of requested files) to assign priorities to e-commerce requests which needs resources and time to check.

In the middleware layer there are Handler threads that are created to deal with clients on one-to-one basis and become a one place buffer then check the specific virtual buffer depending on the type of request which leads to classification stage. Each Handler executes the PSM algorithm below.

Algorithm 3.1: Priority Scheduling Mechanism (PSM) Algorithm

```
if (priority of arriving request== high)
        {
          if(buy_buffer not full)
             Process this request;
           else
              Drop this request;
        } // End if
else if(priority of arriving request== low)
        {
         if (browse_buffer not full)
```

```
{       if(buy_buffer is empty)

            Process this request;

          else

            Drop this request;

}//End if

 else

       Drop this request;

}//End else if
```

3.3. Characteristics of PSM

Priority scheduling mechanisms give preferential treatment to high priority requests; PSM can therefore significantly improve the overall performance of the system by reducing the server response time, reducing rejected requests and by obtaining acceptable system throughput. Scheduling priority mechanisms, such as non-preemptive scheduling (also referred to as head-of-line priority scheduling, HOL-PS), occurs as follows: a high-priority customer can move ahead of all the low-priority customers waiting in the queue, but low-priority customers who are being serviced are not interrupted by high-priority customers as they work preemptively, which is a consequence of processing a request that is done using mutual exclusion by giving the lock to the active thread until it finishes. Therefore, PSM is extremely scalable because it can add

an extra application service and if extra resources are available, there will also be extra locks available. Among the simplest time-priority scheduling schemes, the non-preemptive HOL priority scheduling discipline, according to Miller [19], provides differentiated services. However, priority queuing (PQ) can lead to starvation for low priority customers because the request for a low priority provides a very long wait time before being served, because these requests require more processing and access to the database servers. Hence, the main aim in this work is to find a scheduling strategy that reduces the response time of high priority customers, without losing the property of fairness. Therefore, it is important to evaluate different metrics, including: average response time, throughput, arrival rate, percentage of dropped requests and utilisation. To the best of our knowledge, priority scheduling mechanism combines the benefits of scheduling mechanism, service differentiation, admission control and service degradation techniques and queuing model-based approaches to improve the performance of web servers, as described in the results section that presents every proposed model accompanied by simulation. This research, therefore, synthesises the following reasonable performance management mechanisms which described before in chapter 2.

Admission control mechanism:

The purpose of admission control is to prevent servers from entering overload conditions because the mechanism polices the requests coming from large numbers of e-commerce clients. This system tells clients that their requests will

be accepted, served in a certain amount of time, or rejected immediately; PSM, for instance, rejects low priority requests when there are high priority requests in the system. Different algorithms are implemented, as part of the admission control strategy, which process requests depending on their type (high or low priority as stated below) or the status of the underlying servers. The admission control system is implemented as a set of finite capacity virtual buffers for accommodating the incoming requests.

Session-based admission control:

Sessions are created for each client in the PSM system; this includes the different types of request, such as: browse, review, portal buy and direct buy. Each client has a handler which can be an session-based control for client's session because the handler control the session and met the query only if the system got a capacity to do it. This will be described, in more detail, within the simulation section. Note that the admission control is at request level and it could be at session level with the gatekeeper (see chapter 7).

Service differentiation:

Different priorities are given to each class, service differentiation is implemented by means of scheduling; in other words, scheduling mechanisms need to not only classify requests, but also identify the difference between them. The proposed approach implements mechanisms in order to classify the client requests into different classes, in accordance with the type of request.

Request scheduling:

As previously mentioned, this work applies request scheduling in the form of the priority scheduling mechanism that distinguishes classes of requests and schedules these classes in a priority order. It prioritizes the service of buy requests in front of other requests such as browse and review.

Service degradation technique:

The acceptance probability of low priorities requests are adjusted in this proposed mechanism. More details will be presented in the proposed models section.

Queuing Model technique:

The initializations are shared by all the clients when they start to connect with the server and they are then served in a first-in, first-out (FIFO) order for generated requests; the order is like the specification with a non-deterministic order (see chapter 4).

3.4. Proposed Models

The mechanism proposed in section 3.2, is based on Younas et al. [36], to demonstrate the versatility of the basic architecture model, a number of alternatives models are proposed and extended from PSM and evaluated to resolve several of the following issues:

- To show that the proposed main approach is capable of assigning more than two types of priorities to ecommerce requests, the review model is proposed. The review model adds new parameter to the basic PSM which includes review requests.
- To adapt the basic PSM to reduce the unfairness of low priority customers, the low priority fair model is proposed. This model modifies the basic PSM to improve performance of browse requests.
- To adapt the basic PSM for realistic scenario, the portal model is extended and adds new features to the basic PSM because it is the current requirements of e-commerce applications that involve multiple websites for price comparison and more.

The following sections will provide more details of each of the extended model.

3.4.1.1. Architecture of Review Model

It is important to model the performance of modern e-commerce websites which provide users with facilities for reading and writing reviews on products. User reviews have become an important part of e-commerce because they influence customer purchasing behaviours. In addition, many other approaches [17, 15, 18, 14] do not consider the effect of user reviews and the subsequent drop in requests on the performance of e-commerce web servers.

Reviews are an important source to find out more about the different services of any company and find valuable information, and then decide on the one that

best fulfils customer needs. Moreover, with review model it could be a free advertisement of the company and increases the credibility of the website for customers. Thus, reviews could affect on ecommerce revenue, so it is useful to give priority to that kind of customers.

The proposed class-based priority scheme was therefore extended to classify e-commerce requests into high, middle and low priority requests, instead of two types of requests as shown in Figure 3.3. In e-commerce, some requests (e.g. buy) are generally considered more important than others (e.g. review the product details or browse for new products). In the proposed model, virtual buffers are used to identify the number of active requests of each type. Incoming requests are assigned a priority (high, medium and low) based on their type. ‘Buy’ requests are given priority over ‘browse’ and ‘review’ requests by processing the first request when its buffer is not full. Review requests are processed if the buffer of buy requests is empty and the browse requests are processed if the buffer of review requests and buy requests are empty. The proposed model was tested through several experiments and showed a considerable reduction in the percentage of high priority requests that were not completed, please see chapter 6 for more information. More details are also provided within Algorithm 3.2, presented below.

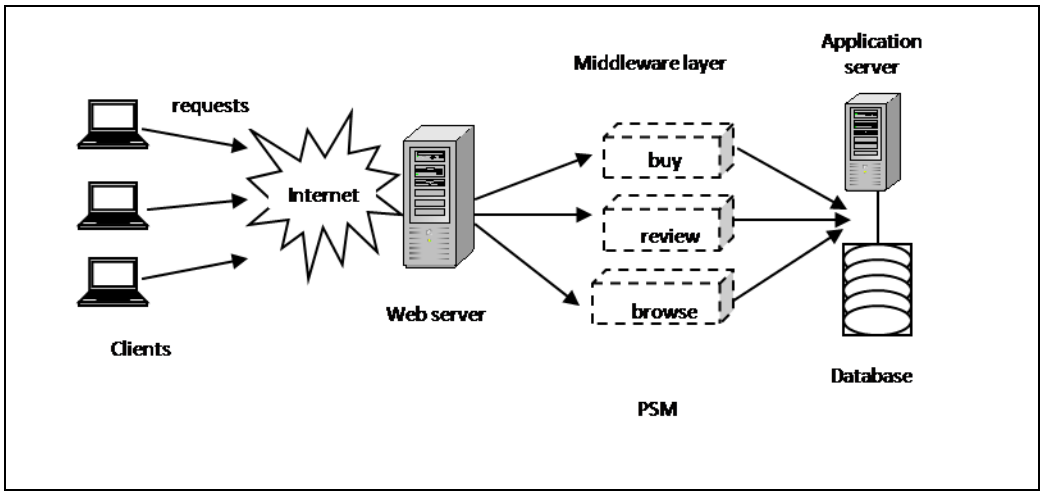

Figure 3.3: PSM with three types of requests

Algorithm 3.2: Review Algorithm

```
if (priority of arriving request== high)

        {

        if(buy_buffer not full)

            Process this request;

         else

            Drop this request;

        } // End if



else if(priority of arriving request== medium)
```

```
{
if (review_buffer not full)
{       if(buy_buffer is empty)
            Process this request;
          else
             Drop this request;
}//End if
else
      Drop this request;
}//End else if
else if(priority of arriving request== low)
{
if (browse_buffer not full)
{       if ((buy_buffer is empty)&&( (review_buffer is empty))
            Process this request;
          else
             Drop this request;
}//End if
else
```

Drop this request;

}//End else if

As shown in Algorithm 3.2 the only difference between Algorithm 3.1 and Algorithm 3.2 is to have an extra condition in Algorithm 3.2 to serve review customers. Therefore, extra virtual buffer to store review customers.

3.4.2. Architecture of Low Priority Fair Model

This model applies a handler thread priority to account for fairness. Some browse customers wait until the sever deals with their requests, as it is busy and delayed due to serving high priority customers. Some buyers prefer to browse many times before making a purchase, while other customers may become frustrated when they browse and then leave without making a purchase. A new scheduling algorithm is added to handle the thread priority on the middleware layer. The initial idea was raising the low priority customers, more than once, by measuring how often they browse within a set interval, like one millisecond. However, this solution does not provide fairness for low priority; despite the increase in priority, a higher priority is still given to buy requests. Hence, another algorithm is produced for the same reason, it reduces the loss of low priority customers, by giving a small delay to processing their requests rather than rejecting them, thus it keeps low priority customers on the web site. Algorithm

3.3 gives more detail about the way the low priority fair model, it works as following:

- If the buffer of payment is not full then process the first buy request.
- If the buffer of payment request is not empty then wait until it becomes empty then process the first review request.
- If the buffer of payment and review requests are not empty then wait until they become empty then process the first browse request.

Algorithm 3.3: Low Priority Fair Model Algorithm

```
if (priority of arriving request== high)
        {
            if(buy_buffer not full)
                Process this request;
           else
                Drop this request;
        } // End if
else if (priority of arriving request== medium)
        {
           if (review_buffer not full)
             {    while   !( buy_buffer is empty)
```

```
                    {  Wait; }

                Process this request;

          }//End if

            else

              Drop this request;

          }//End else if

else if (priority of arriving request== low)

          {

            if (browse_buffer not full)

              {      while  (! ( buy_buffer is empty) || (! (review_buffer is empty))

                        { Wait; }

                  Process this request;

          }//End if

             else

                Drop this request;

          }//End else if
```

As seen in algorithm 3.3 there is a busy waiting for low priority requests inside each handler to avoid lost connection and keep the client on the website.

However, with algorithm 3.3, the response time will increase therefore, it could be useful to add operation timed out scenario or run the additional processes to serve clients simultaneously.

3.4.3. Architecture of Portal Model

Modern e-commerce services are offered in a flexible but complex setup which involves multiple websites; for example, business web portals or price comparison websites obtain the cost of a variety of online products from relevant websites and present them to millions of customers. Though this modern style of service provisioning is very attractive, it significantly increases load on the web servers implementing the e-commerce services. The concern is that overloaded servers will become unresponsive and will drop requests which are beyond their capacity.

Portal model is extended from PSM and new features are therefore proposed that build on the synthesis of performance management mechanisms. It provides a more effective way for managing the performance of modern e-commerce services that offer a flexible but complex setup involving multiple websites; for example, business web portals or price comparison websites, draw the cost of a variety of online products from relevant websites and present them to millions of customers and, offer the most up-to-date information and provide opportunities to save money. Moreover, the portal model can address and benefit both the business and the customer; the model can provide a gateway

for new business opportunities for companies, large and small. Furthermore, it can improve the marketability of a new product or service idea and thus attracts new customers and enables companies to sell their services and products whilst also helping to maintain and manage transactions relating to their business. The customer portal model provides benefit to customers by pooling shared knowledge and experiences, thus offering: convenience, speed and access to ‘comparison pricing’ between multiple companies. In summary, the portal model brings together customers and business organisations who want improved working around the world by serving some requests itself and passing on some others.

Figure 3.4 represents the generalised architecture of modern e-commerce services. In it, two servers are considered: the web server (of the e-commerce service provider) which serves the clients who interact with the target website without the portal, and a portal-server which serves the clients who access to target website within the portal. Thus, the browser client or the buyer client can be served by the same web server.

In the same world, the web server receives requests from different clients, either directly or through business web portals, then in turn it passes the requests to the application server and then to the database server. The PSM is applied on the middleware layer for request scheduling.

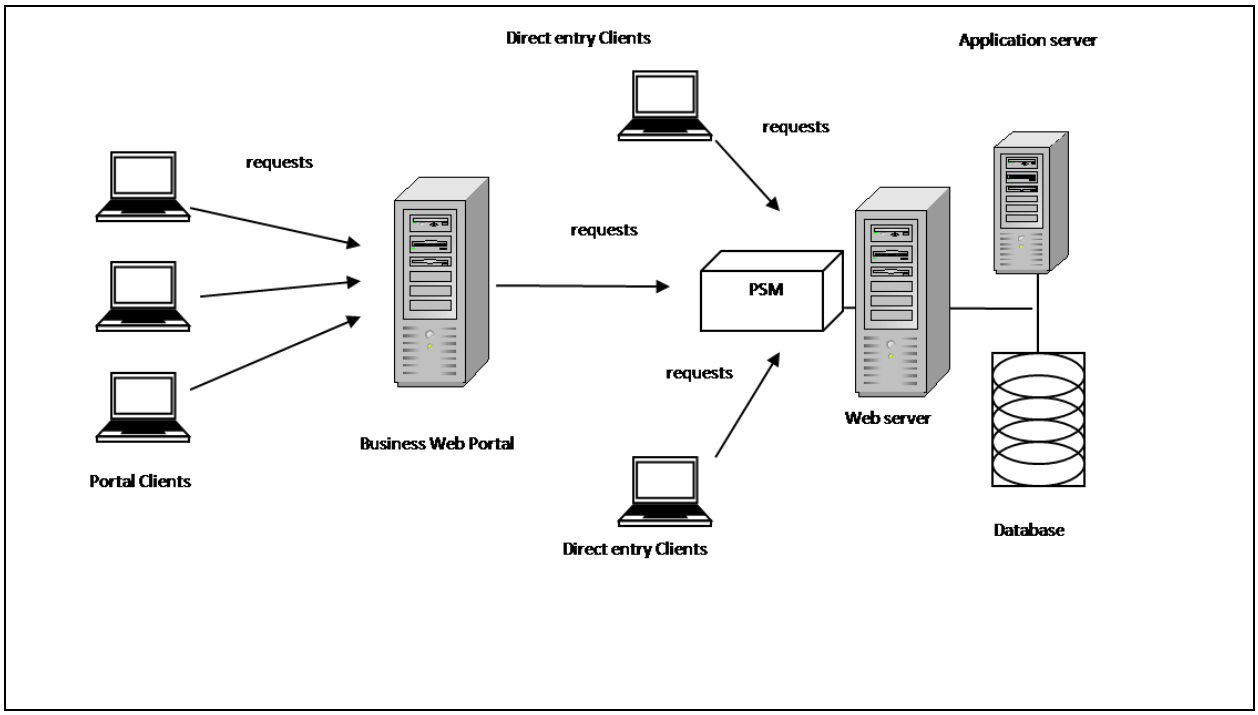

Figure 3.4: Portal model architecture

In this model four different types of requests are classified, including: 'Direct_buy', 'Direct_browse', 'Portal_browse' and 'Portal_buy'. 'Portal_buy' requests should be given high priority compared to other requests, as these requests are more likely to be converted to purchasing orders which generate money for the service provider. A medium priority is given to 'Direct_buy' and a low priority is given to 'Direct_browse', as described in Algorithm 3.4. For 'Portal_browse', there is no competition with other types because it is served directly in the portal server.

The algorithm works as following:

- If the buffer of portal buy request is not full, then process the first portal buy request.

- If the buffer of portal buy request is empty, then process the first direct buy request.
- If the buffers of portal and direct buy requests are empty, then process the first direct browse request.

Algorithm 3.4: Portal Model Algorithm

```
if (priority of arriving request== high)
        {
        if(Portal_ buy_buffer not full)
            Process this request;
         else
            Drop this request;
        } // End if
else if(priority of arriving request== medium)
        {
          if (Direct_buy_buffer not full)
           {    if( Portal_buy_buffer is empty)
                    Process this request;
                else
                    Drop this request;
```

```
        }//End if

          else

             Drop this request;

        }//End else if

else if(priority of arriving request== low)

        {

        if (browse_buffer not full)

          {     if ((Portal_ buy_buffer is empty)&&( Direct_buy_buffer is empty))

                  Process this request;

                 else

                  Drop this request;

        }//End if

        else

            Drop this request;

        }//End else if
```

3.5. Summary

In this chapter, the priority scheduling mechanism (PSM) was introduced, in detail. PSM combines the benefits of scheduling mechanism, including: service differentiation, admission control and service degradation techniques, and the queuing model-based approach to improve the performance of web servers. More properties were investigated and added to the PSM which resulted in extended models of PSM; these extended models and their related algorithms were presented and supported. The algorithm does not give a clear picture of the complicated system so it is useful to use formal specification And UML diagrams to give clear and deep picture of the several components. The next chapter will give formal specifications for PSM and extended models and will investigate the different UML diagrams in details.

Chapter 4: Formal Specification

4.1. Introduction

The objectives of this chapter are to design and develop the approach described in chapter 3 by constructing formal models. The work in this chapter provides design details of the individual components of the different architectures presented in Chapter 3.

The π-calculus [34] was used to specify, design and develop the proposed model and the extended models presented in chapter 3. The specification process allowed rapid investigation of a number of different protocols for scheduling e-commerce requests, as well as providing an architectural prototype for their implementation [38]. In other words, π-calculus allowed a more natural formulation of the model, particularly the dynamic creation of components configured with appropriate communication links by exchanging the names of the channels.

The main benefit of using a compositional modelling language, such as π-calculus, is that the individual layers of multi-tier applications can be studied to fully understand their functionality before they are composed with the other tiers.

This opens the possibility of replacing the complex model of each tier with a simpler model.

The use of π-calculus facilitates behaviour constructions, such as changing priorities and adding client histories, but, there are no limits on the number of clients. It is apparent that the proposed method is appropriate to modelling the complex architecture of modern e-commerce services. Moreover, it is suitable for selecting the best composition that matches the requirement of e-commerce simulations.

In each proposed model the π-calculus was presented to clarify the nature of actions. The implementation of actions was then created, dependent on the chosen language – in this case, Java language. Java was chosen because it can achieve the requirements for implementing actions [41], more details are provided in chapter 5.

This chapter will firstly present a formal specification that will capture the essential PSM model's design features presented in Chapter 3. The formal specification of PSM will then be modified to express the extended models. Unified modelling language (UML) diagrams will then be presented to identify and understand the concepts and relationships between the model's components. UML model is useful to bridge the gap between the abstract π-calculus and the concrete programming language.

Then, the link between the presented UML diagrams and the π-calculus will be established. Finally, a summary of the contents of this chapter will be presented.

4.2. π -Calculus

In theoretical computer science, the π-calculus is a process calculus originally developed by Robin Milner [34], as an extension of his work on the calculus of communicating systems (CCS) [69], following work by Engberg and Nielsen who added mobility to CCS while preserving its algebraic properties. The aim of the π-calculus is to be able to describe and analyse concurrent computations whose configuration may change during the computation [55]. Moreover, the π-calculus obtains simplicity by removing all distinction between variables and constants; communication links are identified by *names*, and computation is represented simply as the communication of names across links.

Notation descriptions of π-calculus [34] are given below in Table4.1:

Table 4.1: Notations of π -calculus

Symbol	Description
$p\|q$	parallel composition
$\overline{\mathrm{x}}\langle \mathrm{y} \rangle$	To send y along x
$x(y)$	To receive y along x
$(new\ x)P$	New communication scope
$\mathrm{A}(\overrightarrow{a}) \stackrel{\text{def}}{=} P_A$	Process definition
$\mathrm{p} + \mathrm{q}$	Choice operator
if ……then	Condition statement
$\vec{a}$	A vector of names
$p.q$	sequencing

The example below perhaps clarifies the use of some of the presented notations from Table 4.1:

$$\bar{\boldsymbol{x}}\langle \boldsymbol{z} \rangle.\, \boldsymbol{z} | \boldsymbol{x}(\boldsymbol{y}).\, \bar{\boldsymbol{y}} \rightarrow \boldsymbol{z} | \bar{\boldsymbol{z}}$$

$\bar{x}\langle z \rangle$ and $x(y)$ are complementary pairs of actions between two processes. y is a formal parameter and z is an actual one. In this example, the parallel composition operator denotes that $\bar{\boldsymbol{x}}\langle \boldsymbol{z} \rangle.\, \boldsymbol{z}$ and $\boldsymbol{x}(\boldsymbol{y}).\, \bar{\boldsymbol{y}}$ are running concurrently. $\bar{x}$ and x react and invoke a substitution ($z/\ y$).

4.3. Formal Specification of Priority Scheduling Mechanism (PSM)

This section gives the description of internal proposed model which descried in general within chapter 3. It presents the main internal components and the formal specification in details of the proposed model.

4.3.1. Main components

The detailed architecture of the proposed model is shown in Figure 4.1. In brief, the main components of the architecture include:

Gatekeeper (GK): This component deals with admission control; it is initializing a new handler for each new client.

Handler: The handler passes links from each client to the appropriate virtual buffer for classification and processing.

Scheduler: The scheduler deals with the thread priorities that are created for each client, access to the processor, and processors may be added here.

Counter: A number of components keep a record of the number of each type of request that are currently handled.

Client: This component represents client side in the system.

The client side generates a new request, only after a previous request is completed. This cycle is then repeated.

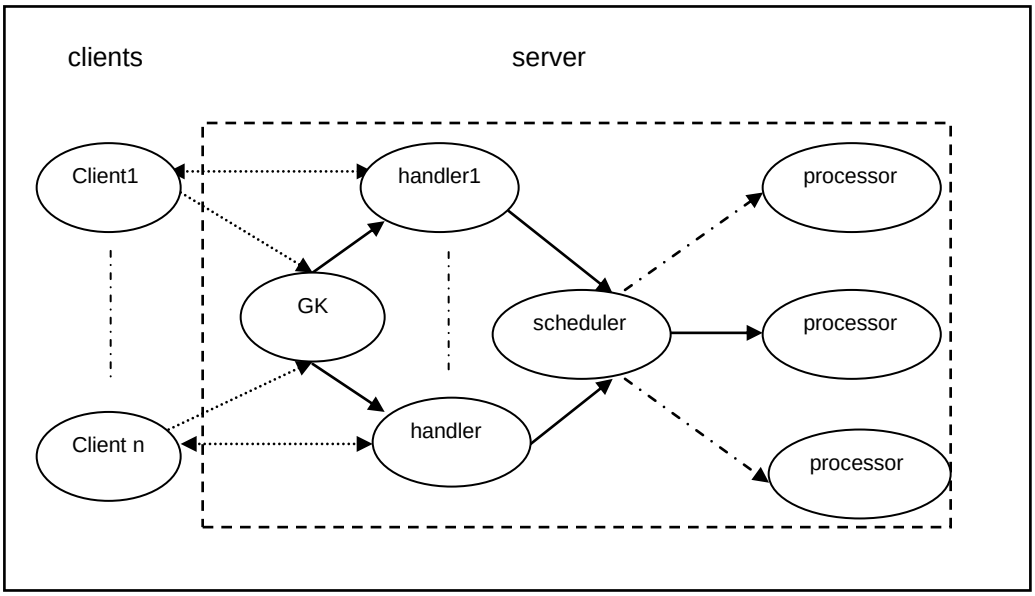

Figure 4.1: Specification architecture of PSM model

Some clients wait patiently for their requests to be completed, and if these requests are rejected, due to server overload, further requests will continue to be submitted until send 'end' message. The server provides different levels of priority of service for different types of request. Low priority requests concern the browsing/searching of E-commerce products and high priority requests concern the actual purchasing phase.

The Java runtime support system includes a scheduler component which does not appear in the specification but it appears in the real implementation, which

replaces a non-deterministic choice. A scheduler's job is to decide the order in which the requests should be handled.

4.3.2. Formally specifying PSM

The following section illustrates in details the formal specification of PSM.

4.3.2.1. The E-Commerce System

The e-commerce system is made up of a population of clients that interact with a web server as shown in Figure 4.1. Formal representation of this system, using π-calculus, is:

$$eCommerce \stackrel{\text{def}}{=} \text{new } connect,\ \ Client|Client| \ldots\ |Client|Server$$

This indicates that the server publishes a single action $connect$ used by all clients to request service that is accessed by the client.

4.3.2.2. The Client

Client behaviour is an important part of the analysis of interactive systems in general and in e-commerce systems in particular. In order to collect experimental results of the performance of the model, it is important to specify a client component. The behaviour of the client is independent of the system used to process its request. The client receives a link from the server which is used to connect it with a dedicated handler. After the client has sent a number (*n*) of

browse and buys requests, it informs the handler that it has finished its requests. The client then terminates the session. The client is specified as follows:

$$Client \triangleq^{\text{def}} connect(link).Client`_0\langle link\rangle$$

$$Client`_i(link) \triangleq^{\text{def}} \overline{link}\langle buy\rangle.link(resp).Client`_{i+1}\langle link\rangle \quad 0 \leq \text{i} < n$$

$$+\overline{link}\langle browse\rangle.link(resp).Client`_{i+1}\langle link\rangle$$

$$Client`_n(link) \triangleq^{\text{def}} \overline{link}\langle end\rangle.0 \quad \text{n} \in \text{N}$$

The specification shows the client making a non-deterministic choice between browse and buy requests; this is right since the aim of this specification is to describe how the component may interact, not why a particular action occurs. The implementation replaces this non-deterministic choice with one based on the relative probabilities of the two actions.

4.3.2.3. The Server

The server is made up of a number of components, including: gatekeeper, handler, processor and a counter.

$$Server \triangleq^{\text{def}} new\ \ bind, \overrightarrow{up}, \overrightarrow{down}$$

$$Gatekeeper|Processor|Counter_0\langle up_1, down_1\rangle\ \ |Counter_0\langle up_2, down_2\rangle$$

This specification indicates that the server publishes information that is used by the counters, each of which uses a distinguished member from the vector $\overrightarrow{up}, \overrightarrow{down}$. The bound action $bind$ is restricted to:

- *Gatekeeper* that passes action for client to handler,

- *Processor* that processes the request and send the response *and*

- Counter that checks action to submit the request from handler to processor.

but there are no controls about which, if any, of these components may use it. The next sections describe the server components in more detail.

The Gatekeeper

When a client connects to the gatekeeper, a fresh action is passed to both the client and a new instance of the handler, allowing these components to interact privately.

$Gatekeeper \stackrel{\text{def}}{=} new\ link\ \overline{connect}\langle link\rangle.Gatekeeper|Handler\ \langle link\rangle$

The Handler

Firstly, the handler receives a request from its client. If the request indicates that the client has finished, then the handler terminates; otherwise, the handler passes links to the appropriate counter of the handler's component, which checks to see whether there is too many requests in the system to process the request. If not, the client is informed that the request has been rejected;

otherwise, the request is processed, the counter is decremented, and the results of the query passed to the client. The handler then waits for the next request to be processed.

$$Handler(link) \stackrel{\text{def}}{=} link(req)$$

$$($$

$$if\ req = end\ then\ 0$$

$$+$$

$$if\ req = buy\ then\ Handler`\langle link, req, up_1, down_1\rangle$$

$$+$$

$$if\ req = browse\ \ then\ Handler`\langle link, req, up_2, down_2\rangle$$

$$)$$

$$Handler`(link, req, full, dec, connect) \overset{\text{def}}{=} full(ans)$$

$$($$

$$if\ ans = yes\ then\ \overline{link}\langle reject\rangle.\ Handler\langle link\rangle$$

$$+$$

$$if\ ans = no\ then\ Process\langle link, req, dec\rangle$$

$$)$$

$$Process(link, req, dec) \overset{\text{def}}{=} \overline{bind}\langle req\rangle.bind(resp).\overline{dec}.\overline{link}\langle resp\rangle.\ Handler\langle link\rangle$$

The Processor

The processor merely receives a request to process, it then returns the results. Its purpose is to model the shared resources of the e-commerce system that need to be accessed under mutual exclusion. The proposed model has only one processor initially, but the model could be expanded to include several.

$$Processor \overset{\text{def}}{=} bind(req).\overline{bind}\langle resp\rangle.\ Processor$$

The processor does not implement scheduling; as mentioned before; the aim of this specification is to describe how the component may interact, not why a particular action occurs. The implementation replaces non-deterministic scheduling with the priority scheduling mechanism.

The Counter

The counter keeps track of how many active requests of a particular type (buy, browse, etc.) there are in the system. When the system reaches its maximum capacity, the counter reports to the handler that no more requests can be accepted. Please remember that this approach is implemented using finite storage capabilities to eliminate the starvation for low priority customer due to the high priority virtual buffer's will never become empty, then low priority request never being processed.

$$Counter_0(up, down) \stackrel{\text{def}}{=} \overline{up}\langle no\rangle.Counter_1\langle up, down\rangle$$

$$Counter_i(up, down) \stackrel{\text{def}}{=} \overline{up}\langle no\rangle.Counter_{i+1}\langle up, down\rangle \qquad 1 \leq \mathrm{i} < max$$

$$+\ down.Counter_{i-1}\langle up, down\rangle$$

$$Counter_{max}(up, down) \stackrel{\text{def}}{=} \overline{up}\langle yes\rangle.Counter_{max}\langle up, down\rangle \qquad max \in N$$

$$+down.Counter_{max-1}\langle up, down\rangle$$

4.4. Formal Specification of Extended Models

This part covers the formal specification of other models that have extended from PSM which were already explained in chapter 3. The main structure of PSM is defined based on the fact that all of the other models have some different parts, in accordance with each of the model's requirements.

4.4.1. Formal Specification of Review Model

The main formal specification is the same, but a third type of request – 'review' – is added. In the review model, counters are used to count the number of active requests of each type.

The modified part covers the server's, client's and handler's formal specifications because the new types of request are added with the counter and other components will stay the same as the main structure of PSM in section 3.2.

4.4.1.1. The Server

As mentioned before the different here from the basic PSM server that the third counter component is added to count review requests.

$Server \stackrel{def}{=} new\ bind, \overrightarrow{up}, \overrightarrow{down}$

$Gatekeeper|Processor|Counter_0\langle up_1, down_1\rangle$

$|\text{Counter}_0\ \langle \text{up}_2, \text{down}_2\rangle|\text{Counter}_0\ \langle \text{up}_3, \text{down}_3\rangle$

4.4.1.2. 4.4.1.2 The Client

$Client \stackrel{def}{=} connect(link).Client`_0\ \langle link\rangle$

$Client`_i(link) \stackrel{def}{=} \overline{link}\ \langle buy\rangle.link(resp).Client`_{i+1}\ \langle link\rangle \quad 0 \leq \text{i} < n$

$+\overline{link}\langle review\rangle.link(resp).Client`_{i+1}\ \langle link\rangle$

$+\overline{link}\langle browse\rangle.link(resp).Client`_{i+1}\ \langle link\rangle$

$Client`_n\ (link) \stackrel{def}{=} \overline{link}\langle end\rangle.0 \qquad \text{n} \in \text{N}$

As described review type is added to this model.

4.4.1.3. The Handler

$$Handler(link) \stackrel{\text{def}}{=} link(req)$$

$$\begin{aligned}
(\quad & if\ req = end\ then\ 0 \\
& + \\
& if\ req = buy\ then\ Handler`\langle link, req, up_1, down_1 \rangle \\
& + \\
& if\ req = review\ then\ Handler`\langle link, req, up_2, down_2 \rangle \\
& + \\
& if\ req = browse\ then\ Handler`\langle link, req, up_3, down_3 \rangle \\
)
\end{aligned}$$

The handler deals with the new review.

4.4.2. Formal Specification of Portal Model

For this project, two different servers are proposed: the web server and the portal server. Each of these servers comprise of five main components, including the: *gatekeeper, scheduler, handler, processor and counter, as* shown

in Figure 4.2. The main structure is already explained in 3.4.3; here the internal details are given.

4.4.2.1. Architecture of Portal Model

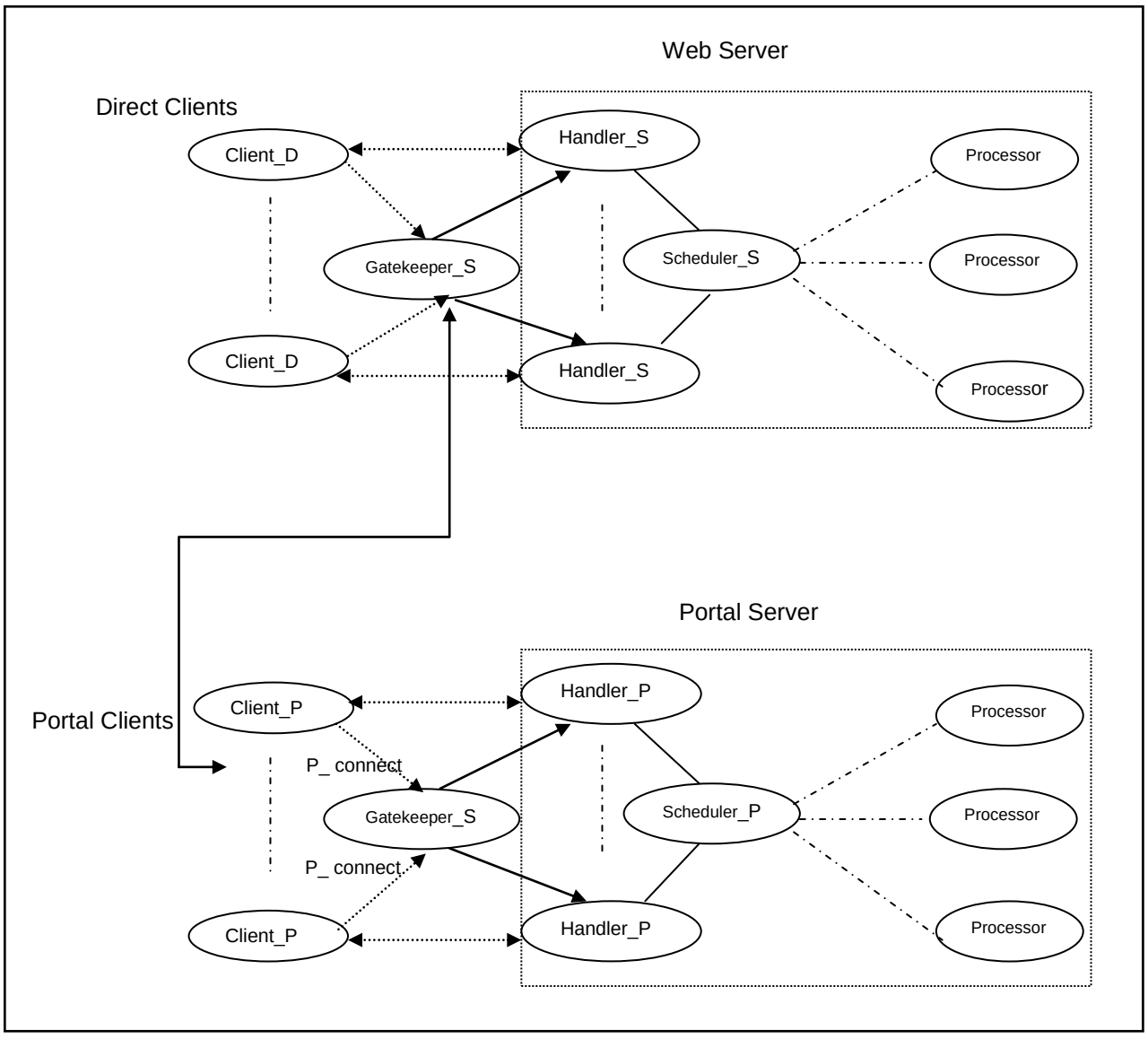

Figure 4.2: Specification architecture of portal model

The modified part covers the server's, client's and handler's formal specifications because the other server and other type of client are added and other components will stay the same as the main structure of PSM.

4.4.2.2. E-Commerce System

As shown in Figure 4.2, modern e-commerce services are composed of a number of clients, the service provider web server and the business web portal server. The following specification represents an interaction between the different components of e-commerce services.

$$eCommerce \stackrel{\text{def}}{=} \text{new } P\_connect, S\_connect$$

$$ClientD| \ldots \ldots |ClientD|Server|Portal|ClientP| \ldots \ldots |ClientP$$

4.4.2.3. Web Server Components

Service Provider Web Server

This web server is composed of a number of components which implement the performance management mechanism. The server specification of the service provider web server is as follows:

$$Server \overset{\text{def}}{=} new\ \overrightarrow{up}, \overrightarrow{down}, do\ Gatekeeper\_S | Processor | Counter_0\langle up_1, down_1\rangle$$

$$|\text{Counter}_0\langle \text{up}_2, \text{down}_2\rangle | \text{Counter}_0\langle \text{up}_3, \text{down}_3\rangle$$

Direct Clients

The client receives a link from the web server which is used to connect it to a dedicated handler. After the client has sent a number of fixed browse, or buy, requests, it informs the Handler_S that it has finished its requests.

$$ClientD \overset{\text{def}}{=} S\_connect(link).ClientD`_0\langle link\rangle$$

$$ClientD`_i(link) \overset{\text{def}}{=} \overline{link}\langle Direct\_buy\rangle.link(resp).ClientD`_{i+1}\langle link\rangle \quad 0 \leq \text{i} < n$$

$$+\overline{link}\langle Direct\_browse\rangle.link(resp).ClientD`_{i+1}\langle link\rangle$$

$$ClientD`_n(link) \overset{\text{def}}{=} \overline{link}\langle end\rangle.0 \qquad \text{n} \in \text{N}$$

Gatekeeper_S

$$Gatekeeper\_S \overset{\text{def}}{=} new\ link\ \overline{S\_connect}\ \langle link\rangle.Gatekeeper\_S\ |Handler\_S\ \langle link\rangle$$

This component deals with admission control and handles the initial direct client requests. When a client connects to the Gatekeeper_S, a fresh action is passed

to both the client and a new instance of the Handler_S, allowing these components to interact privately.

Handler_S

An instance of this component is created for each direct client that connects to the system. It starts by receiving requests from a client and terminates when the client has finished submitting requests. It passes links to the appropriate counter (specified below) which checks to see whether there is any space to accommodate the client's request. If there is capacity, the request is processed, the counter decremented, and the results are passed to the client. If there is no space, the request is rejected and the client is informed accordingly. The handler then waits to process the next request.

$$Handler\_S(link) \triangleq link(req)$$

$$
\begin{aligned}
(\quad & if\ req = end\ then\ 0 \\
& + \\
& if\ req = Portal\_buy\ then\ Handler\_S`\langle link, req, up_1, down_1\rangle \\
& + \\
& if\ req = Direct\_buy\ then\ Handler\_S`\langle link, req, up_2, down_2\rangle \\
& +
\end{aligned}
$$

$$if\ req = Direct\_browse\ \ then\ Handler\_S`\langle link, req, up_3, down_3\rangle$$

$$)$$

$$Handler\_S`(link, req, full, dec, connect) \triangleq full(ans)$$

$$($$

$$if\ ans = yes\ then\ \overline{link}\langle reject\rangle.\ Handler\_S\langle link\rangle$$

$$+$$

$$if\ ans = no\ then\ Process\langle link, req, dec\rangle$$

$$)$$

$$Process(link, req, dec) \triangleq \overline{do}\langle req\rangle.do(resp).\overline{dec}.\overline{link}\langle resp\rangle.\ Handler\_S\langle link\rangle$$

4.4.2.4. Portal Server Components

Portal Server

The portal server is composed of a number of components which implement the performance management mechanism. The portal server specification is as follows:

$$Portal \triangleq new\ up_1, down_1, do\ Gatekeeper\_P | Processor | Counter\langle up_1, down_1\rangle$$

Portal Clients

The portal client receives two links, one from the portal server which connects it with a dedicated Handler_P to process the browse requests, and another link from the web server which connects it with a dedicated Handler_S to process the buy requests. After the portal client has sent a number of browse and buy requests, it informs the specified handler that it has finished its requests.

$$ClientP \triangleq P\_connect(link).S\_connect(link1).ClientP_0`\langle link, link1\rangle$$

$$ClientP`_i(link, link1) \triangleq \overline{link1}\langle Portal\_buy\rangle.link1(resp).ClientP`_{i+1}\langle link, link1\rangle$$

$$+ \overline{link}\langle browse\rangle.link(resp).ClientP`_{i+1}\langle link, link1\rangle \qquad 0 \leq \mathrm{i} < n$$

$$ClientP_n`(link, link1) \triangleq \overline{link}\langle end\rangle.\overline{link1}\langle end\rangle.0 \qquad \mathrm{n} \in \mathrm{N}$$

Gatekeeper_P

This is specified in a similar way to that of Gatekeeper_S, detailed above.

$$Gatekeeper\_P \triangleq new\ link\ \overline{P\_connect}\langle link\rangle.Gatekeeper\_P\ |Handler\_P\langle link\rangle$$

Handler_P

This is also specified in a similar way to that of Handler_S, detailed above.

$$Handler\_P(link) \stackrel{def}{=} link(req)$$

$$($$

$$if\ req = end\ then\ 0$$

$$+$$

$$if\ req = browse\ \ then\ Handler\_P`\langle link, req, up_1, down_1\rangle$$

$$)$$

$$Handler\_P`(link, req, full, dec, connect) \stackrel{def}{=} full(ans)$$

$$($$

$$if\ ans = yes\ then\ \overline{link}\langle reject\rangle.\ Handler\_P\langle link\rangle$$

$$+$$

$$if\ ans = no\ then\ Process\langle link, req, dec\rangle$$

$$)$$

$$Process(link, req, dec) \stackrel{def}{\equiv} \overline{do}\langle req\rangle.do(resp).\overline{dec}.\overline{link}\langle resp\rangle.\ Handler\_P\langle link\rangle$$

In addition, the portal server has the same processor and counter definitions as the web server in 4.3.2.3.

4.5. Unified Modelling Language (UML) of the E-commerce Systems

Unified Modelling Language (UML) [39] is a standard notation which is used to specify high-level design of software systems. It provides structured and graphical notations for the specification of software systems. In this thesis, UML is used in order to simplify the understanding of the design of the proposed models formally specified in the previous sections. The main feature of the UML is that it provides users with diagrammatic view of the complex systems (e.g., modern e-commerce systems) such as they are easy to understand. It is also helpful to have a standardised modelling language, such as UML, to understand the concepts and relationships between the model's components. In the light of introducing formal specifications and a UML model, UML is useful to bridge the complexity of implementation gaps due to the big difference between π-calculus structure and the chosen Java language programming structure.

This thesis restricts the UML specification of the proposed models to using activity diagrams, class diagrams and sequence diagrams. However these are

sufficient to specifying the structure and behaviour of the proposed e-commerce models.

4.5.1. Activity Diagram

The activity diagram records the dependencies between activities [40]. Two activity diagrams are therefore drawn, the client view and the server view.

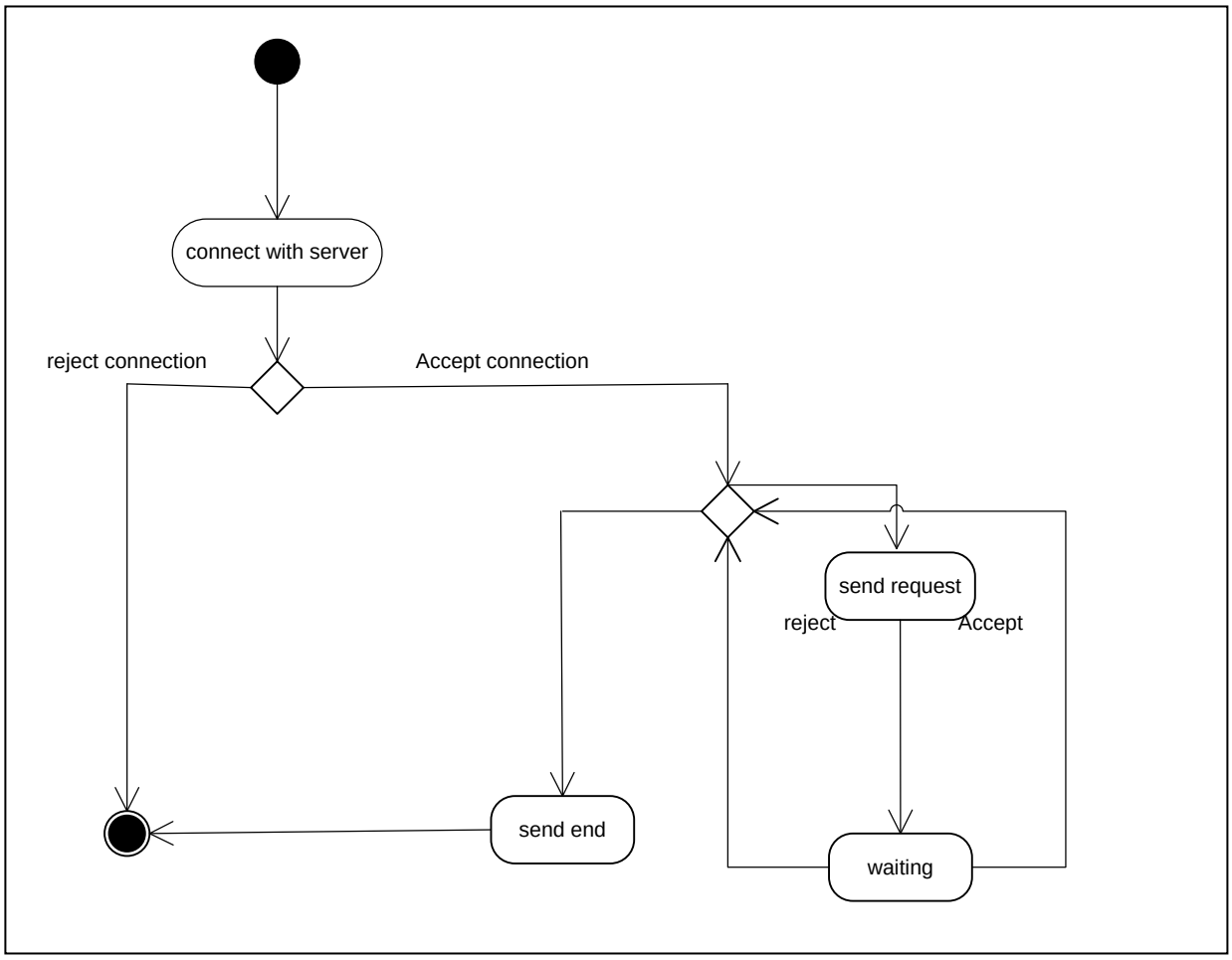

Figure 4.3: Activity diagram (client view)

Figure 4.3 illustrates an activity where the client connects with the server, based on the following:

- The server accepts the connection or rejects it, if there are socket problems. This is a concrete example of adding specific concerns into the implementation which are not appropriate at the specification level.
- When the server accepts the connection, it will create a handler to deal with client requests that corresponds to '$Handler(link) \stackrel{\text{def}}{=} link(req)$' in π-calculus.
- The client will establish the session and can send many requests through its handler until it decides to disconnect, corresponding to: '$Client \stackrel{\text{def}}{=} connect(link).\, Client_0^{`}\langle link\rangle$' in π-calculus.
- In this step, the system applied capacity condition determines whether each incoming request is accepted or rejected.
- The client will send end-messages to release transmission control protocol (TCP) connections with the server, this translates to: '$Client_n^{`}(req) \stackrel{\text{def}}{=} \overline{req}\langle end\rangle.0$' in π-calculus.

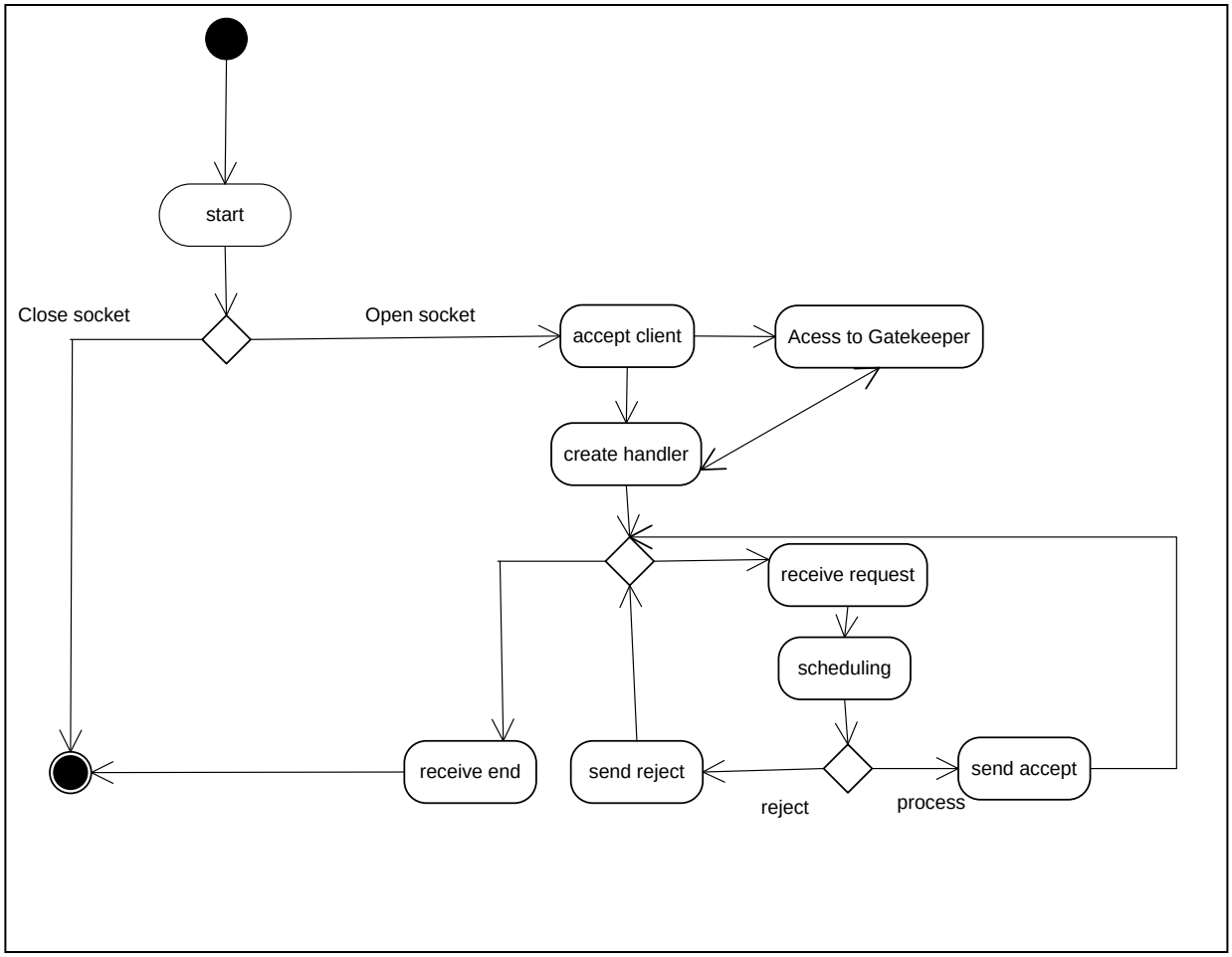

Figure 4.4: Activity diagram (server view)

Figure 4.4 presents a small difference from Figure 4.3; it shows the system from the server side which is based on the following.

Note: the condition shape in the activity diagram denotes choice operator '**+**' in π-calculus and each state in activity diagram denotes action in π-calculus.

- The server accepts the client along the socket; otherwise, it rejects the client by closing the socket.

- The gatekeeper creates handler dynamically for each client. This translates to:

  $$`Gatekeeper \stackrel{\text{def}}{=} new\ link\ \overline{connect}\langle link\rangle.\ Gatekeeper|Handler\ \langle link\rangle'$$

  in π-calculus. The server is therefore made up of gatekeeper and handler components.

- Through the handler the server receives incoming requests, it decides to process or reject these requests according to capacity conditions. This action is repeated until the server receives an end-message from the client.

- The server receives end-messages and then disconnects.

4.5.2. Class Diagram

Class diagrams show classes and their relationship with other classes. Figure 4.5 describes, in detail, the mechanism of the proposed model by defining the classes.

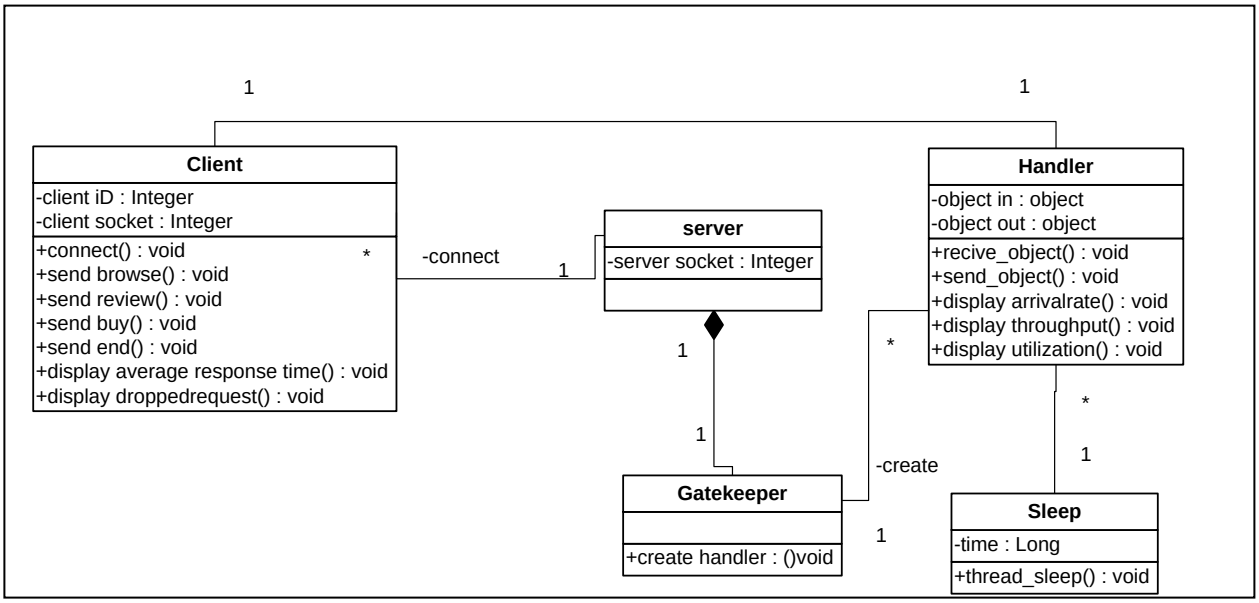

Figure 4.5: Class diagram

The class diagram of the proposed model includes five classes. Note that each class denotes the active component in π-calculus in the model (see Figure 4.1):

- **Client class:** It contains the attributes client ID and client socket. It has operations that mean parallel composition '$p|q$ 'in π-calculus:
    - *connect():* which will connect with the server using the socket number.
    - *send browse()*:responsible for sending browse requests.

- *send review()*:responsible for sending review requests.
- *send buy():*responsible for sending buy requests.
- *send end():*responsible for send end-message.
- *display average response time():* calculate and display average response time for each client.
- *display dropping request():* get the percentages of dropped requests and display them for each client.

- **Server class:** It contains the attribute server socket and it has a composition link with Gatekeeper class.
- **Gatekeeper class:** it has: create handler() operation that call for each client to handle its requests.
- **Handler class:** It contains the attributes of objects in and objects out and has the following operations:
  - receive_object(): responsible for receiving the objects, including the multiple types of requests (browse, review or buy) from client.
  - send_object(): responsible for sending the objects that represent the result (accept or reject the requests) to client.
  - displayarrivalrate(): calculate and display the average arrivals for each client.

- display throughput(): calculate and display the throughput for each client.
- display utilisation(): calculate and display the utilisation of the server at each point of the assumed time.

- **Sleep class:** the sleep class represents processing actions inside the CPU, it contains the attribute time that indicates how long the request takes to process. It has thread_sleep() operations which operates under mutual exclusion.

For all of these methods, *receive()* in class diagram means $x(y)$ in π-calculus, and *send()* method means $\bar{x}\langle y\rangle$ in π-calculus. Figure 4.5 provides more information on these classes and the relationships between them.

- Multiple clients can connect with the server.
- There is one handler for each client, created dynamically by the server.
- When one handler accesses the sleep class, all other handlers will wait until that handler finishes its processing.

4.5.3. Sequence Diagram

Sequence diagrams clearly depict the sequence of events, showing when objects are created and destroyed; they are excellent at depicting concurrent operations.

Figure 4.6 explains the order of the activity that occurred among a set of objects, this was dependent on time-based ordering.

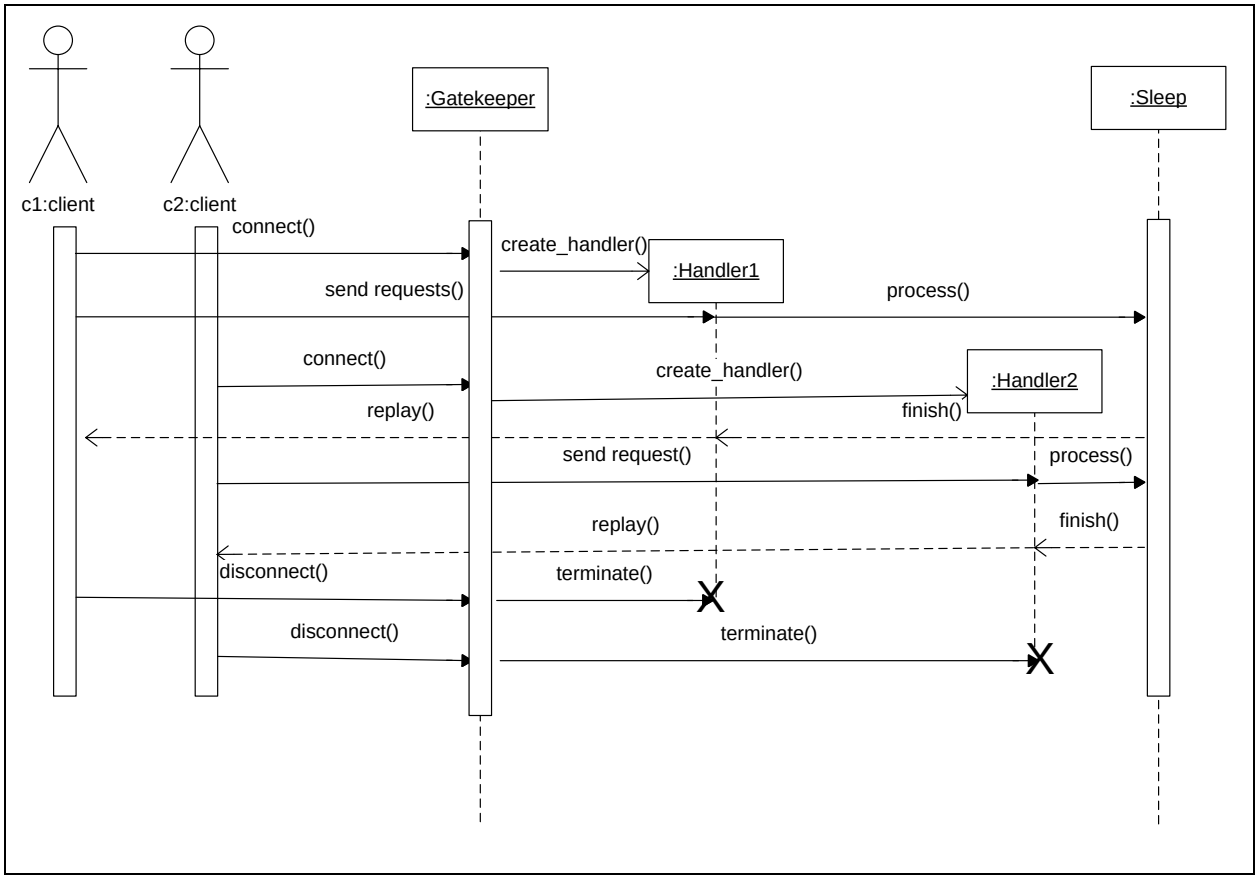

Figure 4.6: Sequence diagram

The complete sequence diagram can be divided into the following steps:

- Client1 initiates TCP connection with the server which means a new communication scope '$(new\ x)P$' in π-calculus.

- The Gatekeeper responsible to create a handler1 dynamically for client1.
- Client1 starts its session and starts to send its requests, thus messages are sent along the channel link ' $\overline{req}\langle type\ of\ request\rangle$'in π-calculus.

- Handler1 processes incoming requests by accessing sleep functions, '$\overline{connect}\langle req\rangle$', *in* π-calculus.

- Sleep class finished from the request processing which corresponds to the Processor performing the '$\overline{connect}\langle resp\rangle$' *action in* π-calculus.

- Handler1 sends a replay message to client1 through the server that sends as link in π-calculus.
- Client1 disconnects with the server corresponding to'$Client_n^{`}\ (req) \stackrel{\text{def}}{=} \overline{req}\langle end\rangle.\ 0$' in π -calculus.

- The server terminates handler1.

- Client2 performs similar actions.

4.6. Summary

This fourth chapter has presented the formal specification of the priority scheduling mechanism (PSM) that helped to consider alternative approaches and refine the one chosen, as well as providing a framework for the implementation, thus raise the confidence that the models have been correctly implemented as they were designed. With the assessment of PSM, π -calculus helped to build this model with its property in easily and quickly way. In addition, all of the modified parts of the formal specification of the extended models have been illustrated in this chapter. To reduce the gap between the formal specification phase and the implementation phase, UML diagrams were illustrated in this chapter. In addition, the link between π -calculus and UML diagrams has been illustrated as a means of identifying the relationship between the formal specification and the UML diagrams. Furthermore, the implementation phase will be described in the next chapter.

Chapter 5: Implementation of the Proposed Models

5.1. Introduction

This chapter describes how the models presented in chapters 3 and 4 are implemented. The proposed model is implemented as a simulation tool using the Java programming language. Simulation is one of the most widely used techniques in Internet and web applications research [61] and it provides a means of evaluating the system. With simulation, it is easy to test and analyse the performance of web applications. If the underlying infrastructure (such as networks, web servers, etc) is not available for measurement, then simulation can be considered as a convenient way to predict the performance and provide more details than an analytical model because it can be made as accurate as desired [60]. This chapter gives a detailed explanation of the simulation model which has been used as a basis for the evaluation of the proposed model. The type of simulation used, in this work, is a multi-actor simulation.

The multi-actor systems are a new modelling paradigm inherited from the object modelling [35]. Moreover, multi-actor simulation is the most popular approach used by software developers to naturally understand, model and develop

important classes of complex distributed systems. In addition, it is used to study the behaviour of models and is adaptable to different scenarios. An actor can be defined in several different ways, depending on the model's viewpoint. For example, in software engineering, an actor refers to a function module consisting of a set of programs that interact with other such modules to perform some form of computation, and it is capable of real-time simulation; thus, multi-actor simulation frameworks were produced in Java to support the research and to evaluate the performance of the proposed models. The multi-actor simulation was developed to reflect the target models as accurately as possible; therefore, for all experiments, the proposed models were solved using multi-actor simulation.

There are different platforms on the use and development of simulation. Quality Network Appliance Provider (QNAP) [70] is a general purpose simulation tool which can be used for simulating different systems but mainly for network simulation. The proposed simulation is developed for web servers supported by real time which is not appropriate to use QNAP. The novelty and contribution of the proposed multi-actor simulation is to demonstrate more realistic and specific characteristics and requirements of web servers. Also there are some real web servers which are implemented using Java and multithreading, etc. In addition, multi-actor simulation can be easily developed using Java; so the proposed simulation can easily change or add any condition to justify the simulation results. Therefore, the proposed simulation model provides multiple solutions.

The structure of the simulation is based on the state of the proposed models which mimic e-commerce networks that may should have server and clients; so, the implementation of the proposed approaches (see section 3.4) were set, as middleware components or actors (e.g. client, server, scheduler, etc.) were seamlessly integrated by the Java programming language. Java language was chosen because it is a common language, so it is easy to extend the proposed simulation by adding new components. In addition, Java language is able to have concurrent systems that closely follow π-calculus specifications which described in detail in Chapter 4. Furthermore, Java facilitates communication between server and client using Transmission control protocol (TCP).

In the simulation framework, a multi-threaded web server is illustrated; any number of threads can be generated for servicing the requests. The number of concurrent threads, or clients, denotes simulation capacity.

Performance metrics is considered and it needs a central reporting object to collect statistic results without interfered, thus mutual exclusion is added into implementation phase to guarantee the exclusive access to shared resources, more details is in section 5.5.

The structure of the main simulation for the priority scheduling mechanism (PSM) is introduced; this is the basic building block that is enhanced to implement the extended models from section 4.4.

In addition, this chapter presents different traffic models because it is a useful way of validating the models under different traffic models; therefore, there is, in part, model evaluation with the self-similar traffic model.

5.2. Simulation Structure

In the simulation model a complete range of components, parameters and methods are used in order to simulate the e-commerce web servers and to evaluate their performance by taking into account various parameters including: response time, throughput, arrival rate, utilisation and percentage of dropped requests. The internal details (i.e. processor, disks and network interface) are not considered in the proposed simulation model.

5.2.1. Simulation Components

The proposed simulation tool is developed in the following setup. It uses Java, JCreator LE 4.00 on Microsoft Windows XP. TCP is used in order to ensure reliable communication and to avoid loss of messages between component systems [62]. The different components of the simulation tool are modelled using object-oriented approach (as described in Chapter 4). Table 5.1 represents the main components or actors of the simulation tool.

Table 5.1: Components of the Simulation Tool

Component	Description
Client	It models client side and includes: ID, socket, number of clients, request, response times and drop requests.
Server	It models server side and includes: sockets, service time distribution objects, sleep objects, client objects, arrival rates, throughputs and utilisation. Note that the request is provided as an argument when a client object is instantiated; therefore, a thread is created for a client's objects, with its arguments.
Request	It models requests that includes its type and the order number.
Sleep	It models accessing CPU time.
Buffer	It models the different virtual buffers that provide: insert requests and removes them after processing, it includes: counters.
Distribution	It models the service time, such as exponential distribution and Poisson distribution functions.
Report	It models result recording methods.

5.2.2. Simulation methods and parameters

The simulation model has many parameters and methods that could be used for produces performance metrics which are presented in section 5.3.

5.2.2.1. Client Parameters

The client code includes in a class which is instantiated multiple times to represents a set of clients who share the same characteristics, but they differ in one characteristic – that is the number of each type of request which is different because this number is generated randomly.

Simulation parameters, governing the generation of clients' sessions, are summarised as follows:

- The number of clients to be simulated is defined as an input parameter *N*; once *N* is indicated, the run method is called for each client until *N* becomes zero or when no client appears in the system.
- The number of requests for each client, *NR*, divides randomly into different types of request. Note that an object of the request class is created.
- Think time means a random delay between requests [72].

5.2.2.2. Server Parameters

- The Handler decides whether to accept or reject service requests, according to the storages capacity and scheduler conditions. A thread for each client is created within the server side – its primary responsibility is to pass on generated requests.

- The service time of request is exponentially distributed with the same mean (μ) for all types of request. The service time is modelled by accessing an instance of the sleep class running in its thread to emulate periods in which a process waits response from a back-end server or database. Synchronisation is necessary for mutual exclusive access of sleep class for reliable communication between created threads.

- The counters' size is assumed as *B* for each type of requests and initialised to '0'. Note that it should be an object of the buffer class, and its primary responsibility is to check the buffer size synchronously and to increase and decrease the counter's value of specified buffer.

- The simulation parameters used have been initialised and presented in Table 5.2:

Table 5.2: Validation configuration parameters

Parameters	Value
N	100-1000
NR	100
Λ	1
μ	100
B	100
Request	Buy, review...etc.
Think time	Uniform distributed or exponential distribution

N: Number of clients, NR: Number of requests, λ:arrive rate, μ:mean service time, B:Buffer size.

5.2.2.3. Session and Time

As mentioned before that the proposed simulation is real time multi actor simulation, so this section explains time and session. The period where all requests are sent over the TCP connection, between the client and server, is called a session – as mentioned within previous chapters. A session is established for each client, this contains different types and different priorities of requests that are generated randomly (independently and uniformly). The duration of the session includes: think time between the interactions that are generated randomly with uniform distribution. The service time is exponentially distributed with a mean, $\mu=100$, for all types of requests. It is the a standard mean for most works [73] to spent at different request execution phases (e.g., waiting for a thread, processing in the database, etc.).

The simulation keeps track of each session by giving an identification (ID) number to each client that is sent along the client and server connection. In addition, each request is completely determined by the object parameter, such as the type of request and the *ID* of the client to make sure that is coming from the same client during a session.

The server uses multiple threads where each connection or client is assigned to a dedicated handler thread so, by varying the number of connections, the load on the server can vary. Moreover, different arrival rates can be produced by changing the number of clients or the waiting time. Figure 5.1 illustrates time-

slicing in generated handlers – i.e. the think time (between the submission of each request from the client side) and the response time (which is the total processing time and waiting time). When the server is under loaded the waiting time tends to be short resulting in small response times and when overloaded the overflow of requests can increase the response time. Note that this is the average response time experienced by the clients. Therefore, the simulation tracks the time and assumes these times are measured in units of seconds; the response time starts when the request is sent from the client, the current time of the system is recorded when the response is received and again when it is finished, the system then records the difference from these two recorded times.

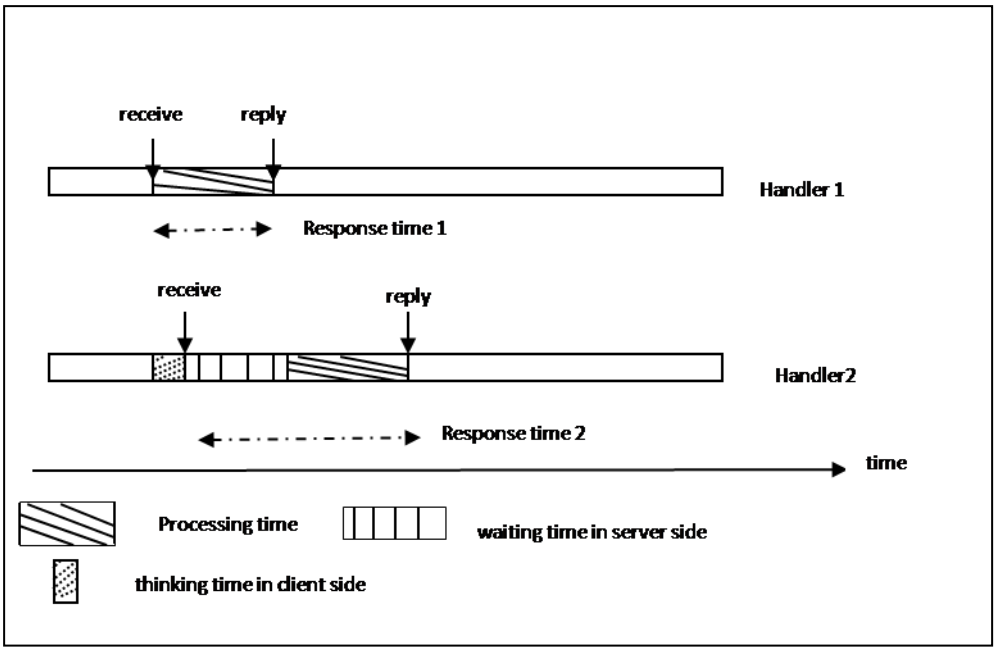

Figure 5.1: Time slicing

5.3. Performance Metrics

To study the behaviour of the system, the following metrics were used (see Figure 5.2), which would have the maximum impact on the performance in the design and analysis of any system and will greatly increase the chances of the system satisfying user performance requirements:

- Throughput: it is obtained by dividing the total number of accepted requests by the client's session time. The overall throughput is to the total throughput for all clients.

- Response time: is observed by the client over different number of clients. Response time is defined as the time difference between when a request is sent and when a successful response is received from the server. The average response time is calculated when the response time is divided by the number of accepted requests.

- The percentage of dropped requests is obtained by dividing the number of rejected requests by the total number of requests and then by multiplying them by 100 to get the percentage.

- The actual average arrival specifies when each client arrives. This is measured by dividing the number of arrivals, within the client's time, by the client's session time. The session time starts when the connection between the client and server starts and then ends when it disconnects.

- The overall arrival rate is equal to the total arrival average for all clients as presented in Figure 5.2 – this is equal to the number of clients multiplied by the average arrival of clients.

- Server utilisation: It can be used to track server performance regressions or improvements and it is calculated as the percentage of time (*R*) during which the server is busy processing requests during a simulation, for all clients the utilisation is $(U = \sum_{i=1}^{n} R_i/\text{simulation\_time})$.

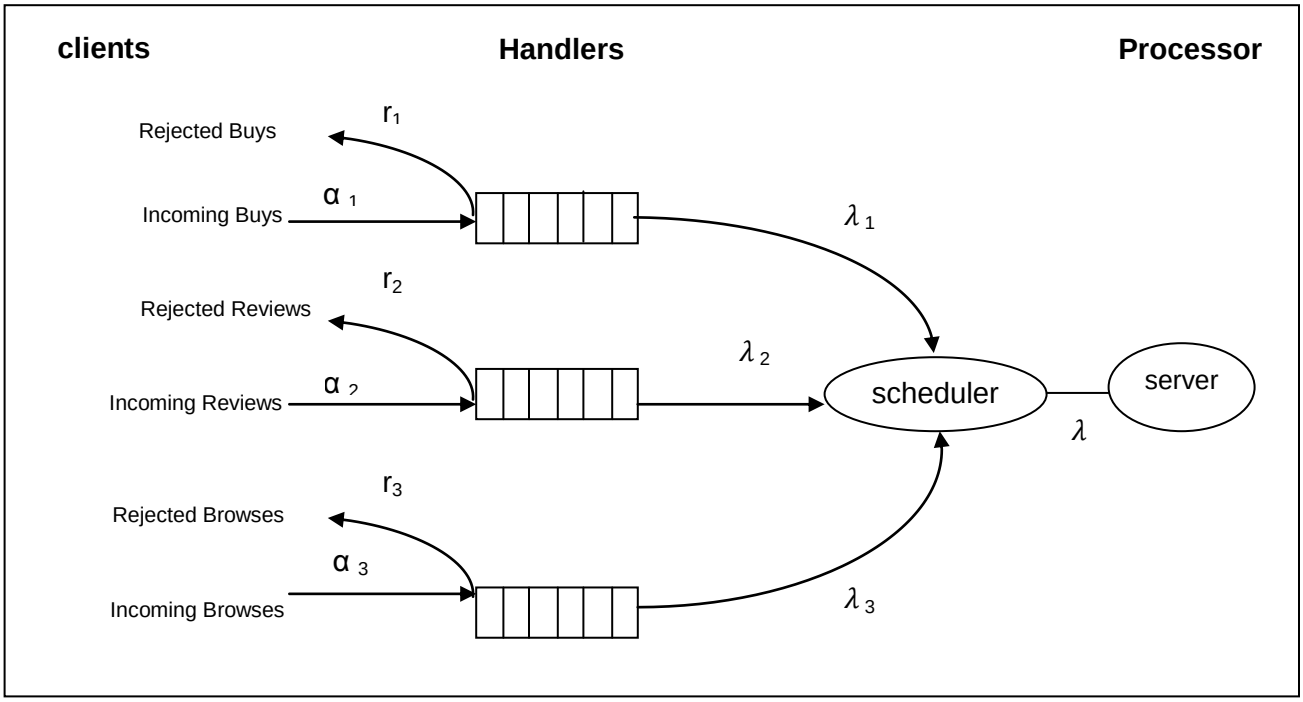

Figure 5.2: Performance Metrics

Figure 5.2 illustrates the main performance metrics as following:

λ in (5.1) is the arrival rate for all clients at shared processor

$$\lambda = \sum_{i=1}^{n} \lambda_i \qquad (5.1)$$

r in (5.2) is the rate of rejected requests

$$\sum_{i=1}^{n} r_i \qquad (5.2)$$

α in (5.3) is arrival rate into admission control system,

$$\sum_{i=1}^{n} \alpha_i \qquad (5.3)$$

(5.4) presents the actual arrival rate which is calculated by subtract rejected request (5.2) from arrival (5.3) for each client

$$\lambda_i = \alpha_i - r_i \qquad (5.4)$$

5.4. The Simulation Traffic Model

The usual starting point when evaluating the performance of e-commerce systems is to use a traffic model which has a major role in analysing real world traffic. In order to design a network, it is important to understand the traffic characteristics of the network. Traffic is a key component of the proposed

simulation as it shapes the e-commerce environment under the TCP protocol which handles most Internet traffic. For instance, using the Poisson distributed process for requests, which arrive with rate λ and exponential distribution for service rate with mean of μ, can generate traffic that is comparable to that expected from a population of real world clients. If the client arrivals are according to Poisson processes, and service times are exponentially distributed, the system can be modelled by a multi-dimensional Markov chain [59]. In addition, a part of the validation considered different traffic models, such as burst traffic which increased confidence in the proposed simulation due to its flexibility in the use of several types of traffic models. In other words, it is easy to adopt a new traffic model, by just changing the instantiation routine of the client object and by not changing the other parts of the simulator.

5.4.1. Poisson distribution

Network arrivals are often modelled as Poisson processes for analytic simplicity. Poisson distribution is a discrete distribution which takes on the values x=0, 1, 2, 3 ... It is often used as a model for the number of events in a specific time period *t*.

The Poisson probability mass function (pmf) is given by:

$$f(x) = \begin{cases} \frac{e^{-\alpha}\,\alpha^{x}}{x!}, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad (5.5)$$

The cumulative distribution function (cdf) is given by:

$$f(x) = \sum_{i=0}^{x} \frac{e^{-\alpha}\,\alpha^{i}}{i!}, \alpha > 0 \qquad (5.6)$$

5.4.2. Exponential distribution

Exponential distribution describes the time between events in a Poisson process at constant average rate λ. The probability density function (pdf) is given by:

$$f(x) = \begin{cases} \lambda\, e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad (5.7)$$

The cumulative distribution function (cdf) is given by:

$$f(x) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad (5.8)$$

The exponential random number generated to represent the service time or the inter-arrival time in a simulation is calculated by solving for *x* in Equation (5.8) using the inverse transform technique as follows:

$$R = 1 - e^{-\lambda x} \qquad (5.9)$$

$$e^{-\lambda x} = 1 - R \quad (5.10)$$

$$-\lambda x = \ln(1 - R) \quad (5.11)$$

$$x = -\frac{1}{\lambda} \ln(1 - R) \quad (5.12)$$

R is a uniform random number distributed on [0, 1] and *λ* is the rate in service completions or arrivals per unit time.

5.4.3. Burst traffic

Traffic that follow a Poisson arrival process, would have a characteristic burst length which would tend to be smoothed by averaging over a long enough time scale. However, measurements of real traffic indicate that significant traffic variance is present on a wide range of time scales.

Self-similar traffic has observable bursts at a wide range of timescale thus it can exhibit as well as long range dependent properties. The properties of self-similar traffic are very different from the properties of traditional models based on Markovian models (e.g., Poisson). The scale-invariant characteristics of the self-similar traffic are in strong contrast to traditional network traffic models which show burstiness at short time scales but are smooth at large time scales [57]. However, self-similar is extremely complex and intractable to practice.

Therefore, in this simulation a simple traffic is provided that includes bursty behaviour.

Bursty traffic is used in simulation which refers to an uneven pattern of requests transmission: sometimes a large number of requests are transmitted while at other times it might be a very small number thus burst traffic could exhibit a high variance.

Barrier synchronisation primitive technique is used to enforce the stopping of execution between a number of threads or processes at a given point and prevents further execution until all threads have reached the given point.

In burst traffic the thinking time is exponential distribution with a mean of 1 second for about 30% of generated requests. For the remaining 70% of requests suppose there is not any specific thinking time between them; in other words there is zero thinking time as presented in Figure 5.3. For more details refer to Figure 5.4.

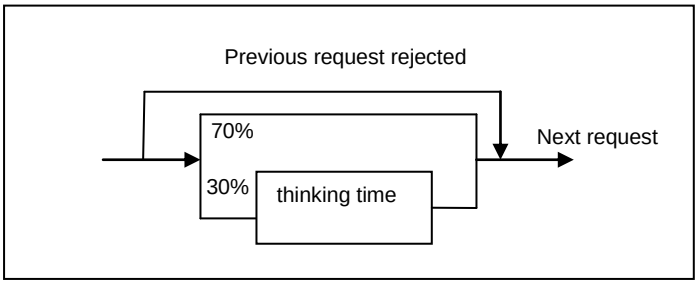

Figure 5.3: Generating requests in Burst Traffic

5.5. Implementation of the Simulation Tool

This section describes the implementation of the aforementioned methods and models of the simulation tool. The implementation is based on the π-calculus formal specification presented in Chapter 4.

The implementation replaces non deterministic choice in the specification with random choice. In addition, to collect the statistic results, a central object is needed thus the implementation exploits the facilities provided by the active object model [53] or the active actors therefore a new feature is added into implementation that not at specification part. However, shared data can cause multiple update problems when two or more threads access shared data and they try to change it at the same time. In order to prevent race conditions occurring, the simulation considers a monitor that encapsulates shared resources or shared variables, such as counters of virtual buffers that are shared between all generated threads. A lock is defined to associate with the monitor and each function requires that it is locked before it continues its execution therefore synchronised methods are used to enforces mutual exclusion and ensures that threads interact safely more details are in section 5.5.5.

A thread is created for each client and the priority of each incoming request is assigned according to its type and class. This is the gatekeeper component, as presented in Chapter 4; to illustrate: a highest priority is given to high priority

class such as buy or portal buy; a middle priority is given to middle priority class, such as review requests; and, a minimum priority is given to low priority classes, such as browse. All clients share the first connection over TCP with the server and they are served in a FIFO order, then a separate connection is produced for each handler and they are served like in the specification with a non-deterministic order. Of course, the order of the thread to execute requires monitoring for mutual exclusion purposes. Requests of the same class are served on a FIFO basis. In addition, requests from the same client are served in FIFO order because the client will not produce the next request until the last one has an answer.

5.5.1. The Client

The client is the only contrived part of the implementation; the other components, described below, could be used without modification in a production version of the protocol.

The non-deterministic choice in the model was implemented as a probabilistic choice, with 80% of requests being browse in the main model and 50% in the extended models because most of customer browses or searches without buying any things. It was assumed that a client would terminate after generating 100 requests. The performance results in the following chapter were obtained from several hundred runs of the simulation. In addition, several variables accumulate metrics such as response time since each client

calculates their own performance metrics and theses metrics can be combined in the end of the simulation to produce the overall results. In addition, each run executed a number of clients [100-1000] running concurrently and with access to a central report class. Therefore, the client was improved to include calls to a reporting object, for statistical purposes.

5.5.2. The Server

The model was conveniently translated because Java allows active processes within other tasks which permit the *processor* and *counter* components to be translated, as tasks, without them being visible to the clients. The capacity of the virtual buffers was assumed to be 100, for each, more details will be given in chapter 6.

The *gatekeeper* component was very easy to implement using Java's condition statement: depending on the type of request received, the *handler* component will direct that request to the appropriate virtual buffer then process it or reject it.

The model shows a non-deterministic choice between processing a buy or a browse request which is implemented as a priority choice: if a buy request is available then process it, otherwise process the browse request.

5.5.3. Requests generation

Each client will send a hundred requests with random types. The client waits for a response which is checked to calculate the response time when it successful or adding to dropping request when it is rejected. After transaction completion, the client will wait an exponentially distributed think time between some requests following burst traffic algorithm (see section 5.4.3) then the process starts again.

Figure 5.4 shows the code of non deterministic choice in specification by generating different types of requests randomly.

```
for(int i = 1; i <=100; i++) {//100 requests for each client

        try{    r1 = generator.nextDouble(); //generate random number

                if(r1>=0.8)   {  try{

                                        type Packetb = new type(3,"hi",id,i);

                                         lastFlushTimeb3 = System.currentTimeMillis()/1000;

                                         out.writeObject(Packetb);

                                         clientt.report.totalportalb() ;

                                         Object datb = in.readObject();

                                             type packetb=(type)datb;
```

```
                                        //check server meassages

                                       if(packetb.getStatus().equals("buyacceptP"))

                                          { lastFlushTimeb2 = System.currentTimeMillis()/1000;

                                            timeb=(lastFlushTimeb2-lastFlushTimeb3);

                                             clientt.report.add_portalb(timeb);

                                         }//end if

                                      else if(packetb.getStatus().equals("buyrejectP"))

                                              {clientt.report.Drop_portalb() ;  } //end else if

                                  }//end try

                          catch(Exception ioEx1){System.out.println("buy exception receive-
"+ioEx1);}

                              }// end high priority

          else if((r1>=0.5)&&(r1<0.8))

                              { // medium priority requests  }

         else if((r1>=0.0)&&(r1<0.5))

                          {// low priority requests}

               }catch(Exception e) { System.err.println("IO Exception main try");}
r2 = delay_r2.nextDouble();

      if(r2>=0.7)
```

```
try {   delay=Math.round(Po.nextExpo(delay_r,100.0)); //think time in burst traffic

                sleep(delay);

  } catch(Exception e) {System.err.println("Interrupted");}

  else

  {System.out.println(" nothing"); } //no think time

}//end for loop
```

Figure 5.4: Requests generating

5.5.4. Handler Creation

Figure 5.5 shows the server when it connects with each client through the specific socket, then it creates a handler for each clients. Here thread denotes the handler and it should be out and in object streams to send and receive the request. This implements *new* action in π-calculus.

```
while(true) {

    // Blocks until a connection occurs:

    Socket socket = s.accept();

    if (x==0) //number of clients

    {       start_time= System.currentTimeMillis()/1000;

          Start_time.println("   "+start_time);//save simulationn start time
```

```
 }

x++;

System.out.println("after s.accept");

try {                    outToClient1 = new ObjectOutputStream(socket.getOutputStream());

                         inFromClient1 =  new ObjectInputStream(socket.getInputStream());

                         t= new ServeOneEcho (socket, inFromClient1, outToClient1);

                          System.out.println("new ServerOne ");

} catch(IOException e) {   // If it fails, close the socket,

                             // otherwise the thread will close it:

                           socket.close();

                          }//end catch

  }//end while
```

Figure 5.5: Handler creation

As seen in Figure 5.5, the counting of clients is implemented for statistic purpose and the start time is considered when the server accepts the first client. The server is switched on always to receive any number of connections at any time thus the infinite loop is considered.

5.5.5. Mutual Exclusion

Figure 5.6 shows the mutual exclusive technique within sleep class. Synchronised method is defined and it has mutually exclusive access to the data encapsulated by the object. Synchronised method has two effects:

- It is not possible for two invocations of synchronised methods on the same object to interleave. When one thread is executing a synchronised method for an object, all other threads that invoke synchronised methods for the same object suspend execution until the first thread is done with the object. Note that, it is random order for the waiting threads who get the access to the method.
- When a synchronised method exits after the specific time *d*, it automatically releases the lock. This guarantees that changes to the state of the object are visible to all threads.[53]

```
public class sleep implements Serializable {

private long t1,t2,total;

public synchronized  long Sleep(long d)

  {    try{

          t1=System.currentTimeMillis();
```

```
            Thread.currentThread().sleep(d);

             t2=System.currentTimeMillis();

             total=t2-t1;

           }

         catch (InterruptedException ie) {System.out.println("InterruptedException "+ie); }

      return total;

  }
```

Figure 5.6: Mutual exclusion

5.6. Simulation Validation

A series of simulations were performed to measure the performance and behavioural specifics of the proposed model and extended models which described in chapter 3. The proposed model and the extended models are classified as closed models [54], which means that the performance concepts can be obtained by varying the number of clients and thus causes the variation in the arrival rate. Furthermore, model validation was conducted by running the models under the same input conditions for a number of different clients further details will be given in chapter 6.

After each run of the simulation, using specific numbers of clients, the Handler response to reject the request when the buffer is full or to continue processing. The sum of all the actual arrival averages was calculated and an arrival rate from that point (100,200,... etc) was presented. At the same time, the sum of all throughputs was produced. The desired measures of performance: average arrival, throughputs, utilisation, average response time and percentage of dropped requests, should be produced for each client in a report form when the simulation ends, via a link between the simulation and Microsoft Office's Excel. This allows the recording of output values and calculates the overall averages of all the clients by using central share report object.

5.7. Summary

Simulations play a vital role in analysing the performance of e-commerce applications. Simulation programs can be used to closely replicate the e-commerce application, to be remodelled, and in many cases can capture details that may be impossible to obtain from analytical models. This is because the latter can become intractable and prevent the introduction of simplified assumptions. Since multi-actor simulations are powerful tools for analysing real world distributed systems, they have been used for simulation and evaluation throughout the thesis. Java programming language is used to implement the simulation because it supports concurrent programming and it is easy to extend the code by adding new components by other researchers.

This chapter has presented the benefits of the proposed simulation tool. In addition, a full description of the specified simulation has been illustrated to explain the related classes and parameters. Several calculations, used to obtain results, were presented and will be illustrated further in the next chapter. Furthermore, in the next chapter the results from the simulation will be used to evaluate the proposed model.

Chapter 6: Evaluation and Experimental Results

6.1. Introduction

This chapter evaluates the performance of the proposed models through a series of experiments. The simulation tool presented in Chapter 5 is used in the evaluation process. A number of cases have been studied using the proposed models. These models are based on Poisson arrivals, burst arrivals and exponential service time. The experiments cover a wide range of input parameterizations, such as number of clients and different traffic techniques, and demonstrate performance metrics. The experiments take into account all the important parameters of performance metrics such as average response time, throughput, arrival rate, utilisation and percentage of dropped requests in different models and varying number of clients. Experimental results show that the proposed models improve the performance of web servers in different e-commerce setup such as conventional e-commerce systems as well as modern e-commerce systems including web portal with user's reviews. The results also give valuable insight into the performance studies of real e-commerce web servers.

6.2. Experimental setup

The set up of all experiments are mostly the same and there is small different as following:

1. In the first set of experiments, there are two types of request. This set of experiments is used as a baseline to present the importance of the priority scheduling mechanism.

2. In the second set of experiments, another type of request is added to the main priority scheduling mechanism to include review customers and to show that the proposed main mechanism is capable of assigning more than two types of priority to ecommerce requests.

3. In the third set of experiments, the Low Priority Fair Model modifies the basic PSM to improve the performance of browse customers. The effects of buffer size on the performance metrics are investigated using the Low Priority Fair Model. In addition, a set of experiments are conducted to study the behaviour of the Low Priority Fair Model under a bursty traffic technique.

4. In the fourth set of experiments, new features are added to the basic PSM which present the advantage of a portal model. One TCP

connection is used in all experiments except portal model, two TCP connections are used as described in simulation chapter. The effects of different basis models on the performance metrics are investigated in Portal Model using the Basic PSM and Fair Low Priority Model.

6.2.1. Traffic

To be more realistic traffics, bursty traffic that described within chapter 5 is used for all experiments except in Low Priority Fair Model where the comparison between smooth and burst traffic is presented. Note that a Low Priority Fair model was chosen because the results obtained here can be generalized to other models.

6.2.2. Clients

Each client generates one request and waits until the server responds. Then the client may or may not go to sleep, depending on the generated random number. The sleep time, or think time, is a random variable with exponential distribution with a mean of one second. After waking up, the client generates a new request, and so on until 100 requests have been made or until he finishes his session. The same experiment is repeated several times using a different number of clients, which varies between 100 and 1000 clients as mentioned in Chapter 5; this is a large enough number of clients to make sure that at the end of each run, requests are executing concurrently. Moreover, the most basic type

of load testing is used to determine the web application's behaviour under both normal and expected peak load conditions and this appears between 100 to 1000 clients.

6.2.3. Buffers

A typical buffer configuration can be any number in size. The small-buffer model is suitable when buffer size does not depend on the number of clients [37]. After many adjustments as shown in Appendix D, 100 is the better size to monitor the performance, so all buffers' sizes are set to 100 for all types of request.

6.3. Experimental Results

This section presents the results in detail, showing the effectiveness of the proposed models. In all experiments, the key is in requests priority. The experiments demonstrate performance metrics such as arrival rate, throughput, utilisation, response time and dropped requests.

6.3.1. Review Model

Review model as mentioned in chapter 3, adds new parameter to the basic PSM which includes review customers. Therefore, it was extended to classify e-commerce requests into high, middle and low priority requests, instead of two types of requests of the basic PSM. The results of the basic PSM and the Review model are jointly presented given the fact that they share the two types

of requests of ‘browse’ and ‘buy’. That is, the former only deals with ‘browse’ and ‘buy’ while the latter adds another class which is the ‘review’ requests. That is by excluding the ‘review’ requests the Review model converts to the basic PSM.

There are three different requests: browse, review and buy, which are randomly generated from different clients. The number of different types of request varies according to [1]; for example, 20% buy requests, 30% review requests and 50% browse requests. Generally, there are more ‘browse’ requests than ‘buy’ or ‘review’ requests. The results show a clear improvement in the performance of high-priority requests over medium- and low-priority ones. Another aim of the Review model experiments is to show that the proposed approach is capable of assigning more than two types of priority to ecommerce requests. That is, it meets the requirements of modern e-commerce application's development and integration services in order to increase business efficiencies by taking e-commerce applications to a new level.

To study the effect of changing the model load on the performance metrics, the system examined when the service rate is similar for all types of requests with burst traffic.

6.3.1.1. Arrival rate

To consider approximating the behaviour of the proposed model, the arrival rate is presented.

The arrival rates were varied by varying the number of clients; they increased dramatically with more clients or more time. Figure 6.1 illustrates the model arrival rate in terms of the arrived requests of buy, review and browse per second. The X-axis is the number of emulated clients, and the Y-axis refers to arrival rate. As seen in Figure 6.1, the arrival rate increases very fast due to the small thinking time and the large number of dropped requests regarding priority condition for browse and review requests which is: If the buffer of payment requests is empty then process the first review request and If the buffer of payment requests and review requests are empty then process the first browse request. In addition, the buffer status could affect the arrival rate as soon the arrival rate starts to grow above 100 clients due to the full buffer for buy requests that have size of 100 and has the highest priority.

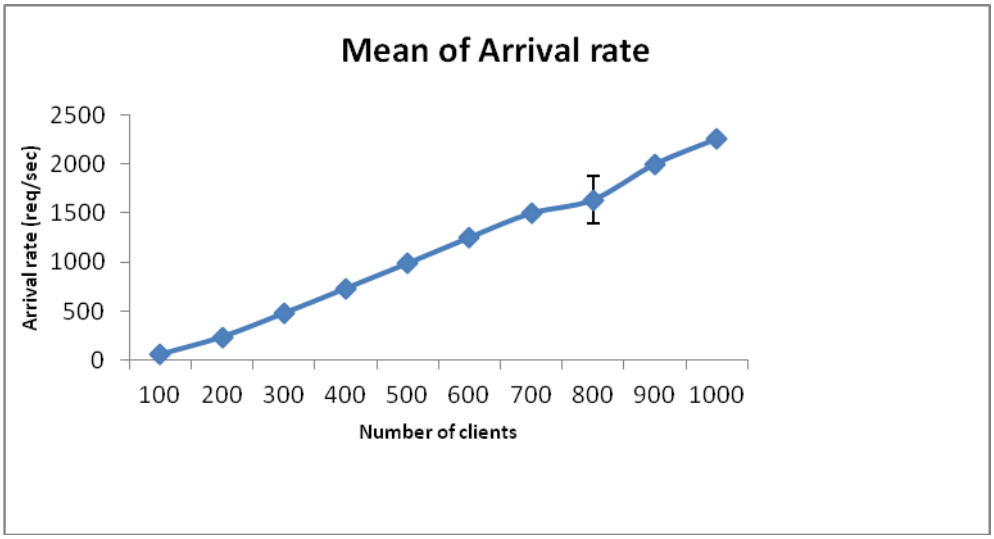

Figure 6.1: Arrival rate in Review Model

6.3.1.2. Throughput

Figure 6.2 shows the throughput to ensure that the model does not introduce excessive processing overheads particularly under overload conditions. The throughput in Figure 6.2 shows the accepted requests per second. It is clear that, by increasing the traffic load, the throughput will slightly increase. The throughput reaches a maximum value when the arrival rate ensures that the processor is fully utilized. In other word, the system exhibits a higher peak throughput of 17 requests per second at 900 clients. The throughput is unaffected by increasing the arrival rate because the system brings some of competition resources into handler rather than completing requests.

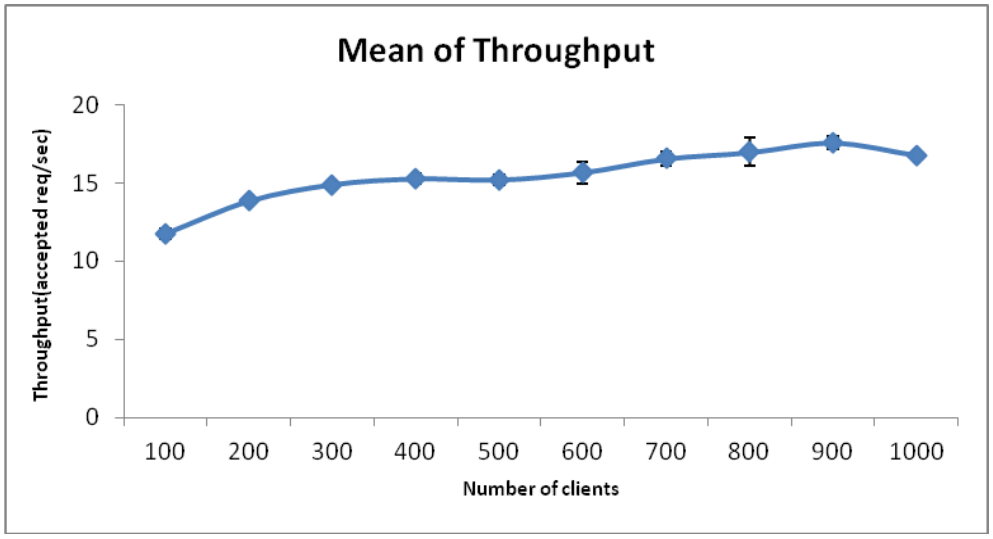

Figure 6.2: Throughput of Review Model

6.3.1.3. Utilisation

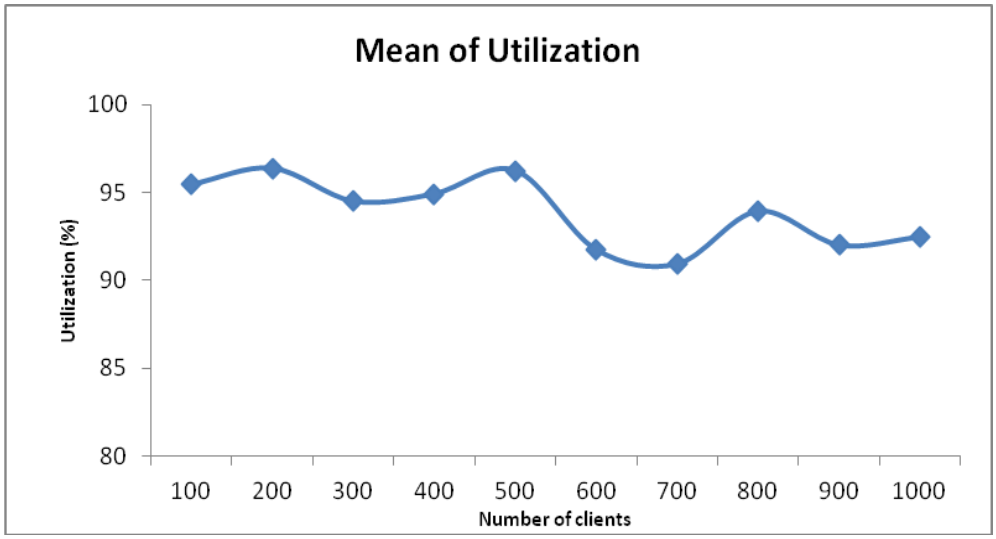

Figure 6.3: Utilisation of server in Review Model

Figure 6.3 shows the utilisation of the server in terms of the percentage of time during which the server is busy processing requests. The X-axis is the number of emulated clients, and the Y-axis refers to utilisation. The utilisation of each run is measured. Most of the handlers are processing high-priority requests. When the number of clients is small, the server is under-utilized and the throughput increases by increasing the number of clients, as shown in Figure 6.2. In the first of the simulation when the client number becomes 100, the utilisation is 95% then it increases due to processing requests state and when the number of clients starts to reaches 1000 the server utilizes 92% thus no time for processing rejected requests, therefore, this small change has no any affects on the throughput. Moreover, the throughput remains constant even if the

number of clients increases. Then the throughput decrease slightly because the rate of incoming requests exceeds the capacity of the server and this situation leads to drop most of incoming requests.

6.3.1.4. Response Time

The observed response time as seen by the client side (see Figure 6.4) can give clear information about the server performance. When the server is close to 100% utilized, it shows that the protocol gives best value by completing most of the highest-priority buy requests.

Three curves are presented in Figure 6.4: these denote buy, review and browse requests and the X-axis is again the number of emulated clients, but the Y-axis is average response time in second. Figure 6.4 shows that larger numbers of clients produce larger loads on the server, which results in the increased average response time of high-priority requests because the server processes most buy requests. However, due to the large percentage of low- and medium-priority requests being dropped, these will have a low average response time. With more generated requests, the number of low- and medium-priority requests completed will decrease because most of these requests will be rejected. However, Figure 6.4 demonstrates this, as the number of high-priority requests completed before the buffer becomes full is high at the beginning of the simulation then it starts to decrease due to the large number of requests. The

marks in the graph for Buy requests show that the error being less than 5% at all points even at 800 clients.

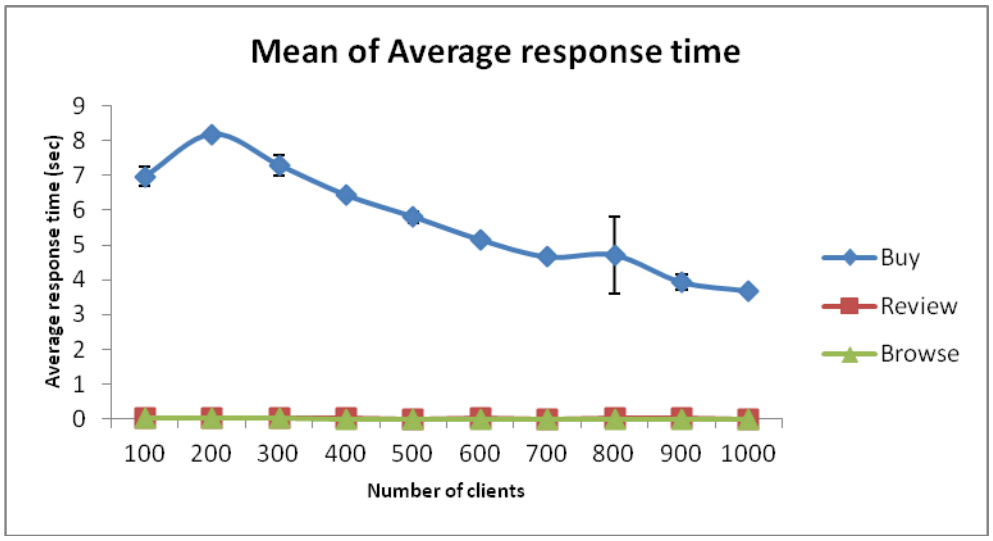

Figure 6.4: Average response time in Review Model

6.3.1.5. Dropped Requests

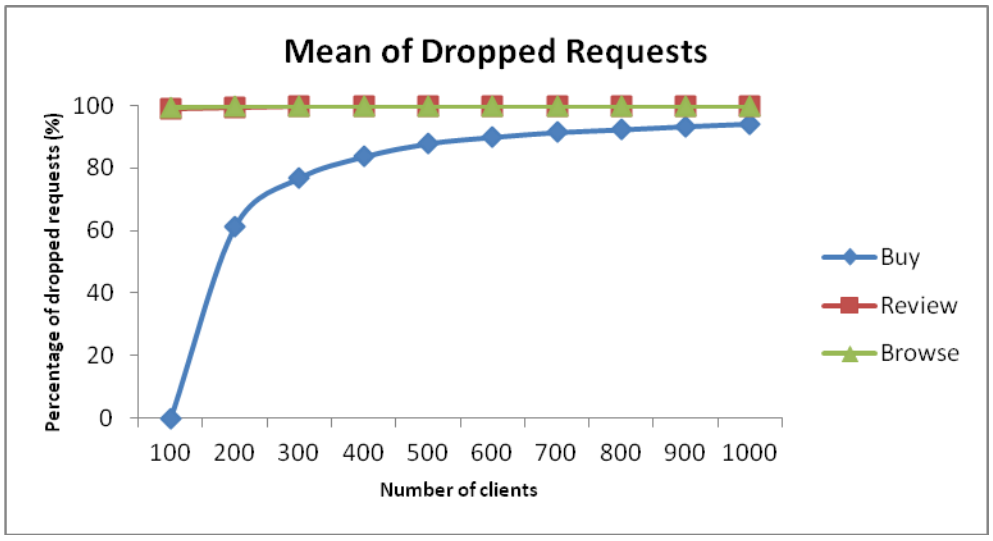

Figure 6.5: Dropped requests in Review Model

Figure 6.5 presents the effect of an increasing number of clients on the percentage of dropped requests for each type. It can be seen that the occurrence of the request being dropped steadily increases for all types upon an increase in the number of clients. This is clearly due to the finite capacity of the buffer for incoming requests.

There are rejected buy requests under heavy load conditions at the same point above 100 clients, so that the average response time becomes small (see Figure 6.4). However, there is a huge number of other rejected low-priority requests. The browse requests start to be rejected a little earlier than review ones because the processor is being occupied with review requests and occupied fully with buy requests which have higher priority.

In summary, after the buffer becomes full the throughput arrival rate and the utilisation are consistent at the same point (Figures 6.2 and 6.3). The throughput reaches the peak point when the server reaches the maximum value of the utilisation as shown in Figure 6.3 and this happen when the arrival rate increases dramatically due to the rejected requests in the same point (Figure 6.1).

6.3.2. Low Priority Fair Model

The low priority fair model as mentioned in chapter 3 modifies the basic PSM to improve performance of browse customers. It gives a small delay to processing

browse requests rather than rejecting them, thus it keeps low priority customers on the web site.

To evaluate the effectiveness of the Low Priority Fair Model, two sets of experiments are designed with different buffer sizes. In the first set of experiments two different types of traffic generation are used – smooth and burst– using exponential distribution and setting the buffer size to 1. The smooth traffic follows a dynamic stochastic process such as random process. In the second set of experiments the buffer size is increased to 100 as mentioned in section 6.2.3 that the better size is 100 after many adjustments. The following results represent the first set of experiments; to be more realistic traffics, suppose the thinking time is randomly generated the probability of exactly one event in one second it means with a mean of 1 second between each request, and is labelled ‘random case’. For ‘burst case’ the thinking time is exponential distribution with a mean of 1 second for about 30% of generated requests. For the remaining 70% of requests suppose there is not any specific thinking time between them; in other words there is zero thinking time (More details are presented in chapter 5).

6.3.2.1. First set of Low Priority Fair Model’s experiments

The following results are maintained when the buffer size is set to 1 for all types of requests.

6.3.2.1.1. Arrival Rate

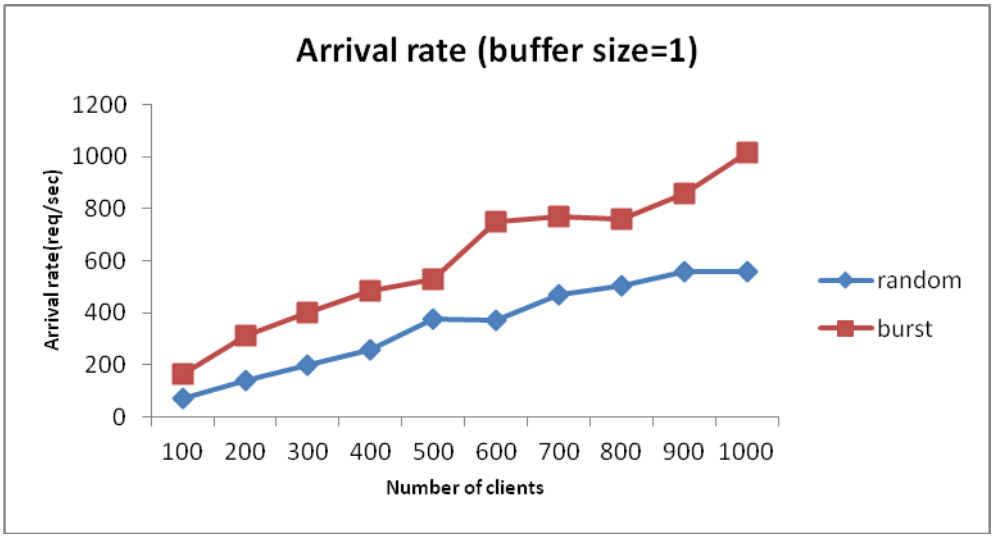

Figure 6.6: Arrival rate in Low Priority Fair Model (buffer size=1)

The difference in arrival rate between the two presented cases is clear (see Figure 6.6). In the burst case, the number of arrivals is higher than in the random case, due to the different speeds of request generation in the two cases. It seems that in the burst traffic the requests are generated more quickly than in the random case due to rejected requests with a small buffer size.

In Figure 6.6, the arrival rate increases until the server begins to utilize 100% and no more requests are received. Note that as the size of the buffer increases, the arrival rate in the two cases will become almost the same (see Appendix C).

6.3.2.1.2. Throughput

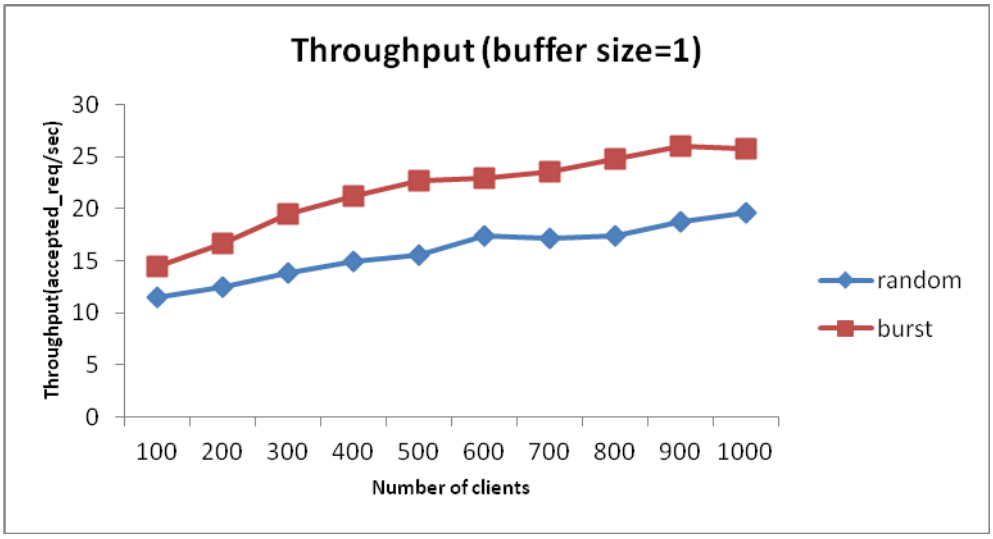

Figure 6.7: Throughput of Low Priority Fair Model (buffer size=1)

Figure 6.7 shows the throughput of the two cases. As shown, the throughput increases with the increase of arriving traffic until the total arrival rate exceeds the service rate; from that point, after dropping, the system throughput will not exceed the service rate for processing a request, which is chosen randomly from an exponential distribution with a mean of 100 ms for each type of request. When the server has high utilisation, the throughput remains constant even though the number of arrivals increases. As shown in Figure 6.7, the throughput of random case is slightly lower than in burst one due to the lower number of requests and more thinking time, even though the service time is equal.

6.3.2.1.3. Utilisation

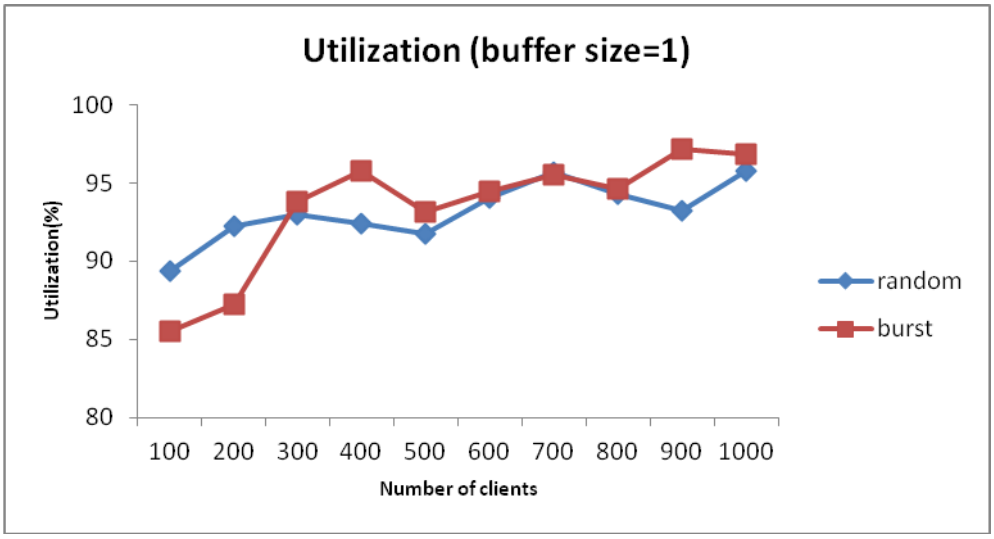

Figure 6.8: Utilisation of server in Low Priority Fair Model (buffer size=1)

Utilisation becomes higher in the burst case than in the random case as presented in Figure 6.8 due to increasing completed requests, which explains the higher throughput in the burst case (see Figure 6.7). The higher arrival rates (see Figure 6.6) will give higher throughput if there is a capacity and, as shown in Figure 6.8, the utilisation is under 100%. Note that the variation in the utilisation is due to the variation in resources availability. Moreover, when the utilisation spikes at a high level, it means more requests will be completed.

6.3.2.1.4. Dropped Requests

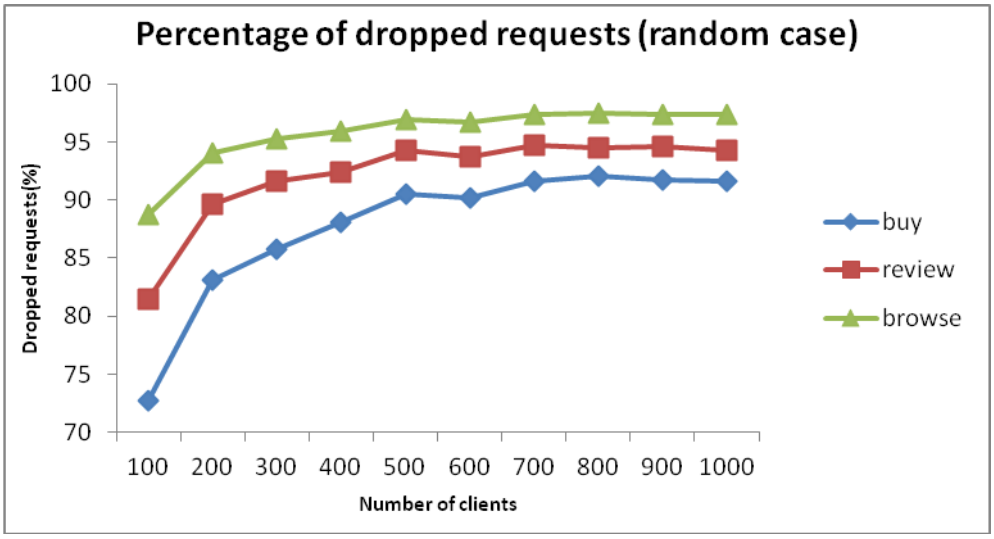

Figure 6.9: Percentage of dropped requests in Low Priority Fair Model (random case, buffer size=1)

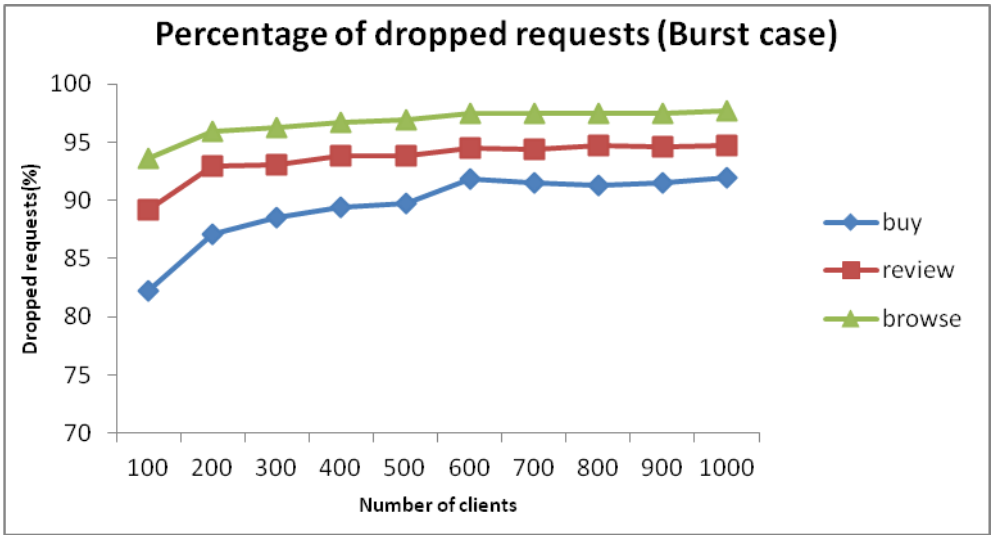

Figure 6.10: Percentage of dropped requests in Low Priority Fair Model (burst case, buffer size=1)

As the results from arrivals and utilisation of the two cases show, the higher utilisation means higher arrivals and more rejected requests. Figure 6.9 shows the higher amount of dropped requests in the random case. With more requests or with a fully-utilized server, the percentage of all types of completed requests will decrease because most of these requests will be rejected by the web server. This improves the relationship between dropped requests and buffer size, as will be clear in the second set of experiments when the buffer is set to 100. In addition, this improves the response time for the clients as less time is spent on rejecting incoming requests (See Figure 6.9 and Figure 6.10).

6.3.2.1.5. Response Time

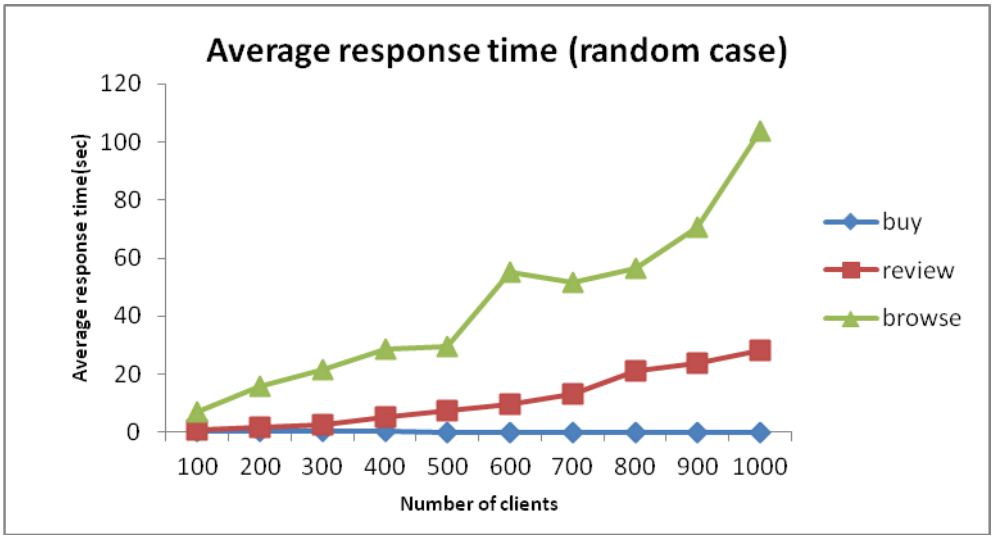

Figure 6.11: Average response time of requests in Low Priority Fair Model (random case, buffer size=1)

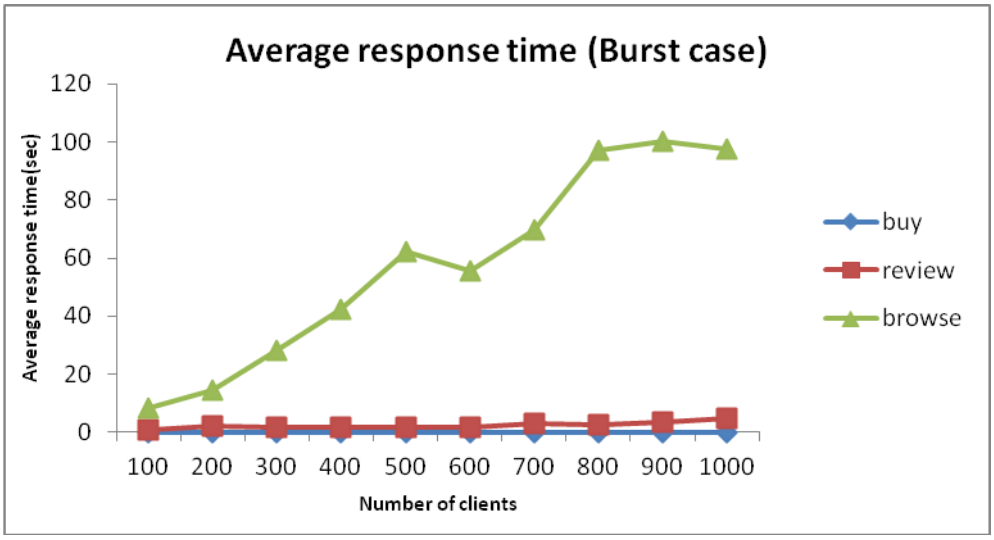

Figure 6.12: Average response time of requests in Low Priority Fair Model (burst case, buffer size=1)

The proposing of the Low Priority Fair Model clearly appears in the average response time of browse requests in Figure 6.11 and Figure 6.12. The high response time spent on low-priority requests is due to the waiting that is caused by processing rather than dropping requests. However, this high response time does not lead to an acceptable state because the buffer size is very small and most of requests from each type are dropped, this is particularly true for browse requests, here the average is taken over a small number of completed requests leading to more variance and a curve that is not smooth.

6.3.2.2. Second set of Low Priority Fair Model's experiments

The following results are maintained under the same setting of the first set of experiments except the buffer size has been changed to 100.

6.3.2.2.1. Arrival Rate

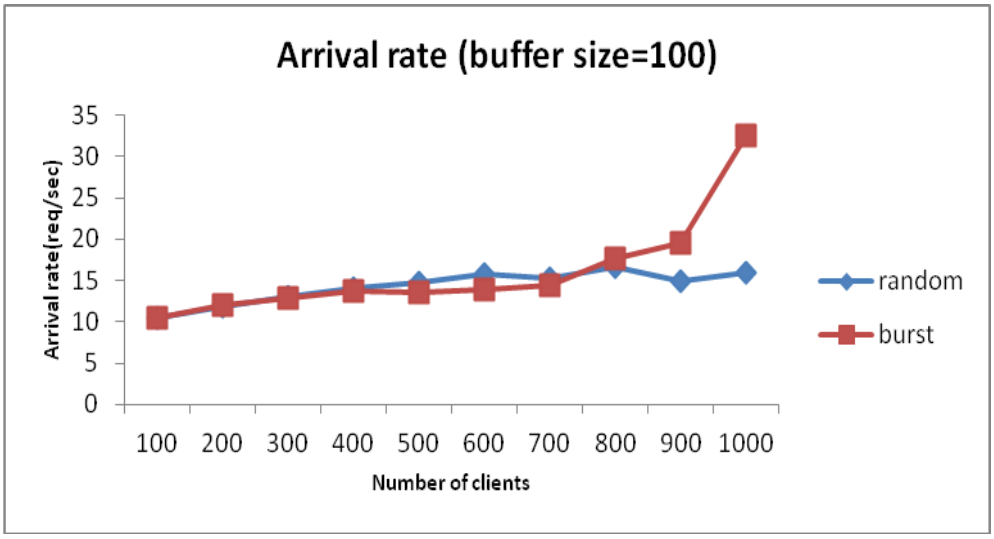

Figure 6.13: Arrival rate in Low Priority Fair Model (buffer size=100)

There is not any significant difference between the arrival rates in the random case and those in the burst case (See Figure 6.13). Therefore, the arrival rate is increased by increasing the number of clients generating requests and then starts to increase faster due to the rejected requests. Note that when the buffer's size increases, the burst traffic will absorb by the large buffer.

6.3.2.2.2. Throughput

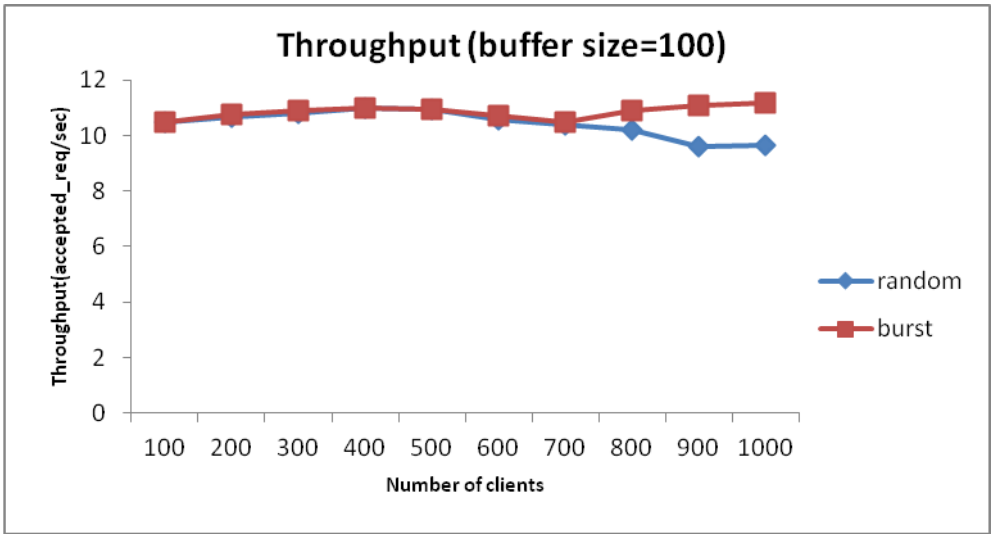

Figure 6.14: Throughput of Low Priority Fair Model (buffer size=100)

The throughputs achieved with both cases are presented in Figure 6.14. There is no significant difference in peak throughput in both cases; however, the throughput starts to decrease clearly in the random case due to the rejected requests because of context-switching overheads. In other word, with low traffic the processor uses most of computational resources to process requests however, high traffic means it has less because handlers are using it. Moreover, the burst traffic could effect on the service rate to become variance.

6.3.2.2.3. Utilisation

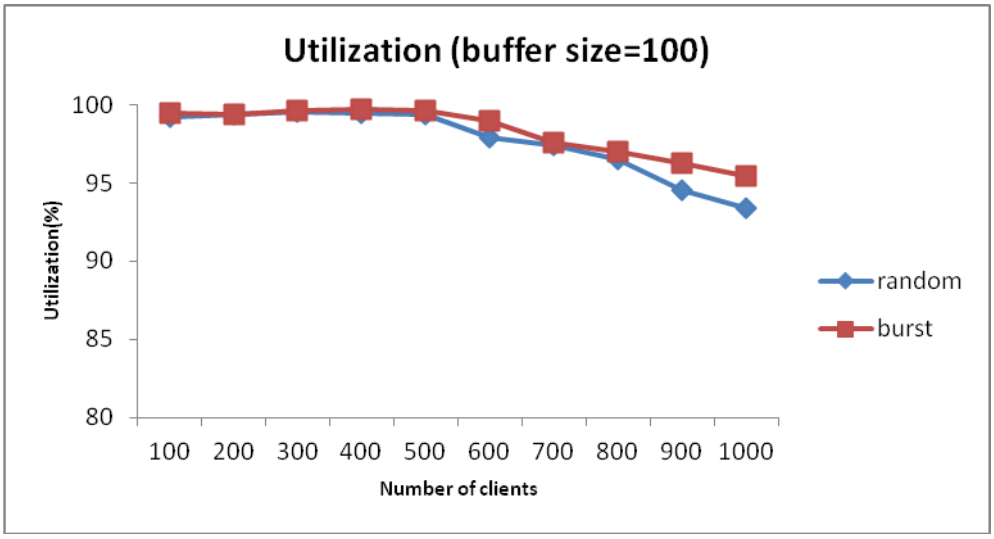

Figure 6.15: Utilisation of server in Low Priority Fair Model (buffer size=100)

Figure 6.15 indicate the utilisation of the two proposed cases when the buffer size is set to 100 and there is no significance difference. It appears that with more incoming requests the server will reject most of them thus the utilisation becomes lower in the two cases.

6.3.2.2.4. Dropped Requests

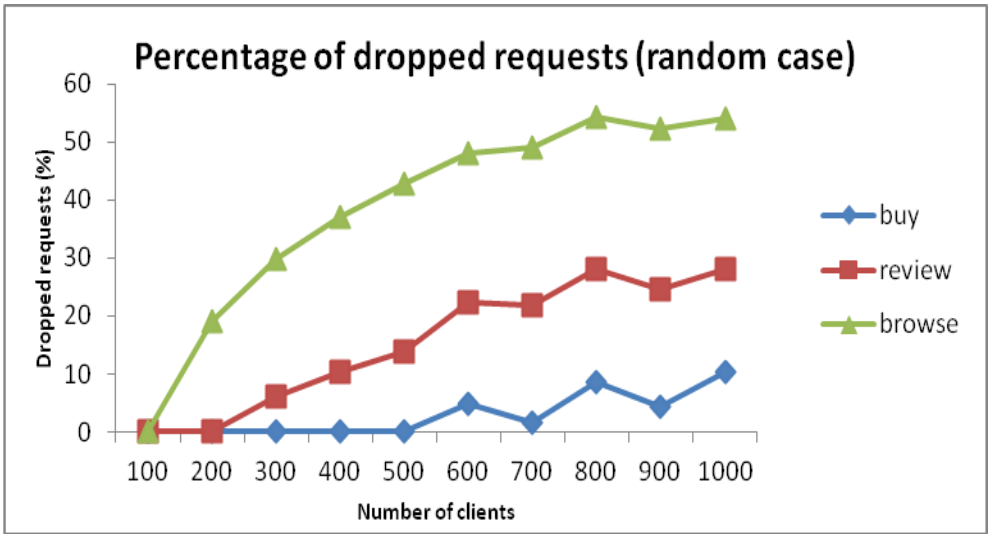

Figure 6.16: Percentage of dropped requests in Low Priority Fair Model (random case, buffer size=100)

Figure 6.16 shows that the number of dropped requests in the random case is slightly higher than in the burst case (Figure 6.17): this because buffers start to fill faster in the random case due to less thinking time and more generated requests.

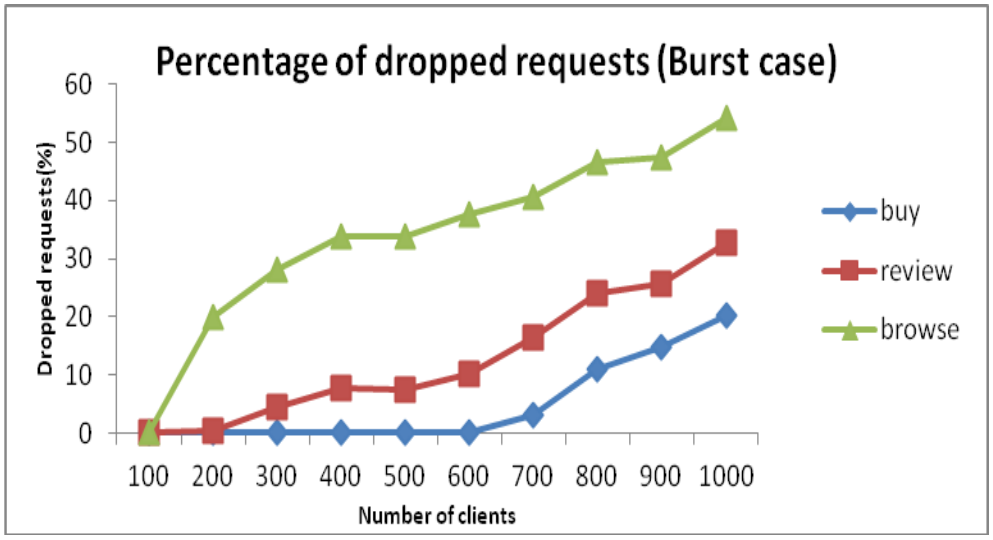

Figure 6.17: Percentage of dropped requests in Low Priority Fair Model (burst case, buffer size=100)

6.3.2.2.5. Response Time

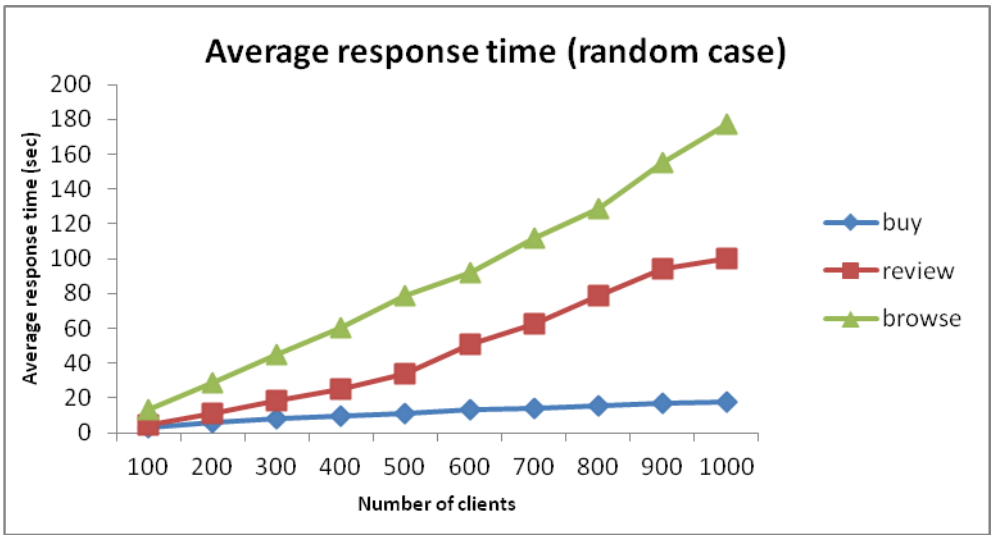

Figure 6.18: Average response time of requests in Low Priority Fair Model (random case, buffer size=100)

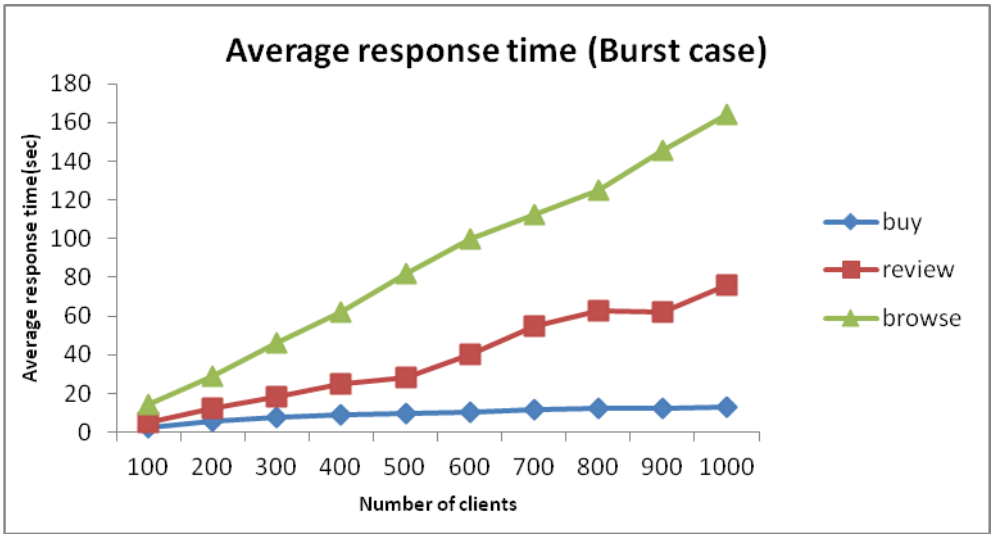

Figure 6.19: Average response time of requests in Low Priority Fair Model (burst case, buffer size=100)

It appears that the response times are low for small numbers of clients. As expected, the response time is related to the length of the buffer: note that the sharp increase in average response time that occurs when the server is fully utilized and the buffer starts to fill particularly in the beginning of the simulation. Both cases in Figure 6.18 and Figure 6.19 show an improvement for buy requests regarding their high priority when keeping the server busy during their presence in the system. At the same time there is an improvement for browse requests compared with the main priority scheduling mechanism and review model: this is clear in Figure 6.17; the percentage of dropped browse requests becomes lower. The average response time is consistent between both cases. In general, the average response time for all types grows with the increase of

the traffic load but is always less than 2 seconds for each request. Low Priority Fair Model gives better response time, as it provides lower rejected requests.

The results indicate that the two cases behave similarly when the buffer size is set to 100.

In summary, it is clear that the buffer size and the type of generated traffic effected on the performance of the server. When the buffer size is set to 1, it leads to generate more arrivals because it becomes full quickly therefore more requests are rejected. Moreover, with small size of buffer and more generated requests, there is more chance to accept more buy requests and thus will increase the completed buy requests in the system. For response time it increased with big buffer due to queuing time and busy waiting overhead without processing which given to low priority requests for more details see Algorithm 3.3 in chapter 3. In order, to reduce busy waiting time, timeout is considered for low priority requests see Appendix E for more details.

In terms of the effectiveness of different forms of traffic, it is clear that there is a difference between burst and random cases when the buffer is set to 1. In order, the generated requests are faster with the burst case and thus give higher throughput, and this appears from the throughput of the burst case which is more than in the random case (Figure 6.7).Therefore, this difference appears because the small buffer.

In addition, the conducted results show the trade off between buffer size and response time. When the buffer becomes large the request takes more time to wait and process and give large response time. Note that response time is calculated from client view.

6.3.3. Portal Model

Portal model adds new features to the basic PSM thus it's the current requirements of e-commerce applications that involve multiple websites for price comparison and more. Portals are important to capture recent developments in new e-commerce web sites.

The client configured the requests in two different TCP ports; one connects with the web server and the other connects with the portal server.

This section investigates the effectiveness of the Portal Model, two sets of experiments are designed with different based models. In the first set of experiments basic PSM is used as the basis of web server and it gives the higher priority to the customer who comes from portal web sites to complete the purchase step it called “Portal_buy”. Then the priority is given to "Direct_buy" who comes directly to the web sites to purchase, and then the lowest priority is given to "Direct_browse" who browse directly at web site.

In the second set of experiments the Fair Low Priority Model is used as the basis of web server and it gives a small delay to processing "Direct_browse"

requests rather than rejecting them. Note that the portal server serves only browse requests sent by portal clients, and it called "Portal_browse" in both sets of experiments for more details see section 3.4.3.

6.3.3.1. Portal Model Based on the basic PSM

This section presents the results of portal model that depends on the basic PSM at web server.

6.3.3.1.1. Arrival Rate

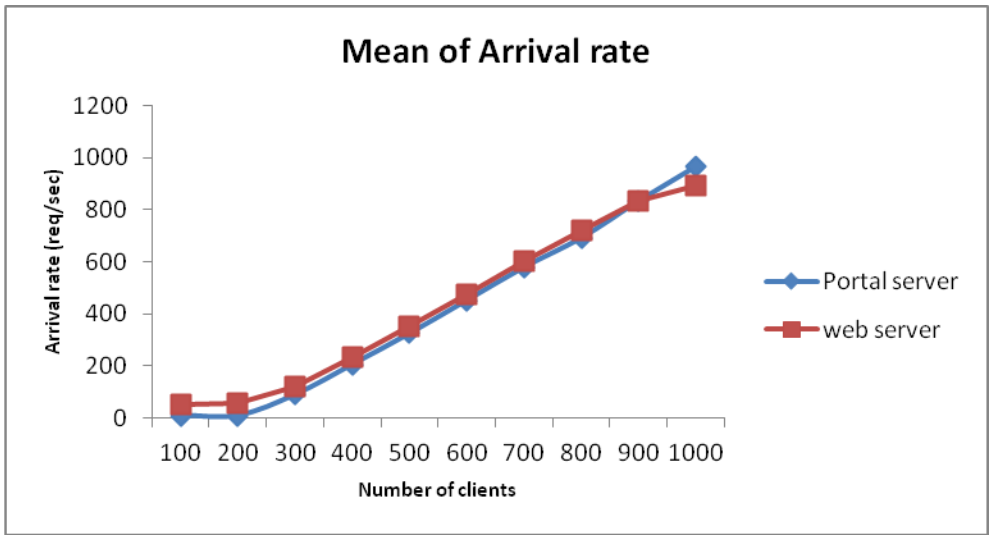

Figure 6.20: Mean of arrival rate in Portal Model based on basic PSM

Figure 6.20 illustrates the difference between the mean arrival rates in the portal and web server, the X-axis is again the number of emulated clients, but the Y-axis is arrival requests per second. As shown, the arrivals in the web server are

slightly higher than in the portal because the number and the type of requests arriving to the web server are higher than in the portal server. There are three types of request: direct browse, direct buy and portal buy. However, the small amount in browse generation caused a low arrival rate because the portal server serves only browse requests sent by portal clients.

6.3.3.1.2. Throughput

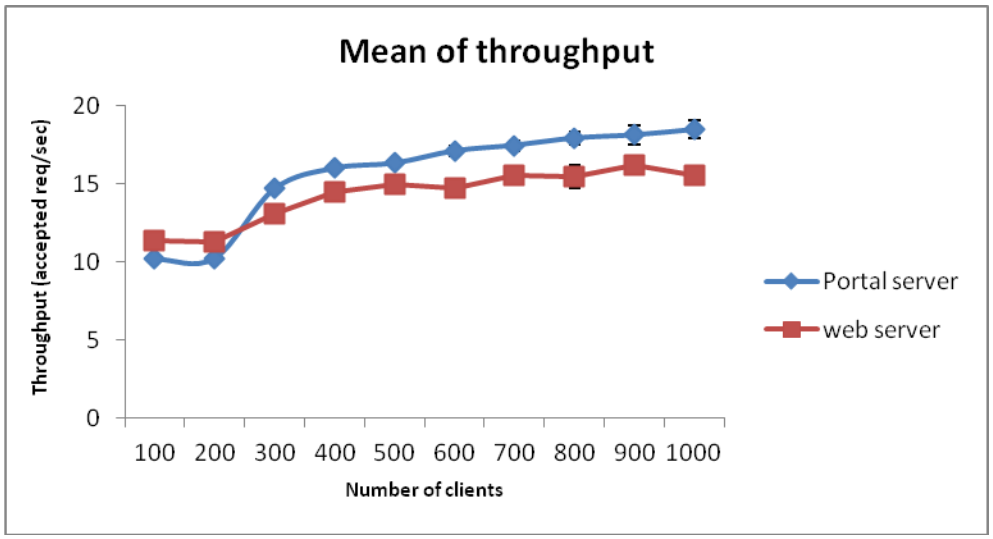

Figure 6.21: Mean of throughput in Portal Model based on basic PSM

Figure 6.21 shows the throughput mean in the portal and web server, the X-axis is again the number of emulated clients, but the Y-axis is accepted requests per second. It is clear that by increasing the arrival rate, the throughput will slightly increase. The throughput reaches a maximum value when the arrival rate ensures that the processor is fully utilized. The throughput is the number of

accepted requests per second. Accepted requests are sent from portal clients as well as direct clients to the web server.

Throughput in the portal server is the number of accepted browse requests that are sent by portal clients per second. Note that the throughput in the portal server is higher than in the web server due to a large number of accepted portal browse requests. On the other hand, with the priority scheduling mechanism at web server there are some of rejected low priority requests thus can affect the throughput results to become lower because of more rejected requests.

6.3.3.1.3. Utilisation

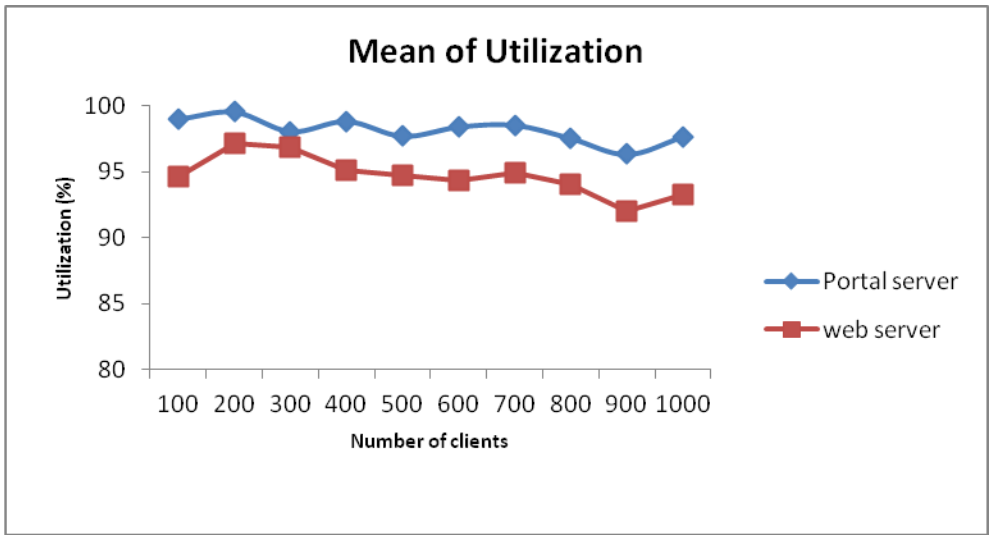

Figure 6.22: Mean of utilisation of Portal Model based on basic PSM

Figure 6.22 presents the utilisation of the two different servers as seen at portal curve it becomes fully utilized quickly regarding to accepted browse requests.

6.3.3.1.4. Dropped Requests

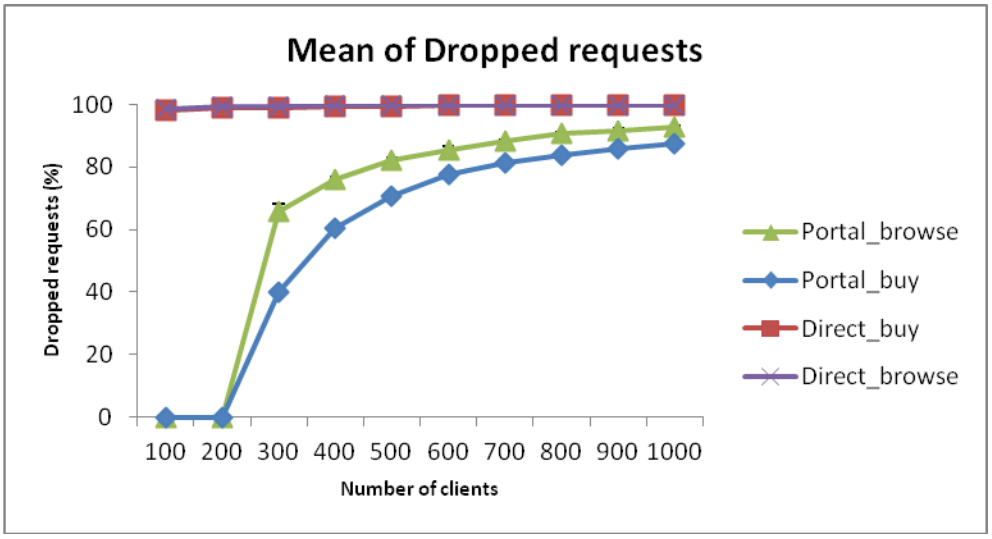

Figure 6.23: Mean of dropped requests in Portal Model based on basic PSM

As shown in Figure 6.23 most handlers process Portal_buy requests first, then Direct_buy then Direct_browse. With more requests, or with a fully-utilized server, the percentages of all types of completed requests will decrease because most of these requests will be rejected by the web server.

6.3.3.1.5. Response Time

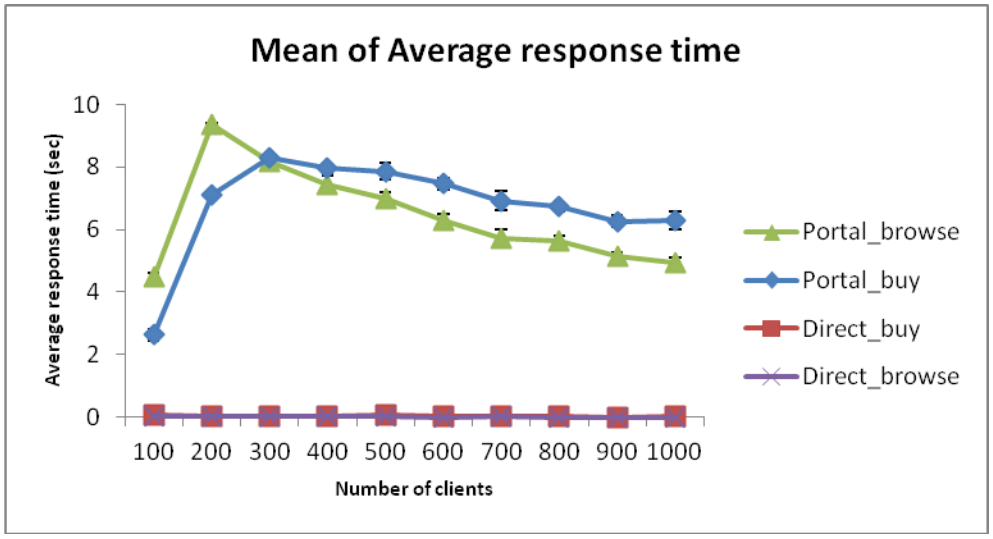

Figure 6.24: Mean of Average response time in Portal Model based on basic PSM

Figure 6.24 shows the average response time of all types of request that serve at the web server and portal server. As expected, response times are low for small numbers of clients, whereas larger numbers of clients (direct and portal) produce a heavy load on the web server, resulting in a slightly increased average response time for high-priority requests because the web server processes most Portal_ buy requests. However, due to the large percentage of low-priority requests being dropped, these will have a low average response time. Average response times (Figure 6.24) are consensus with the dropped requests (Figure 6.23) at the same point of 300 clients the dropped requests starts to increase dramatically thus cause lower response time.

Moreover, Figure 6.24 shows that a higher arrival rate for Portal_Browse produces a higher average response time because the portal server receives a large number of browse requests. As shown in Figure 6.23, with more generated requests the number of Portal_Browse requests completed will decrease because most of these requests will being rejected, thus diminishing response time because there is no time spent on rejected requests.

6.3.3.2. Portal Model Based on Fair Low Priority Model

This section presents the results of portal model that depends on the Fair Low Priority Model at web server.

6.3.3.2.1. Arrival Rate

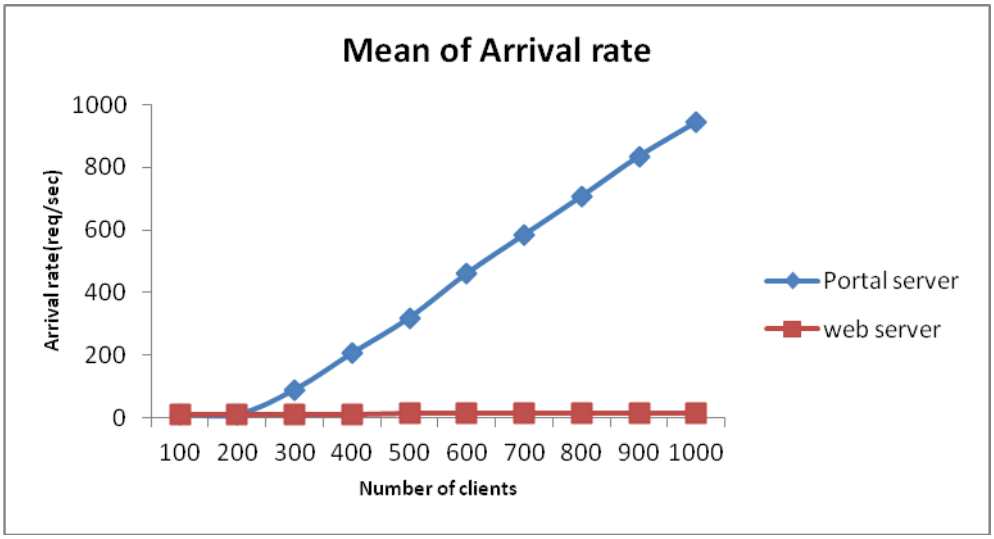

Figure 6.25: Mean of arrival rate in Portal Model based on Fair Low Priority Model

Figure 6.25 illustrates the difference between the mean arrival rates in the portal and web server, the X-axis is again the number of emulated clients, but the Y-axis is arrival requests per second. As shown, there is a significant different between arrivals in the web server and portal server. The arrivals are higher in the portal server than in the web server due to the delay in processing low priority requests thus keep the web server busy (see Figure 6.27) most of the time and the clients will not generate any more requests until they receive the response from the previous one. Note that there are three types of request: direct browse, direct buy and portal buy which served by web server. Figure 6.25 shows that the arrivals increases in portal server due to the amount of rejected requests with more clients, note that the portal server serves only browse requests sent by portal clients.

6.3.3.2.2. Throughput

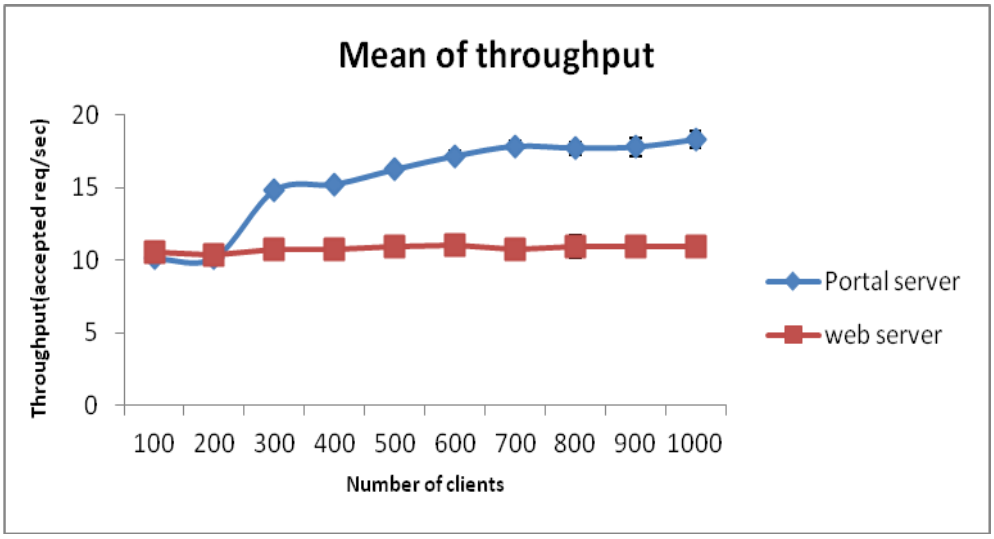

Figure 6.26: Mean of throughput in Portal Model based on Fair Low Priority Model

Figure 6.26 shows the throughput mean in the portal and web server, the X-axis is again the number of emulated clients, but the Y-axis is accepted requests per second. It is clear that by increasing the arrival rate, the throughput will slightly increase. The throughput reaches a maximum value when the arrival rate ensures that the processor is fully utilized. The throughput is the number of accepted requests per second. Accepted requests are sent from portal clients as well as direct clients to the web server.

Throughput in the portal server is the number of accepted browse requests that are sent by portal clients per second. Note that the throughput in the web server is lower than in the portal server due to a busy loop which causes a long

simulation time. In addition, there is no waiting loop at the portal server and no competition between types of requests because as, mentioned before, portal server servers only browse requests.

6.3.3.2.3. Utilisation

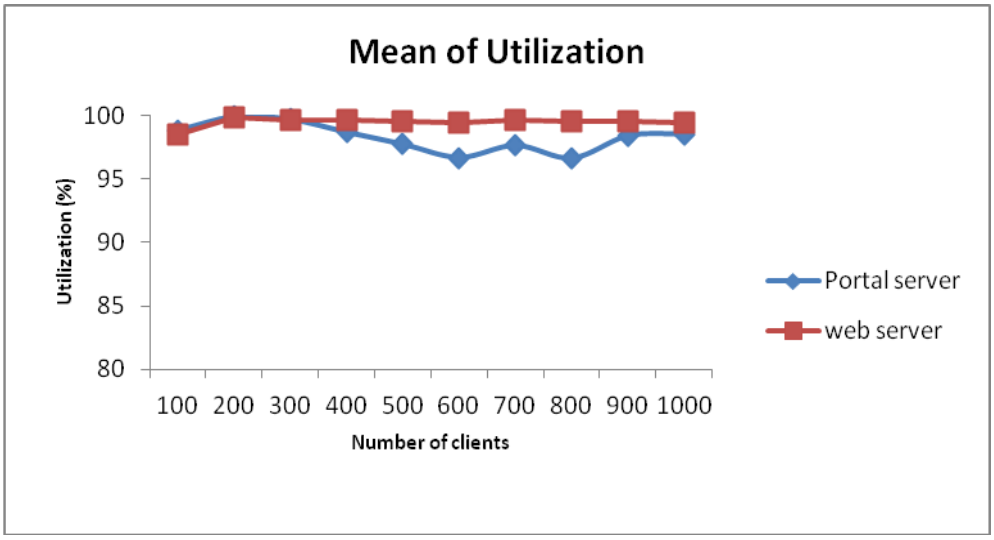

Figure 6.27: Mean of utilisation in Portal Model based on Fair Low Priority Model

Figure 6.27 presents the utilisation of the two different servers as seen at web server curve it becomes fully utilized quickly regarding to accepted low priority requests rather than reject them. The portal server curve is slightly lower than web server due to fully buffer of browse requests which rejects these requests.

6.3.3.2.4. Dropped Requests

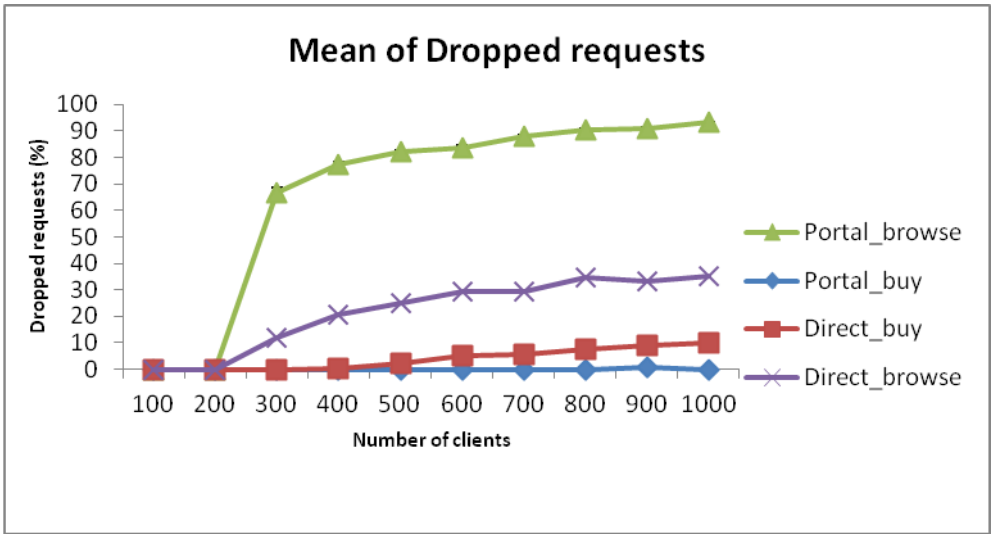

Figure 6.28: Mean of dropped requests in Portal Model based on Fair Low Priority Model

As shown in Figure 6.28 most handlers process Portal_buy requests first, then Direct_buy then Direct_browse. With more requests, or with a fully-utilized server, the percentages of all types of completed requests will decrease because most of these requests will be rejected by the web server. With more generated "Portal_browse", the portal server will reject most of them regarding to the full buffer status.

6.3.3.2.5. Response Time

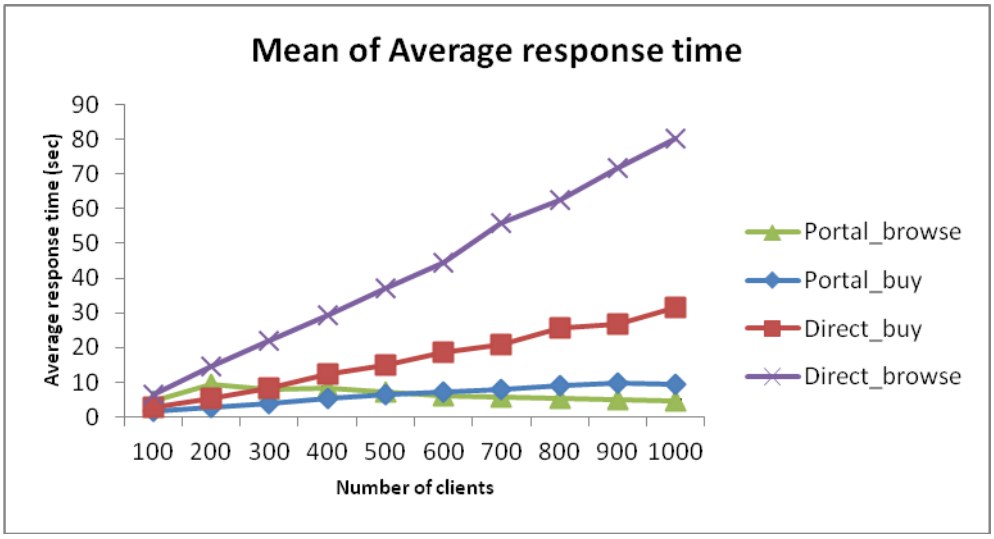

Figure 6.29: Mean of average response time in Portal Model based on Fair Low Priority Model

Figure 6.29 shows the average response time of all types of request that serve at the web server and portal server. As expected, response times are low for small numbers of clients, whereas larger numbers of clients (direct and portal) produce a heavy load on the web server, resulting in a slightly increased average response time for high-priority requests because the web server processes most "Portal_ buy" requests. However, due to the large number of the created handlers that doing busy waiting loops not processing any request, thus will have a high average response time for "Direct_browse". Average response times (Figure 6.29) are compatible with the dropped requests (Figure 6.28) at the same point of 300 clients the dropped "Portal_browse" requests

starts to increase dramatically thus cause lower response time because there is no time spent on rejected requests..

Moreover, Figure 6.29 shows that a higher arrival rate for "Portal_Browse" at 300 clients produces a lower average response time because the portal server rejects browse requests

In summary, Portal model used two different servers to serve different requests. Portal server responsible for serving browse customers who will redirect to serve by web server when he or she wants to browse more for the specific product in order to purchase it. Two sets of experiments are presented in portal model using different model basis. The portal server results are not effective with the different basis models because it considers only the web server. The results concluded that with more arrivals in the portal server, the portal server processes requests more quickly because there is one type of request (browse) with the highest priority and this means no competition. Therefore, it increased the throughput of the portal server and reduced the number of rejected browses, which is the key to keeping more customers on portal web sites.

The results indicate that the highest-priority customers take less time to process because they are prioritized, while the lowest-priority customers lose more requests during the first set of experiments.

The results concluded that the portal model with a Fair Low Priority basis gave better results with a good effect on the performance of high-priority requests.

This appears in the graph showing the percentage of dropped requests in Figure 6.28, as shown lower dropped of all requests is approved because the Fair Low Priority model avoids rejecting the low-priority customers and also gives the system more time to process the higher-priority customers. However, a higher response time is produced due to busy waiting loops or a long simulation time and thus leads to lower throughput of the system compared with the first set of experiments. In addition, the utilisation of the web server is higher with a Fair Low Priority model basis because most of the incoming requests are completed.

In addition, as shown at Figure 6.20 and Figure 6.25, different basis models at web server leads to different speeds of requests generating.

6.4. Summary

This chapter presents the experimental results based on the implementation of the proposed simulation described in Chapter 5. The performance has been assessed by looking at different combinations of conditions, such as varying the number of clients, changing the traffic flow or varying the buffer size and changing the used model basis. The results clearly demonstrate how different load settings can provide different response times and dropped requests.

The results demonstrate the effectiveness of the proposed models and show a marked improvement of the performance of high-priority requests, though low-priority requests experience reduced performance including rejected requests.

However, in the Low Priority Fair Model there is a clear improvement in the performance of low-priority requests, brought about by reducing the rejected requests without any effect on the performance of high-priority requests.

The results indicated that with Fair Low priority model basis in portal model, gave better performance than in basic PSM model.

In all situations the CPU is fully utilized. Throughput stays relatively consistent with response time, which is in turn consistent with dropped requests graphs. In addition, with more rejected requests the arrivals will increase due to zero time spent on rejected requests: therefore it will make request generating faster.

From the conducted results it appears two arguments to add extra separate hardware tier to the PSM. Firstly, because of more arrivals and more rejected requests. Secondly, because of busy waiting loops inside handlers which raise the response time without processing any requests.

Chapter 7: Conclusion

7.1. Introduction

This chapter provides the conclusion of the research work presented in this thesis. It summarises the main contributions of the proposed research.

The chapter also gives a critical evaluation of the proposed models and identifies the areas that need further improvements and defines directions for further research work.

7.2. Contributions and Critical Evaluation

Due to the rapid growth of e-commerce capabilities and applications, which are now available everywhere, e-commerce has identified certain effective requirements needed from the performance of these applications. In particular, if the web server is overloaded, poor performance can result, due to either a huge rate of requests being generated, which are beyond the server's capacity, or due to a saturation of the communication links capacity, which connects the web server to the network. Many approaches have been proposed to control server overload. The aim of this research was to investigate what was required to exploit the capabilities of e-commerce, Priority Scheduling Mechanism has been

developed to treat each request differently, and to therefore assign different priorities to different classes of e-commerce requests. In e-commerce, some requests, e.g. payments, are generally considered more important than others, i.e. search or browse, due to their revenue-raising capabilities. The aim of PSM is to increase the performance of high-priority requests at the expense of those of lower priority. The contribution is the design of PSM system based on combining more than one technique to solve overload issues on web servers. This work encompassed a mixture of several techniques, including: admission control mechanism, session-based admission control, service differentiation, request scheduling, and queuing model-based approaches.

User reviews have become an important part of e-commerce because they influence customer purchasing behaviours therefore the proposed class-based priority scheme was extended to classify e-commerce requests into high, middle and low priority requests, instead of two types of requests and included review customers. Moreover, this thesis considers the loss of low priority customers and modifies the basic PSM to improve performance of browse customers, by giving a small delay to processing their requests rather than rejecting them, thus it keeps low priority customers on the web site.

PSM is extended to have new features in portal Model that build on the synthesis of performance management mechanisms. It provides a more effective way for managing the performance of modern e-commerce services that offer a flexible but complex setup involving multiple websites.

The conclusions of this thesis can be summarised as follows:

- Several models have developed from PSM. These models were extended from the basic PSM and new parameters were added, such as a review model or modification of the basic PSM to the Low Priority Fair Model, after the discovery of incompetence with low-priority customers or to add new features such as portal models. The benefits of PSM are clear: the use of unmodified commodity software components reduces development effort tremendously. As a result, this approach does not require extensive modifications to the operating system or a complete re-write of the server.

- The formal specification of the proposed models is considered to allow alternative approaches and to refine the one chosen, as well as providing a framework for its implementation and increasing confidence that the correctly implemented model is developed from the designed scheme. π-calculus is very effective way and it is useful starting point for the implementation phase.

- In order to maintain the performance metrics i.e. arrival rate, throughput, average response time, dropped requests, and utilisation, a multi-actor simulation is implemented using the Java programming language. Java supported all the requirements which are needed to build PSM such as: threading technique, client/server application and object oriented approach. The multi-actor simulation was developed to reflect the target

models as accurately as possible; therefore, for all experiments, the proposed models were solved using multi-actor simulation.

- A number of cases have been studied using the proposed models: these models are based on Poisson arrivals, burst arrivals and exponential service time. The experiments cover a wide range of input parameterizations, e.g. number of clients and different traffic techniques, which demonstrate performance metrics. Multiple experiments can give strengths to the proposed model and help to keep the validity of the results higher.

- The results demonstrate the effectiveness of the proposed models, and show a marked improvement in the performance of high-priority requests, although low-priority requests (including rejected requests) experienced a reduced performance. However, in the Low Priority Fair Model, there is a clear improvement in the performance of low-priority requests, brought about by reducing the rejected requests without any effect on the performance of high-priority requests.

- Further types of requests can be defined simply by including extra components and modifying the Gatekeeper to sort requests appropriately, e.g. a review model.

- PSM can accommodate multiple servers, like the portal model, because the virtual buffers can be accessed by N servers as easily as by one.

- Performance depends on the behaviour of the server and the client's reaction to that behaviour, and any small changes in that behaviour ultimately lead to changes in performance. For example, given more chance to accept low priority requests can increase their response times and decrease the percentage of rejected requests. As a result, it will be small changes in server specification.

7.3. Future Work

In addition to the work reported in this thesis, several advances are suggested as recommendations for future work, as follows:

- Future work will consider the distribution of request arrivals in the proposed model using different metrics such as counter values. This will not only improve the design of our model, but will also allow more realistic testing that can be used in real e-commerce systems. Therefore, it may add to the tasks of a gatekeeper i.e. to count the number of sessions and implement different conditions depending on that number.

- Future work intends to introduce the traffic generation algorithm to study, in depth, how sensitive different simulation studies are to certain characteristics of the background traffic; in addition, to explore the application of this methodology with alternative traffic models or optimization objectives.

- Future research might also evaluate proposed scheduling protocols to see if it is possible to match protocols to traffic types.

- Further investigation of the behaviour of clients is necessary to gauge more realistic results. For example, when the clients have their requests rejected, it would be useful to add a feature to PSM that could send a message from a client when the amount of rejected requests becomes unacceptable.

- It is useful to use TPC-W, which gives one the performance characteristics of various servers in the context of a simulated e-commerce workload.

References:

[1] Menascé,D,A., Almeida, V,A, F., Fonseca, R., and Mendes, M.A. (1999) A Methodology for Workload Characterization of E-commerce Sites. ACM Conference on Electronic commerce, Denver, Colorado, USA, pp. 119-128.

[2] Almeida, J.,Dabu, M., Manikutty, A., and Cao P. (1998) Providing differentiated quality-of-service in web hosting services. ACM sigmetrics Workshop on Internet server Performance (WISP'98),Madison, WI, USA.

[3] Jakob Nielsen "Why People Shop on the Web" http://www.useit.com/alertbox/990207.html.

[4] VanderMeer,D.,Datta, A., Dutta,K., Thomas, H., and Ramamritham, K. (June 2004) Proxy- Based Acceleration of Dynamically Generated Content on the World Wide Web: An Approach and Implementation, ACM Transactions on Database system (TODS), Vol 29,No.2,pp.403-443.

[5] Elnikety, S., Nahum, E., Tracey, J., and Zwaenepoel, W. (May 17-22,2004) A Method for Transparent Admission Control and Request scheduling for in E-Commerce Web Sites", Proc. Of ACM WWW Conference, New York, USA.

[6] Jia, D., Dutkiewicz, E., and Chicharo, J, F.(2000) Performance Analysis of QoS Mechanisms in IP Networks, Computers and Communications, Proceedings. ISCC 2000. Fifth IEEE Symposium, pp.359 - 363.

[7] Tae Il, Jeong., Jae Ho, Jeong., and Sung Jo Kim.(1997) An efficient Scheduling Mechanism using multiple Multimedia Traffic in ATM Switching Nodes, Local Computer Networks, 22nd Annual Conference in USA,pp.347-356.

[8] Kamra, A., Misra, V., Nahum, E.M. (2004) Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. In: 12th IEEE international workshop on quality of service. IWQOS 2004, pp.47-56.

[9] Muppala, S., Xiaobo, Zhou. (2011) Coordinated session-based admission control with statistical learning for multi-tier internet applications, Journal of Network and Computer Applications, volume 34, pp. 20–29.

[10] Gupta, V., Burroughs, M., Harchol-Balter, M. (2010) Analysis of scheduling policies under correlated job sizes, Performance Evaluation, Elsevier, volume 67, pp. 996–1013.

[11] Crovella, M., Frangioso, R., Harchol-Balter, M.(11-14 October 1999) Connection scheduling in web servers, ACM, Second symposium of Internet Technologies and System (USITS'99),Boulder, CO,USA, volume 2.

[12] Rawat, M., Kshemkayani, A. (16-18 April 2003) SWIFT: Scheduling in web servers for fast response time, Second IEEE International

Symposium on Network Computing and Applications (NCA 2003),Cambridge, MA, USA,pp.51-58.

[13] Harchol-Balter, M., Schroeder, B., Bansal, N. and Agrawal, M.(2003) Size-based scheduling to improve web performance, ACM Transactions on Computer systems (TOCS,2003),21(2):207-233.

[14] Chuan, Yue., Haining, Wang.(2009) Profit-aware overload protection in E-commerce Web sites, Journal of Network and Computer Applications, 32, pp. 347– 356.

[15] Alonso, J., Guitart, J., and Torres, J. (2007) Differentiated Quality of service for e-Commerce Applications through Connection Scheduling based on System-Level Thread Priorities, Parallel, Distributed and Network-Based Processing 15th EUROMICRO International Conference, 7-9 Feb. 2007, pp.72 – 76.

[16] Schroeder, B., and Harchol-Balter, M. (2006) Web servers under overload: How scheduling can help, ACM Transactions on Internet Technology, 6(1):20-52.

[17] Kaszo, M., Legany, C.(2007) Analyzing Customer Behavior Model Graph (CBMG) using Markov Chains, The 11th International Conference on Intelligent Engineering Systems (INES 2007), 29 June – 1 July, 2007, Budapest, Hungary.

[18] Elnikety, S., Nahum, E., Tracey, J., and Zwaenepoel, W.(2004) A Method for Transparent Admission Control and Request scheduling for in E-

Commerce Web Sites, Proc. Of ACM WWW Conference, May 17-22, 2004, New York, USA.

[19] Miller, D. (1981) Computation of steady-state probabilities for m/m/1 priority queues, Operations Research, vol. 29, no. 5, pp. 945–958.

[20] Bhatti, N., and Friedrich, R., (1999) Web server support for tiered services, IEEE Network, 1999, 13(5):64-71.

[21] Voigt, T., Tewari, R., Freimuth, D., and Mehra, A. (2001) Kernel mechanisms for service differentiation in overload web servers, USENIX Annual Technical Conference, Boston, MA, USA, 2001, p 189-202.

[22] Li, K., and Jamin, S. (2000) A measurement-based admission-controlled web server, 19th IEEE INFOCOM, 2000,p 651-659.

[23] Verma, A., and Ghosal, S. (2003) On admission control for profit maximization of networked service providers, 12th International World Wide Web Conference (WWW'03), Budapest, Hungary, 2003, pp 128-137.

[24] Carlstrom, J., and Rom, R. (2002) Application-aware admission control and scheduling in web servers, twenty-first IEEE INFOCOM, 2002, New York, USA, p 506-515.

[25] Chen, H., and Mohapatra, P. (2003) Overload control in QoS-aware web servers, Computer Networks, 2003, 42(1):119-133.

[26] Urgaonkar, B., and Shenoy, P.(2008) Cataclysm: Scalable overload policing for internet applications, Journal of Network and Computer Applications (JNCA), 2008, 31:891-920.

[27] Abdelzaher, T., Shin, K., and Bhatti, N. (2002) Performance guarantees for web server end-systems: A control-theoretical approach, IEEE Transactions on Parallel and distributed Systems, 2002, 13(1):80-96.

[28] Abdelzaher, T., and Bhatti, N., (1999) Web content adaptation to improve server overload behavior, Computer Networks, 1999, 31(11-16):1563-1577.

[29] Andrzejak, A., Arlitt, M., and Rolia, J.(2002) Bounding the resource savings of utility computing models, Technical Report HPL-2002-339,HP Labs.

[30] Chandra, A., and Shenoy, P.(2003) Effectiveness of dynamic resource allocation for handling internet flash crowds, Technical report TR03-37, Department of computer science, University of Massachusetts, USA.

[31] Almeida, J., Almeida, V., Ardagna, D., Francalanci, C., and Trubian, M. (2006) Resource management in the autonomic service-oriented architecture. Third International Conference on Autonomic Computing (ICAC'06), Dublin, Ireland, 2006, pp 4–92.

[32] Liu, Z., Squillante, M., and Wolf, J.(2001) On maximizing service-level-agreement profits, Third ACM Conference on Electronic Commerce (EC 2001), Tampa, FL, USA, 2001,pp, 213–223.

[33] Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., and Wood, T. (2008) Agile dynamic provisioning of multi-tier internet applications, ACM Transactions on Adaptive and Autonomous Systems (TAAS) 2008; 3(1):1–39.

[34] Milner, R. (1999) Communicating and Mobile systems: the π -Calculus. Cambridge University Press.

[35] Henk W. M. Gazendam. (2005) Coordination Mechanisms in Multi-Actor Systems, Planning in Intelligent Systems, Wiley Online Library.

[36] Younas, M., Awan, I., Chao, K.-M., Chung, J.-Y.(2008) Priority scheduling service for e-commerce web servers, Journal of Information Systems and E-Business Management 6(1), 69-82, 2008.

[37] Wischik, D. (2006) Buffer sizing theory for bursty TCP flows, Proceedings of the 2006 International Zurich Seminar on Communications, February 21-24,2006. pp. 98-101.

[38] Rodrigues, N.F., and Barbosa, L.S.(2005) Architectural Prototyping: From CCS to. Net. Electronic Notes in Theoretical Computer Science, 2005, 130:151–167.

[39] Dennis, A., Wixom, B., Tegarden, D., Ghattas, R., and McKee, S. (2005) System Analysis and Design with UML Version 2.0. Prentice Hall, 2005.

[40] Stevens, P., and Pooley, R. (2006) Using UML software engineering with objects and components, Second edition. Harlow, Eng. ; New York : Addison-Wesley.

[41] Holton, D.R.W. (2008), Implementing π -calculus style actions. Technical report.

[42] Trent, G. & Sake, M. WebSTONE: The first Generation in HTTP Server Benchmarking,(1995).Available from:

http://www.sgi.com/products/?/WebFORCE/WebStone. [Accessed 15th June 2011]

[43] TPC-W, Transactional Web e-Commerce Benchmark, (2005), Available from: http://www.tpc.org/. [Accessed 10th June 2011]

[44] Wang, H., and Dittmann, L., (2010) Downlink resource management for QoS scheduling in IEEE 802.16 WiMAX networks, Computer Communications 33 940–953, 2010.

[45] IEEE 802.16, (2011), Available from: http://en.wikipedia.org/wiki/IEEE_802.16. [Accessed 10th August 2011]

[46] Almeida, J., Almeida, V., Ardagna, D., Cunha, I., Francalanci, C., and Trubian, M., (2010) Joint admission control and resource allocation in virtualized servers, Journal of Parallel and Distributed Computing, 2010, 70, 344_362.

[47] Bertini, L., Leite, J., Mossé, D.,(2010) Power and performance control of soft real-time web server clusters, Information Processing Letters, 2010, 110: 767–773.

[48] Totok, A., and Karamcheti, V. (2010) RDRP: Reward-Driven Request Prioritization for e-Commerce web sites, Electronic Commerce Research and Applications, 2010, 9: 549–561.

[49] Boone, B., Hoecke, S., Seghbroeck, G., Joncheere, N., Jonckers, V., Turck, F., Develder, C., and Dhoedt, B. (2010) SALSA: QoS-aware load balancing for autonomous service brokering, The Journal of Systems and Software, 2010, 83: 446–456.

[50] Guitart, J., Carrera, D., Beltran, V., Torres, J., and Ayguadé, E.(2008) Dynamic CPU provisioning for self-managed secure web applications in SMP hosting platforms, Computer Networks, 2008, 52(7):1390–1409.

[51] Huang, C.J., Chuang, Y. T., Cheng, C. L.,(2005) An admission control scheme for proportional differentiated services enabled internet servers using support vector regression, International Journal of Simulation Systems, Science & Technology Special Issue on: Soft Computing for Modeling and Simulation, 2005, Volume 6, pp,10-11.

[52] Kasigwa, J., Baryamureeba, V., and Williams, D.(2005) Dynamic Admission Control for Quality of Service in IP Networks, World Academy of Science, Engineering and Technology 8, 2005.

[53] A J, Wellings. (2004) Concurrent and Real-Time Programming in Java. Wiley, University of York.

[54] Menascé, D, A., Almeida, V, A, F., (2000) Scaling for E-Business Technologies, Models, Performance, and Capacity Planning. Upper Saddle River, New Jersey. Prentice-Hall Inc.

[55] Pi_calculus,(2012), Available from: http://en.wikipedia.org/wiki/Pi_calculus.[Accessed 10th June 2011]

[56] Kant, K., Tewari, V., and Lyer, R., (2001) Geist: A generator for E-commerce & Internet server Traffic, Performance Analysis of Systems and Software, 2001. ISPASS. 2001 IEEE International Symposium, p 49 – 56.

[57] Park, K., and Willinger, W. (2000) "Self-Similar Network Traffic: An Overview", in Self-Similar Network Traffic and Performance Evaluation K. Park and W. Willinger, Eds., ed New York; Chichester John Wiley & Sons, Inc., 2000, pp. iii-xlix.

[58] Trivedi, K. S., (2002) Some Important Distributions, in Probability and Statistics with Reliability, Queuing and Computer Science Applications 2ed: JOHN WILEY & SONS, INC., 2002, pp. 144-146.

[59] Horváth, G., and Telek, M., (2003), An approximate analysis of two class wfq systems, PMCCS-6, September 5-7, 2003, pp. 43–45.

[60] Domanski, B.,(1999) Simulation versus Analytic Modeling in Large Computing Environments, Predicting the Performance Impact of Tuning Changes,1999.

[61] Floyd, S., and Paxson, V., (2001) Difficulties in simulating the internet, IEEE/ACM Trans. Netw., 2001, vol. 9, pp. 392-403.

[62] Graba, J.,(2003). An Introduction to Network Programming with Java. Sheffield Hallam University, UK, 2003.

[63] IBM, developer Works, Available from: http://www.ibm.com/developerworks/websphere/newto/. [Accessed 10th January 2012]

[64] Web wiz, What are Active Server Pages (Classic ASP), Available from: http://www.webwiz.co.uk/kb/asp-tutorials/what-is-asp.htm. [Accessed 10th January 2012].

[65] Oracle, Java Server Pages Technology, Available from: www.oracle.com/technetwork/java/javaee/jsp/. [Accessed 10th January 2012].

[66] The Perl Programming Language, (2002), Available from: www.perl.org/. [Accessed 10th January 2012].

[67] Townsend, M. (2011), Target Works to Fix Crashing Website Before Black Friday: Retail, Bloomberg Businessweek, November 10, 2011, Available from: http://www.businessweek.com/news/2011-11-10/target-works-to-fix-crashing-website-before-black-friday-retail.html. [Accessed 20th December 2011].

[68] Blodget, H.(2011), Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data, Business Insider, April 28,2011, Available from: http://articles.businessinsider.com/2011-04-28/tech/29958976_1_amazon-customer-customers-data-data-loss. [Accessed 20th December 2011].

[69] Milner,R.(1980) A Calculus of Communicating Systems, Springer Verlag, ISBN 0-387-10235-3.

[70] QNAP Systems, Available from: http://www.qnap.com. [Accessed 20th December 2011].

[71] Robert Chartier (2010). N-Tier Application Architecture [Online image][Figure].Available at: http://www.webopedia.com/quick_ref/app.arch.asp. [Accessed: 20 April 2010].

[72] About Think Times, Available from: http://msdn.microsoft.com/en-us/library/ms184790%28v=vs.80%29.aspx. [Accessed 20th June 2010].

[73] 3GPP, TSG Services and System Aspects, QoS Concept, 3G TS23,107, Version1.3.0,1999.

Appendix A: Source code

In this Appendix, some of the source codes from the simulation is detailed.

1- Exponential distribution Function

```
import java.util.Random;

import java.lang.Object;

import java.io.Serializable;

import java.io.*;

public class exponential implements Serializable

{

      public synchronized long Exponential(long mean)

   {

       Random random = new Random();

       long E;

       double f;

       f = -Math.log(Math.random());

       E=(long)(f*mean);
```

```
        return E;

    }//end Exponential

}//end class
```

Figure 30: Code of Exponential distribution method

2- Client in portal model

```
while (x < MAX_THREADS) {

            if(x%2==0)

          {

            try{

                link = new Socket(addr, 1111); //1- connected to portal(browse requests)

                outToServer11 = new ObjectOutputStream(link.getOutputStream()); // out 2 portal

                inFromServer11 =  new ObjectInputStream(link.getInputStream()); // in from portal

               try{

                  System.out.println("client Bind to portal");

                  Thread t1=new Portalclient(link, inFromServer11, outToServer11,

                              Portalclient.threadCount(),tp);
```

```
            }catch (IOException e){System.out.println("\nError....! " +e);}

      }//end try

        catch (IOException e){System.out.println("\nUnable to set up port! " +e);}

}//end if

      else

      {

       try{

           socket1 = new Socket(addr, 8881); //1- connected web server

            outToServer2 = new ObjectOutputStream(socket1.getOutputStream());//out 2 webserver

           inFromServer2 =  new ObjectInputStream(socket1.getInputStream());//in from webserver

         try{

                  System.out.println("client Bind to server1");

                Thread t3=new Directclient(socket1, inFromServer2, outToServer2,

                          Directclient.threadCount2());

        }catch (IOException e){System.out.println("\nError....! " +e);}

      }//end try

        catch (IOException e){System.out.println("\nUnable to set up port to server1! " +e);}

       }//end else

 x++;
```

```
} //end while
```

Figure 31: Code of client in portal model

as seen in Figure 31, there are two types of clients :Portal clients who connect with portal server and direct clients who connect with web server.

3- Buffer class in portal model

```
import java.io.Serializable;

import java.util.Vector;

public class Buffer implements Serializable {

private int i=0,in=0,out=0,max=100,totalaccept=0,sum=0;

private int[] size1=new int[max];

private long time=0,S_time=0;

/*************************************************************************/

 public synchronized  void add_sizep(int g)

  {

    while(i==max)

    {      try{

           wait();}catch(InterruptedException e){System.out.println("add size1 report  "+e ); }
```

```
    }//end while
     in=(in+1)%max;
     ++i;
     size1[in]=g;
     System.out.println("add size1="+size1[in]);
     notifyAll();
  }//end add to buffer
  public synchronized  void remove_sizep()
  {
     while(i==0) {
           try{
           wait();}catch(InterruptedException e){System.out.println("remove size1 report  "+e );}
             } //end while
           out=(out+1)%max;
           --i;
     notifyAll();
     System.out.println("remove size1 report  "+size1[out]);
       }//end remove from buffer
////////////////////////////////////////////////////////////////////////////////////////
```

```
  public synchronized void record_time(long t)
  {
     time=time+t;
    System.out.println("time1=  "+time);
  }//end record time of processing
////////////////////////////////////////////////////////////////////////////////
   public synchronized void reset_time()
  {
      time=0;
   }//end reset time
////////////////////////////////////////////////////////////////////////////////
  public synchronized void record_simulationtime(long t)
  {
      S_time=t;
   }//end record simulation time
////////////////////////////////////////////////////////////////////////////////
   public synchronized long print_simulationtime()
  {
     return S_time;
```

```
  }//end print simulation time

////////////////////////////////////////////////////////////////////////////////

  public synchronized void reset_simulationtime()

  {

     S_time=0;totalaccept=0;sum=0;

  }//end reset simulation time

////////////////////////////////////////////////////////////////////////////////

  public synchronized long print_time()

  {

     return time;

   }//end print processing time

////////////////////////////////////////////////////////////////////////////////

  public synchronized boolean empty_b ()

  {

     if(i==0)

     return true;

     else

          return false;

   }//end checking buffer if empty
```

```
//////////////////////////////////////////////////////

  public synchronized boolean full_b ()

  {

    if(i==max)

     return true;

     else

          return false;

    }//end checking buffer if full

//////////////////////////////////////////////////////////////

} //end class Report
```

Figure 32: Code of Report class in portal model

Figure 32 presents the share Buffer class which is used to identify the number of active requests of each type and used synchronised methods for mutual exclusive purposes as described in chapter 5.

4- Handler Body in server side

```
if ( flag == 11 ) // portal buy

                    {

                         sum1=sum1+1;//number of incoming requests
```

```
if(!(server2.report.full_b1()))  //check buffer size
 {
        try{
            try{
                server2.report.add_size1(sum1);//add to size
                type ddp=new type(11,"buyacceptP",server2.x,0);//create object
server2.T1=server2.thread.Sleep(server2.E.Exponential(100));//time spending in CPU
                server2.report.record_time1(server2.T1);//record processing time
                out.writeObject(ddp);//send 'accept' object to client
                 }catch(Exception ioEx1){System.out.println("add size1  "+ioEx1
                                                                        );}
        try{
                server2.report.remove_size1();//remove
         }catch(Exception ioEx1){System.out.println("remove size1  "+ioEx1 );}
                totalaccept=totalaccept+1;//total of accepted
                out.flush();
        }catch(Exception ioEx1){System.out.println("XXXXXX   Exception in
                    accept portal payment writing  "+ioEx1 );}
}//end if check size
```

```
else    //if the buffer full

 {

      try{

          d1++;

          type dddp=new type(11,"buyrejectP",server2.x,0);

          out.writeObject(dddp);//send 'reject'

          server2.report.record_time1(0);

           out.flush();

      }//end try

      catch(Exception ioEx1){System.out.println("XXXXXX   Exception in drop

                              portal payment writing  "+ioEx1);}

 }//end else

}//end portal
```

Figure 33: Code of Handler in server

Figure 33 presents the implementation of Handler which is created for each client to classification and processing requests. Note that the appropriate buffer is checked when receive the request using Buffer class.

Appendix B: Work flow

This appendix describes the main phases in research project.

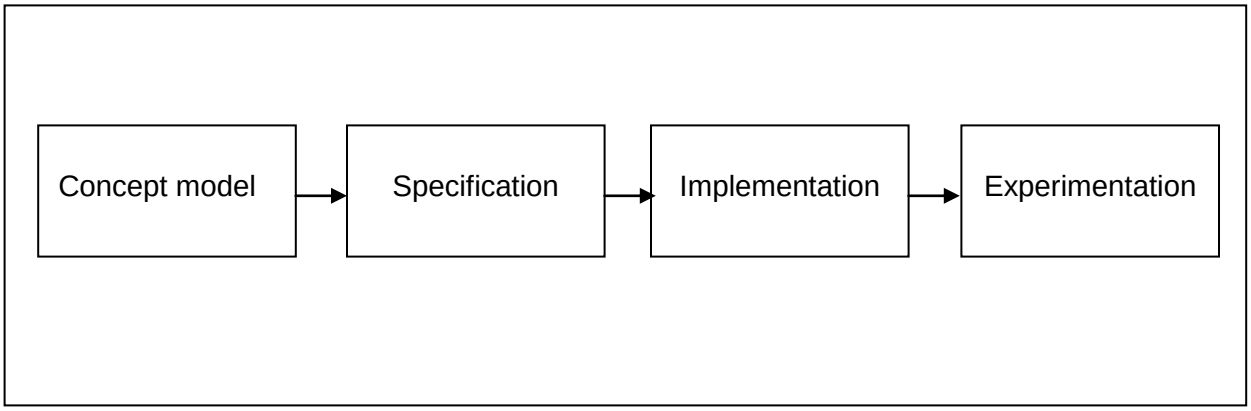

Figure 34: work flow phases

As seen in Figure 34, Determine the concepts of the proposed model is the first phase in this research which is mentioned in chapter 3 with details. Then the proposed model is formally specified using the π-calculus in early stage of models design which is presented in details in chapter 4. Java programming language was chosen for implementation more details are provided in chapter 5. Finally, the experiments cover a wide range of input parameterizations to demonstrate performance metrics in chapter 6.

Appendix C: Arrival rate in random and burst cases with different buffer's size

This appendix shows the relation between two different types of traffics: random and burst when the buffer size increased. This tradeoff is worth considering to adjust the best size of the buffer.

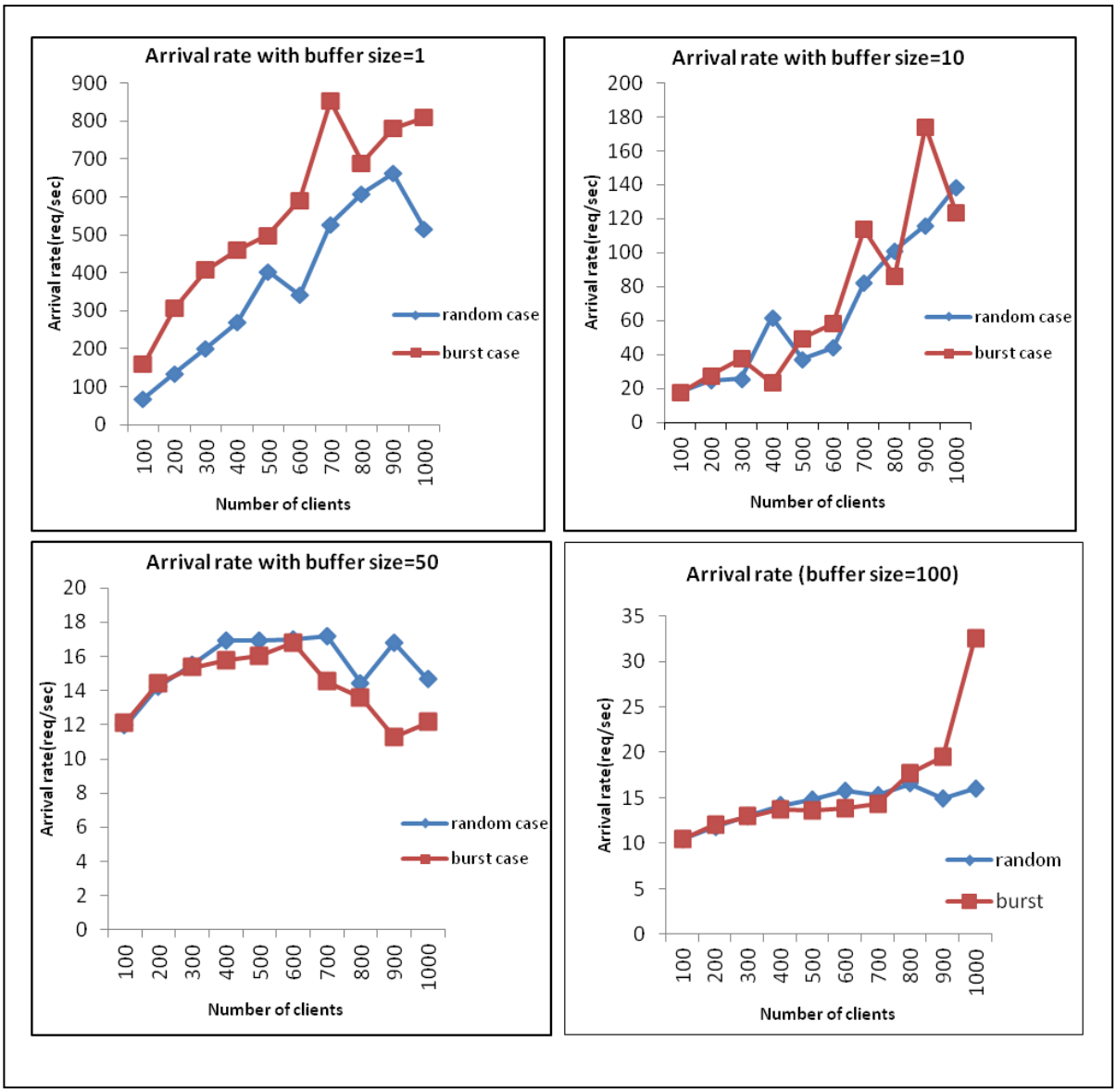

Figure 35: Arrival rate in random and burst cases

Figure 35 presents the obtained arrival rate in two cases – random and burst – when the buffer size is set to 1, 10, 50 and 100. It shows that when the size of the buffer increases, the arrival rate of random and burst traffic becomes similar. Note that when the buffer's size increases, the burst traffic will absorb by the large buffer therefore, it is sufficient to get good performance.

In addition, figure 35 shows that the generated requests becomes faster with small size of buffer due to rejected requests.

Appendix D: Adjustments buffer size and arrival rate

This appendix presents the multiple adjustments to set the best size for the buffer. One of the performance metrics is chosen to see the effect of different buffer sizes on obtained results.

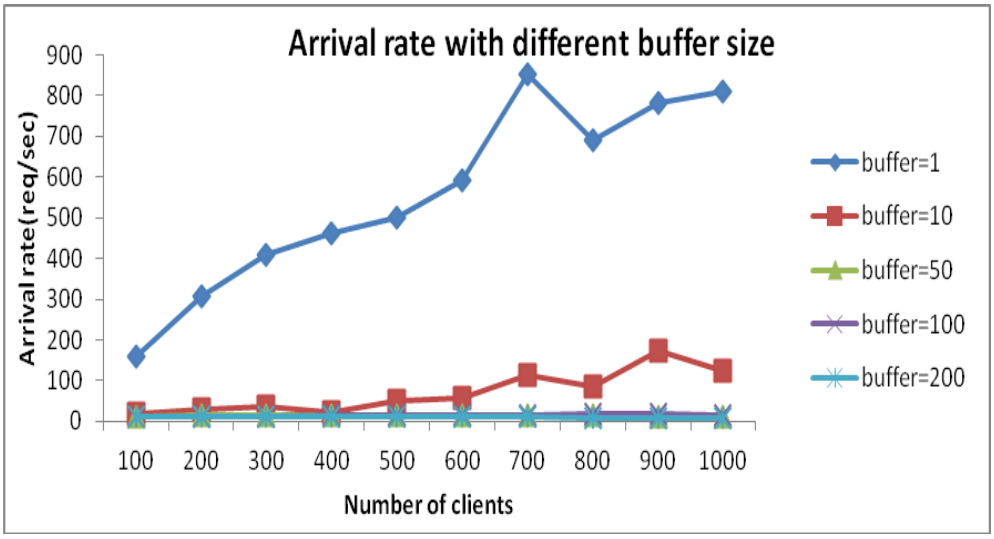

Figure 36: Buffer's size and arrival rate

The buffer's size is set to 1, 10, 50,100 and 200 and at each point the arrival rate is calculated. Figure 36 shows that when the buffer's size is set to 1, it gives a high arrival rate due to the high percentage of dropped requests. Then, when the size becomes 10, it also gives a high arrival rate and this size leads to high dropped requests as well.

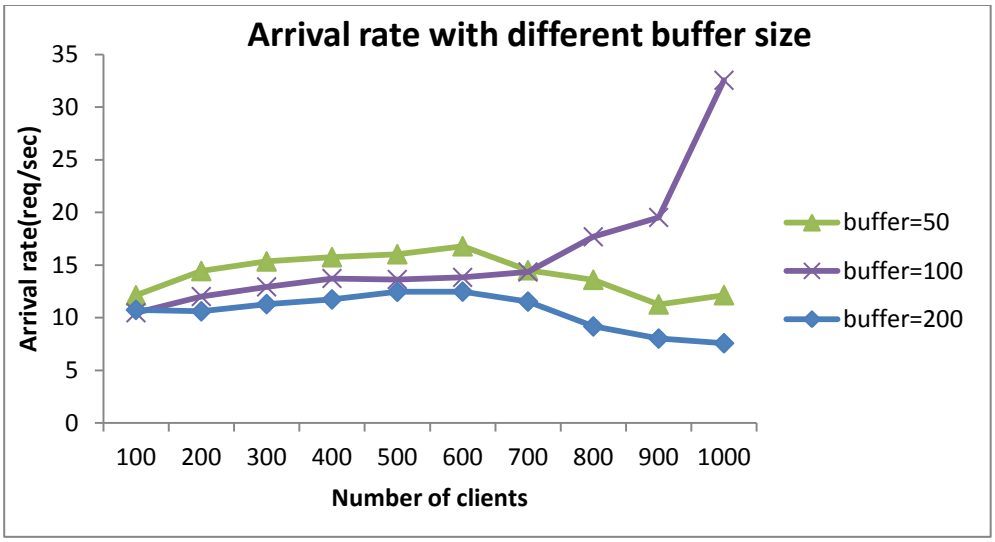

Figure 37: Buffer's size=50,100 &200

Figure 36 does not show clearly the arrival rate when the buffer's size is set to 50,100 and 200 due to the small arrival rate compared with the previous sizes of 1 and 10. In order to show the different arrival rates in cases of 50,100 and 200, Figure 37 is presented.

As shown when the buffer's size sets to 200, the arrival rate decreases due to long round trip time for request thus will delay to generate the next one. In addition, the response times for lower priority requests increase due to long waiting time that spend to process all high priority requests in their buffer.

Figure 37 shows that the best buffer size is 100 because it gives an acceptable arrival rate and lower rejected requests. Therefore, it gives better results because there are enough places to accept incoming requests and this brings balance to the obtained results during the simulation.

Appendix E: Timeout for low priority requests

In this Appendix, timeout for low priority requests is considered to reduce the waiting overhead which is produced in low priority fair model. Timeout value for maximum waiting is set to be 5 s. If this timeout expires, the request is processed.

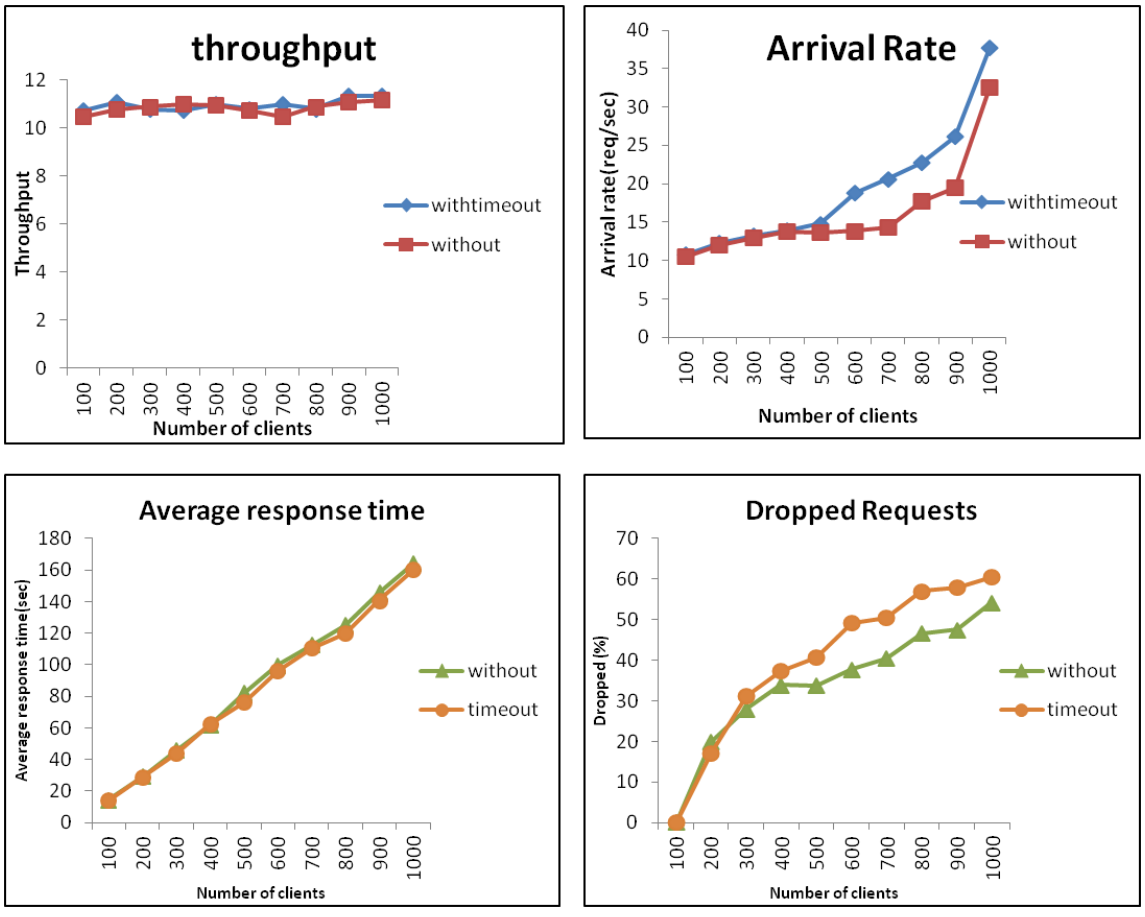

Figure 38: Timeout Performance

As seen in Figure 38, with timeout the arrival rate is higher because the time of waiting loop decreased and this effect on the requests generation speed.

In addition, for the throughput is similar in the two cases. Figure 38 shows that with timeout for browse requests, the response time is lower. However, the rejected requests is higher consistent with the response time.

In summary with timeout it gives better results because it reduces the obtained response time and generate more requests. Note that the timeout value effects on the response time.