A Comparative Evaluation of Web Server Systems: Taxonomy and Performance

by

Manikandaprabhu Ganeshan

A thesis
presented to the University of Manitoba
in partial fulfilment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba, Canada, 2005

Signature

Abstract

The Internet is an essential resource to an ever-increasing number of businesses and home users. Internet access is increasing dramatically and hence, the need for efficient and effective Web server systems is on the rise. These systems are information engines that are accessed through the Internet by a rapidly growing client base. These systems are expected to provide good performance and high availability to the end user. They are also resilient to failures at both the hardware and software levels. These characteristics make them suitable for servicing the present and future information demands of the end consumer.

In recent years, researchers have concentrated on taxonomies of scalable Web server system architectures, and routing and dispatching algorithms for request distribution. However, they have not focused on the classification of commercial products and prototypes, which would be of use to business professionals and software architects. Such a classification would help in selecting appropriate products from the market, based on product characteristics, and designing new products with different combinations of server architectures and dispatching algorithms.

Currently, dispatching algorithms are classified as content-blind, content-aware, and Domain Name Server (DNS) scheduling. These classifications are extended, and organized under one tree structure in this thesis. With the help of this extension, this thesis develops a unified product-based taxonomy that identifies product capabilities by relating them to a classification of scalable Web server systems and to the extended taxonomy of dispatching algorithms. As part of a detailed analysis of Web server systems, generic queuing models, which consist of a dispatcher unit and a Web server unit are built. Some performance metrics, such as throughput, server performance, mean queue size, mean waiting time, mean service time and mean response time of these generic queuing models are measured for evaluation. Finally, the correctness of generic queuing models are evaluated with the help of theoretical and simulation analysis.

Acknowledgements

First, I need to acknowledge the Department of Computer Science, University of Manitoba for giving me the opportunity to pursue my master's degree.

I would like to express my sincere gratitude and appreciation to my supervisors, Dr. Randal Peters and Dr. Rasit Eskicioglu for their personal guidance, encouragement, support, and motivation all through this research. It was a great pleasure to conduct this thesis under their supervision. I want to thank them for all their help, interest, and valuable hints.

I am deeply grateful to professors, Dr. Attahiru S. Alfa and Dr. Neil Arnason, who have guided me in the right way of my ”queueing theory” research. Their ideals, concepts, detailed and constructive suggestions have had a remarkable influence on this thesis and my entire study in the field of queuing theory. I would also like to thank Dr. Robert D. McLoed, who monitored my work and took effort in reading and providing me with valuable comments on this thesis.

I owe my warm and sincere thanks to my friends, Hossein Pourreza and Ye Li, who have been of great support throughout my research.

I am deeply indebted to all my friends in Winnipeg, who are of great value to me, for their moral support and timely help. Without them, I could not have completed this thesis successfully.

I dedicate this thesis to my beloved parents, whom I consider the most important people in my life. Needless to say that I am grateful to my sisters and in-laws, whose inspirations make me work hard all the time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Internet is a popular medium for communication, commerce, knowledge dissemination, and information exchange. Much of the functionality provided through the Internet is generally serviced by a variety of *Web server systems*, which are physically characterized as a cluster of quasi-independent servers connected by a local area network. The intrinsic features of these systems include: high performance, high availability, high scalability, and resiliency to failures. The servers in a Web server system collectively service requests and handle failures. Such configurations typically incorporate significant amounts of redundancy and are easily expanded by adding more identically configured components. This makes them suitable for handling large amounts of Web traffic. One important challenge in designing these Web server systems is to effectively balance the load across servers and their storage sub-systems.

The demand for Internet access is increasing dramatically and, hence, the need for efficient Web server systems is on the rise. Interest in Web server systems has received greater attention in recent years. A larger number of commercial products [Aka99, Nor00, Nor02, F5N00, Res00, Cis02, Fou02, Zeu02, Cis00, Mic00] are available in the market and are in use today. Furthermore, several research groups are actively pursuing and developing Web server system prototypes [LVS98, APE96, KBM94, AYHI96, DCH$^+$97, ZBCS99], each with varying goals in mind.

A number of factors contribute to the rising demand for the Internet. One is the proliferation of high speed Internet access to the end consumer through better availability, and affordability of technology such as Cable and Digital Subscriber Line (DSL) [DSL04] modems. This demand is further augmented by an increased use of Cable/DSL routers by home and small business users, who can easily share an Internet connection between multiple computers within their respective establishments.

Another factor is the growing user demand for enriched content that includes high-definition images (jpg [Joi04]), animation (flash, shock wave, QuickTime VR [Mac04, App04]), audio (mp3), and video (mpg, avi, QuickTime) [Mov04, AVI04]. Web server systems can improve the delivery of these media types by exploiting content-aware methods, and the ability to evaluate and compare various mechanisms is an important challenge. Some additional factors are the increased need for encryption, authentication, and general security in the support of e-commerce transactions with respect to application service providers from the banking, shopping, business purchasing, and delivery sectors.

The flexibility of multi-tier Web server systems can make them susceptible to increased traffic by the nature of their designs. For example, the use of XML [Nor98], and XSLT [Ken00] make it fairly easy to open up Web server systems to a large number of new devices. This means that multi-tier Web server systems utilize common data and business layers to retrieve back-end data in XML format and apply business rules in a consistent fashion. The resulting XML can then be transformed using an appropriate XSL [Ken04] style sheet to produce the appropriate HTML [Eri95] for a requesting device. By detecting the browser string of different devices and browsers, an appropriate selection of XSL style sheets can be made. For example, to support Palm and PocketPC devices in an existing XML/XSL-based Web site, all that is needed is to write the appropriate XSL style sheets for the requesting browsers. Thus, with relatively little effort, Web server systems can dramatically increase their potential client base to include a whole new set of devices and hence, a whole new set of traffic demands.

Web Services is yet another technology that many Web server systems are beginning to offer and it is becoming increasingly popular. Use of this technology allows applications to act as clients of Web server systems. These applications can place a much greater demand on Web server systems than a user sitting at a browser. As a result, architectures and algorithms need to evolve to better support this expanding set of new clients.

1.1 Objectives

Several research efforts are ongoing in Web-based information systems to satisfy user needs for efficient Web services. Research interest in this field is likely to increase because of the number of users accessing the Web and expanding developments in Web technologies.

The goals of this research are to:

- extend an existing taxonomy [CCCY02] of dispatching algorithms.
- develop a unified product-based taxonomy, which uses the taxonomic categories developed in [CCCY02].
- build generic queuing models to evaluate Web server systems with the help of metrics such as throughput, server performance, mean queue size, mean waiting time, mean service time, and mean response time.
- analyze the correctness of models using theoretical and simulated results of the model.
- validate and verify these models for accuracy and correctness.

This thesis reviews the state-of-the-art in commercial and prototype Web server systems, and devises a *unified product-based taxonomy* that identifies product capabilities by relating them to a classification of scalable Web server systems and a taxonomy of dispatching algorithms. To accomplish a unified taxonomy that encompasses a large set of products, the set of dispatching algorithms identified by Cardellini et al. [CCCY02], Colajanni et al. [CYD98], and Schroeder et al. [SGR00] are extended to include a number of additional classifications. The unified product-based taxonomy forms a basis to analyze and quantify the fundamental characteristics of the products and prototypes. Thus, this new taxonomy is a significant step towards a methodology for classifying products with respect to their capabilities. Generic queuing models are built and the above mentioned performance metrics are evaluated. The correctness of these models is also analyzed by comparing the theoretical and simulation results of these models. The findings are valuable to business professionals to help them compare products against one another when evaluating a Web server system deployment. Furthermore, these models could be extended by designing and analyzing models with exact product specifications, which in turn help software architects in identifying weaknesses in existing products. New products with enhanced capabilities that support emerging client bases such as mobile devices and Web service applications could be developed.

1.2 Basic Architecture of Web server systems

This section analyzes the flow of messages in the Internet. It first considers the basic message flow structure and then describes the details in terms of message flow, protocols, ports, and standards. This section is meant as an overview of message sequence interchange and will help establish the basis for analyzing web server systems in the following chapters.

When a client demands a request from a server, it initially sends a connection request message to the server. The server receives the connection request message and sends back a connection reply message to the client, based on its availability and then any necessary information is transferred. The general flow of requests from a client to a server is represented in Figure 1.1 [KR03].

Internet activities are administered by protocols, which are standards involving the communication between entities like client and server. For communication between computers, the International Organization for Standardization (ISO) developed communication architectures, Open System Interconnection (OSI) model and Transmission Control Protocol /Internet Protocol (TCP/IP) model that describe the linking standards. The OSI model consists of seven layers, physical, data link,

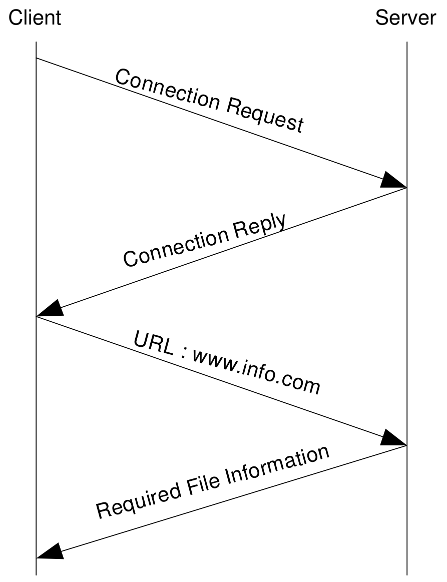

Figure 1.1: Message flow—Client to Server

network, transport, session, presentation and application. In the TCP/IP model, the number of layers is reduced to five, which are physical, data link, network, transport and application accommodating the necessary requirements. Some of the protocols used in different layers are HTTP (HyperText Transfer Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), DNS (Domain Name Server) [Application Layer], TCP (Transmission Control Protocol), UDP (User Datagram Protocol) [Transport Layer], and IP (Internet Protocol) [Network Layer] [KR03].

In the Internet, Web servers are computers that administer and serve-up Web pages. They have specific IP address and are typically referenced by a domain name. Domain names are "user-friendly" identifiers of IP addresses. Virtually any modern computer can be modified into a Web server by installing server software and linking it to the Internet. When a client visits a Web page, with the help of the URL of that site, the browser initiates the Web server connection through the Internet. DNS servers are responsible for mapping a domain name to an IP address of a Web server system and the respective server that stores client desired information.

The flow of the requests from a client to a target server consists of two phases: (i) the look-up phase and (ii) the request-response phase. The basic architecture of the request flow in Web server systems is shown in Figure 1.2. In the look-up phase, a URL request from a client is directed to the Authoritative DNS servers (A-DNS) (step 1), which are responsible for the translation of the Web site name to the IP address of the target server. The client uses the IP address obtained from an A-DNS server (step 2) to establish a TCP connection to the destination. The request-response phase involves forwarding requests to the target server (step 3) and delivering the processed requests to clients (step 4).

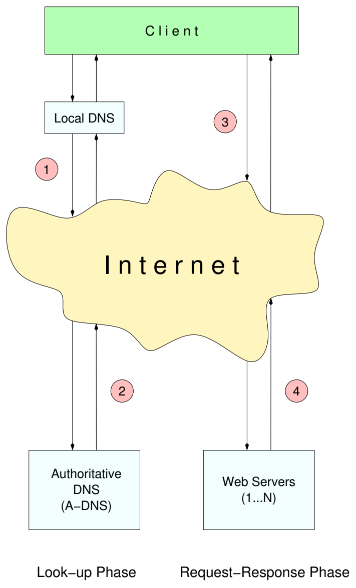

Figure 1.2: Basic Architecture of Web Server Systems

Look-up Phase:

The entities involved in the look-up phase are the client and DNS servers. DNS consists of a collection of name servers and application layer DNS protocols for translating the host name to an IP address [KR03, Bla00]. DNS servers run over the transport layer protocol UDP at port 53 to communicate with the clients for IP address translation. The name servers found in DNS are local DNS, root name DNS, intermediate DNS, and authoritative DNS servers. The flow of requests to the DNS from the client is shown in Figure 1.3.

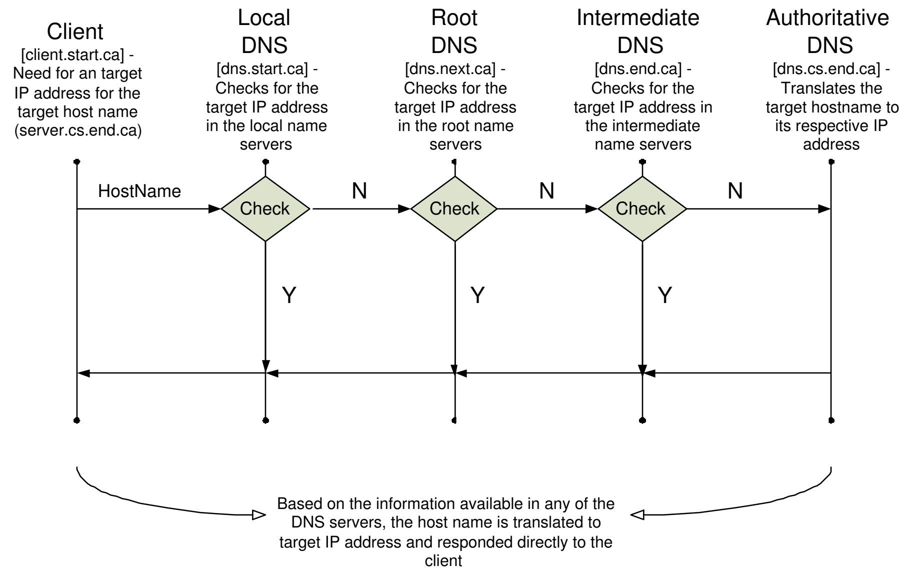
Client
[client.start.ca] - Need for an target IP address for the target host name (server.cs.end.ca)
Local DNS
[dns.start.ca] - Checks for the target IP address in the local name servers
Root DNS
[dns.next.ca] - Checks for the target IP address in the root name servers
Intermediate DNS
[dns.end.ca] - Checks for the target IP address in the intermediate name servers
Authoritative DNS
[dns.cs.end.ca] - Translates the target hostname to its respective IP address
HostName
Check
N
Y
Based on the information available in any of the DNS servers, the host name is translated to target IP address and responded directly to the client

Figure 1.3: Flow of requests—Client to DNS servers for IP address translation

As an example, let us consider a client with a host name client.start.ca that requests a URL *server.cs.end.ca/index.html*. The URL consists of two parts, a host name and the path name. The client browser is responsible for extracting the host name from the URL. The client needs to target the Web server, which holds the specific information. In order to obtain the IP address of the target server, the client contacts the DNS. First, the client application checks the client side of the DNS and responds directly if the target IP address is found or passed to the collection of name servers in DNS. The flow of DNS servers, is such that, it reaches local DNS, root DNS, intermediate DNS and authoritative DNS for the IP address translation of the target server. Based on the information found in any level of DNS servers, the host name is translated to an IP address that is sent back to the client. The local DNS are DNS servers pertaining to that locality, such as a department or a company. Suppose our local name server is *dns.start.ca*. It will check for the IP address translation for *server.cs.end.ca*. Alternatively, the client query is passed on to root DNS, which are grouped based on a zone or an area if it is not listed in the local DNS. Let us consider the IP address of root DNS as *dns.next.ca*. From the root DNS, if the IP address of the target server is not translated, the client query is supplied to the intermediate DNS (address: *dns.end.ca*) and then A-DNS (address: *dns.cs.end.ca*), in which every host for domain cs.end.ca is registered and a DNS record for translating the host name (*server.cs.end.ca*) to IP address should be found.

Request-Response Phase:

In the request-response phase section, the HTTP (HyperText Transfer Protocol) application layer protocol and its Web activities are overviewed [KR03, Bla00]. HTTP explains the standards for communication with the client and the server. The client and server communicate with the help of HTTP message requests as shown in Figure 1.4 [KR03]. The HTTP client initiates a TCP connection to the target server. After the establishment of the connection, the client and target server processes exchange information by accessing TCP through a socket interface.

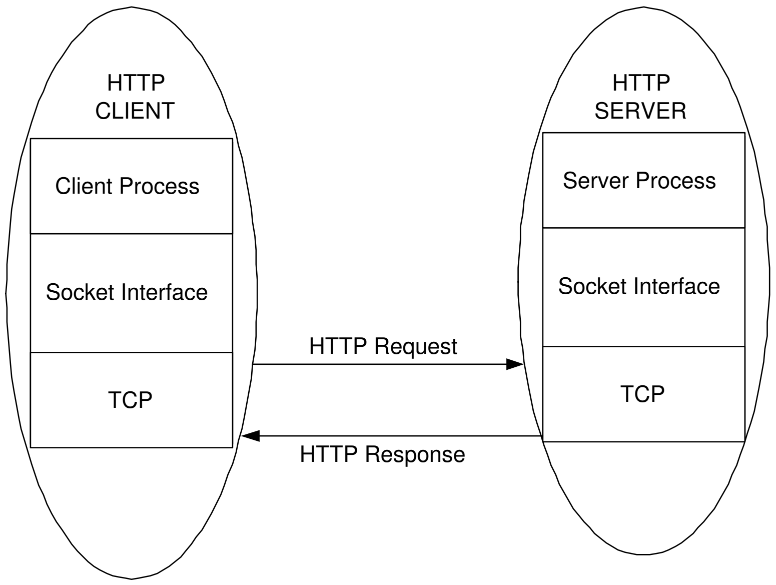

Figure 1.4: HTTP client and HTTP server communication

The message sequence is as follows:

- The HTTP client initiates a TCP connection to the server server.cs.end.ca at port number 80, which is the default HTTP port number
- After the acknowledgement from the server for a TCP connection, the HTTP client sends a HTTP request message via a socket interface with both the host name and path name *server.cs.end.ca/index.html*
- The HTTP server accepts the client requests and extracts the object */index.html* and responds to the client via a socket interface
- The HTTP client accepts the server response and the TCP connection is terminated only after client extracts the required file information

HTTP connections are of two types: Non-persistent (HTTP/1.0) and persistent (HTTP/1.1). In non-persistent connections, a single Web object is exchanged over a TCP connection. Persistent connections involve the transfer of multiple Web objects in the same TCP connection. Persistent connections can be explained by two mechanisms: with pipelining and without pipelining. In persistent connection without pipelining, after establishing TCP connection, each client request waits for the response from the server before its next request. In persistent connection with pipelining, new client requests can be supplied to the server, before its first response from the server.

The decision to choose the request route from a client to the target server can be done at several places. The four possible ways for this routing decision of the client requests to the target server are [NM00]: (i) Web client-based, (ii) DNS-based, (iii) dispatcher-based, and (iv) server-based. In the first approach, the client, who originates the request, is responsible for request routing. In the DNS-based approach, routing decisions are made during the look-up phase and are handled by the A-DNS servers. The dispatcher-based approach deals with switching devices for routing the requests to the target server. In this approach, the basic scheduling algorithms include Round Robin (RR), Least Connected, and Least Loaded techniques [TA01]. In the server-based approach, the Web server systems processes the requests by themselves or redirects them to other servers in the Web server system.

In this thesis, we concentrate on the Layer-4 one-way architecture, which involves a dispatcher unit for distribution of client requests to the server unit. Figure 1.5 illustrates the physical flow of client requests to a target server, via Layer-4 dispatcher. The Layer-4 dispatcher acts as a dispatcher unit and is responsible in keeping track of target server IP addresses. The IP address of the Layer-4 dispatcher is alone visible to the client, while the IP addresses of the target servers are hidden.

The client contacts the dispatcher, which in turn connects to the target server. In the one-way architecture, the target server responds to the client without passing through the dispatcher again.

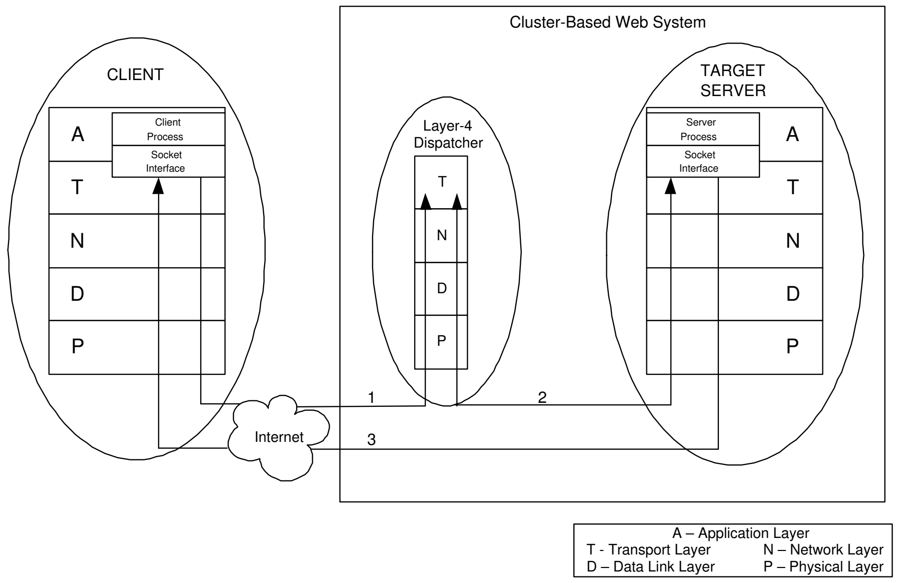

Figure 1.5: Physical Flow of Requests—Client, Dispatcher Unit and Server Unit

Let us consider the packet single rewriting one-way architecture in detail. IBMs TCP router supports the packet single rewriting one-way architecture. The inbound client packets reach the dispatcher, whose IP address is alone visible to the client. The dispatcher replaces the destination address of the inbound packets from VIP address of the dispatcher to the IP address of the target server and TCP/IP header sum is recalculated. In the target server, before sending the response packets to the client, IP address of the target server is replaced with the VIP address of the dispatcher, such that the client contacts the dispatcher for all its processes.

The travel path of the requests from the client, dispatcher and the target server is explained in Figure 1.6 [KR03]. In the first part of the handshake, the client

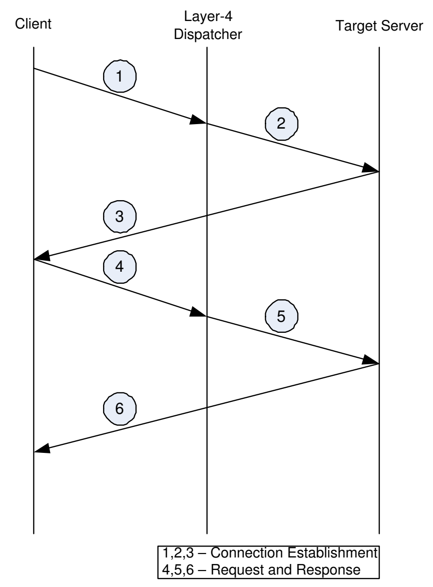
Client
Layer-4 Dispatcher
Target Server
1
2
3
4
5
6
1,2,3 – Connection Establishment
4,5,6 – Request and Response

Figure 1.6: Handshakes: Client–Dispatcher–Server

initiates a call to get a TCP connection with the target server. The dispatcher routes the request to the target server, which responds directly to the client. The steps 1, 2, and 3 illustrate the connection establishment phase with the client and server. After the connection is established, the client acknowledges the dispatcher, which in turn connects with the target server for the data transfer.

In summary, this section describes the message sequence between a client and a server. Various tasks involved during this communication process are explained in detail. It gives an overview of the flow of messages from a client to a server, and vice versa.

1.3 Key Contributions

The significant contributions of this thesis are as follows:

- The extended taxonomy of dispatching algorithms brings content-blind, content-aware, DNS scheduling algorithms and a few other additional classifications under a single taxonomy, which provides a broad overview of dispatching algorithms found in the field.
- The unified product-based taxonomy could help allow business professionals to explore and pick products in the market based on their architectural and dispatching characteristics. This research could also be applied to designing products with new combinations of Web server system architectures and dispatching algorithms.
- Evaluation of generic queuing models form the basis of Web server modeling. The identified metrics used in such an evaluation contribute to analyzing the Web server systems in depth.

1.4 Organization of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 discusses the previous work on the existing taxonomies of scalable Web server systems and dispatching algorithms, along with the extended taxonomy of dispatching algorithms. Chapter 3 describes the classification of the various products and prototypes in reference to the integrated product analysis taxonomy. Chapter 4 presents the generic queuing model architecture, its operation details, and the theoretical and simulation analysis for random and round robin dispatching policies. Finally, chapter 5 concludes the thesis and outlines future work.

Chapter 2

Taxonomy

Taxonomy is the process of grouping any entity. Identification has always been a challenging task in science. Taxonomy attempts to identify various classes and organize them into broader categories. These classifications are of great explanatory value and display a vivid picture of the existing diversity. The knowledge of any system can be summarized by integrating the system components into a taxonomy.

As demands for Web-based services and complexity in Web applications are escalating, researchers are increasingly interested in various Web system related issues. Research has been done in architectural and dispatching solutions for Web server systems. The classification of scalable Web server systems and the taxonomy of dispatching algorithms are explained in this chapter in detail. An extended taxonomy of dispatching algorithms is developed and dispatching algorithms are grouped under one graph.

Several on-going research efforts categorize the various Web server technologies. In this thesis, Web server architectures and dispatching algorithms are elaborated, as they form the basis for building a new product taxonomy. This chapter deals with the scalable Web server system architectures and the dispatching algorithms in particular.

Scalable Web server systems are broadly classified into "scale-up" and "scale-out" categories [CCCY02]. The scale-up category deals with a single server and is divided into hardware and software scale-up. Hardware scale-up [DGLS99] involves adding more resources to an existing server, thus relieving short-term pressure only. Software scale-up [NBK02] includes employing efficient Web server algorithms with suitable dispatching policies and building effective operating systems. The scale-out category [DGLS99], which involves multiple servers, is further divided into local scale-out and global scale-out categories. Local scale-out systems include servers that lie within a single local network. The complete classification of Web server systems as discussed in [GEP02] is shown in Figure 2.1. In the global scale-out category, servers are located at different geographical locations. In this thesis, we concentrate more on the local scale-out Web server systems and discuss their server architectures and dispatching algorithms, in detail.

2.1 Server Architectures

Local scale-out distributed architectures are divided into three main classes: (i) cluster-based Web systems, (ii) virtual Web clusters, and (iii) distributed Web systems [CCCY02]. In a cluster-based Web system, a dispatcher is involved in the distribution of the requests. Only the virtual IP address (VIPA) of the dispatcher is visible to the clients and all actual servers IP addresses are hidden from the clients. In virtual Web clusters no separate dispatching device is used for request routing.

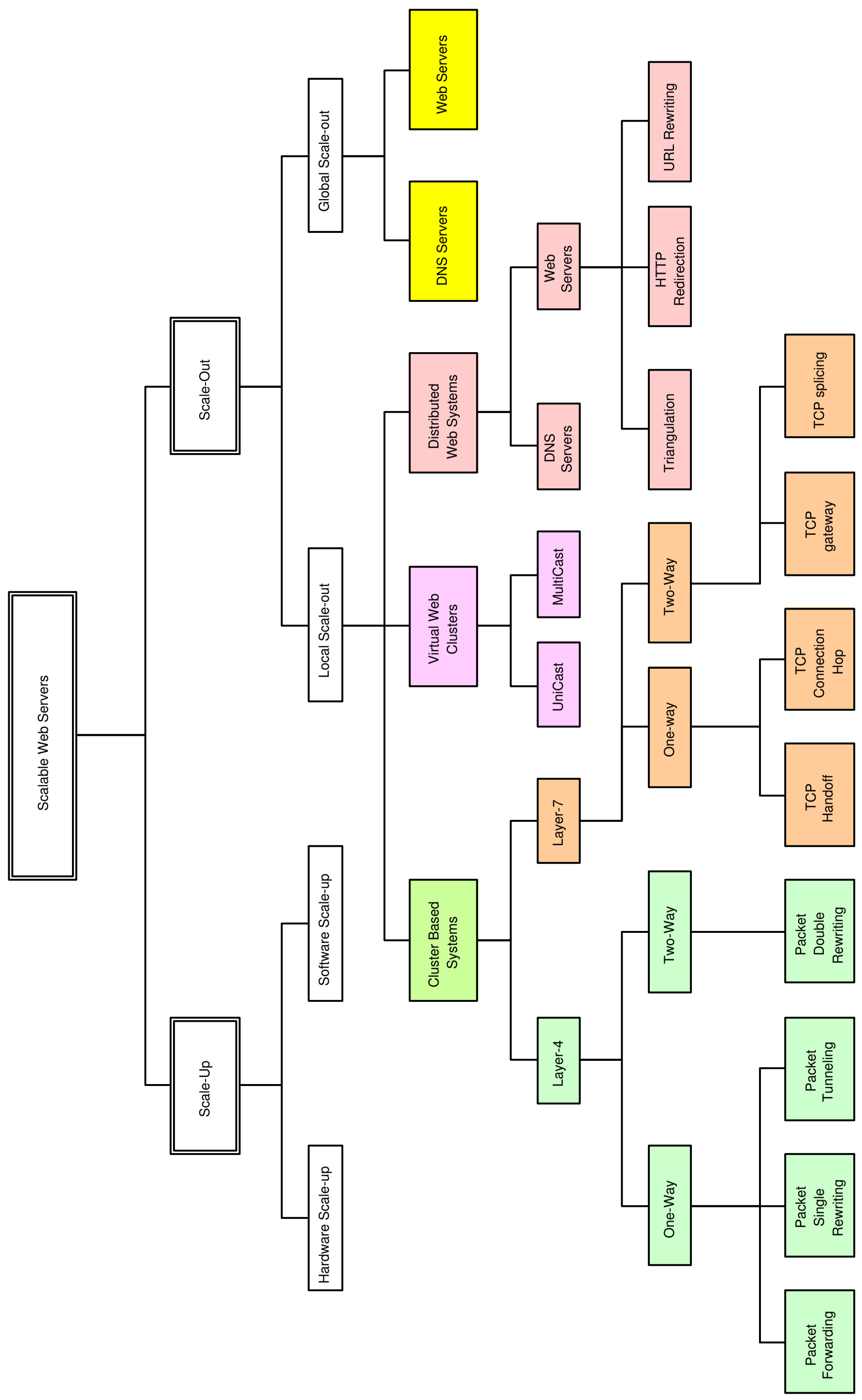

Figure 2.1: Detailed Taxonomy of Web Server Architectures

All the servers in the Web server system share the same VIPA and this VIPA is the only one visible to the client. Request routing in virtual Web clusters is done at the media access control (MAC) layer. A filtering mechanism is employed in each server of the Web server system to accept or reject the requests. Distributed Web systems have a visible IP architecture where the IP addresses of the servers are visible to clients. Routing of requests can be done at the DNS or Web server level in distributed Web systems.

2.1.1 Cluster-Based Web Systems

Cluster-based Web system deals with a dispatcher unit for routing of packets to the target server, as shown in Figure 2.2. Cluster-based systems are classified either as Layer-4 or Layer-7, according to the OSI layer protocol stack. Depending on the data flow, Layer-4 and Layer-7 are further classified as one-way and two-way architectures. One-way architectures allow only inbound packets to flow through the dispatcher, while the target servers communicate the outbound packets directly to the client. In two-way architectures both inbound and outbound packets flow through the (front-end) dispatcher. In a one-way Layer-4 architecture the routing techniques involved are packet single rewriting, packet tunneling, and packet forwarding [ADZ99]. In the packet single rewriting technique the dispatcher rewrites the destination address of inbound packets. The VIPA and the earlier destination address of inbound packets are rewritten to the target server's IP address to process the requests. The outbound packets are delivered directly to the client. IP tunneling involves encapsulation of IP datagrams within IP datagrams. The inbound packet is encapsulated with an IP datagram, which holds the VIPA and the target server's IP

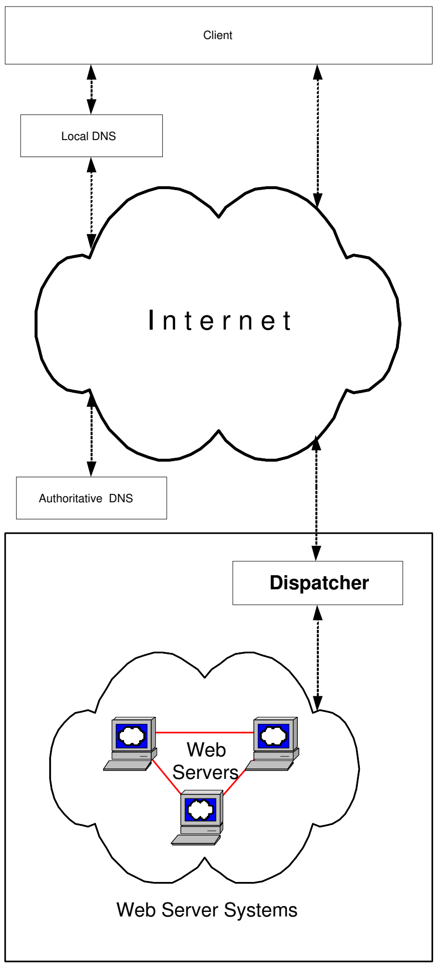

Figure 2.2: Cluster-Based Web System Architecture

address as its source and destination addresses, respectively. Once inbound packets reach the target server the encapsulated datagram is stripped for processing requests. In packet forwarding the same VIPA is shared by all Web servers and the dispatcher. The unique private addressing of each server in the Web server system is done at the MAC layer. A dispatcher directs inbound packets to a target server whose MAC address is used for destination reference.

In packet double rewriting of two-way architecture the dispatcher rewrites the destination address field of inbound packets by changing the VIPA to the unique IP address of the target server. The source address of outbound packets is also rewritten by changing it from the IP address of the target server to the VIPA.

In the Layer-7 category a dispatcher establishes a TCP connection and the HTTP requests are confirmed before any decisions are made. In the one-way architecture of Layer-7 TCP hand-off and TCP connection hop mechanisms are involved. In the TCP hand-off mechanism the dispatcher hands off the TCP connection to a target server once the connection is established. Persistent connection is achieved with the help of multiple hand-off and back-end forwarding mechanisms. In the multiple hand-off method a hand-off protocol is used to migrate the connections between Web servers [PAB^{+}98, HKM98]. The back-end forwarding mechanism deals with forwarding requests from a server to other servers in Web server system when they could not serve a particular request. In the TCP connection hop method the packets are hopped to the target server after the connection is established. The packets are encapsulated with the help of a TCP-based encapsulation protocol when using TCP connection hop. Responses are delivered directly to clients as these techniques fall under one-way architecture.

A two-way architecture in Layer-7 deals with TCP gateway and TCP splicing [MB98] for routing decisions. In a TCP gateway mechanism a proxy is responsible for client-server communication, which runs in the dispatcher at the application level. An open persistent TCP connection is maintained by the proxy for distribution of the requests to a target server. The TCP splicing method forwards packets at the network level between the network interface card and the TCP/IP stack. Once the client to switch connection and TCP persistent connection between the switch and servers are established both connections are spliced together for communication. IP packets are forwarded at the TCP layer on the dispatcher without passing up to the application layer on the Web switch.

2.1.2 Virtual Web Clusters

In virtual Web cluster architecture request routing is content-blind. This server architecture, as represented in Figure 2.3, avoids the single point of failure because it has no dispatcher to direct the client requests to a target server. All servers in the Web server system have a filtering mechanism, which computes a hash function on the respective client's IP address or the port number. Based on the match between the hash value and server's own value the packet is accepted or discarded. As VIPA is shared by all the servers in the Web server system the request routing is done at the MAC layer. Unicast MAC address and multicast MAC address are the two approaches used in the servers of virtual Web clusters for routing [Mic00]. In a unicast approach the original MAC address of the network adapter (called the cluster adapter) is reassigned as the cluster MAC address. This common cluster MAC address is assigned to all the servers in the virtual Web cluster. For intra-network communication a second adapter is used in each server. The second adapter's MAC address is mapped as the server dedicated IP address to communicate between the servers in the Web server system.

In a multicast method all the servers have a virtual common MAC address, which are their references to the virtual Web cluster. All the servers in the cluster hold their unique built-in MAC address for intra-network communication. The servers in the multicast approach works with one network adapter for both client to

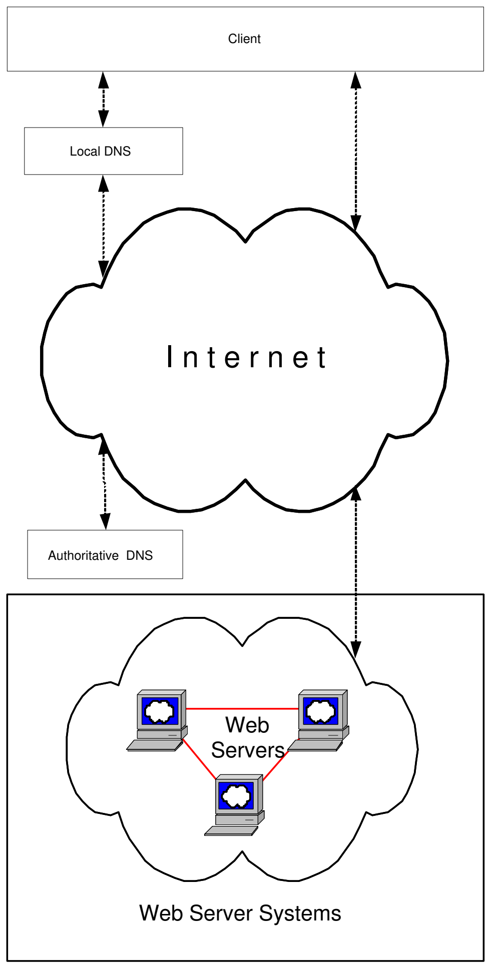

Figure 2.3: Virtual Web Cluster Architecture

server and within server traffic.

2.1.3 Distributed Web Systems

Authorized DNS server are included in the distributed Web system architecture as shown in Figure 2.4. The routing mechanisms for distributed Web systems consist of DNS-based routing and Web server routing. In DNS-based mechanisms routing is done during the look-up phase at the beginning of the Web transaction. A-DNS (Authoritative DNS) systems are responsible for mapping the name of a Web site to the IP address of the respective Web servers. The A-DNS systems respond to every address request with a tuple, which consists of an IP address of one of the servers in the distributed Web system and a Time-To-Live (TTL) value determining the validity period of the host-name address mapping. Web server routing can be implemented by one of the three methods: (i) Triangulation, (ii) HTTP redirection, and (iii) URL rewriting [AB00, Aka99]. The triangulation technique is done at the TCP/IP level while HTTP redirection and URL rewriting mechanisms work at the application level. Triangulation involves a packet-tunneling method for distribution of requests and is also called Distributed Packet Rewriting (DPR) [AB00]. In the DPR approach each Web server keeps track of information about the other servers in the Web server system. Periodic updating of load information is maintained. All servers in the Web server system participate in connection routing. Incoming requests can address any server in the Web server system as all the individual addresses of the servers are published. A request may be redirected to a different server based on the very first packet (SYN packet) received from the client. Each SYN packet contains the Web server system state and relative load. In the HTTP redirection mechanism Web server systems respond to client requests using status codes of 301 or 302 to support URI (Uniform Resource Identifier) based redirection. Status code 301 implies that the requested resource is assigned a new URI and has moved permanently while status code 302 infers that a temporary URI is assigned to the requested resource. The drawback in this mechanism is the extra round trip latency during the redirection of resources. In the URL rewriting mechanism the first contacted server changes the links for the embedded objects dynamically to the

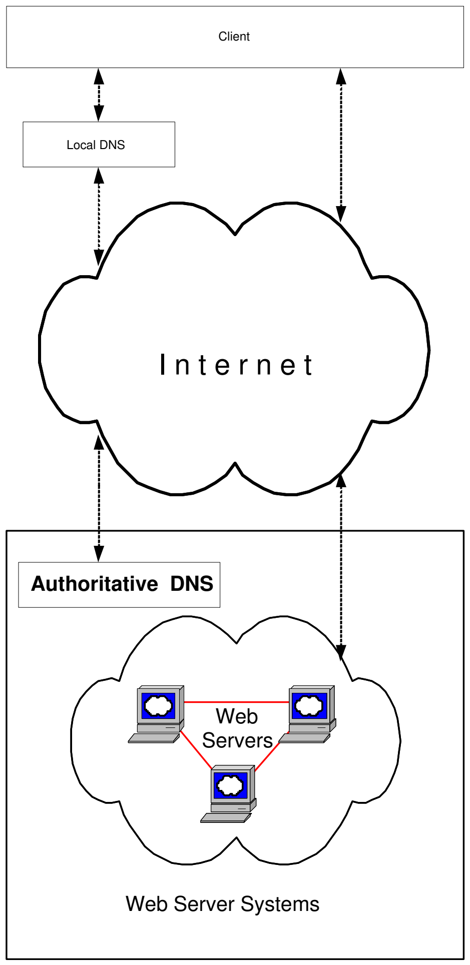

Figure 2.4: Distributed Web System Architecture

target server during the redirection.

2.2 Dispatching Algorithms

Dispatching algorithms are responsible for selecting the best-suited server to respond to client requests [CCCY02]. They are static, dynamic, or adaptive. In static algorithms the scheduling decisions are made without considering any dynamic state information, and hence they are fast. Most commercial products prefer these simple algorithms for doing server selection. *Random* and *Round Robin (RR)* are examples of static algorithms [CYD98]. In dynamic algorithms server selection is based on the run-time system state information. Dynamic algorithms may be further classified as client-state-aware policies, server-state-aware policies, and combined client-server-state-aware policies. In client-state-aware policies request routing is done on the basis of some client information (such as client IP address or TCP port). A server-state-aware policy is based on past or current server load conditions, the latency time to service requests, and server availability. In combined client-server-state-aware policies, a dispatcher selects the target server using both client and server state information. In adaptive algorithms the heterogeneity of Web servers and changes in the Web server system are considered and dispatching is adapted accordingly [CCY99].

The dispatching algorithms are broadly classified into content-blind and content-aware dispatching [CCCY02]. Dispatching algorithms that do not consider message content in performing target server selection and which work at the TCP/IP layer are termed content-blind. In content-blind dispatching the routing decisions are based only on IP header information and port numbers, and they do not consider the actual content information in the packets. In content-aware dispatching policies the process of selecting the target server may also consider detailed information about the content of client requests. Content-aware dispatching policies work at the application layer [Cis00, F5N00, LVS98, Fou02, HKM98]. Figure 2.5 shows the taxonomy of dispatching algorithms [CCCY02].

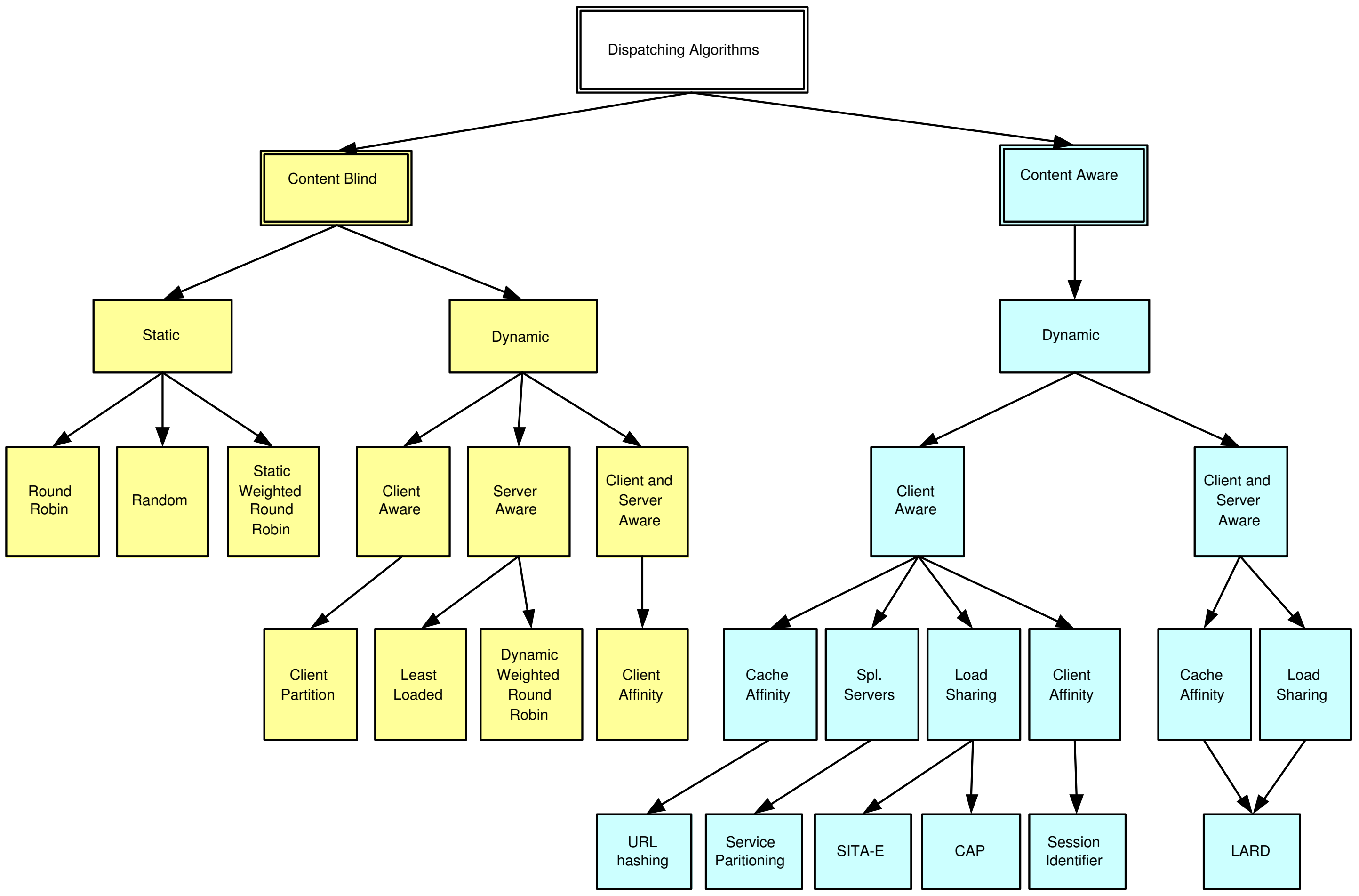

Figure 2.5: Taxonomy of Dispatching Algorithms

2.2.1 Content-Blind Dispatching Policies

Random and RR are two examples of static algorithms that use content-blind dispatching. In Random algorithm any server in the Web server system can be picked for request processing. RR algorithms select servers in a cyclic manner. As static algorithms do not consider the current state information, they have the disadvantage of possibly selecting faulty servers. A modified version of RR algorithm is the *Static Weighted Round Robin (SWRR)* algorithm in which servers are assigned an integer weight indicating their capacity. In client-state-aware dynamic policies the dispatcher uses the client's source IP address and TCP port number for server selection. Servers are statically partitioned and the same clients are assigned to the same servers based on a hash function, which is applied to the client's IP address.

Server-state-aware policies use a load index of Web servers to decide on the target server's address. Server-state-aware policies often use *Least Loaded* and *Dynamic Weighted Round Robin (DWRR)* algorithms as their routing approach. Algorithms like *Least Connections (LC)* and *Fastest Response Time (FRT)* fit into the least loaded approach. In LC new requests are assigned to the servers with the fewest active connections, while in FRT Web servers with smallest latency time in the last observation interval are assigned a new connection, as they respond faster. In DWRR algorithm each server is assigned a dynamic weight, which is proportional to the server load state. The dispatcher gathers load information periodically from the Web servers and the dynamic weights are computed. A *Client Affinity* algorithm has also been proposed for client-server-state-aware policy, which is responsible for the assignment of the same client to the same server.

2.2.2 Content-Aware Dispatching Policies

Content-Aware dispatching use client-state-aware and client-server-state-aware policies, and they work at the application level. *Cache Affinity*, *Specialized Servers*, *Load Sharing*, and *Client Affinity* algorithms fall under client-state-aware policies. Cache Affinity algorithms partition the file space among servers using a hash function applied to each URL. Some Web sites provide heterogeneous services and their Web servers can be partitioned according to the services they handle. These specialized servers are employed to manage certain types of requests like dynamic content, multimedia files, and streaming video. This content-aware dispatching algorithm is referred as *Service Partitioning* that comes under the specialized server category.

Size Interval Task Assignment with Equal load (SITA-E) and *Client-Aware-Policy (CAP)* support load sharing. SITA-E algorithm deals with assigning tasks of certain size range to individual servers. The advantages of using the SITA-E is that it limits the range of task sizes assigned to each server and reduces the waiting time of the tasks. A popular content-aware algorithm is the *Locality-Aware Request Distribution (LARD)*, which considers both client and server information for its server selection. In LARD the incoming requests are directed to the same servers from the same clients until a given load threshold is reached. When the threshold is exceeded requests are redirected to the least loaded server or the server with the fewest connections.

2.3 Extended Taxonomy of Dispatching Algorithms

Merging Domain Name Server (DNS) scheduling in the content-blind and content-aware classification could extend the earlier dispatching algorithms taxonomy. Distributed Web server systems use DNS scheduling algorithms for request routing. They are content-blind and are associated with the content-blind category. Incorporating the server-state-content-aware policy to the earlier taxonomy further extends the content-aware classification. A good example for this extended server-state-content-aware policy is the Harvard Array of Cluster Computers (HACC) prototype [ZBCS99], which uses the *Least Loaded* algorithm for distributing client requests. This thesis extends the taxonomy of dispatching algorithms and the new categories are represented as round objects as shown in Figure 2.6.

In this combined taxonomy the client-state-aware component is extended with DNS scheduling algorithms like *Proximity* and *Multi-Tier Round Robin (MTRR)*. Proximity algorithms focus on network proximity information such as round trip delays for the selection of servers. MTRR algorithms use client information hidden load weight, which is the average data request rate from a domain to Web site during each measurement period. For different ranges of hidden load weight various RR chains are used in MTRR. Other extensions found in the updated dispatching taxonomy is the inclusion of *Least Residual Load* and *Adaptive TTL* algorithms in the client-server-state-aware policy. Least Residual algorithm selects the respective Web server based on both domain and server information [CYD98]. The hidden load weight along with Web server systems of minimum number of residual requests are considered for request processing in the least residual policy. The adaptive TTL algorithm involves the dynamic reduction of the TTL value. In adaptive

TTL the choice of the Web server depends on the hidden load weight algorithms and an appropriate TTL period value assigned by the DNS servers. Adaptive TTL does not take any proximity into consideration.

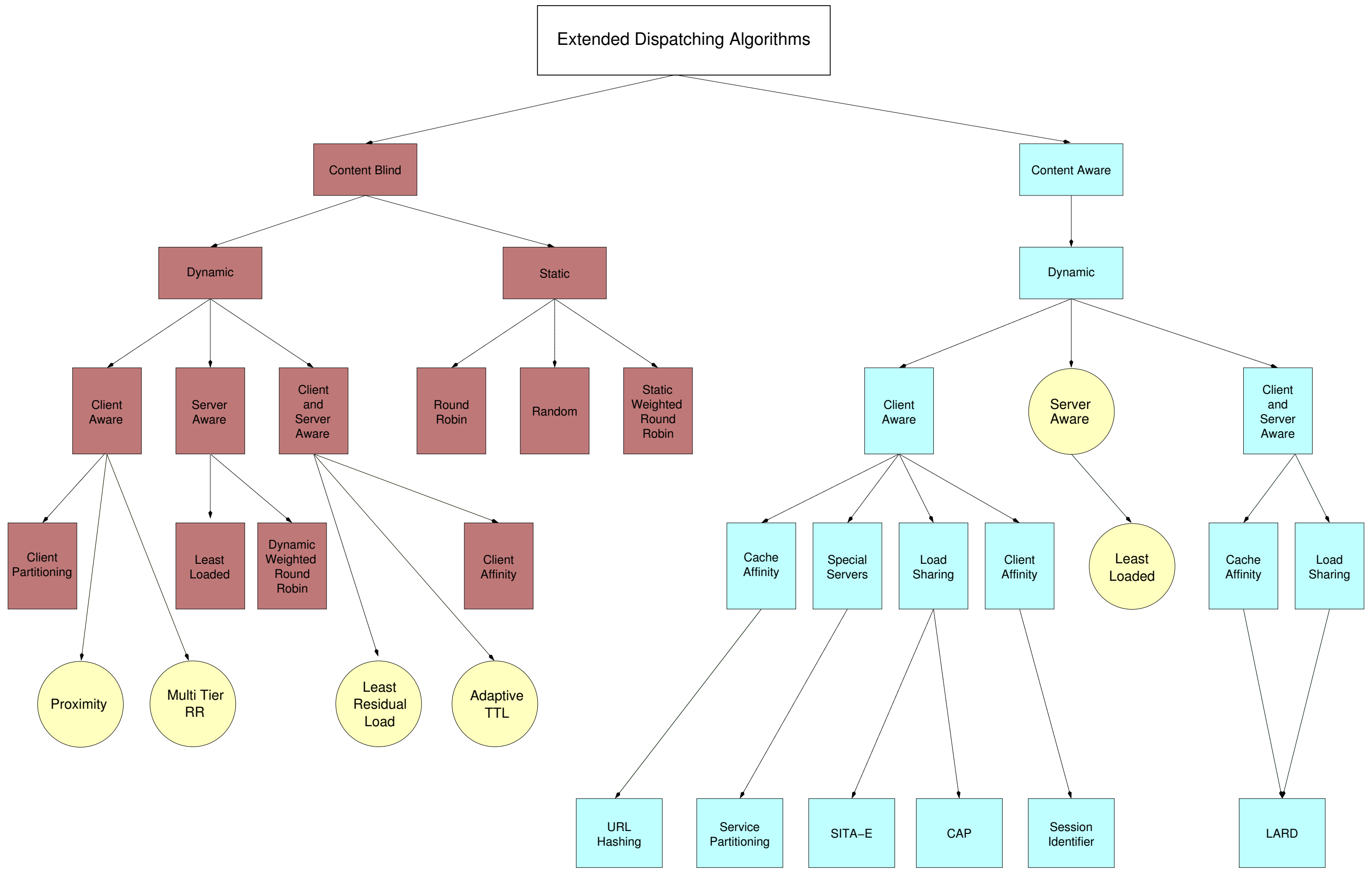

Figure 2.6: Extended Taxonomy of Dispatching Algorithms

Chapter 3

Products and Prototypes

In recent years many companies and organizations have concentrated on building commercial products, developing research prototypes, and expanding their knowledge in Web server systems. Both educational institutions and industries have come up with innovative and advanced solutions in this field. A classification of these products and prototypes gives a better understanding of these entities that are launched in the market. In order to have a clearer picture of the Web server systems categorization, a taxonomy with classifications of server architectures, dispatching algorithms, products, and prototypes is very much necessary and helpful. This motivated the development of a unified product-based taxonomy in this thesis. This chapter gives an overview of the commercial products and research prototypes found in the market along with the unified product-based taxonomy.

Commercial products select simple dispatching algorithms for their routing as they are cost effective and easy to implement. Research prototypes tend to explore new areas and provide ingenious solutions.

3.1 Related Work

The need for efficient load balancing techniques in Web server systems rises due to an increase in the number of Internet applications. With the help of load balancing technology available servers in the Web server system are utilized to their maximum potential. The dispatcher, which is responsible for distributing the load among Web servers, receives all client requests and selects the best server for the job. The load balancing techniques used in these commercial products and research prototypes are built in such a way that they suit different Web server architectures and dispatching algorithms based on client requirements.

Commercial products and prototypes are implemented on specialized hardware or in software. Request routing techniques are implemented on hardware in some of the products like Foundry Networks' ServerIron [Fou02], Cisco's LocalDirector (LD) [Cis00], and Cisco's DistributedDirector (DD) [Cis02]. In the Linux Virtual Server (LVS) [LVS98], the Resonate's Central Dispatch (CD) [Res00], Microsoft's Network Load Balancing (NLB) [Mic00], Akamai [Aka99], IBM's Network Dispatcher (ND) [IBM01] and Zeus's Load Balancer (LB) [Zeu02], the routing techniques are implemented in software. Certain products like F5 Network's BIG-IP [F5N00] and Nortel Networks' Alteon Web Switch (AWS) family [Nor00] use both hardware and software in their implementation for forwarding their requests to Web servers. Non-commercial software solutions are identified as research prototypes. Magicrouter [APE96], Harvard Array of Cluster Computers (HACC) [ZBCS99], ClubWeb [CC01], and ScalaServer [PAB$^+$98] prototypes use software for implementing their routing decisions. Some of the other prototypes are TCP Router [DKMT96], ONE-IP [DCH$^+$97], National Center for Supercomputing Applications

(NCSA) [KBM94], Scalable Web Server (SWEB) [AYHI96], Distributed Packet Rewriting (DPR) [AB00], and Internet2 Distributed Storage Infrastructure project (I2-DSI) [BM98].

3.1.1 Hardware Solutions

ServerIron supports high performance Layer-2 through Layer-7 switching. ServerIron accommodates Internet traffic management applications like Reliable Server Load Balancing (RSLB), Global Server Load Balancing (GSLB), FireWall Load Balancing (FWLB), and Transparent Cache Switching (TCS). The key benefits of this product are server and application availability, maximum scalability, and simple configuration.

Cisco's LD is a hardware solution that load balances TCP/IP traffic across multiple servers and is highly scalable. LD connected with Catalyst 6000 family switch supports Accelerated Server Load Balancing (ASLB) that allows it to accelerate TCP sessions and to provide high availability. LD security features filter incoming traffic and allow only essential requests to pass through real and virtual servers, thus protecting the Web server system.

DD provides efficient distributed Internet services globally. DD are very suited for load balancing geographically dispersed servers. With the help of a routing table, intelligence in the network infrastructure, and Distributed Response Protocol (DRP), DD forwards the client requests to the closest proximity available server increasing the server access performance.

3.1.2 Software Solutions

LVS is a software tool that runs on the Linux operating system. It supports network services with high scalability and availability. Load balancing is done at the IP level (Layer 4) in LVS. The three load balancing techniques used in LVS are virtual server via Network Address Translation (NAT), via IP tunneling, and via direct routing.

In CD the traffic management software ensures optimal performance for e-business applications and online services. CD includes useful load balancing API that provides effective traffic management solutions. CD also accommodates enhanced denial of service protection and end-to-end transaction time measurement.

Microsoft's NLB uses a clustering software technology that supports a virtual Web cluster architecture. Some of the key features of NLB are secure network communications, routing and remote access services, Virtual Private Networking (VPN), and dynamic DNS services. Due to virtual Web clustering in NLB, a high availability, scalability and easier manageability of Web servers is achieved.

Akamai provides e-business infrastructure services and assures faster and easier implementation. Akamai servers are distributed worldwide and proximity between servers and clients are reduced to a larger extent, which provides efficient services to their customers.

IBM's ND is capable of supporting large-scale high-load Web sites. Its TCP connection router provides a fast IP packet forwarding kernel extension to the TCP/IP stack. This software product began as a research prototype supporting scalable Internet services [HKM98]. The highlights and the usefulness of this prototype were recognized and the prototype was produced as an IBM product.

LB is a pure software application that provides excellent solutions for Web server management, traffic management, and e-commerce applications. LB works on Layer-7, which the application layer for routing the requests to respective servers. LB application runs on the front-end machines in order to route the incoming requests to the back-end server machines. A Zeus Admin server, which can be installed in the front-end, back-end or on separate dedicated machines, monitors the performance of Web servers and manages LB.

3.1.3 Both Hardware and Software Solutions

BIG-IP product deals with intelligent load balancing and excellent fail over detection mechanisms. Its exceptional traffic control mechanisms assure Quality of Services to end users. Health monitors in BIG-IP helps in real time performance monitoring and to test the availability of servers and applications in the Web server system. BIG-IP uses a specialized hardware switch combined with some control functions implemented in software for request routing.

AWS product family consists of Alteon ACEdirector series, Alteon 180 series, and the Alteon Web Switching Module for the Passport 8600. The main advantages of AWS are intelligent traffic management, multi-application support, network scalability, high performance security, rapid deployment, and fail-safe network assurance. Alteon Web OS [Nor02] software implemented on AWS provides advanced filtering, application redirection, effective server load balancing, and security services.

3.1.4 Prototypes

In Magicrouter a software prototype developed at University of California, Berkeley implements fast packet interposing to intercept packets to different destinations. Fast packet interposing allows a user level process to modify the packet's data when flowing through device drivers to redirect to a different route.

HACC concentrates on locality enhancement and dynamic load balancing. A Smart router found in HACC consists of two layers: (i) High Smart Router (HSR) and (ii) Low Smart Router (LSR). Dynamic load balancing is implemented with the help of the Performance Data Helper (PDH) interface. PDH acts as a monitoring agent that gathers the machine's performance statistics.

The ClubWeb prototype uses Client-Aware policy (CAP) as its dispatching policy. It classifies user requests based on the impact on server resources like CPU, network interface, and disk. In this prototype the Web cluster architecture consists of a dispatcher for distributing the client requests and a LAN connection that connects the back-end servers and Web servers with the dispatcher.

ScalaServer focuses on the resource management in cluster-based Web systems. ScalaServer, developed at the Rice University and uses a content-based request distribution technique. A persistent connection is established with the client and respective Web server by the dispatcher allowing multiple requests to the same server, which in turn reduces client latency and server overhead.

The TCP Router uses a combination of TCP routing and DNS techniques for load balancing the requests. TCP router provides an effective server failure detection mechanism to enhance high availability of Web servers for processing requests at higher rates.

ONE-IP dispatches packets at the IP level. The two techniques used in ONE-IP for forwarding requests to appropriate servers are routing-based dispatching, where a central dispatcher is involved, and broadcast-based dispatching, which uses a broadcast mechanism and local filtering.

In NCSA a RoundRobin DNS dispatching mechanism is used for request distribution. It allows dynamic scalability and helps in increasing the number of servers and the load capacity of the virtual server. A distributed File System mechanism employed in NCSA manages the synchronized set of documents found in the Web server system.

SWEB is implemented on distributed memory machines and is well suited for client requests that demand large digitalized documents. SWEB monitors and utilizes multiple system resources that effectively supports high scalability and server availability.

DPR is a DNS prototype approach that distributes requests without using centralized resources. With the help of periodic multicast, each server in the Web server system maintains the load information of the other servers. This helps the servers in the Web server system to redistribute the client requests to any available respective servers for processing.

I2-DSI concentrates on the engineering characteristics of available services. The three main components found in I2-DSI are: replicated server systems, transfer file, and intelligent redirection mechanisms. With the I2-DSI architecture, the desired proximity between the client and servers is well achieved, even though the servers are located at any point in the network.

3.2 Unified Product-Based Taxonomy

This section discusses the unified product-based taxonomy in detail. The unified product-based taxonomy is build based on Web server architectures and dispatching algorithms of the products and prototypes. The Content-Blind Layer-4, and Content-Aware Layer-7 classification fall under the cluster-based Web server architecture. Figures 3.1 and 3.3 summarize the unified product-based taxonomy of Content-Blind Layer-4, virtual Web cluster, and distributed Web system architectures that uses content-blind algorithms for distributing client requests. Figure 3.5 illustrates the Content-Aware Layer-7 category of products and prototypes in unified product-based taxonomy. Figures 3.2, 3.4, and 3.6 represent the unified product-based taxonomy in a tabular format.

With the help of this unified product-based taxonomy the various products and prototypes found in the market are identified distinctly with respect to their server architectures and dispatching algorithms. It gives a comprehensive outlook of prevailing products and prototypes helping the research community in designing new products and prototypes with different combinations of architectures and dispatching algorithms.

3.2.1 Content-Blind Layer-4 Products and Prototypes

Cisco's LD, Magicrouter, BIG-IP, TCP Router, LVS, IBM's ND, and ONE-IP are some of the products and prototypes that support cluster-based content-blind Layer-4 server architecture. All the products and prototypes in this content-blind category work at the TCP/IP level and are implemented on specialized hardware or software. Some of the hardware-implemented Layer-4 products are LD and ServerIron.

Products like LVS and ND, and prototypes Magicrouter, TCP Router and BIG-IP are software implementations that uses content-blind information for routing the requests. The content-blind Layer-4 architectures are further classified as one-way and two-way based on their request flow. Some products like LVS and BIG-IP handle more than one server architecture for request routing.

One-Way Architecture

ND, ONE-IP, LVS, and BIG-IP support the packet forwarding one-way architecture of the Layer-4 category. ND uses the Dynamic Weighted RR algorithm for dispatching. In this algorithm every server calculates a weighted value dynamically based on its server load. The incoming requests are distributed to servers according to this dynamic value of servers. The fault tolerance mechanism in ND consists of the cache consistency protocol that runs along with the primary dispatcher and acts as a backup mechanism if heartbeat messages from the primary dispatcher stop [SGR00]. ND is also effective for TCP gateway two-way Layer-7 architecture.

ONE-IP supports routing-based and broadcast-based dispatching algorithms for distributing the requests to servers. In the routing-based dispatching policy a dispatcher is responsible in the selection of servers based on a hash function, while in the broadcast-based policy the Web server systems handle distribution of incoming requests directly by local filtering techniques. A hash table is maintained to monitor the updates of server information. These products use a Client Partition dispatching policy for their server selection.

LVS can be configured to support one-way architectures like packet tunneling and packet forwarding along with a packet double rewriting two-way architecture.

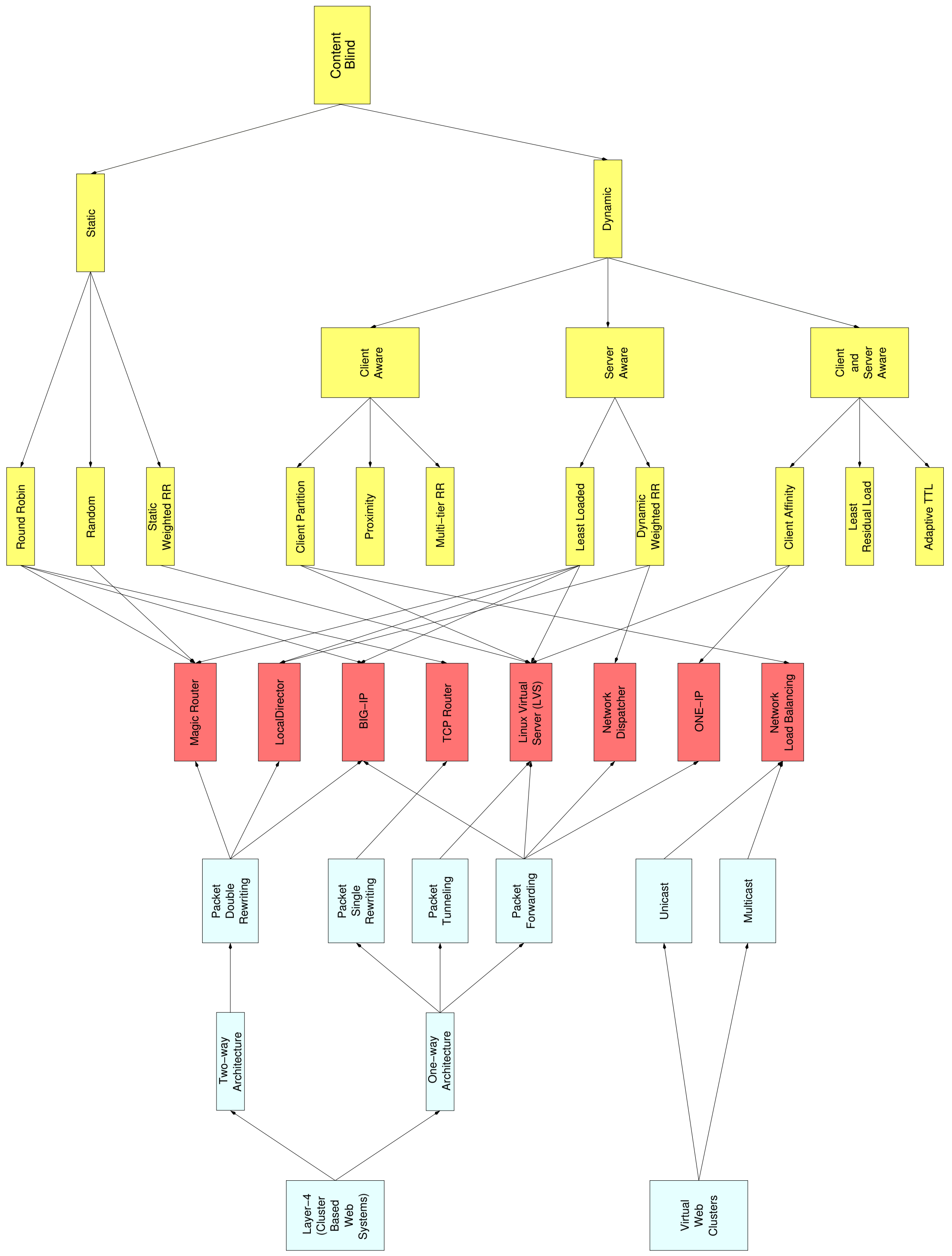

Figure 3.1: Content-Blind Classification I

				Products and Prototypes							
				Magic Router	LocalDirector	BIG-IP	TCP Router	Linux Virtual Server (LVS)	Network Dispatcher	ONE-IP	Network Load Balancing
Server Architectures	Layer-4 (Cluster Based Web Systems)	Two Way	Packet Double Rewriting	X	X	X					
		One Way	Packet Single Writing				X				
			Packet Tunneling					X			
			Packet Forwarding			X		X	X	X	
	Virtual Web Clusters		Unicast								X
			Multicast								X
Dispatching Algorithms	Static		Round Robin (RR)	X		X	X				
			Random	X							
			Static Weighted RR					X			
	Dynamic	Client Aware	Client Partition					X			X
			Proximity								
			Multi-tier RR								
		Server Aware	Least Loaded	X	X	X		X			
			Dynamic Weighted RR		X				X		
		Clent and Server Aware	Client Affinity					X		X	
			Least Residual Load								
			Adaptive TTL								

Figure 3.2: Content-Blind Classification I

This is a special product that favors both one-way and two-way architectures. The dispatching policies such as Static Weighted RR, Least Loaded, and Client Affinity can be used for the request distribution. The LVS consists of a virtual server built on a Web server system of real servers. Only the virtual server is visible to the client and the real servers are hidden to users. Suitable dispatching policies are selected based on their respective configuration.

In the TCP Router prototype a packet single rewriting architecture is employed. TCP Router concentrates on high scalability and availability. It keeps a constant check on the faulty servers so that they can be removed from the Web server system immediately.

Two-Way Architecture

LD, Magicrouter, and BIG-IP encourage the content-blind two-way architecture. LD supports RR, Weighted RR (WRR), Least Connections (LC), and Fastest Response Time (FRT) dispatching algorithms for its request distribution. They come under content-blind, server-aware, and least-loaded categories in the taxonomy of dispatching algorithms. LD has a hot-standby fail over mechanism, which is a backup dispatcher unit that works in case of a failure of the primary dispatcher [SGR00]. Failed servers in the Web server system are detected when they do not respond to the incoming request and are put in a testing phase. Once failed servers regain their power to handle active connections, they are updated in the active list of LD. LD uses reassigned and threshold commands to detect server failures. Security features of LD include SecureAccess, SecureBind, SecureBridging, and SecureIP. The *Assign*, *secure* and *static* commands of the LD product help in restricting access to servers and thus, avoid unwanted traffic [Cis00].

Magicrouter uses RR and Random dispatching policies for server selection. In addition to these policies, Magicrouter also supports an Incremental Load Policy that selects a server based on the current load of the server and the number of active connections. They have primary or secondary backup mechanisms for recovery of faulty dispatchers. Servers are checked with ARP queries periodically and unproductive servers are detected [SGR00].

BIG-IP supports both packet double rewriting of two-way and packet forwarding of one-way architectures. The requests are routed by RR and Least-Loaded dispatching algorithms. These BIG-IP products are also capable of assisting Layer-7 server architectures.

3.2.2 Content-Aware Layer-7 Products and Prototypes

Layer-7 products (Figures 3.3 and 3.4) work at the application level and can be grouped based on their architectures. ScalaServer and ClubWeb handle TCP hand offs and CD deals with TCP connection one-way architectures. The two-way architectures involve TCP gateway and TCP splicing policies for server selection. HACC, ClubWeb, and ND use a TCP gateway policy. Request routing in these prototypes are implemented in software. Alteon Web OS, BIG-IP, ServerIron and LB use a TCP splicing method. In this section we investigate the products and prototypes that are content-aware.

One-Way Architecture

The prototypes that use the TCP hand off mechanisms are ScalaServer and ClubWeb while the Central Dispatch uses the TCP connection hop technique. ScalaServer

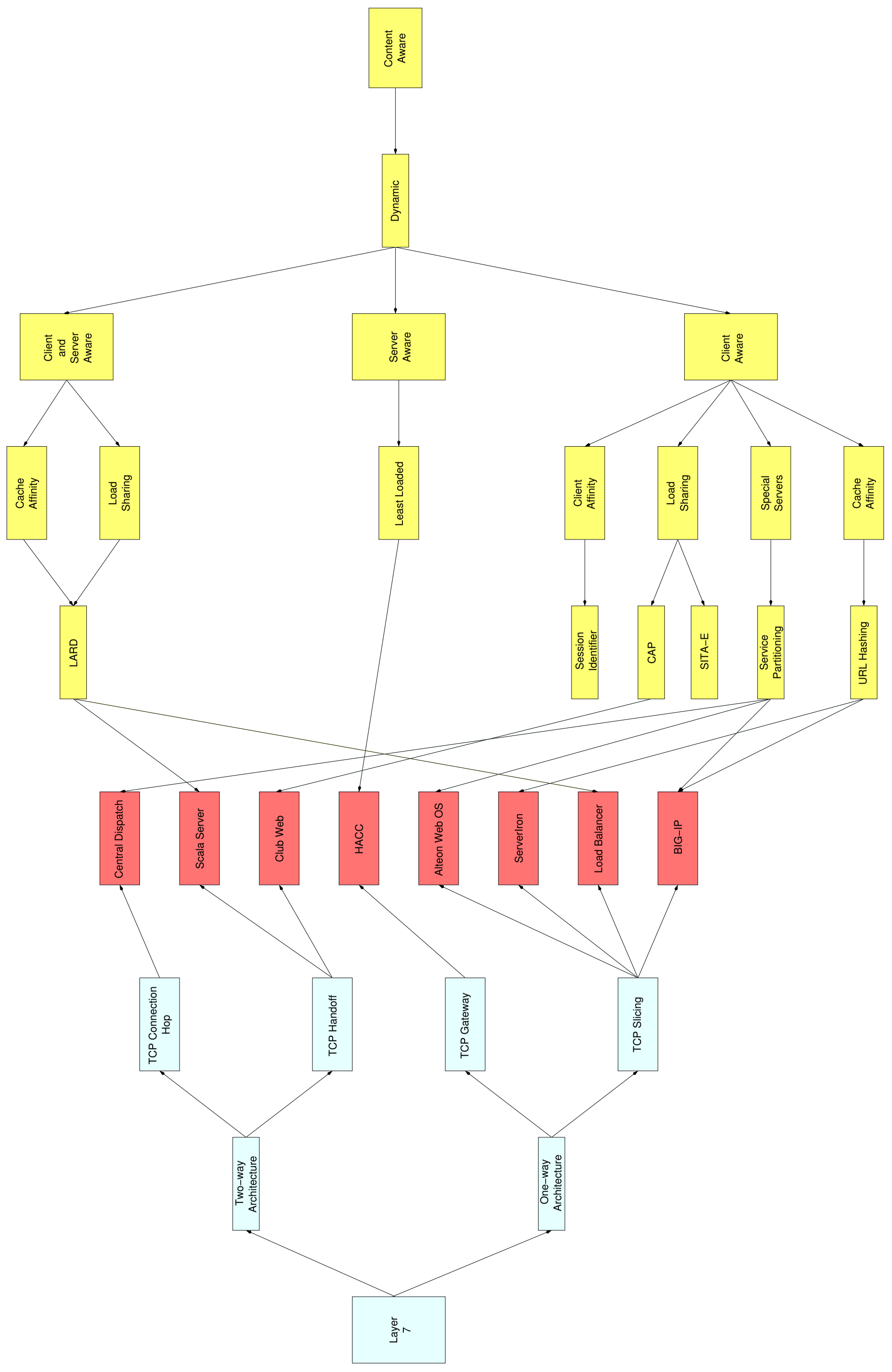

Figure 3.3: Content-Aware Classification I

	Dispatching Algorithms							Server Architectures			
	Content Aware							Cluster Based Systems			
	Dynamic										
	Client Aware					Server Aware	Client/Server Aware	Layer 7			
	Cache Affinity	Special Servers	Load Sharing		Client Affinity	Least Loaded	Local Sharing, Cache Affinity	One Way		Two Way	
Products and Prototypes	URL Hashing	Service Partitioning	SITA-E	CAP	Seesion Identifier		LARD	TCP Slicing	TCP Gateway	TCP Handoff	TCP Connection Hop
Central Dispatch		X									X
Scala Server							X			X	
Club Web				X						X	
HACC						X			X		
Alteon Web OS		X						X			
ServerIron	X							X			
Load Balancer							X	X			
BIG-IP	X	X						X			

Figure 3.4: Content-Aware Classification I

uses the LARD policy for selecting the servers from the Web server system. The FreeBSD operating system is modified to support a TCP Hand off protocol. ClubWeb prototype uses a modified Linux kernel to support this protocol. The ClubWeb uses the CAP technique for dispatching and the stress on the components is shared among Web servers.

The CD distributes HTTP requests to target servers based on client and server information. To support the TCP connection hop mechanism, the installation of a kernel module on both the dispatcher and the servers are required. The dispatcher is responsible to parse the URL for its requested content and then a target server is assigned based on the services offered by Web servers. A service partitioning algorithm can be employed for request handling.

Two-Way Architecture

HACC distributes requests based on the locality of reference found in individual Web servers. Client desired data is divided into smaller parts and spread within the servers in the Web server system. A Smart Router with LSR and HSR distributes incoming requests to the target server. The incoming requests are queued in LSR. When a HTTP request is passed to LSR, the URL is extracted and copied to HSR of the Smart router. HSR identifies the target server that can handle the request and responds to LSR with required information. LSR now establishes a connection and forwards the queued data over this connection. After the first request, subsequent requests are assigned to the same server for better locality of reference. For dynamic load balancing a performance monitoring thread is generated to collect periodic performance data from each server. This thread gathers each server's load statistics such as CPU utilization, disk activity, paging activity, and queuing requests.

In ClubWeb the dispatcher uses the Client Aware Policy (CAP) for its selection of the target server. The incoming requests can stress different Web System resources like CPU, disk, or network. A dispatcher is responsible for estimating the impact produced by the requests on Web System resources. CAP avoids overload on each component by sharing the stress between Web servers.

Alteon Web OS is a software application that runs on the AWS family for request distribution and connection management. Service partitioning of the content-aware policy is used to select Web servers. Web servers that are responsible in processing client requests are partitioned based on the services they can handle. The requests are directed to specific servers where its respective service type is offered.

BIG-IP supports both service partitioning and URL hashing dispatching algorithms. In the URL hashing algorithm a hash function is applied to the URL and static partitioning of the files are performed. The dispatcher for request distribution uses the same hash function. BIG-IP manages partial hardware and software implementations in their front-end device.

Dispatching algorithms like RR, Least Connection, and URL hashing are assisted by ServerIron for server load balancing and request distribution. With the help of URL hashing, ServerIron investigates HTTP request information that internally guides it one of the respective Web server. Future requests that contain similar information are forwarded to the same server.

LB employs the LARD policy for dispatching its requests to the servers. In LARD the same servers that handle a particular type of requests are assigned repeatedly until a threshold is reached. If the servers reach a threshold value, then the dispatcher is responsible for distributing the load to least loaded servers.

3.2.3 Virtual Web Clusters

Network Load Balancing (NLB) product support the virtual Web cluster architecture and the content-blind approach is used for request routing. The target server is identified based on the client IP address and port. As NLB is a virtual Web cluster product, each server has a filtering mechanism to accept or reject the request. This filtering technique involves computation of a hash function on the client IP address or port number. The client partition policy is used for target server selection. NLB favors both unicast MAC address and multicast MAC address modes of virtual Web cluster architecture. This product analysis taxonomy of virtual Web clusters is shown in Figures 3.1 and 3.2.

3.2.4 Distributed Web Systems

The Web server systems in the distributed Web systems can be located globally or locally in a network. NCSA, SWEB, and Akamai are some of the products that support local distributed Web systems. DD and I2-DSI use global distributed Web system architecture for routing decisions as given in Figures 3.5 and 3.6.

In NCSA server and SWEB the DNS-RR algorithm is used for server selection. The drawback of DNS-RR is its ignorance of the server capacity and availability. DD can act as a primary DNS and can also be configured for the HTTP redirection approach. The Director Response Protocol is a simple User Datagram Protocol (UDP) based application that delivers the proximity information between Web servers and clients to DD. The dispatching algorithms used in DD are multi-tier RR and Least Residual Load. In the I2-DSI project a smart DNS uses a proximity algorithm for name-to-address resolution.

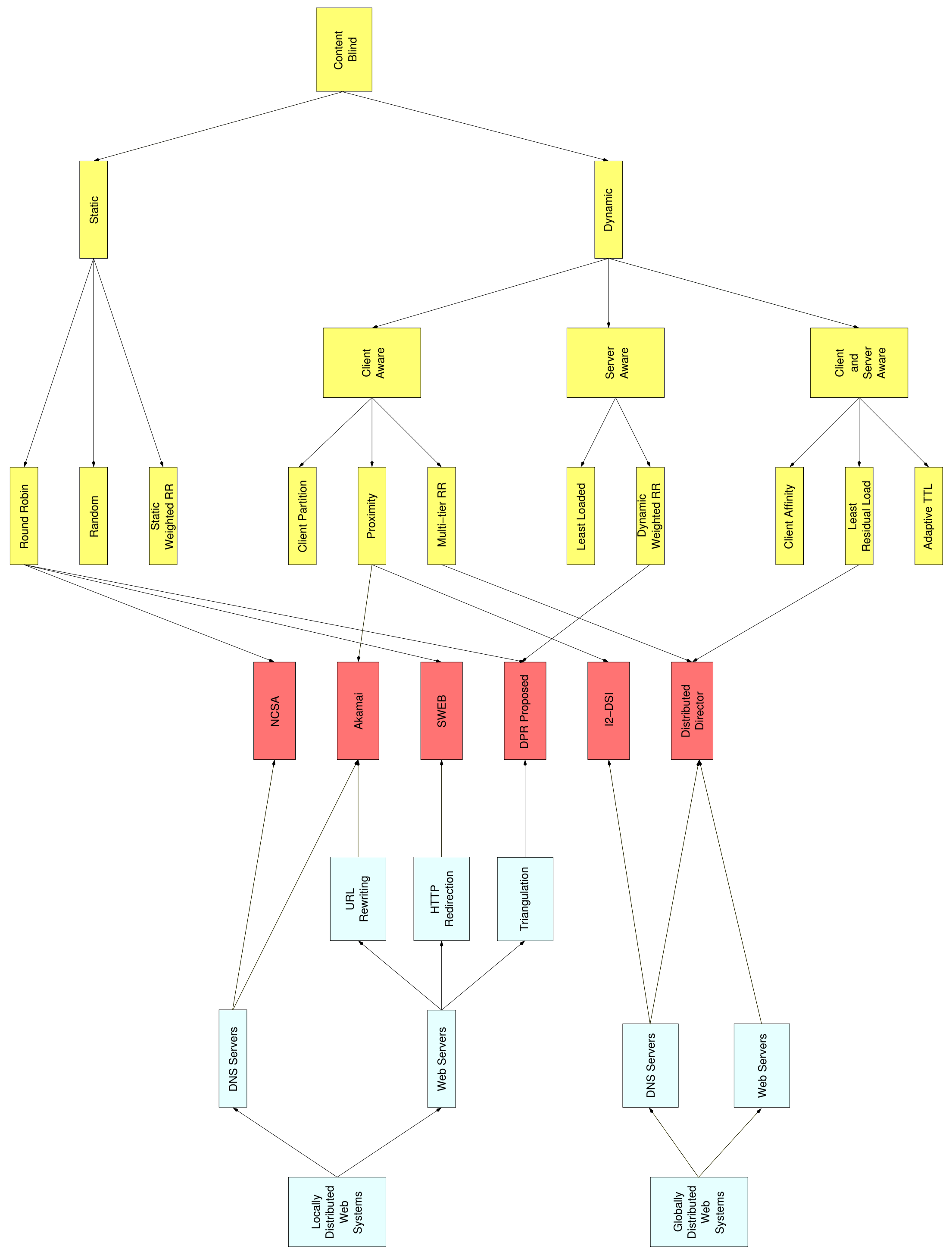

Figure 3.5: Content-Blind Classification II

				Products and Prototypes					
				NCSA	Akamai	SWEB	DPR Proposed	I2-DSI	Distributed Director
Server Architectures	Locally Distributed Web Systems		DNS Servers	X	X				
		Web Servers	URL Rewriting		X				
			HTTP Redirection			X			
			Triangulation				X		
	Globally Distributed Web Systems		DNS Servers					X	X
			Web Servers						X
Dispatching Algorithms	Content Blind	Static	Round Robin	X		X	X		
			Rando m						
			Static Weigted RR						
		Dynamic							
		Client Aware	Client Partition						
			Proximity		X			X	
			Multi-tier RR						X
		Server Aware	Least Loaded						
			Dynamic Weighted RR				X		
		Clent and Server Aware	Client Affinity						
			Least Residual Load						X
			Adaptive TTL						

Figure 3.6: Content-Blind Classification II

Akamai is configured for a DNS-based approach and URL rewriting in the server selection process. In the Akamai infrastructure a dynamic page is generated that contains the URLs of the embedded objects found in the customer's Web site. The servers close to the clients are assigned the request for processing. In the DNS-based approach, the EdgeSuite software running on Akamai servers help to serve the proximity information to DNS [Aka99].

Chapter 4

Generic Queuing Models

Nowadays, Web researchers concentrate on serving data at a faster rate to the clients and analyzing the performance of Web server systems. From the unified product-based taxonomy, we get an overview of the products and prototypes in the market along with their server architectures and dispatching algorithms. Now that the taxonomy is built, we need to focus on the performance analysis of Web server systems. In earlier research work, different Web performance models have been built to meet their specific requirements [MA02].

In this thesis, we design generic queuing models that suit products with Layer-4 one-way architectures and content-blind static algorithms like Random and RoundRobin (RR). Figure 4.1 schematically represents the product-based taxonomy with respect to the generic queuing models. The products found in the market that satisfy the Layer-4 one-way architecture and content-blind static algorithms are BIG-IP, and TCP Router.

Layer-4 one-way architectures are dispatcher-based models, which deal with

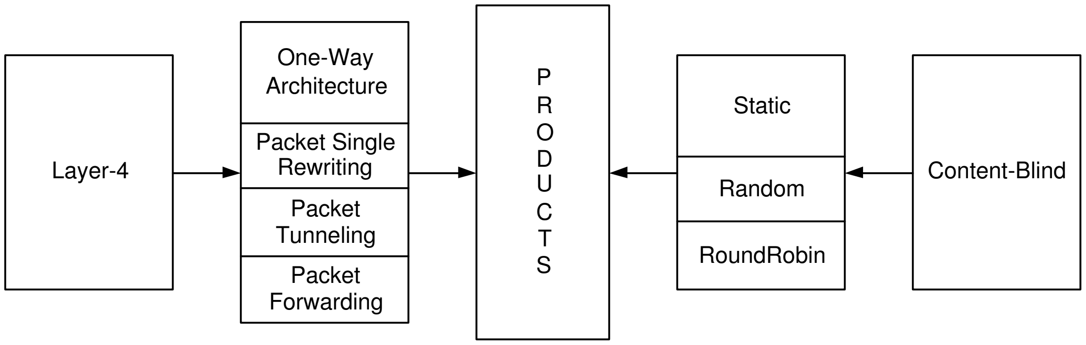

Figure 4.1: Product-Based Taxonomy with respect to Generic Queuing Models

a dispatcher to distribute the client requests to the target server. The processed requests, after being served are delivered to the client directly from the target server. The one-way architectures like packet single rewriting, packet tunneling, and packet forwarding are well suited architectures for the models used in this research. The dispatching algorithms that are taken into consideration for the selection of servers are Random and RR. The choice of Random and RR is due to the fact that most of the commercial products prefer simple algorithms for their server selection. The static algorithms, Random and RR are fast solutions as they do not rely on any current state of the system. Random and RR algorithms are implemented in the dispatcher to distribute the client requests to the target server.

The overall picture of the generic queuing models is depicted in figure 4.2. In this thesis, the models are designed with a dispatcher unit and a server unit, which are responsible for distributing and serving client requests, respectively.

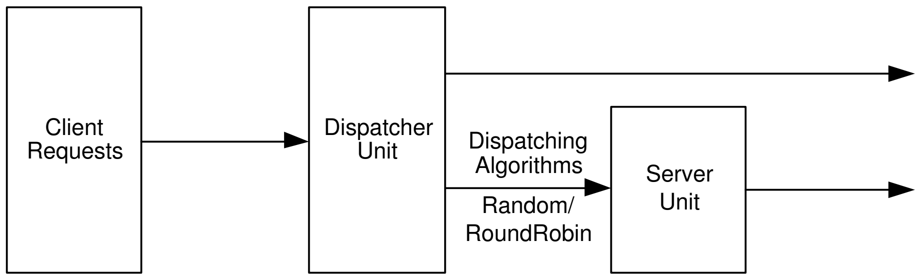

Figure 4.2: Generic Queuing Models

4.1 Model Description

In the designed generic queuing models, the two categories of requests that leave the dispatcher unit are:

- the client requests that leave the system directly from the dispatcher unit, without service, and
- the client requests that are supplied to the server unit from the dispatcher unit, for service.

For example, let us consider a client, who wants to browse a Web page. There are two possibilities with respect to the response from a server to the client, namely, a response from the server with necessary information, and no response from the server. The “no response” possibility could be attributed to busy traffic, faulty server, or server busy. In this research, we consider a case, where one of the servers is shut down or faulty and the client never gets a response from this server. So, in order to compensate this lack of response from the faulty server, generic queuing models are designed with requests that leave the dispatcher unit, without service.

In this thesis, the generic queuing models are designed with and without feedback from the server unit to the dispatcher unit. In models without feedback, all the requests that are assigned to the server unit leave the server unit directly after being serviced. While in models with feedback, some of the requests from the server unit are reprocessed by sending them back to the dispatcher unit.

In real world scenario, when a client queries information in the Internet, the client either gets a reply immediately or after some delay. One of the reasons for this delay is that the requests, which are not serviced for the first time, are reprocessed for service. In other words, the unanswered requests are fed back to the Web server system for processing again, which causes the delay. Therefore, the need for designing generic queuing models with feedback is mandatory in the Web server modeling research. In this research, we deal with four models to analyze the performance of Web server systems. The models designed are: (1) *Random Policy Model Without Feedback*, (2) *RoundRobin Policy Model Without Feedback*, (3) *Random Policy Model With Feedback*, and (4) *RoundRobin Policy Model With Feedback*.

The designed models are theoretically analyzed and simulated for evaluation by estimating their performance metrics. These models are then validated and verified to determine the closeness of theoretical and simulated results.

These models are a network of simple queuing systems, which means that the resources are interconnected. The outputs of the dispatcher unit are given as the inputs to the server unit. Queuing networks are broadly classified as open, closed, and mixed [All90, Jai91]. In open queuing networks the requests or customers enter the system from outside the system and leave the system. The closed queuing networks have no external arrivals or departures and the requests circulate within the system. Mixed networks are partially open and partially closed networks, where some requests enter or leave the system, and others circulate among their servers indefinitely.

The queuing network in these models are designed in such a way that, the requests entering the dispatcher is supplied to the next queue in the server unit. The product form network analyzes the joint probability of queue lengths in *m* queues by multiplying the individual probabilities. These product form solutions were extended by Jackson [Jac63] proving the computation validity of joint probability in any arbitrary open network of *m* queues.

Jackson's network accommodates models with single request class, unlimited total number of requests, poisson arrivals, and exponential service distributions along with load dependencies. Jackson's network is well suited for analyzing the models used in this research.

BCMP (Basket, Chandy, Muntaz, and Palacios) models [BGMT98] are more generalized versions of Jackson networks allowing different classes of customers, different service requirements, and different service distributions. There are four different types of model categories that fulfill BCMP networks, The possible queuing disciplines found in these type of BCMP models are *-/M/m (FCFS)*, *-/G/1 (PS)*, *-/G/infinity (IS)*, and *-/G/1 (LCFS:PR)*, where

- means that arrival is no more Poisson and is unknown,
M represents Exponential service,
m is the number of servers,
FCFS is first come first served,
G signifies general distribution,
PS is processor sharing,
IS is infinite server, and
LCFS: PR symbolizes last come first served with preemptive resume.

The feedback models used in this research falls within the range of BCMP networks.

4.1.1 Generic Queuing Models Without Feedback

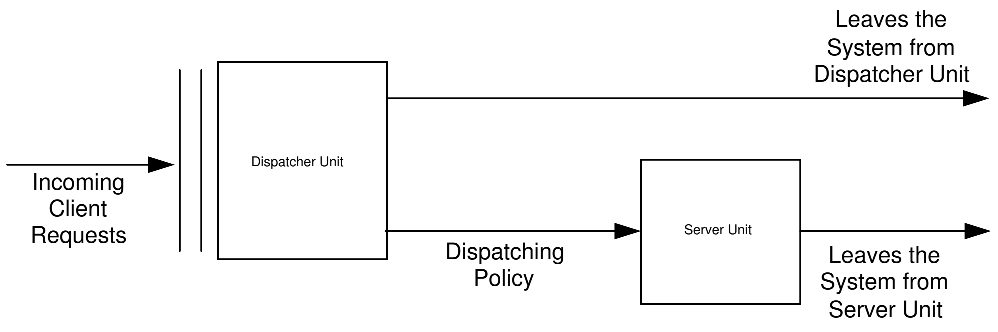
Leaves the System from Dispatcher Unit
Dispatcher Unit
Incoming Client Requests
Server Unit
Dispatching Policy
Leaves the System from Server Unit

Figure 4.3: Design—Generic Queuing Models Without Feedback

The generic queuing models without feedback consist of two units: a dispatcher and a server as shown in Figure 4.3. The dispatcher unit is responsible for accepting the incoming client requests and distributing them to the server unit for processing, based on their dispatching policies. Some of the requests that are not assigned to the server unit leave the system directly from the dispatcher unit. The dispatcher unit can have more than one dispatcher depending on the product specifications. The server unit involves one or more servers to process incoming requests. The server unit accepts requests from the dispatcher unit for service. The processed requests leave the system immediately after service in these generic queuing models without feedback. In this thesis, all the models deal with one dispatcher and four servers for processing the client requests.

Figure 4.4 represents the model description of the generic queuing model without feedback. The dispatcher unit consists of one dispatcher, which is termed as

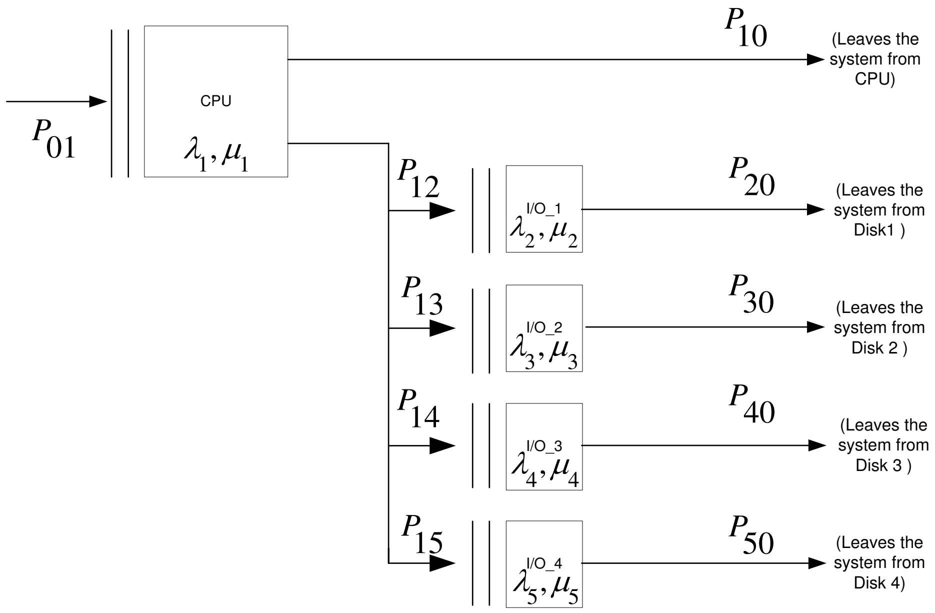

Figure 4.4: Structure—Generic Queuing Models Without Feedback

CPU. It accepts client requests and distributes them to the server unit based on selected dispatching policies. In these models, four Input/Output (I/O) servers are considered in the server unit to service requests from the dispatcher unit. As this model is implemented without feedback, requests are not reassigned to the dispatcher unit from the server unit. As shown in figure 4.4, P_{01} is the routing probability from outside the system to CPU. While P_{12}, P_{13}, P_{14}, and P_{15} are routing probabilities from CPU to I/0 servers respectively. The routing probabilities P_{10}, P_{20}, P_{30}, P_{40}, and P_{50} leave the system from CPU, and I/O servers respectively. $\lambda_1, \lambda_2, \lambda_3, \lambda_4,$ and λ_5 symbolize the throughput of CPU, I/O_1, I/O_2, I/O_3, and I/O_4 servers respectively. $\mu_1, \mu_2, \mu_3, \mu_4,$ and μ_5 represent the service rates of the CPU, and the four I/O servers. The models are named based on their dispatching policy and feedback mechanism.

Random Policy Model Without Feedback

In this model, the dispatcher unit implements a random algorithm for distributing client requests. Any server from the server unit is picked randomly for service with the help of a routing probability. As this generic model has no feedback, all the requests assigned to the server unit are assumed to be serviced.

RoundRobin Policy Model Without Feedback

The model structure, and description of a *RoundRobin Policy Model Without Feedback* are similar to that of the *Random Policy Model Without Feedback*. This model differs only in the dispatching algorithm that distributes the incoming requests to the server unit. In this model, the dispatcher is responsible in distributing the requests to the servers in a RR fashion. The incoming requests leave the system or assigned to the server unit based on a routing probability. Any request that enters the server unit is assigned and processed by the servers I/O_1, I/O_2, I/O_3, and I/O_4 respectively in a RR fashion.

4.1.2 Generic Queuing Models With Feedback

Generic queuing models with feedback are designed with a dispatcher, and a server unit along with their feedbacks. Figure 4.5 shows the design of generic queuing models with feedback. The dispatcher and server unit have the same functionality of distributing and serving incoming requests, as generic queuing model without feedback, except that some of the requests from server unit are reassigned to the dispatcher unit since it is a feedback model.

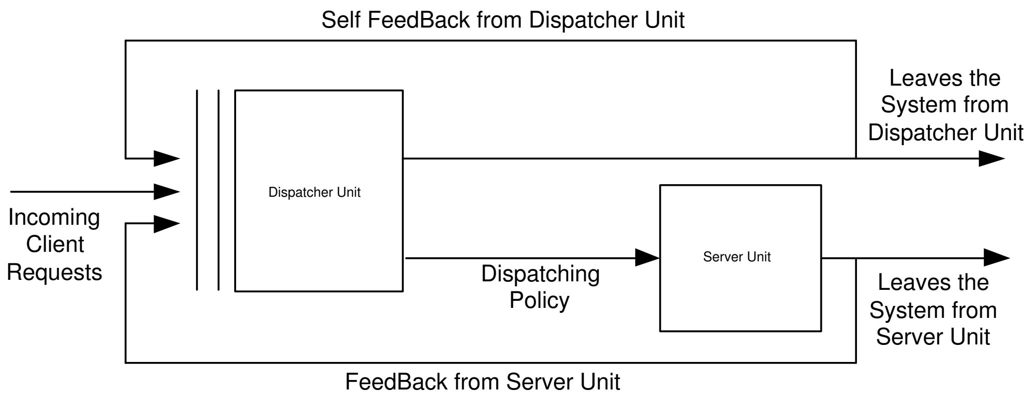
Self FeedBack from Dispatcher Unit
Leaves the System from Dispatcher Unit
Dispatcher Unit
Incoming Client Requests
Dispatching Policy
Server Unit
Leaves the System from Server Unit
FeedBack from Server Unit

Figure 4.5: Design—Generic Queuing Models With Feedback

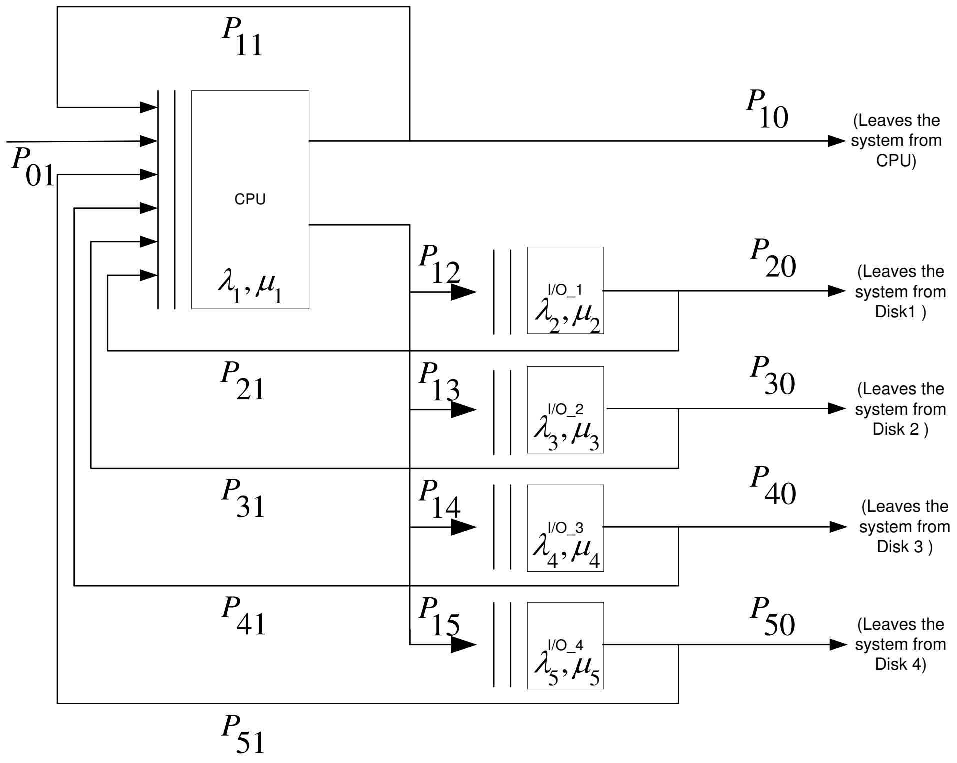

Figure 4.6: Structure—Generic Queuing Models With Feedback

The request flow of these feedback models are shown in Figure 4.6. Some of the requests that are assigned to the server unit are supplied back to the dispatcher unit for reprocessing again. The dispatcher unit is also provided with self feedback requests of the dispatcher. In generic queuing models with feedback, routing probabilities, arrival rates, and service rates are similar to the generic queuing model without feedback except for the routing probabilities P_{11}, P_{21}, P_{31}, P_{41}, and P_{51}, which are feedback probabilities from CPU, and I/O servers (I/O_1, I/O_2, I/O_3, and I/O_4) to CPU respectively.

Random Policy Model With Feedback

The random dispatching algorithm is implemented in this feedback model. The servers from the server unit are selected in a random fashion by the dispatcher unit for service. The dispatcher unit receives feedback from the server unit along with the additional self feedback. The feedback requests are again reprocessed in the dispatcher unit and serviced by the servers in the server unit randomly.

RoundRobin Policy Model With Feedback

As the model name suggests, this model is a feedback model, which implements RR algorithm for dispatching client requests. This model is more complex than other models as the requests reach the server unit from the dispatcher deterministically in a RR fashion, along with a feed back mechanism.

4.2 Model Evaluation

The model evaluation techniques widely used are analytical modeling, simulation, and measurement [Jai91]. Analytical modeling involves mathematical analysis of a model to understand the system behavior of the model. Performance metrics of the system resources are calculated analytically to evaluate their performance. Analytical modeling is cost-effective as the whole system is interpreted mathematically. Simulation is the process of reflecting the real system to predict and evaluate the performance of a system. Measurement techniques deal with experimental modeling of a system. For any research, the choice of model evaluation techniques to analyze their performance is vital. In this thesis, analytic modeling and simulation techniques are selected for evaluating these models. Measurement approach is expensive compared to other evaluation techniques, as it requires real-time devices, and tools to build the whole system.

The key elements of a queuing model are: calling population, the arrival process, service mechanism, and the number of available resources. Calling population, which can be finite or infinite, is the number of customers or client requests that arrive to the system. If the client requests that enter the queuing system are large, then the calling population is considered infinite. In this thesis, an infinite calling population is considered. Arrival process is the manner in which the client requests reach the queuing system. The service mechanism explains the service offered by the servers to process client requests. The number of available resources is the total capacity of the system resources, which are responsible for processing client requests. The available resources in all the models of this thesis are structured in such a way that each model has one dispatcher and four servers for serving the client requests.

4.2.1 Analytical Modeling

This section deals with the theoretical analysis of the models studied in this research. The basic assumptions, queuing discipline and performance metrics play a major role in the analytical modeling of a system. In the theoretical approach of a model, several simplifying assumptions are applied, as the real world system cannot be replicated exactly as such. The queuing discipline for each model varies based on their behavior. Performance metrics are calculated, and compared with the simulation results in the later sections for each model.

Assumptions

The following assumptions hold for all the models used in this thesis. The assumptions are chosen to suit the considerations of any generic model like the utilization of any device, ρ (λ/μ) is always less than 1. The mean inter-arrival time of the requests to the dispatcher unit is assumed as 1.5 time units. The mean arrival rate, λ_0, which is the inverse of mean inter-arrival time is calculated. The service rates for the dispatcher (μ_1) and I/O servers $(\mu_2, \mu_3, \mu_4, \mu_5)$ are assumed to be 1.5 requests/time units and 0.5 requests/time units respectively. The routing probability to the dispatcher unit from outside the system, P_{01}, is set as 1.

The requests that are supplied from the dispatcher unit to the server unit and the requests that leave the system from the dispatcher unit are tested for different ratios. In order to be consistent, all the models are experimented with the same percentage of requests, such that, 80% of the requests are supplied from the dispatcher unit to the server unit and the remaining 20% of the requests leave the dispatcher unit directly, without service.

In generic queuing models without feedback, 20% of the requests from the dispatcher is assumed to leave the system (P_{10} = 0.2), and 80% of the requests from the dispatcher is fed to the server unit. So, the routing probabilities from the dispatcher unit to the server unit P_{12}, P_{13}, P_{14}, and P_{15} are fixed as 0.2. While the other routing probabilities from the server unit to outside the system P_{20}, P_{30}, P_{40}, and P_{50} are set as 1, explaining the fact that all the requests assigned to the server unit are processed.

The assumptions in the generic queuing models with feedback are 10% of the requests from the dispatcher leave the system along with a 10% self feedback, and 80% of the dispatcher requests is supplied to the server unit. In the server unit 90% of the requests are serviced, and 10% of the requests are fed back to the dispatcher unit. The feedback routing probabilities P_{11}, P_{21}, P_{31}, P_{41}, and P_{51} are fixed as 0.1, and the routing probabilities ($P_{20}, P_{30}, P_{40}, P_{50}$) that leave the servers in the server unit are set to 0.9.

Queuing Discipline

A queuing discipline of a system is classified by a standard notation termed as Kendal notation [Jai91]. In Kendal notation, a queuing system is represented in the form A/S/m/B/K/SD, where

A - Arrival time distribution
S - Service time distribution

m - Number of servers
B - Number of buffers
K - population size
SD - Service discipline

The arrival and service time distributions are generally denoted by a one letter symbol like *M* for Exponential, E_k for Erlang with parameter k, *PH* for phase-type, etc. In most cases with infinite buffers and infinite population size, the Kendal notation is of the form *A/S/m*.

Performance Metrics

Performance metrics evaluate the efficiency of a system. The selection of metrics has a high impact in the analysis of a system's performance. Factors like time, rate and resource are taken into consideration, as they give a better view of the responsiveness, productivity and utilization of a system, respectively. The performance of a system can be measured by the time taken to perform the service, the rate at which the service is performed, and the resources consumed while performing the service.

In this thesis, a set of performance metrics like throughput, utilization, mean queue size, mean waiting time, and mean response time are selected for model evaluation. Throughput is the rate at which requests are processed per unit time. It is measured in requests per time units. Utilization is the traffic intensity of a resource. It is the fraction of time when a server is busy. Mean queue size is described as the number of requests waiting in the queue before they get serviced. Mean waiting time is the average time that a request waits for its service. This is the time interval between the time when a request arrives to the system till its service starts. Response time is the total time by a client request in a system and is calculated from the time a client sends a request till it receives a response.

In order to calculate these performance metrics of these models, the mean value analysis algorithm is chosen, as they are simpler and easier to derive open queuing network models [Jai91]. Theoretical analysis for each model is explained in the following sections.

Random Policy Model Without Feedback

In the *Random Policy Model Without Feedback*, the dispatcher and each server in the server unit handles the *M/M/1* queuing discipline individually, where the first *M* stands for the exponential inter arrival time; the second *M* stands for the exponential service time and the number *1* denotes the number of servers [Tri02]. Based on the earlier research efforts in this field, the choice of exponential distribution is best suited for the theoretical analysis of these model types [MA02].

Throughputs of CPU and I/O servers are calculated mathematically by mean value analysis and are drawn into equations with the help of their routing probabilities (figure 4.4). Since input rates ($n\lambda_n$) must equal to the output rates at each server, we have:

$$\lambda_1 = \lambda_0 \tag{4.1}$$

$$\lambda_2 = \lambda_1 P_{12} \tag{4.2}$$

$$\lambda_3 = \lambda_1 P_{13} \tag{4.3}$$

$$\lambda_4 = \lambda_1 P_{14} \tag{4.4}$$

$$\lambda_5 = \lambda_1 P_{15} \tag{4.5}$$

We know that from the assumptions,

λ_0 = 0.667 requests/time units,
μ_1 = 1.5 requests/time units,
μ_2 = μ_3 = μ_4 = μ_5 = 0.5 requests/time units.

By solving the Equations 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 throughputs $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$ are calculated:

λ_1 = 0.667,
λ_2 = λ_3 = λ_4 = λ_5 = 0.133,
P_{12} = P_{13} = P_{14} = P_{15} = 0.2.

The performance metrics of an *M/M/1* queuing model are calculated from the formulae given below [Jai91].

$$Utilization, \rho = \frac{\lambda}{\mu} \tag{4.6}$$

$$Mean\ number\ of\ jobs\ in\ the\ queue, E(n_q) = \frac{\rho^2}{1-\rho} \tag{4.7}$$

$$Mean\ Waiting\ Time, E(w) = \rho(\frac{1/\mu}{1-\rho}) \tag{4.8}$$

$$Mean\ Service\ Time, E(s) = \frac{1}{\mu} \tag{4.9}$$

By substituting the values of throughputs and service rates of CPU and I/O servers in the *M/M/1* queue formulae, the metric values are computed, as shown in Table 4.1.

RoundRobin Policy Model Without Feedback

In the *RoundRobin Policy Model Without Feedback* two different queuing disciplines are found at the dispatcher and server unit, respectively. The dispatcher unit

Metrics / **Devices**	**Throughput**	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**
CPU	0.667	0.445	0.356	0.533
I/O_1	0.133	0.267	0.097	0.727
I/O_2	0.133	0.267	0.097	0.727
I/O_3	0.133	0.267	0.097	0.727
I/O_4	0.133	0.267	0.097	0.727

Table 4.1: Metrics—Random Policy Model Without Feedback

handles a *M/M/1* queuing discipline similar to the *Random Policy Model Without Feedback*, with their exponential inter-arrival and service time. Using mean value analysis and *M/M/1* queue formulae discussed earlier in this section, the performance metrics of this model's dispatcher unit (CPU) is calculated (see Table 4.2).

Metrics / **Devices**	**Throughput**	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**
CPU	0.667	0.445	0.356	0.533

Table 4.2: Metrics—RoundRobin Policy Model Without Feedback—Dispatcher Unit (M/M/1)

As the server unit employs RR algorithm, the arrival behavior of requests to each server is found in a different manner than before. The new arrival in each server needs to wait in stages, as the request flows in a RR fashion in the servers. The inter-arrival time of the client requests in the servers is no longer exponential. So, the server unit is now considered to handle general independent inter-arrival queuing discipline, and exponential service time. The queuing discipline of the server unit in the *RoundRobin Policy Model Without Feedback* is now generalized as *GI/M/1*, where

GI is symbol for general independent inter-arrival time distribution
M is the symbol for exponential service time
1 denotes the number of servers

The working process of the server unit is analyzed closely to understand the nature of the queuing discipline. The behavior of the requests assigned to I/O servers gives a clue that I/0 servers in this model employ the Erlang distribution, as the requests navigate similar to stages in each server. The arrival of requests in each server of the server unit, is such that every request waits for four stages till it gets its next request. The queuing discipline of the server unit is concluded as Er_k */ M / 1*, where *k* is the number of stages or hosts [SHB00].

By using the mean value analysis, the throughputs and utilizations of I/O servers are calculated as shown in the *Random Policy Model Without Feedback*. In order to calculate the metrics, mean queue size and mean waiting time for the Er_k */ M / 1* server unit model, the *GI/M/1* Queue Formulae should be transformed and derived, such that it suits Er_k */ M / 1* queuing discipline [All90]. In *GI/M/1*, the steady state probability that an arriving request will find the system empty, π_0, is the unique solution of the equation,

$$1 - \pi_0 \quad = \quad A^*[\mu\pi_0] \; : \; for\ all\ 0 < \pi_0 < 1 \tag{4.10}$$

where,

μ is the service rate
π_n is the probability that an arriving request will find ‘n’ requests
$A^*[\mu\pi_0]$ is the Laplace Stiltjes transform of the inter-arrival time, τ

In this model, as each server in the server unit needs to wait for four stages (other servers in the server unit, along with one leaving the system) before its next arrival, the value of k is specified as 4.

The Laplace Stiltjes transform of $A^*[\theta]$ for Er_k is

$$(\frac{k\lambda_i}{k\lambda_i+\theta})^k \tag{4.11}$$

$$(1-\pi_0) = (\frac{k\lambda_i}{k\lambda_i+\theta})^k \tag{4.12}$$

$$(1-\pi_0)((k\lambda_i+\mu_i\pi_0)^k) = (k\lambda_i)^k \tag{4.13}$$

where,

k = 4
θ = $\mu_i\pi_0$
μ_i = 0.5 requests/time units, i = 2, 3, 4, and 5
λ_i = 0.133 net requests/time units, i = 2, 3, 4, and 5

Now π_0 is calculated by substituting the k, μ_i, and λ_i values in the above equation:

π_0 = 0.916

From π_0, mean queue size and mean waiting time in the queue are calculated analytically, with the help of the formulae given below [All90] and are tabulated in Table 4.3.

$$Utilization, \rho = \frac{\lambda}{\mu} \tag{4.14}$$

$$Mean\ Service\ Time, E(s) = \frac{1}{\mu} \tag{4.15}$$

$$Mean\ number\ of\ jobs\ in\ the\ queue, E(n_q) = \frac{\rho(1-\pi_0)}{\pi_0} \tag{4.16}$$

$$Mean\ Waiting\ Time, E(w) = \frac{E(s)(1-\pi_0)}{\pi_0} \tag{4.17}$$

Metrics / **Devices**	**Throughput**	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**
I/O_1	0.133	0.267	0.024	0.183
I/O_2	0.133	0.267	0.024	0.183
I/O_3	0.133	0.267	0.024	0.183
I/O_4	0.133	0.267	0.024	0.183

Table 4.3: Metrics—RoundRobin Policy Model Without Feedback—Server Unit (Er_k/M/1)

Random Policy Model With Feedback

In a *Random Policy Model With Feedback* some of the requests from the server unit are fed back to the dispatcher unit. Although this model has a feedback, follows the *M/M/1* queuing discipline model still describes the marginal distribution of performance results for both the dispatcher and server units. As discussed earlier in this section, using mean value analysis, throughputs of CPU and I/O servers are measured mathematically based on the assumptions of the feedback and routing probabilities.

$$\lambda_1 = \lambda_0 P_{01} + \lambda_1 P_{11} + \lambda_2 P_{21} + \lambda_3 P_{31} + \lambda_4 P_{41} + \lambda_5 P_{51} \tag{4.18}$$

$$\lambda_2 = \lambda_1 P_{12} \tag{4.19}$$

$$\lambda_3 = \lambda_1 P_{13} \tag{4.20}$$

$$\lambda_4 = \lambda_1 P_{14} \tag{4.21}$$

$$\lambda_5 = \lambda_1 P_{15} \tag{4.22}$$

By solving these equations throughputs λ_1, λ_2, λ_3, λ_4 and λ_5 are calculated:

$\lambda_1 = 0.813$,
$\lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = 0.163$.

Metrics	**Throughput**	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**
Devices				
CPU	0.813	0.542	0.641	0.789
I/O_1	0.163	0.325	0.157	0.964
I/O_2	0.163	0.325	0.157	0.964
I/O_3	0.163	0.325	0.157	0.964
I/O_4	0.163	0.325	0.157	0.964

Table 4.4: Metrics—Random Policy Model With Feedback

Table 4.4 shows the performance metrics of the *Random Policy Model With Feedback* calculated using the *M/M/1* queue formulae.

RoundRobin Policy Model With Feedback

The feedback arrivals along with RR fashion distribution to the servers make the *RoundRobin Policy Model With Feedback* a very interesting and complex model. The dispatcher unit of the *RoundRobin Policy Model With Feedback* manages the *M/M/1* queuing discipline. The performance metrics of the dispatcher unit are calculated from the formulae specified in previous section (Table 4.5). In the server

Metrics	**Throughput**	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**
Devices				
CPU	0.813	0.542	0.641	0.789

Table 4.5: Metrics—RoundRobin Policy Model With Feedback—Dispatcher Unit (M/M/1)

unit, the arrivals in each server of this model are Er_k with a feedback. For the feedback models, the added effect caused by its feedbacks (self-feedback, and the feedback from the server unit supplied to the dispatcher unit) reflects the performance of the system to a larger extent. In this model, 10% of the requests from server unit is fed back to the dispatcher unit along with 10% self-feedback of the dispatcher unit. With the mean value analysis specified in the previous section and using the procedure to calculate the π_0 value of the Er_k */ M / 1* model (as specified in the *RoundRobin Policy Model Without Feedback* section), the performance metrics of the server unit is theoretically analyzed.

The steps involved in determining the performance metrics of the server unit model are,

- Compute the throughput and utilization using mean value analysis

$$\lambda_1 = \lambda_0 P_{01} + \lambda_1 P_{11} + \lambda_2 P_{21} + \lambda_3 P_{31} + \lambda_4 P_{41} + \lambda_5 P_{51} \quad (4.23)$$

$$\lambda_2 = \lambda_1 P_{12} \quad (4.24)$$

$$\lambda_3 = \lambda_1 P_{13} \quad (4.25)$$

$$\lambda_4 = \lambda_1 P_{14} \quad (4.26)$$

$$\lambda_5 = \lambda_1 P_{15} \quad (4.27)$$

  By solving these equations throughputs $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ and λ_5 are calculated:

  λ_1 = 0.813,
  λ_2 = λ_3 = λ_4 = λ_5 = 0.163.

- Compute the π_0 value with the help of the Laplace Stiltjes transform of $A^*[\theta]$ for Er_k. Substituting the values,

k = 4
μ_i = 0.5 requests/time units, i = 2, 3, 4, and 5
λ_i = 0.163 requests/time units, i = 2, 3, 4, and 5

in Equations 4.11–4. 13 (as shown above) we get,

π_0 = 0.871

- Compute the performance metrics with the help of the formulae

  From π_0, mean queue size and mean waiting time in the queue is calculated analytically, with the help of the formulae shown in previous section (Equations 4.14–4.17) [All90].

Metrics	**Throughput**	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**
Devices				
I/O_1	0.163	0.325	0.048	0.297
I/O_2	0.163	0.325	0.048	0.297
I/O_3	0.163	0.325	0.048	0.297
I/O_4	0.163	0.325	0.048	0.297

Table 4.6: Metrics—RoundRobin Policy Model With Feedback—Server Unit (Er_k/M/1)

Table 4.6 shows the performance metrics of the *RoundRobin Policy Model With Feedback.*

Derivation—Total Response Time

Little's law states that the mean number of requests in the system is the product of the arrival rate and the mean time each request spends in the system[Jai91]. We use little's law to derive the total response time formula, as shown below.

$$L = \lambda \times E(R) \tag{4.28}$$

where

λ = 0.667 requests/time units
L = $\sum_{i=1}^{5} (E(N_q)_i + \rho_i)$
E(R) - Total response time

By substituting the values of all the devices in the system, we calculate the total response time in each model (Appendix A).

Testing with More Feedbacks

In order to analyze the consistency of this model, the feedbacks from the server unit to the dispatcher unit are varied and tested. Feedback from each server to the dispatcher unit is increased from 10% to 50%, and the results are analyzed. Mean value analysis algorithm and the steps involved in the theoretical calculation of dispatcher unit (*M/M/1*) and server unit (Er_k*/M/ 1*) explained earlier in this section are followed to determine the performance metric values. We know the service rates in the server unit and dispatcher unit, as shown below.

μ_1 = 1.5 requests/time units
μ_2 = μ_3 = μ_4 = μ_5 = 0.5 requests/time units.

Substituting the 50% feedback routing probability in the system,

P_{11} = P_{21} = P_{31} = P_{41} = P_{51} = 0.5,
P_{10} = P_{20} = P_{30} = P_{40} = P_{50} = 0.5.

we calculate the throughput and utilization using mean value analysis for each device.

λ_1 = 1.334
$\lambda_2 = \lambda_3 = \lambda_4 = \lambda_5$ = 0.267.

The utilization of any device, ρ_i is calculated from the formula,

$$\rho_i = (\frac{\lambda_i}{\mu_i}) \tag{4.29}$$

where

i = 1,2,3,4 and 5.

Therefore,

ρ_1 = 0.889
$\rho_2 = \rho_3 = \rho_4 = \rho_5$ = 0.534

In the server unit, to calculate the performance metric values, π_0 value is calculated with the help of the equation below.

$$(1 - \pi_0) = (\frac{k\lambda}{k\lambda + \theta})^k \tag{4.30}$$

$$(1 - \pi_0)((k\lambda + \mu\pi_0)^k) = (k\lambda)^k \tag{4.31}$$

where

k = 4
μ = 0.5 requests/time units
λ = 0.267 requests/time units
$\theta = \mu\pi_0$

Now π_0 is calculated by substituting the k, μ, and λ values in the above equation:

π_0 = 0.659.

The performance metric values for the dispatcher unit (*M/M/1*) and the server unit (Er_k*/M/ 1*) are computed, as shown in Table 4.7.

Metrics / **Devices**	**Throughput**	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**
CPU	1.334	0.889	7.120	5.339
I/O_1	0.267	0.534	0.276	1.035
I/O_ 2	0.267	0.534	0.276	1.035
I/O_3	0.267	0.534	0.276	1.035
I/O_4	0.267	0.534	0.276	1.035

Table 4.7: Metrics—RoundRobin Policy Model With Extra Feedback

4.2.2 Simulation

Models are simulated in several ways depending on the convenience, time, and availability of resources. General-purpose programming languages like Java and C are used simulate models. There are also specific simulation languages and simulation packages available for easier implementation of models. Simulation packages are more comfortable than simulation languages and general-purpose languages, as these packages have a library of data structures, built-in routines and algorithms that are essential for simulation. Simulation packages are time saving and easier to work with, which makes it a popular choice for simulation modeling. Now, the selection of a proper simulation package in developing a simulation model is necessary based on user criteria and specifications [LK91].

In this research, SSJ, a simulation package is used for implementing these generic queuing models [LMV02]. SSJ consists of a collection of classes implemented in Java programming language, which provides extensive facilities for simulation programming. It has predefined classes that are responsible for generating random numbers for various distributions, collecting statistics, managing a simulation clock, and a list of future events, synchronizing the interaction between simulated concurrent processes, etc. SSJ permits event view, process view, continuous, and mixed simulations. All these advantages made SSJ to be the choice for simulating models used in this research.

The models, *Random Policy Model Without Feedback*, *RoundRobin Policy Model Without Feedback*, *Random Policy Model With Feedback*, and *RoundRobin Policy Model With Feedback* are simulated as per the assumptions and specifications mentioned in theoretical analysis. A calling population of 35,000 requests[1] is simulated, and fed into the dispatching unit. The calling population is considered infinite. The generated requests are supplied to the dispatcher unit. The dispatcher unit is considered with one dispatcher to accept the incoming requests and distributed to the server unit with four servers based on their respective dispatching policy.

In the *Random Policy Model Without Feedback*, the dispatcher unit is implemented in such a way that, 80% of the incoming requests reach the server unit with each server accepting 20% of requests randomly. The left over 20% of the incoming requests leave the system directly from the dispatcher unit. The request flow in the simulated models replicates the routing probability found in the theoretical analysis of the models. As this research model has no feedback, all the requests that enter server unit leave the system directly after service (Figure 4.4).

The *RoundRobin Policy Model Without Feedback* is implemented with the dispatcher unit distributing requests in a RR fashion to the server unit. Each server

[1]This count of 35,000 is arbitrary and the generation of requests is done with the help of random number generators of SSJ using exponential arrival times.

in the server unit receives requests one after the other in an orderly fashion. This queuing model is free from any feedback (Figure 4.4).

The implementation of the *Random Policy Model With Feedback* is a little different from the models without feedback, as it involves a feedback from server unit to dispatcher unit and also a self feedback. 10% of the requests from the dispatcher leaves the system directly and 10% is fed back (self-feedback) to the dispatcher unit. From each server in the server unit, 90% of the requests leave the system and 10% of the requests from the server unit is fed back to the dispatcher unit (Figure 4.6).

The *RoundRobin Policy Model With Feedback* is implemented with a feedback mechanism, where the incoming requests to the server unit flows in a RR fashion one after the other to each server (Figure 4.6). All the assumptions are replicated in simulation, as depicted in the theoretical analysis of these models. The simulated results for all these models are tabulated in Table 4.8, as shown below.

Metrics	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**	**Mean Service Time**
Devices				
CPU	0.440	0.342	0.515	0.663
I/O_1	0.273	0.097	0.730	2.048
I/O_2	0.265	0.102	0.744	1.943
I/O_3	0.263	0.100	0.750	1.981
I/O_4	0.266	0.090	0.680	2.008

(a) Metrics—Random Policy Model Without Feedback

Metrics	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**	**Mean Service Time**
Devices				
CPU	0.445	0.347	0.516	0.662
I/O_1	0.273	0.019	0.142	2.035
I/O_2	0.275	0.024	0.182	2.051
I/O_3	0.275	0.028	0.205	2.047
I/O_4	0.280	0.026	0.190	2.082

(b) Metrics—RoundRobin Policy Model Without Feedback

Metrics	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**	**Mean Service Time**
Devices				
CPU	0.541	0.638	0.787	0.668
I/O_1	0.320	0.158	0.982	1.990
I/O_2	0.316	0.139	0.864	1.964
I/O_3	0.330	0.165	1.001	2.006
I/O_4	0.327	0.154	0.937	1.996

(c) Metrics—Random Policy Model With Feedback

Metrics	**Utilization**	**Mean Queue Size**	**Mean Waiting Time**	**Mean Service Time**
Devices				
CPU	0.542	0.635	0.781	0.666
I/O_1	0.323	0.048	0.293	1.978
I/O_2	0.327	0.046	0.283	2.001
I/O_3	0.321	0.047	0.289	1.964
I/O_4	0.332	0.051	0.310	2.037

(d) Metrics—RoundRobin Policy Model With Feedback

Table 4.8: Simulated Outputs—Performance Metrics

4.3 Model Validation

The key aspect of modeling analysis lies in validating the designed model. Model Validation is a process of determining whether a model is built reasonably [Jai91]. Assumptions, input parameter values, distributions, output values and conclusions of a model are considered for model validation. There are techniques like expert intuition, real system measurements, and theoretical results to perform validity tests of a model. Expert intuition is brainstorming between people of related areas to obtain expert advice from their analysis. Measuring real system parameters is a more preferable model validation technique. But the fact is that it is not feasible, and too expensive to incorporate a real system practically. In this thesis, theoretical analysis is selected and compared with simulation outputs for model validation. If the metric values of the theoretical results fall within the calculated confidence interval of the simulated results, the closeness of a model is determined. This comparative analysis of theoretical modeling and simulation ensures that both simulation and theoretical analysis are correct though no assessment of realism. In this thesis, model validation is done for *Random Policy Model Without Feedback*, *RoundRobin Policy Model Without Feedback*, *Random Policy Model With Feedback*, and *RoundRobin Policy Model With Feedback.*

4.3.1 Comparison of Analytical and Simulation Results

The main concern of simulation modeling is how well it is implemented when compared with their respective theoretical models. With the same assumptions for both simulation and theoretical models, we calculate the performance metrics and analyze the closeness of the results. Tables 4.9, 4.10, 4.11, and 4.12 help to visualize the

Devices	**Metrics**	**Simulated Output**	**Theoretical Value**
CPU	Utilization	0.440	0.445
	E(N_q)	0.342	0.356
	E(W)	0.515	0.533
	E(S)	0.663	0.667
I/O_1	Utilization	0.273	0.267
	E(N_q)	0.097	0.097
	E(W)	0.730	0.727
	E(S)	2.048	2.000
I/O_2	Utilization	0.265	0.267
	E(N_q)	0.102	0.097
	E(W)	0.744	0.727
	E(S)	1.943	2.000
I/O_3	Utilization	0.263	0.267
	E(N_q)	0.100	0.097
	E(W)	0.750	0.727
	E(S)	1.981	2.000
I/O_4	Utilization	0.266	0.267
	E(N_q)	0.090	0.097
	E(W)	0.680	0.727
	E(S)	2.008	2.000

Table 4.9: Comparative Analysis—Random Policy Model Without Feedback

Devices	**Metrics**	**Simulated Output**	**Theoretical Value**
CPU	Utilization	0.445	0.445
	E(N_q)	0.347	0.356
	E(W)	0.516	0.533
	E(S)	0.662	0.667
I/O_1	Utilization	0.273	0.267
	E(N_q)	0.019	0.024
	E(W)	0.142	0.183
	E(S)	2.035	2.000
I/O_2	Utilization	0.275	0.267
	E(N_q)	0.024	0.024
	E(W)	0.182	0.183
	E(S)	2.051	2.000
I/O_3	Utilization	0.275	0.267
	E(N_q)	0.028	0.024
	E(W)	0.205	0.183
	E(S)	2.047	2.000
I/O_4	Utilization	0.280	0.267
	E(N_q)	0.026	0.024
	E(W)	0.190	0.183
	E(S)	2.082	2.000

Table 4.10: Comparative Analysis—RoundRobin Policy Model Without Feedback

Devices	**Metrics**	**Simulated Output**	**Theoretical Value**
CPU	Utilization	0.541	0.542
	E(N_q)	0.638	0.641
	E(W)	0.787	0.789
	E(S)	0.668	0.667
I/O_1	Utilization	0.320	0.325
	E(N_q)	0.158	0.157
	E(W)	0.982	0.964
	E(S)	1.990	2.000
I/O_2	Utilization	0.316	0.325
	E(N_q)	0.139	0.157
	E(W)	0.864	0.964
	E(S)	1.964	2.000
I/O_3	Utilization	0.330	0.325
	E(N_q)	0.165	0.157
	E(W)	1.001	0.964
	E(S)	2.006	2.000
I/O_4	Utilization	0.327	0.325
	E(N_q)	0.154	0.157
	E(W)	0.937	0.964
	E(S)	1.996	2.000

Table 4.11: Comparative Analysis—Random Policy Model With Feedback

Devices	**Metrics**	**Simulated Output**	**Theoretical Value**
CPU	Utilization	0.542	0.542
	$E(N_q)$	0.635	0.641
	E(W)	0.781	0.789
	E(S)	0.666	0.667
I/O_1	Utilization	0.323	0.325
	$E(N_q)$	0.048	0.048
	E(W)	0.293	0.297
	E(S)	1.978	2.000
I/O_2	Utilization	0.327	0.325
	$E(N_q)$	0.046	0.048
	E(W)	0.283	0.297
	E(S)	2.001	2.000
I/O_3	Utilization	0.321	0.325
	$E(N_q)$	0.047	0.048
	E(W)	0.289	0.297
	E(S)	1.964	2.000
I/O_4	Utilization	0.332	0.325
	$E(N_q)$	0.051	0.048
	E(W)	0.310	0.297
	E(S)	2.037	2.000

Table 4.12: Comparative Analysis—RoundRobin Policy Model With Feedback

theoretical and simulated performance metric values of the *Random Policy Model Without Feedback*, *RoundRobin Policy Model Without Feedback*, *Random Policy Model With Feedback*, and *RoundRobin Policy Model With Feedback* respectively.

4.3.2 Closeness

Although the theoretical and simulated results are closer, we are not too sure of how close these results are needed to be validated as an acceptable model. In order to determine the closeness of the models, the confidence interval for performance metrics like mean queue size, mean waiting time, and total response time by replication technique is calculated [Mac87, LK91]. In the replication technique the results are analyzed by simulating the whole model for a fixed number of turns and by calculating the overall mean.

4.3.3 Confidence Interval (CI)

A confidence Interval involves the estimation of the mean and standard deviation for the performance metrics. In this research a 95% CI of the results is considered. It is calculated by the formula,

$$CI = \bar{X} \pm 2 * \frac{\sigma}{\sqrt{n}} \tag{4.32}$$

where,

$\bar{X}$ = Sample Mean
σ = Standard deviation
n = Number of turns

A sample mean is simulated by estimating the averages of the populated mean of each metric. In this thesis the number of replications is arbitrarily chosen as 100, 200, and 500 to obtain the CI of the metrics mean.

If the theoretical values of the selected metric lies between the calculated CI (simulation) of that metric, then one can confirm the closeness of the simulated and theoretical results. This procedure of calculating the closeness of the simulated and theoretical results holds for all the models in this thesis. The calling population of 45,000 is simulated and supplied to the dispatcher unit. In these models, the sample mean, and standard deviation for mean queue size, mean waiting time and total response time of the dispatcher and I/0 servers are simulated with 100, 200, and 500 replications. The results for each metric of the models are tabulated in tables A.1 to A.12 in appendix A.

From these tables it is clear that the theoretical mean queue size, mean waiting time, and mean response time metric values of the CPU and I/O servers in each model fall in between the calculated CI of the simulated results. This comparison and the closeness of the results help us to validate the simulation and theoretical models that are designed in this research.

The theoretical results for the *RoundRobin Policy Model With Extra Feedback* are compared with its simulated outputs and the CI is calculated for 100 replications. The values are tabulated in the appendix A (Table A.13). Interestingly, the theoretical results fall within the range of the simulated outputs proving the consistency of the model, even when increased in feedback.

4.4 Steady State Performance

A steady state performance of a model is analyzed with the help of a transient removal method. In the transient removal approach, the simulation models are studied by removing the initial part of the simulation and inspecting the model with other parameters like heavy load to determine the steadiness. The heuristics for analyzing the steady state performance of a model through transient removal are long runs, proper initialization, truncation, initial data deletion, moving average of independent replications, and batch means. In this thesis the models are simulated for long runs and their consistencies are determined [Jai91].

4.4.1 Long runs (LR)

Models are simulated for long runs and the outputs are analyzed by varying the input parameters. In this thesis, the input requests supplied to the dispatcher unit is varied from 10,000 to 500,000, and outputs are investigated in each model. The CI for the designed models, *Random Policy Model Without Feedback (RMWF)*, *RoundRobin Policy Model Without Feedback (RRMWF)*, *Random Policy Model With Feedback (RMF)*, and *RoundRobin Policy Model With Feedback (RRMF)* is evaluated theoretically and simulated for 100, 200 and 500 replications.

The output graphs (Figures 4.7 - 4.14) help us to formulate an interesting conclusion about the behavior of these models when they are supplied with a wider range of input requests. From the output graphs we also confirm that the CI in these models converges to a smaller range for higher value of input requests. This gives us a clue that the servers in these models attain a saturation point with the increase in the number of requests. The values for these graphs are tabulated in the appendix as B–E. Accuracy of any model can never be 100%. The calculated 95% CI of the simulated results fall in the range of the theoretical results ensuring the model consistency.

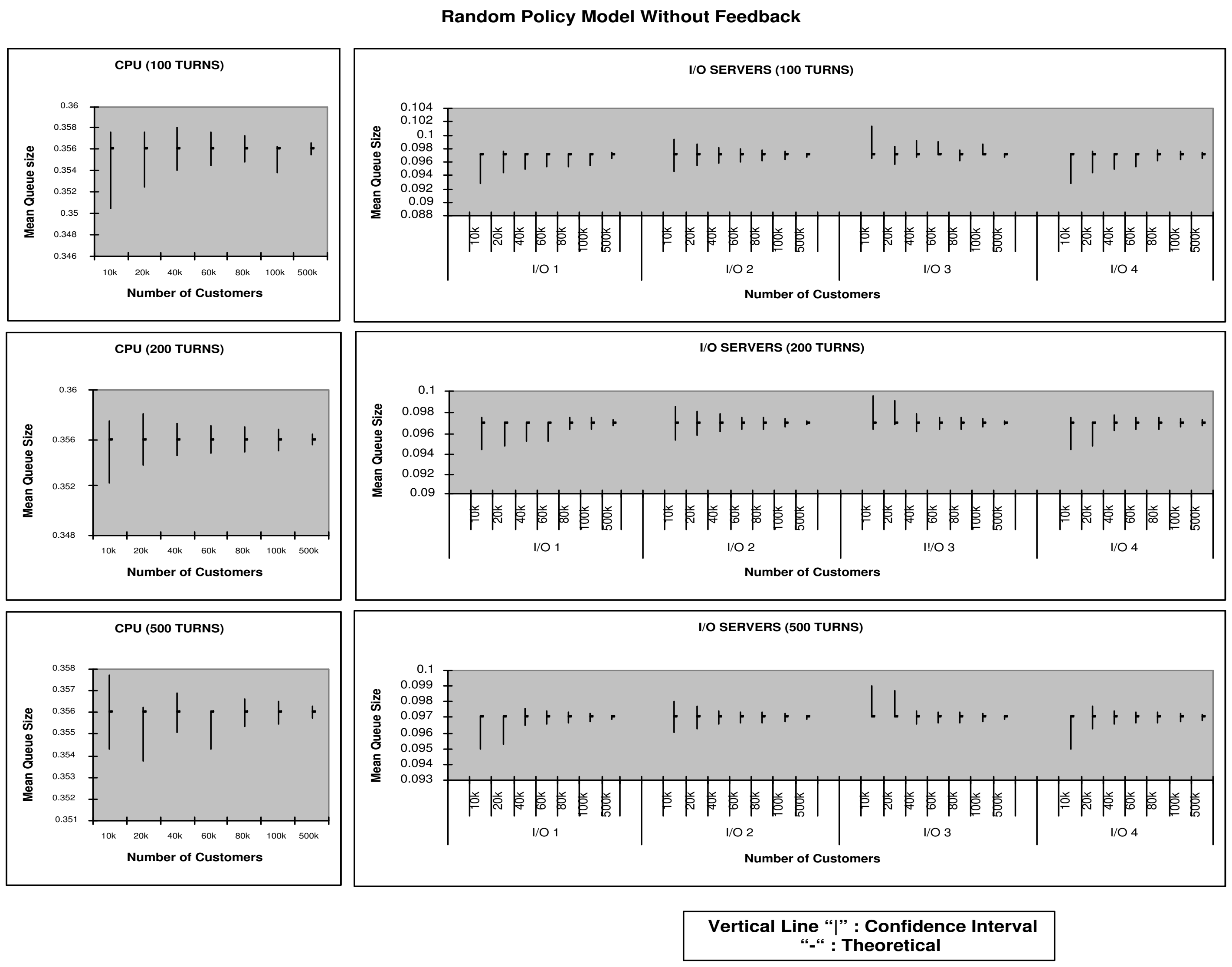

Figure 4.7: LR—Mean Queue Size—RMWF (n = 100, 200 and 500)

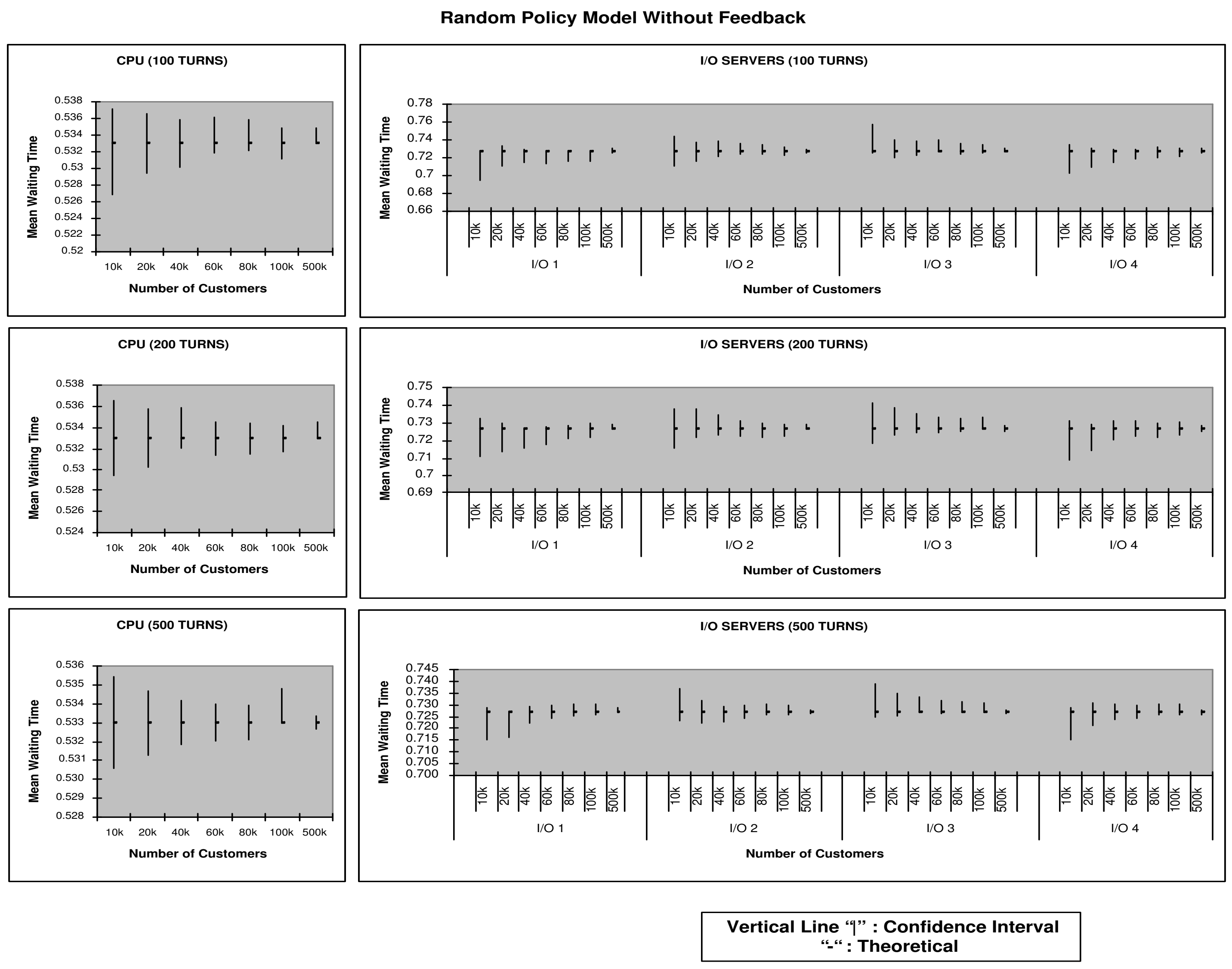

Figure 4.8: LR—Mean Waiting Time—RMWF (n = 100, 200 and 500)

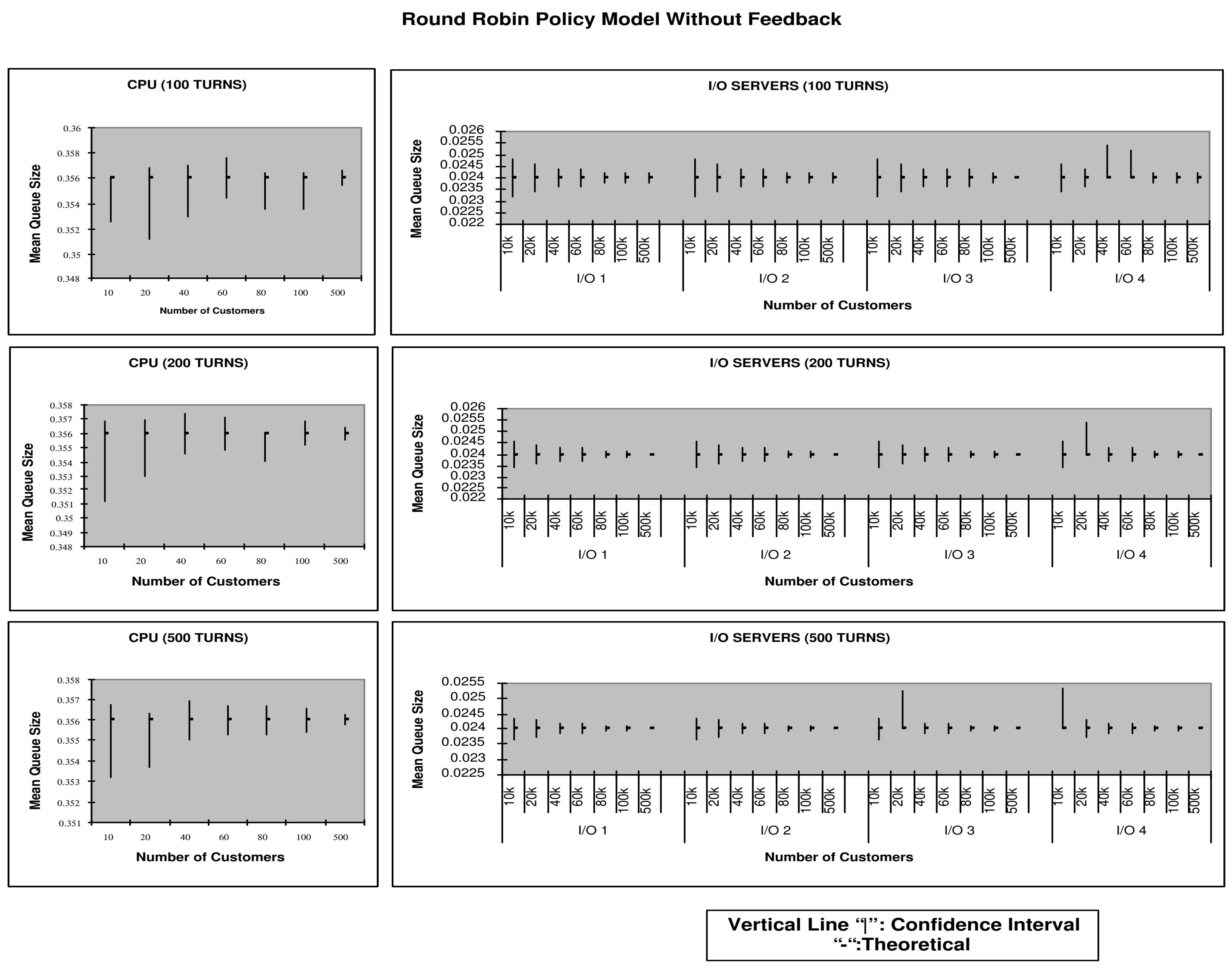

Figure 4.9: LR—Mean Queue Size—RRMWF (n = 100, 200 and 500)

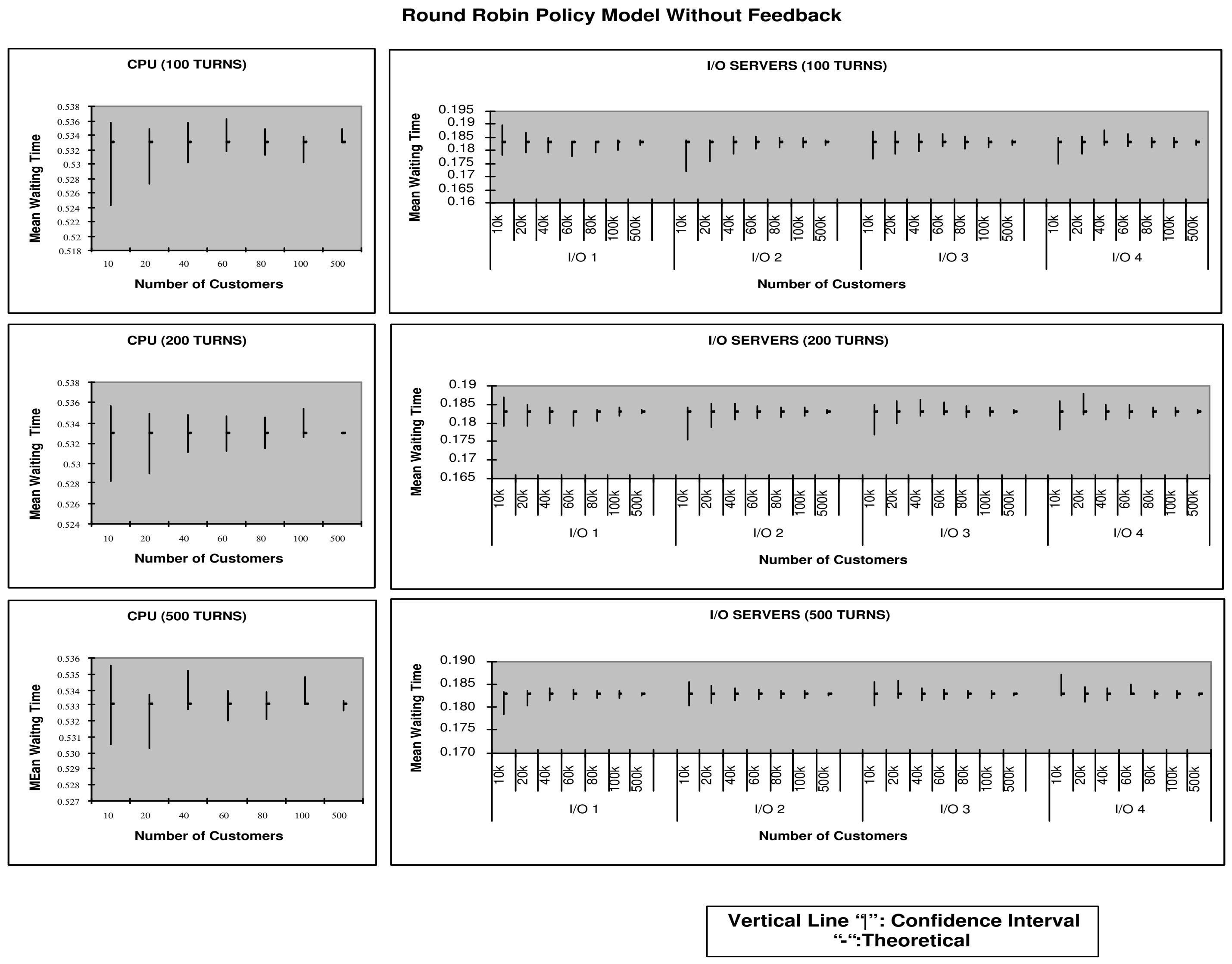

Figure 4.10: LR—Mean Waiting Time—RRMWF (n = 100, 200 and 500)

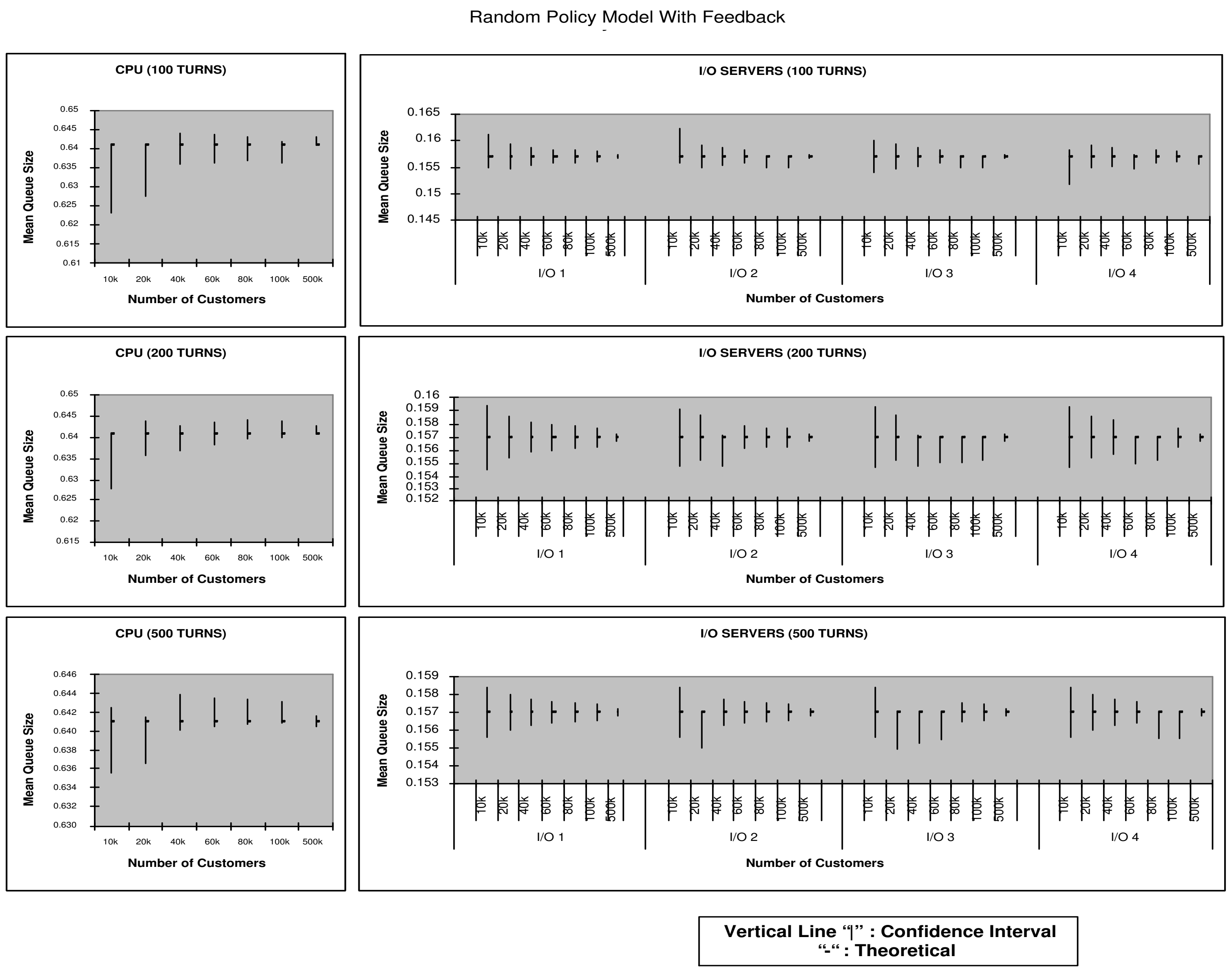

Figure 4.11: LR—Mean Queue Size—RMF (n = 100, 200 and 500)

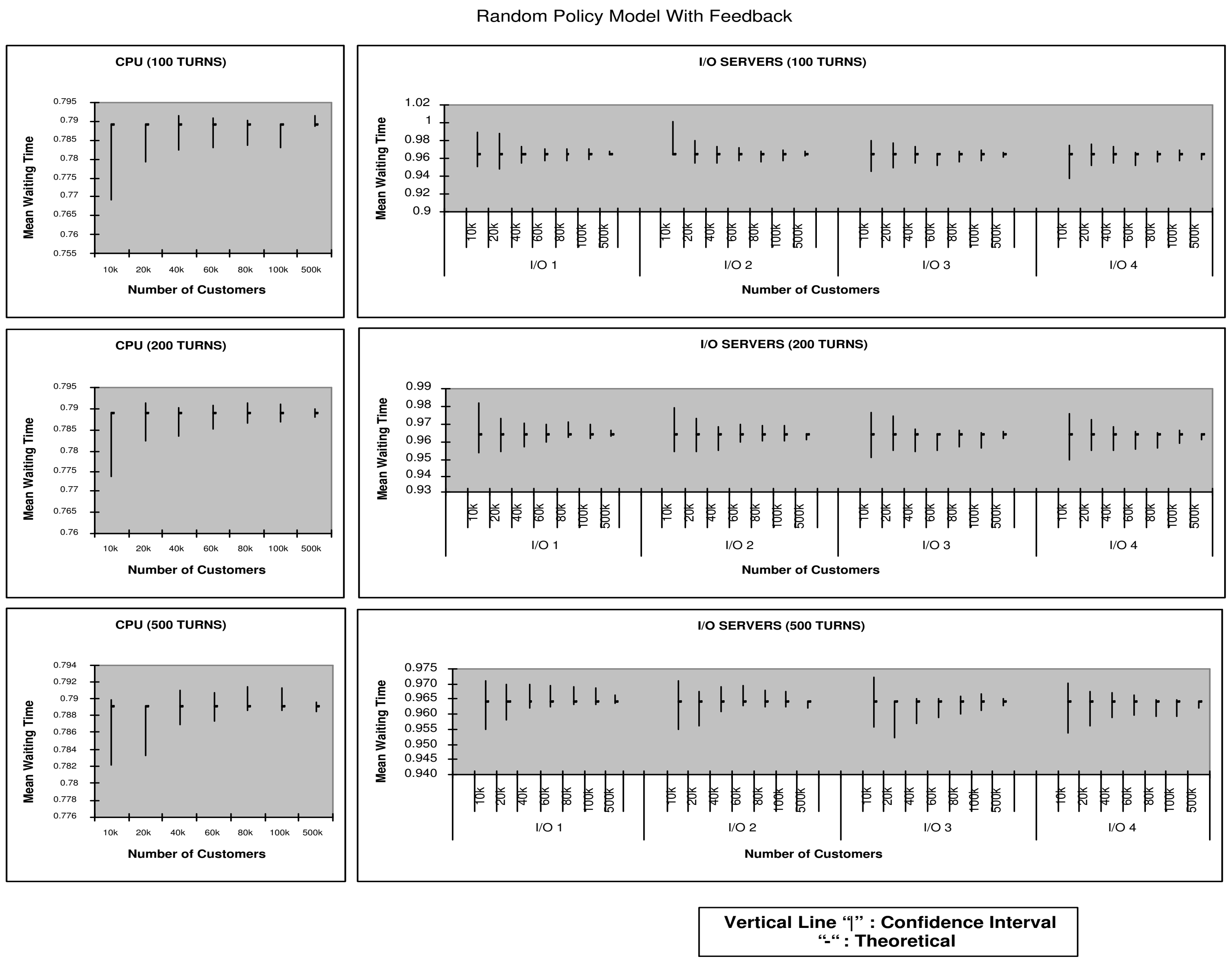

Figure 4.12: LR—Mean Waiting Time—RMF (n = 100, 200 and 500)

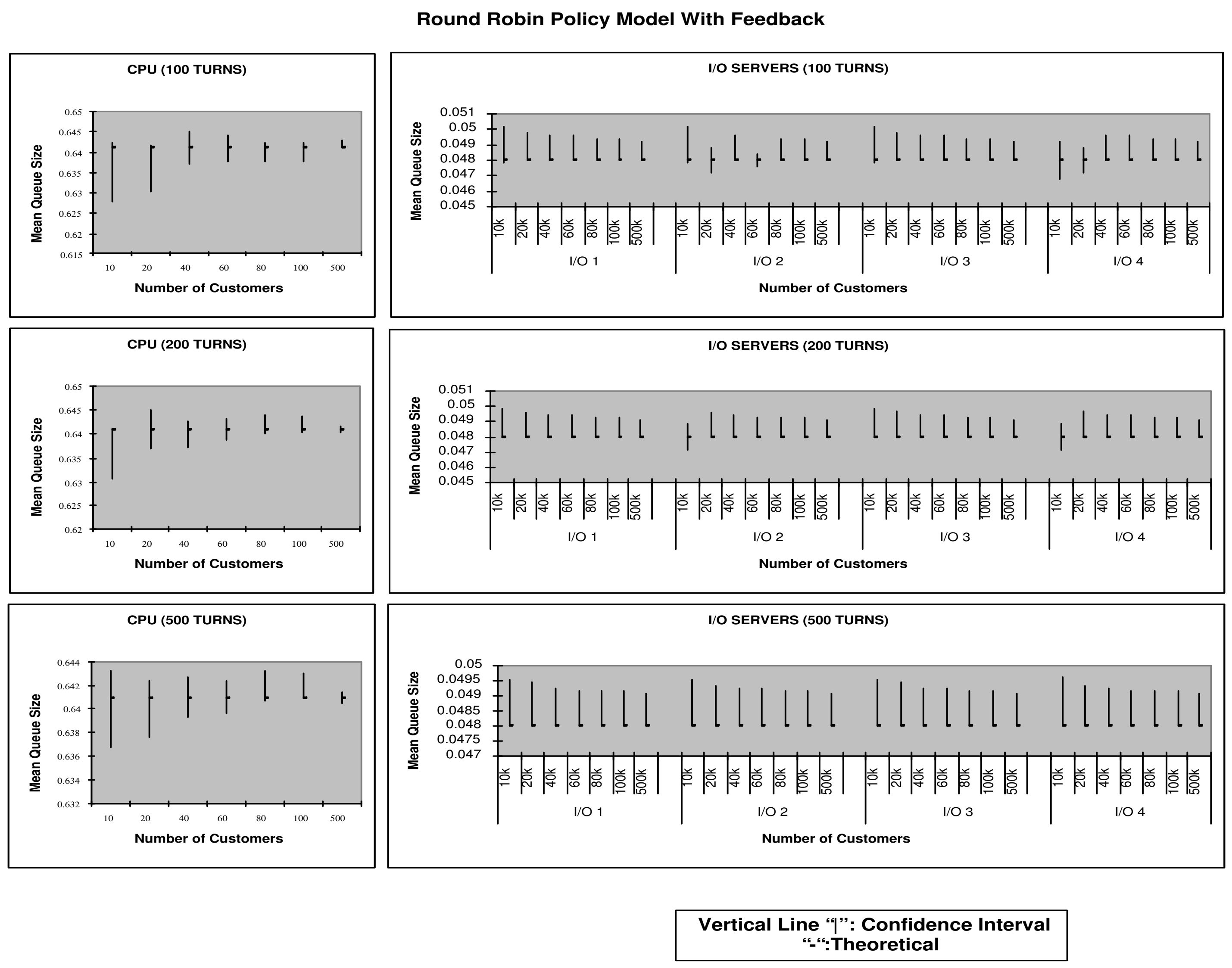

Figure 4.13: LR—Mean Queue Size—RRMF (n = 100, 200 and 500)

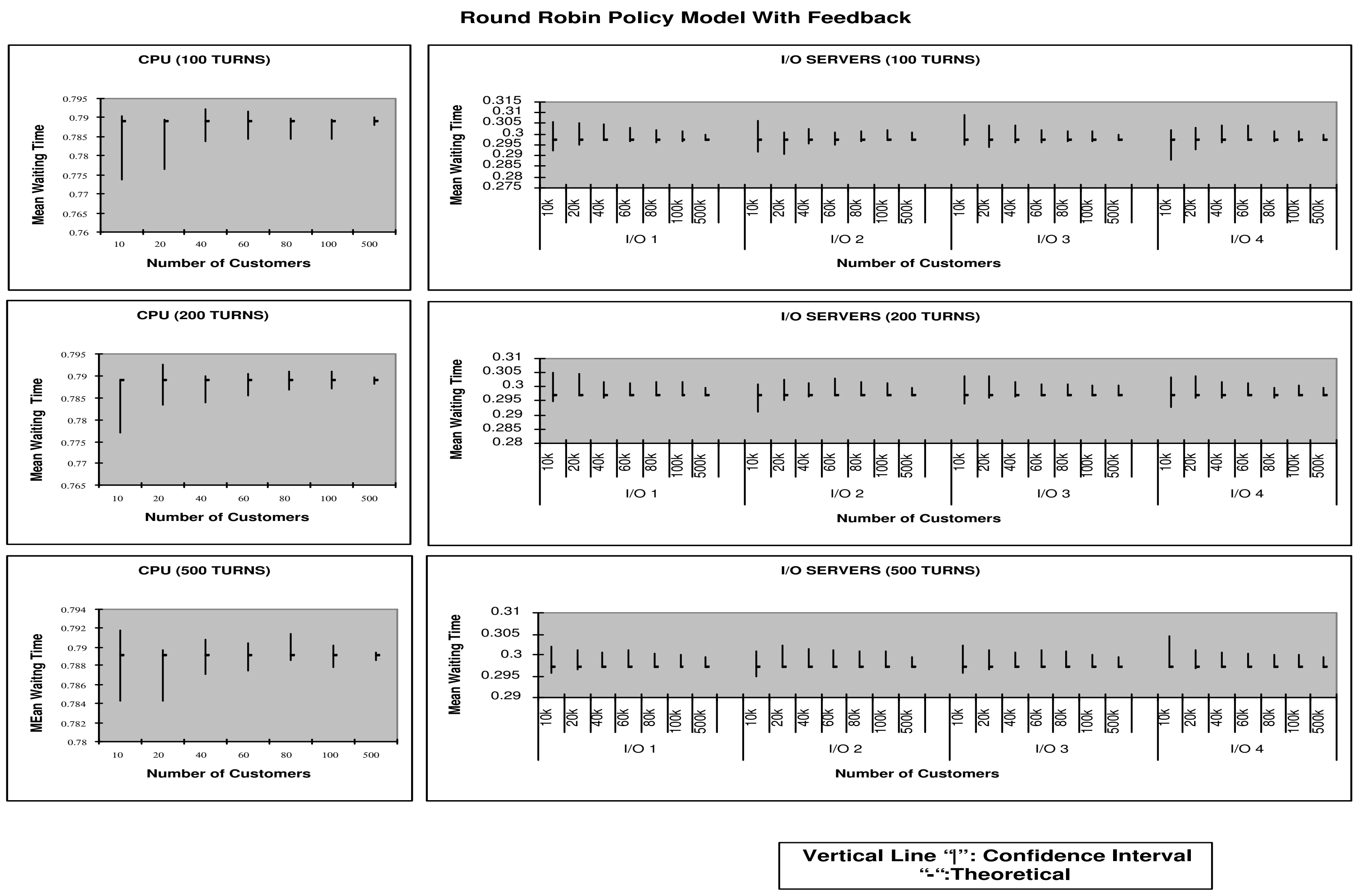

Figure 4.14: LR—Mean Waiting Time—RRMF (n = 100, 200 and 500)

Chapter 5

Conclusions and Future Work

The demand for Internet access is increasing dramatically. Web server systems are the information engines that service this rapidly growing client base. They are expected to provide good performance and high availability to the end user. The need for efficient and effective Web server systems is on the rise. An important step toward fulfilling this need is to identify the state-of-art in this area and to have a clear understanding of existing products and prototypes. In this thesis, we reviewed a number of commercial and prototype Web server systems, and devised a unified product-based taxonomy that identified product capabilities by relating them to a classification of Web server architectures and to an extended taxonomy of dispatching algorithms. The previous work on Web server systems [CCY99, CCCY02, CYD98, SGR00] were extended to produce a base taxonomy of dispatching algorithms that encompasses a more complete set of products.

The unified taxonomy is a significant step towards a methodology for classifying products based on their capabilities. It acts as a bridge between the taxonomy of server architectures and the taxonomy of dispatching algorithms, and this provides a clearer understanding of how the products relate to one another. This work is valuable to business professionals to help compare products when deciding on a Web server system deployment. It is also useful to software architects to help improve existing products and to develop new products with enhanced capabilities.

In a competitive market, commercial Web server systems need to provide feature-rich and efficient scalable services at low cost. Generic queuing models for Web server systems were designed to evaluate Web server systems in this thesis. They were configured without and with feedback mechanisms. Random and RR algorithms were used as dispatching policies in these models. Metrics like utilization, mean queue size, mean waiting time, and response time of these models were calculated theoretically and simulated using SSJ for performance evaluation. The models were validated and steady state performance to confirm the closeness and consistency in these models. The results achieved give a clear understanding that the designed models were sensible, acceptable models and very useful in analyzing Web server systems. The theoretical analysis of the feedback models is a useful contribution for Web server modeling research. The percentage of requests to the server unit was tested for different ratios and the models were consistent in all performance tests. The limitations found in these models are that the designed models are specific to Layer-4 one-way architecture and static content-blind dispatching algorithms, random and RR. The models were not tested for real-time measurements, and were evaluated using simulation and theoretical analysis.

Any research attains its goal only when it could be extended. The generic models used in this thesis forms the basis for modeling Web server systems. For further research these generic queuing models could be extended more with specific product specifications in order to analyze them individually. Models could also be extended with different architectures and dispatching algorithms suiting product specifications. An effective analysis of available products is an attractive area for future work. Different new metrics could be devised and the performance of these models based on end-user goals for the specific metrics also could be analyzed. A subsequent step to this research could be grading of products with the help of their performance metrics. This will provide a basis for classifying products, comparing them to one another, and recognizing the novel features of new products. A product rating system of this nature would be advantageous in a Web server system deployment, as it would evaluate the performance of individual products.

Abbreviations

API Application Program Interface

AVI Audio Video Interleave

AWS Alteon Web Switch

ASLB Accelerated Server Load Balancing

CAP Client-Aware Policy

CD Central Dispatch

CPU Central Processing Unit

DD Distributed Director

DNS Domain Name Server (System or Service)

DPR Distributed Packet Rewriting

DRP Distributed Response Protocol

DSL Digital Subscriber Line

DWRR Dynamic Weighted Round Robin

FRT Fast Response Time

FWLB Firewall Load Balancing

GSLB Global Server Load Balancing

HACC Harvard Array of Cluster Computers

HTML Hyper Text Markup Language

HTTP Hyper Text Transfer Protocol

I2-DSI Internet2 Distributed Storage Infrastructure

JPG/JPEG Joint Photographic Experts Group

LARD Locality-Aware Request Distribution

LD Local Director

LVS Linux Virtual Server

MAC Media Access Control

MP3 MPEG1 Audio Layer 3

MPG/MPEG Moving Picture Experts Group

MTRR Multi-Tier Round Robin

NAT Network Address Translation

NCSA National Center for Supercomputing Applications

NLB Network Load Balancing

RMF Random Policy Model With Feedback

RMWF Random Policy Model Without Feedback

RR Round Robin

RRMF RoundRobin Policy Model With Feedback

RRMEF RoundRobin Policy Model With Extra Feedback

RRMWF RoundRobin Policy Model Without Feedback

RSLB Reliable Server Load Balancing

SITA-E Size Interval Task Assignment with Equal load

SWEB Scalable Web Server

SWRR Static Weighted Round Robin

TCP/IP Transport Control Protocol/Internet Protocol

TCS Transparent Cache Switching

TTL Time To Live

UDP User Datagram Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

VIPA Virtual IP address

VPN Virtual Private Networking

XML Extensible Markup Language

XSL Extensible Stylesheet Language

XSLT Extensible Stylesheet Language Transformations

Appendix A

Confidence Interval (CI)

Models:

- *Random Policy Model Without Feedback* (RMWF)
- *RoundRobin Policy Model Without Feedback* (RRMWF)
- *Random Policy Model With Feedback* (RMF)
- *RoundRobin Policy Model With Feedback* (RRMF)
- *RoundRobin Policy Model With Extra Feedback* (RRMEF)

Performance Metrics:

- Mean Queue Size
- Mean Waiting Time
- Total Response Time

Replications:

- n = 100, 200 and 500

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.356	0.010	[0.358,0.354]	0.356
I/O_1	0.096	0.005	[0.097,0.095]	0.097
I/O_2	0.097	0.006	[0.098,0.096]	0.097
I/O_3	0.098	0.006	[0.099,0.097]	0.097
I/O_4	0.096	0.005	[0.097,0.095]	0.097

(a) n=100

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.356	0.009	[0.357,0.355]	0.356
I/O_1	0.096	0.005	[0.097,0.095]	0.097
I/O_2	0.097	0.006	[0.098,0.096]	0.097
I/O_3	0.097	0.006	[0.098,0.096]	0.097
I/O_4	0.097	0.005	[0.098,0.096]	0.097

(b) n=200

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.356	0.010	[0.357,0.355]	0.356
I/O_1	0.097	0.006	[0.098,0.096]	0.097
I/O_2	0.097	0.005	[0.097,0.097]	0.097
I/O_3	0.097	0.005	[0.097,0.097]	0.097
I/O_4	0.097	0.005	[0.097,0.097]	0.097

(c) n=500

Table 5.1: RMWF—Mean Queue Size

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.533	0.014	[0.536,0.530]	0.533
I/O_1	0.722	0.036	[0.729,0.715]	0.727
I/O_2	0.730	0.042	[0.738,0.722]	0.727
I/O_3	0.731	0.041	[0.739,0.723]	0.727
I/O_4	0.722	0.038	[0.730,0.714]	0.727

(a) n=100

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.534	0.013	[0.536,0.532]	0.533
I/O_1	0.721	0.037	[0.726,0.716]	0.727
I/O_2	0.729	0.040	[0.735,0.723]	0.727
I/O_3	0.730	0.040	[0.736,0.724]	0.727
I/O_4	0.726	0.037	[0.731,0.721]	0.727

(b) n=200

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.533	0.013	[0.534,0.532]	0.533
I/O_1	0.726	0.040	[0.730,0.722]	0.727
I/O_2	0.726	0.038	[0.729,0.723]	0.727
I/O_3	0.730	0.037	[0.733,0.727]	0.727
I/O_4	0.727	0.037	[0.730,0.724]	0.727

(c) n=500

Table 5.2: RMWF—Mean Waiting Time

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	3.380	0.027	[3.385,3.375]	3.384

(a) n=100

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	3.381	0.026	[3.385,3.377]	3.384

(b) n=200

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	3.382	0.027	[3.384,3.38]	3.384

(c) n=500

Table 5.3: RMWF—Total Response Time

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.356	0.010	[0.358,0.354]	0.356
I/O_1	0.024	0.002	[0.024,0.024]	0.024
I/O_2	0.024	0.002	[0.024,0.024]	0.024
I/O_3	0.024	0.002	[0.024,0.024]	0.024
I/O_4	0.025	0.002	[0.025,0.025]	0.025

(a) n=100

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.355	0.011	[0.357,0.353]	0.356
I/O_1	0.024	0.002	[0.024,0.024]	0.024
I/O_2	0.024	0.002	[0.024,0.024]	0.024
I/O_3	0.024	0.002	[0.024,0.024]	0.024
I/O_4	0.024	0.002	[0.024,0.024]	0.024

(b) n=200

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.355	0.010	[0.356,0.354]	0.356
I/O_1	0.024	0.002	[0.024,0.024]	0.024
I/O_2	0.024	0.002	[0.024,0.024]	0.024
I/O_3	0.024	0.002	[0.024,0.024]	0.024
I/O_4	0.025	0.002	[0.025,0.025]	0.024

(c) n=500

Table 5.4: RRMWF—Mean Queue Size

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.533	0.014	[0.536,0.530]	0.533
I/O_1	0.182	0.013	[0.185,0.179]	0.183
I/O_2	0.182	0.015	[0.185,0.179]	0.183
I/O_3	0.183	0.015	[0.186,0.180]	0.183
I/O_4	0.185	0.013	[0.188,0.182]	0.183

(a) n=100

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.532	0.014	[0.534,0.530]	0.533
I/O_1	0.182	0.013	[0.184,0.180]	0.183
I/O_2	0.183	0.014	[0.185,0.181]	0.183
I/O_3	0.183	0.014	[0.185,0.181]	0.183
I/O_4	0.184	0.014	[0.186,0.182]	0.183

(b) n=200

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.533	0.014	[0.534,0.532]	0.533
I/O_1	0.183	0.013	[0.184,0.182]	0.183
I/O_2	0.183	0.014	[0.184,0.182]	0.183
I/O_3	0.183	0.013	[0.184,0.182]	0.183
I/O_4	0.184	0.014	[0.185,0.183]	0.183

(c) n=500

Table 5.5: RRMWF—Mean Waiting Time

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	2.946	0.021	[2.950,2942]	2.946

(a) n=100

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	2.945	0.022	[2.948,2.942]	2.946

(b) n=200

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	2.947	0.022	[2.949,2.945]	2.946

(c) n=500

Table 5.6: RRMWF—Total Response Time

Devices	**Simulated Output—E(N_q)**			**Theoretical Value—E(N_q)**
	Mean	**SD**	**CI**	
CPU	0.640	0.021	[0.644,0.636]	0.641
I/O_1	0.157	0.008	[0.159,0.155]	0.157
I/O_2	0.157	0.008	[0.159,0.155]	0.157
I/O_3	0.157	0.009	[0.159,0.155]	0.157
I/O_4	0.157	0.009	[0.159,0.155]	0.155

(a) n=100

Devices	**Simulated Output—E(N_q)**			**Theoretical Value—E(N_q)**
	Mean	**SD**	**CI**	
CPU	0.640	0.021	[0.644,0.640]	0.641
I/O_1	0.157	0.008	[0.158,0.156]	0.157
I/O_2	0.156	0.008	[0.158,0.156]	0.157
I/O_3	0.156	0.008	[0.157,0.155]	0.157
I/O_4	0.157	0.009	[0.158,0.156]	0.157

(b) n=200

Devices	**Simulated Output—E(N_q)**			**Theoretical Value—E(N_q)**
	Mean	**SD**	**CI**	
CPU	0.642	0.021	[0.644,0.640]	0.641
I/O_1	0.157	0.008	[0.158,0.156]	0.157
I/O_2	0.157	0.008	[0.158,0.156]	0.157
I/O_3	0.156	0.008	[0.157,0.155]	0.157
I/O_4	0.157	0.008	[0.158,0.156]	0.157

(c) n=500

Table 5.7: RMF—Mean Queue Size

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.787	0.023	[0.792,0.782]	0.789
I/O_1	0.964	0.046	[0.973,0.955]	0.964
I/O_2	0.964	0.047	[0.973,0.955]	0.964
I/O_3	0.964	0.048	[0.974,0.954]	0.964
I/O_4	0.964	0.048	[0.974,0.954]	0.964

(a) n=100

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.787	0.024	[0.790,0.784]	0.789
I/O_1	0.964	0.046	[0.971,0.957]	0.964
I/O_2	0.962	0.045	[0.968,0.956]	0.964
I/O_3	0.961	0.047	[0.968,0.954]	0.964
I/O_4	0.962	0.048	[0.969,0.955]	0.964

(b) n=200

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.789	0.023	[0.791,0.787]	0.789
I/O_1	0.966	0.045	[0.970,0.962]	0.964
I/O_2	0.965	0.045	[0.969,0.961]	0.964
I/O_3	0.961	0.047	[0.965,0.957]	0.964
I/O_4	0.963	0.045	[0.967,0.959]	0.964

(c) n=500

Table 5.8: RMF—Mean Waiting Time

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	4.664	0.046	[4.673,4.655]	4.664

(a) n=100

Devices	**Simulated Outputs—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	4.662	0.047	[4.669,4.655]	4.664

(b) n=200

Devices	**Simulated Outputs—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	4.668	0.048	[4.672,4.664]	4.664

(c) n=500

Table 5.9: RMF—Total Response Time

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.641	0.017	[0.644,0.638]	0.641
I/O_1	0.049	0.003	[0.050,0.048]	0.048
I/O_2	0.049	0.003	[0.050,0.048]	0.048
I/O_3	0.049	0.003	[0.050,0.048]	0.048
I/O_4	0.049	0.004	[0.050,0.048]	0.048

(a) n=100

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.640	0.018	[0.643,0.637]	0.641
I/O_1	0.049	0.003	[0.049,0.049]	0.048
I/O_2	0.049	0.003	[0.049,0.049]	0.048
I/O_3	0.049	0.003	[0.049,0.049]	0.048
I/O_4	0.049	0.003	[0.049,0.049]	0.048

(b) n=200

Devices	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	0.642	0.019	[0.644,0.640]	0.641
I/O_1	0.049	0.003	[0.049,0.049]	0.048
I/O_2	0.049	0.003	[0.049,0.049]	0.048
I/O_3	0.049	0.003	[0.049,0.049]	0.048
I/O_4	0.049	0.003	[0.049,0.049]	0.048

(c) n=500

Table 5.10: RRMF—Mean Queue Size

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.789	0.019	[0.793,0.785]	0.789
I/O_1	0.300	0.018	[0.304,0.296]	0.297
I/O_2	0.299	0.017	[0.302,0.296]	0.297
I/O_3	0.300	0.018	[0.304,0.296]	0.297
I/O_4	0.301	0.021	[0.305,0.297]	0.297

(a) n=100

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.787	0.002	[0.790,0.784]	0.789
I/O_1	0.299	0.018	[0.302,0.296]	0.297
I/O_2	0.300	0.017	[0.302,0.298]	0.297
I/O_3	0.299	0.017	[0.301,0.297]	0.297
I/O_4	0.299	0.019	[0.302,0.296]	0.297

(b) n=200

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	0.789	0.021	[0.791,0.787]	0.789
I/O_1	0.299	0.018	[0.301,0.297]	0.297
I/O_2	0.299	0.017	[0.301,0.297]	0.297
I/O_3	0.299	0.018	[0.301,0.297]	0.297
I/O_4	0.298	0.018	[0.300,0.296]	0.297

(c) n=500

Table 5.11: RRMF—Mean Waiting Time

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	4.664	0.046	[4.673,4.655]	4.664

(a) n=100

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	4.662	0.047	[4.669,4.655]	4.664

(b) n=200

Devices	**Simulated Output—E(R)**			**Theoretical Value—E(R)**
	Mean	**SD**	**CI**	
Total system	4.668	0.048	[4.672,4.664]	4.664

(c) n=500

Table 5.12: RRMF—Total Response Time

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(N$_q$)**
	Mean	**SD**	**CI**	
CPU	7.172	0.680	[7.308,7.036]	7.120
I/O_1	0.282	0.014	[0.285,0.279]	0.276
I/O_2	0.277	0.013	[0.280,0.274]	0.276
I/O_3	0.278	0.014	[0.281,0.275]	0.276
I/O_4	0.276	0.013	[0.279,0.273]	0.276

(a) Mean Queue Size (n=100)

Devices	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
	Mean	**SD**	**CI**	
CPU	5.374	0.494	[5.473,5.275]	5.339
I/O_1	1.056	0.051	[1.066,1.046]	1.035
I/O_2	1.039	0.048	[1.049,1.029]	1.035
I/O_3	1.042	0.050	[1.052,1.032]	1.035
I/O_4	1.035	0.048	[1.045,1.025]	1.035

(b) Mean Waiting Time (n=100)

Table 5.13: RRMEF—Performance Metrics

Appendix B

Steady State Performance—Long Runs (LR)

Model:

- *Random Policy Model Without Feedback* (RMWF)

Replications:

- n = 100, 200 and 500

Performance Metrics:

- Mean Queue Size
- Mean Waiting Time

Input Parameters:

- Ranging from 10,000 to 500,000 input requests

Devices	**No. of Customers**	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
		Mean	**SD**	**CI**	
CPU	10k	0.354	0.018	[0.358,0.350]	0.356
	20k	0.355	0.013	[0.358,0.352]	0.356
	40k	0.356	0.010	[0.358,0.354]	0.356
	60k	0.356	0.008	[0.358,0.354]	0.356
	80k	0.356	0.006	[0.357,0.355]	0.356
	100k	0.355	0.006	[0.356,0.354]	0.356
	500k	0.356	0.003	[0.357,0.355]	0.356
I/O_1	10k	0.095	0.011	[0.097,0.093]	0.097
	20k	0.096	0.008	[0.098,0.094]	0.097
	40k	0.096	0.005	[0.097,0.095]	0.097
	60k	0.096	0.004	[0.097,0.095]	0.097
	80k	0.096	0.004	[0.097,0.095]	0.097
	100k	0.096	0.003	[0.097,0.095]	0.097
	500k	0.097	0.002	[0.097,0.097]	0.097
I/O_2	10k	0.097	0.012	[0.099,0.095]	0.097
	20k	0.097	0.008	[0.099,0.095]	0.097
	40k	0.097	0.006	[0.098,0.096]	0.097
	60k	0.097	0.005	[0.098,0.096]	0.097
	80k	0.097	0.004	[0.098,0.096]	0.097
	100k	0.097	0.003	[0.098,0.096]	0.097
	500k	0.097	0.001	[0.097,0.097]	0.097
I/O_3	10k	0.099	0.012	[0.101,0.097]	0.097
	20k	0.097	0.007	[0.098,0.096]	0.097
	40k	0.098	0.006	[0.099,0.097]	0.097
	60k	0.098	0.005	[0.099,0.097]	0.097
	80k	0.097	0.004	[0.098,0.096]	0.097
	100k	0.098	0.003	[0.099,0.097]	0.097
	500k	0.097	0.001	[0.097,0.097]	0.097
I/O_4	10k	0.095	0.011	[0.097,0.093]	0.097
	20k	0.096	0.008	[0.098,0.094]	0.097
	40k	0.096	0.005	[0.097,0.095]	0.097
	60k	0.096	0.004	[0.097,0.095]	0.097
	80k	0.097	0.004	[0.098,0.096]	0.097
	100k	0.097	0.003	[0.098,0.096]	0.097
	500k	0.097	0.002	[0.097,0.097]	0.097

Table 5.14: LR—RMWF (n=100)—Mean Queue Size

Devices	No. of Customers	Simulated Output—E(N_q)			Theoretical Value—E(N_q)
		Mean	**SD**	**CI**	
CPU	10k	0.355	0.018	[0.358,0.352]	0.356
	20k	0.356	0.015	[0.358,0.354]	0.356
	40k	0.356	0.009	[0.357,0.355]	0.356
	60k	0.356	0.008	[0.357,0.355]	0.356
	80k	0.356	0.007	[0.357,0.355]	0.356
	100k	0.356	0.006	[0.357,0.355]	0.356
	500k	0.356	0.003	[0.356,0.356]	0.356
I/O_1	10k	0.096	0.011	[0.098,0.094]	0.097
	20k	0.096	0.008	[0.097,0.095]	0.097
	40k	0.096	0.005	[0.097,0.095]	0.097
	60k	0.096	0.005	[0.097,0.095]	0.097
	80k	0.097	0.004	[0.098,0.096]	0.097
	100k	0.097	0.004	[0.098,0.096]	0.097
	500k	0.097	0.002	[0.097,0.097]	0.097
I/O_2	10k	0.097	0.011	[0.099,0.095]	0.097
	20k	0.097	0.008	[0.098,0.096]	0.097
	40k	0.097	0.006	[0.098,0.096]	0.097
	60k	0.097	0.004	[0.098,0.096]	0.097
	80k	0.097	0.004	[0.098,0.096]	0.097
	100k	0.097	0.003	[0.097,0.097]	0.097
	500k	0.097	0.001	[0.097,0.097]	0.097
I/O_3	10k	0.098	0.011	[0.100,0.096]	0.097
	20k	0.098	0.008	[0.099,0.097]	0.097
	40k	0.097	0.006	[0.098,0.096]	0.097
	60k	0.097	0.004	[0.098,0.096]	0.097
	80k	0.097	0.004	[0.098,0.096]	0.097
	100k	0.097	0.003	[0.097,0.097]	0.097
	500k	0.097	0.001	[0.097,0.097]	0.097
I/O_4	10k	0.096	0.011	[0.098,0.094]	0.097
	20k	0.096	0.008	[0.097,0.095]	0.097
	40k	0.097	0.005	[0.098,0.096]	0.097
	60k	0.097	0.004	[0.098,0.096]	0.097
	80k	0.097	0.004	[0.098,0.096]	0.097
	100k	0.097	0.003	[0.097,0.097]	0.097
	500k	0.097	0.002	[0.097,0.097]	0.097

Table 5.15: LR—RMWF (n=200)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(N_q)**			**Theoretical Value—E(N_q)**
		Mean	**SD**	**CI**	
CPU	10k	0.356	0.019	[0.358,0.354]	0.356
	20k	0.355	0.014	[0.356,0.354]	0.356
	40k	0.356	0.001	[0.357,0.355]	0.356
	60k	0.355	0.008	[0.356,0.354]	0.356
	80k	0.356	0.007	[0.357,0.355]	0.356
	100k	0.356	0.006	[0.357,0.355]	0.356
	500k	0.356	0.003	[0.356,0.356]	0.356
I/O_1	10k	0.096	0.011	[0.097,0.095]	0.097
	20k	0.096	0.008	[0.098,0.095]	0.097
	40k	0.097	0.006	[0.097,0.096]	0.097
	60k	0.097	0.005	[0.097,0.097]	0.097
	80k	0.097	0.004	[0.097,0.097]	0.097
	100k	0.097	0.003	[0.097,0.097]	0.097
	500k	0.097	0.001	[0.097,0.097]	0.097
I/O_2	10k	0.097	0.011	[0.098,0.096]	0.097
	20k	0.097	0.008	[0.098,0.096]	0.097
	40k	0.097	0.005	[0.097,0.097]	0.097
	60k	0.097	0.004	[0.097,0.097]	0.097
	80k	0.097	0.004	[0.097,0.097]	0.097
	100k	0.097	0.003	[0.097,0.097]	0.097
	500k	0.097	0.001	[0.097,0.097]	0.097
I/O_3	10k	0.098	0.011	[0.099,0.097]	0.097
	20k	0.098	0.008	[0.099,0.096]	0.097
	40k	0.097	0.005	[0.097,0.097]	0.097
	60k	0.097	0.004	[0.097,0.097]	0.097
	80k	0.097	0.004	[0.097,0.096]	0.097
	100k	0.097	0.003	[0.097,0.097]	0.097
	500k	0.097	0.001	[0.097,0.097]	0.097
I/O_4	10k	0.096	0.011	[0.097,0.095]	0.097
	20k	0.097	0.008	[0.098,0.096]	0.097
	40k	0.097	0.005	[0.097,0.097]	0.097
	60k	0.097	0.004	[0.097,0.097]	0.097
	80k	0.097	0.004	[0.097,0.097]	0.097
	100k	0.097	0.003	[0.097,0.097]	0.097
	500k	0.097	0.002	[0.097,0.097]	0.097

Table 5.16: LR—RMWF (n=500)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.532	0.026	[0.537,0.527]	0.533
	20k	0.533	0.018	[0.537,0.529]	0.533
	40k	0.533	0.014	[0.536,0.530]	0.533
	60k	0.534	0.011	[0.536,0.532]	0.533
	80k	0.534	0.009	[0.536,0.532]	0.533
	100k	0.533	0.009	[0.535,0.531]	0.533
	500k	0.534	0.004	[0.535,0.533]	0.533
I/O_1	10k	0.710	0.076	[0.725,0.695]	0.727
	20k	0.722	0.057	[0.733,0.711]	0.727
	40k	0.722	0.036	[0.729,0.715]	0.727
	60k	0.719	0.030	[0.725,0.713]	0.727
	80k	0.721	0.026	[0.726,0.716]	0.727
	100k	0.721	0.025	[0.726,0.716]	0.727
	500k	0.728	0.012	[0.730,0.726]	0.727
I/O_2	10k	0.727	0.082	[0.743,0.711]	0.727
	20k	0.727	0.054	[0.738,0.716]	0.727
	40k	0.730	0.042	[0.738,0.722]	0.727
	60k	0.730	0.031	[0.736,0.724]	0.727
	80k	0.729	0.028	[0.735,0.723]	0.727
	100k	0.727	0.024	[0.732,0.722]	0.727
	500k	0.728	0.010	[0.730,0.726]	0.727
I/O_3	10k	0.741	0.081	[0.757,0.725]	0.727
	20k	0.730	0.052	[0.740,0.720]	0.727
	40k	0.731	0.041	[0.739,0.723]	0.727
	60k	0.733	0.032	[0.739,0.727]	0.727
	80k	0.730	0.027	[0.735,0.725]	0.727
	100k	0.730	0.023	[0.735,0.725]	0.727
	500k	0.729	0.010	[0.731,0.727]	0.727
I/O_4	10k	0.719	0.081	[0.735,0.703]	0.727
	20k	0.720	0.054	[0.731,0.709]	0.727
	40k	0.722	0.038	[0.730,0.714]	0.727
	60k	0.724	0.029	[0.730,0.718]	0.727
	80k	0.726	0.028	[0.732,0.720]	0.727
	100k	0.726	0.024	[0.731,0.721]	0.727
	500k	0.728	0.012	[0.730,0.726]	0.727

Table 5.17: LR—RMWF (n=100)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.533	0.025	[0.537,0.529]	0.533
	20k	0.533	0.002	[0.536,0.530]	0.533
	40k	0.534	0.013	[0.536,0.532]	0.533
	60k	0.533	0.011	[0.535,0.531]	0.533
	80k	0.533	0.010	[0.534,0.532]	0.533
	100k	0.533	0.009	[0.534,0.532]	0.533
	500k	0.534	0.004	[0.535,0.533]	0.533
I/O_1	10k	0.722	0.077	[0.733,0.711]	0.727
	20k	0.722	0.056	[0.730,0.714]	0.727
	40k	0.721	0.037	[0.726,0.716]	0.727
	60k	0.723	0.033	[0.728,0.718]	0.727
	80k	0.725	0.028	[0.729,0.721]	0.727
	100k	0.726	0.026	[0.730,0.722]	0.727
	500k	0.728	0.011	[0.730,0.726]	0.727
I/O_2	10k	0.727	0.076	[0.738,0.716]	0.727
	20k	0.730	0.057	[0.738,0.722]	0.727
	40k	0.729	0.040	[0.735,0.723]	0.727
	60k	0.727	0.029	[0.731,0.723]	0.727
	80k	0.726	0.027	[0.730,0.722]	0.727
	100k	0.726	0.024	[0.729,0.723]	0.727
	500k	0.728	0.009	[0.729,0.727]	0.727
I/O_3	10k	0.730	0.079	[0.739,0.723]	0.727
	20k	0.731	0.054	[0.736,0.723]	0.727
	40k	0.730	0.040	[0.733,0.724]	0.727
	60k	0.729	0.030	[0.733,0.725]	0.727
	80k	0.729	0.025	[0.733,0.725]	0.727
	100k	0.730	0.022	[0.733,0.727]	0.727
	500k	0.727	0.010	[0.728,0.726]	0.727
I/O_4	10k	0.720	0.078	[0.731,0.709]	0.727
	20k	0.722	0.053	[0.729,0.715]	0.727
	40k	0.726	0.037	[0.731,0.721]	0.727
	60k	0.727	0.030	[0.731,0.723]	0.727
	80k	0.726	0.027	[0.730,0.722]	0.727
	100k	0.727	0.024	[0.730,0.724]	0.727
	500k	0.727	0.012	[0.729,0.725]	0.727

Table 5.18: LR—RMWF (n=200)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.533	0.027	[0.535,0.531]	0.533
	20k	0.533	0.019	[0.535,0.531]	0.533
	40k	0.533	0.013	[0.534,0.532]	0.533
	60k	0.533	0.011	[0.534,0.532]	0.533
	80k	0.533	0.010	[0.534,0.532]	0.533
	100k	0.534	0.009	[0.535,0.533]	0.533
	500k	0.533	0.004	[0.533,0.533]	0.533
I/O_1	10k	0.722	0.077	[0.729,0.715]	0.727
	20k	0.721	0.056	[0.726,0.716]	0.727
	40k	0.726	0.040	[0.730,0.722]	0.727
	60k	0.727	0.032	[0.730,0.724]	0.727
	80k	0.728	0.028	[0.731,0.725]	0.727
	100k	0.728	0.025	[0.730,0.726]	0.727
	500k	0.728	0.010	[0.729,0.727]	0.727
I/O_2	10k	0.730	0.078	[0.737,0.723]	0.727
	20k	0.727	0.054	[0.732,0.722]	0.727
	40k	0.726	0.038	[0.729,0.723]	0.727
	60k	0.727	0.031	[0.730,0.724]	0.727
	80k	0.728	0.027	[0.730,0.726]	0.727
	100k	0.728	0.023	[0.730,0.726]	0.727
	500k	0.727	0.010	[0.728,0.726]	0.727
I/O_3	10k	0.732	0.079	[0.739,0.725]	0.727
	20k	0.730	0.054	[0.735,0.725]	0.727
	40k	0.730	0.037	[0.733,0.727]	0.727
	60k	0.729	0.030	[0.732,0.726]	0.727
	80k	0.729	0.026	[0.731,0.727]	0.727
	100k	0.729	0.023	[0.731,0.727]	0.727
	500k	0.727	0.010	[0.728,0.726]	0.727
I/O_4	10k	0.722	0.074	[0.729,0.715]	0.727
	20k	0.726	0.053	[0.731,0.721]	0.727
	40k	0.727	0.037	[0.730,0.724]	0.727
	60k	0.727	0.030	[0.730,0.724]	0.727
	80k	0.728	0.026	[0.730,0.726]	0.727
	100k	0.728	0.024	[0.730,0.726]	0.727
	500k	0.727	0.011	[0.728,0.726]	0.727

Table 5.19: LR—RMWF (n=500)—Mean Waiting Time

Appendix C

Steady State Performance—Long Runs (LR)

Model:

- *RoundRobin Policy Model Without Feedback* (RRMWF)

Replications:

- n = 100, 200 and 500

Performance Metrics:

- Mean Queue Size
- Mean Waiting Time

Input Parameters:

- Ranging from 10,000 to 500,000 input requests

Devices	**No. of Customers**	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
		Mean	**SD**	**CI**	
CPU	10k	0.353	0.002	[0.353,0.353]	0.356
	20k	0.354	0.014	[0.357,0.351]	0.356
	40k	0.355	0.010	[0.357,0.353]	0.356
	60k	0.356	0.008	[0.358,0.354]	0.356
	80k	0.355	0.007	[0.356,0.354]	0.356
	100k	0.355	0.007	[0.356,0.354]	0.356
	500k	0.356	0.003	[0.357,0.355]	0.356
I/O_1	10k	0.024	0.004	[0.025,0.023]	0.024
	20k	0.024	0.003	[0.025,0.023]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.001	[0.024,0.024]	0.024
I/O_2	10k	0.024	0.004	[0.025,0.023]	0.024
	20k	0.024	0.003	[0.025,0.023]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.001	[0.024,0.024]	0.024
I/O_3	10k	0.024	0.004	[0.025,0.020]	0.024
	20k	0.024	0.003	[0.025,0.020]	0.024
	40k	0.024	0.002	[0.024,0.020]	0.024
	60k	0.024	0.002	[0.024,0.020]	0.024
	80k	0.024	0.002	[0.024,0.020]	0.024
	100k	0.024	0.001	[0.024,0.020]	0.024
	500k	0.024	0.000	[0.024,0.020]	0.024
I/O_4	10k	0.024	0.003	[0.025,0.023]	0.024
	20k	0.024	0.002	[0.024,0.024]	0.024
	40k	0.025	0.002	[0.025,0.025]	0.024
	60k	0.025	0.001	[0.025,0.025]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.001	[0.024,0.024]	0.024

Table 5.20: LR—RRMWF (n=100)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(N_q)**			**Theoretical Value—E(N_q)**
		Mean	**SD**	**CI**	
CPU	10k	0.354	0.020	[0.356,0.350]	0.356
	20k	0.355	0.014	[0.357,0.353]	0.356
	40k	0.356	0.010	[0.356,0.354]	0.356
	60k	0.356	0.008	[0.356,0.354]	0.356
	80k	0.355	0.007	[0.356,0.354]	0.356
	100k	0.356	0.006	[0.357,0.355]	0.356
	500k	0.356	0.003	[0.356,0.356]	0.356
I/O_1	10k	0.024	0.004	[0.025,0.023]	0.024
	20k	0.024	0.003	[0.024,0.024]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024
I/O_2	10k	0.024	0.004	[0.025,0.023]	0.024
	20k	0.024	0.003	[0.024,0.024]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024
I/O_3	10k	0.024	0.004	[0.025,0.023]	0.024
	20k	0.024	0.003	[0.024,0.024]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024
I/O_4	10k	0.024	0.004	[0.025,0.023]	0.024
	20k	0.025	0.003	[0.025,0.025]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024

Table 5.21: LR—RRMWF (n=200)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
		Mean	**SD**	**CI**	
CPU	10k	0.355	0.020	[0.357,0.353]	0.356
	20k	0.355	0.015	[0.356,0.354]	0.356
	40k	0.356	0.011	[0.357,0.355]	0.356
	60k	0.356	0.008	[0.357,0.355]	0.356
	80k	0.356	0.008	[0.357,0.355]	0.356
	100k	0.356	0.007	[0.357,0.355]	0.356
	500k	0.356	0.003	[0.356,0.356]	0.356
I/O_1	10k	0.024	0.004	[0.024,0.024]	0.024
	20k	0.024	0.003	[0.024,0.024]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024
I/O_2	10k	0.024	0.004	[0.024,0.024]	0.024
	20k	0.024	0.003	[0.024,0.024]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024
I/O_3	10k	0.024	0.004	[0.024,0.024]	0.024
	20k	0.025	0.003	[0.025,0.025]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024
I/O_4	10k	0.025	0.004	[0.025,0.025]	0.024
	20k	0.024	0.003	[0.024,0.024]	0.024
	40k	0.024	0.002	[0.024,0.024]	0.024
	60k	0.024	0.002	[0.024,0.024]	0.024
	80k	0.024	0.001	[0.024,0.024]	0.024
	100k	0.024	0.001	[0.024,0.024]	0.024
	500k	0.024	0.000	[0.024,0.024]	0.024

Table 5.22: LR—RRMWF (n=500)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.530	0.029	[0.536,0.524]	0.533
	20k	0.531	0.019	[0.535,0.527]	0.533
	40k	0.533	0.014	[0.536,0.530]	0.533
	60k	0.534	0.011	[0.536,0.532]	0.533
	80k	0.533	0.009	[0.535,0.531]	0.533
	100k	0.532	0.009	[0.534,0.530]	0.533
	500k	0.534	0.004	[0.535,0.533]	0.533
I/O_1	10k	0.184	0.028	[0.190,0.178]	0.183
	20k	0.183	0.002	[0.187,0.179]	0.183
	40k	0.182	0.015	[0.185,0.179]	0.183
	60k	0.180	0.012	[0.182,0.178]	0.183
	80k	0.181	0.010	[0.183,0.179]	0.183
	100k	0.182	0.009	[0.184,0.180]	0.183
	500k	0.183	0.004	[0.184,0.182]	0.183
I/O_2	10k	0.178	0.030	[0.184,0.172]	0.183
	20k	0.180	0.021	[0.184,0.176]	0.183
	40k	0.182	0.016	[0.185,0.179]	0.183
	60k	0.183	0.012	[0.185,0.181]	0.183
	80k	0.183	0.010	[0.185,0.181]	0.183
	100k	0.183	0.009	[0.185,0.181]	0.183
	500k	0.183	0.004	[0.184,0.182]	0.183
I/O_3	10k	0.182	0.027	[0.187,0.177]	0.183
	20k	0.183	0.022	[0.187,0.179]	0.183
	40k	0.183	0.016	[0.186,0.180]	0.183
	60k	0.184	0.013	[0.187,0.181]	0.183
	80k	0.183	0.012	[0.185,0.181]	0.183
	100k	0.183	0.010	[0.185,0.181]	0.183
	500k	0.183	0.004	[0.184,0.182]	0.183
I/O_4	10k	0.180	0.025	[0.185,0.175]	0.183
	20k	0.182	0.017	[0.185,0.179]	0.183
	40k	0.185	0.014	[0.188,0.182]	0.183
	60k	0.184	0.011	[0.186,0.182]	0.183
	80k	0.183	0.010	[0.185,0.181]	0.183
	100k	0.183	0.009	[0.185,0.181]	0.183
	500k	0.183	0.004	[0.184,0.182]	0.183

Table 5.23: LR—RRMWF (n=100)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.532	0.026	[0.536,0.528]	0.533
	20k	0.532	0.021	[0.535,0.529]	0.533
	40k	0.533	0.013	[0.535,0.531]	0.533
	60k	0.533	0.012	[0.535,0.531]	0.533
	80k	0.533	0.011	[0.535,0.531]	0.533
	100k	0.534	0.010	[0.535,0.533]	0.533
	500k	0.534	0.004	[0.535,0.533]	0.533
I/O_1	10k	0.183	0.028	[0.187,0.179]	0.183
	20k	0.182	0.020	[0.185,0.179]	0.183
	40k	0.182	0.015	[0.184,0.180]	0.183
	60k	0.181	0.012	[0.183,0.179]	0.183
	80k	0.182	0.010	[0.183,0.181]	0.183
	100k	0.183	0.009	[0.184,0.182]	0.183
	500k	0.183	0.003	[0.183,0.183]	0.183
I/O_2	10k	0.180	0.025	[0.184,0.176]	0.183
	20k	0.182	0.017	[0.185,0.179]	0.183
	40k	0.183	0.012	[0.185,0.181]	0.183
	60k	0.183	0.010	[0.185,0.181]	0.183
	80k	0.183	0.009	[0.184,0.182]	0.183
	100k	0.183	0.008	[0.184,0.182]	0.183
	500k	0.183	0.003	[0.183,0.183]	0.183
I/O_3	10k	0.181	0.029	[0.185,0.177]	0.183
	20k	0.183	0.022	[0.186,0.180]	0.183
	40k	0.184	0.016	[0.186,0.182]	0.183
	60k	0.184	0.012	[0.186,0.182]	0.183
	80k	0.183	0.011	[0.185,0.181]	0.183
	100k	0.183	0.009	[0.184,0.182]	0.183
	500k	0.183	0.003	[0.183,0.183]	0.183
I/O_4	10k	0.182	0.027	[0.186,0.178]	0.183
	20k	0.185	0.020	[0.188,0.182]	0.183
	40k	0.183	0.014	[0.185,0.181]	0.183
	60k	0.183	0.013	[0.185,0.181]	0.183
	80k	0.183	0.010	[0.184,0.182]	0.183
	100k	0.183	0.009	[0.184,0.182]	0.183
	500k	0.183	0.003	[0.183,0.183]	0.183

Table 5.24: LR—RRMWF (n=200)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.533	0.028	[0.536,0.530]	0.533
	20k	0.532	0.019	[0.534,0.530]	0.533
	40k	0.534	0.014	[0.535,0.533]	0.533
	60k	0.533	0.011	[0.534,0.532]	0.533
	80k	0.533	0.010	[0.534,0.532]	0.533
	100k	0.534	0.009	[0.535,0.533]	0.533
	500k	0.533	0.004	[0.533,0.533]	0.533
I/O_1	10k	0.181	0.028	[0.184,0.178]	0.183
	20k	0.182	0.020	[0.184,0.180]	0.183
	40k	0.183	0.015	[0.184,0.182]	0.183
	60k	0.183	0.011	[0.184,0.182]	0.183
	80k	0.183	0.010	[0.184,0.182]	0.183
	100k	0.183	0.009	[0.184,0.182]	0.183
	500k	0.183	0.004	[0.184,0.182]	0.183
I/O_2	10k	0.183	0.030	[0.186,0.180]	0.183
	20k	0.183	0.020	[0.185,0.181]	0.183
	40k	0.183	0.015	[0.184,0.182]	0.183
	60k	0.183	0.012	[0.184,0.182]	0.183
	80k	0.183	0.010	[0.184,0.182]	0.183
	100k	0.183	0.009	[0.184,0.182]	0.183
	500k	0.183	0.003	[0.183,0.183]	0.183
I/O_3	10k	0.183	0.030	[0.186,0.180]	0.183
	20k	0.184	0.021	[0.186,0.182]	0.183
	40k	0.183	0.015	[0.184,0.182]	0.183
	60k	0.183	0.012	[0.184,0.182]	0.183
	80k	0.183	0.010	[0.184,0.182]	0.183
	100k	0.183	0.009	[0.184,0.182]	0.183
	500k	0.183	0.003	[0.183,0.182]	0.183
I/O_4	10k	0.185	0.027	[0.187,0.183]	0.183
	20k	0.183	0.019	[0.185,0.181]	0.183
	40k	0.183	0.015	[0.184,0.182]	0.183
	60k	0.184	0.012	[0.185,0.183]	0.183
	80k	0.183	0.010	[0.184,0.182]	0.183
	100k	0.183	0.010	[0.184,0.182]	0.183
	500k	0.183	0.003	[0.183,0.183]	0.183

Table 5.25: LR—RRMWF (n=500)—Mean Waiting Time

Appendix D

Steady State Performance—Long Runs (LR)

Model:

- *Random Policy Model With Feedback* (RMF)

Replications:

- n = 100, 200 and 500

Performance Metrics:

- Mean Queue Size
- Mean Waiting Time

Input Parameters:

- Ranging from 10,000 to 500,000 input requests

Devices	**No. of Customers**	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
		Mean	**SD**	**CI**	
CPU	10k	0.630	0.035	[0.637,0.623]	0.641
	20k	0.633	0.027	[0.638,0.628]	0.641
	40k	0.640	0.021	[0.644,0.636]	0.641
	60k	0.640	0.018	[0.644,0.636]	0.641
	80k	0.640	0.015	[0.643,0.637]	0.641
	100k	0.639	0.014	[0.642,0.636]	0.641
	500k	0.642	0.006	[0.643,0.641]	0.641
I/O_1	10k	0.158	0.016	[0.161,0.155]	0.157
	20k	0.157	0.012	[0.159,0.155]	0.157
	40k	0.157	0.008	[0.159,0.155]	0.157
	60k	0.157	0.006	[0.158,0.156]	0.157
	80k	0.157	0.006	[0.158,0.156]	0.157
	100k	0.157	0.005	[0.158,0.156]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_2	10k	0.159	0.016	[0.162,0.156]	0.157
	20k	0.157	0.011	[0.159,0.155]	0.157
	40k	0.157	0.008	[0.159,0.155]	0.157
	60k	0.157	0.006	[0.158,0.156]	0.157
	80k	0.156	0.005	[0.157,0.155]	0.157
	100k	0.156	0.005	[0.157,0.155]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_3	10k	0.157	0.015	[0.160,0.154]	0.157
	20k	0.157	0.012	[0.159,0.155]	0.157
	40k	0.157	0.009	[0.159,0.155]	0.157
	60k	0.157	0.006	[0.158,0.156]	0.157
	80k	0.156	0.006	[0.157,0.155]	0.157
	100k	0.156	0.005	[0.157,0.155]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_4	10k	0.155	0.016	[0.158,0.152]	0.157
	20k	0.157	0.011	[0.159,0.155]	0.157
	40k	0.157	0.009	[0.159,0.155]	0.157
	60k	0.156	0.007	[0.157,0.155]	0.157
	80k	0.157	0.006	[0.158,0.156]	0.157
	100k	0.157	0.005	[0.158,0.156]	0.157
	500k	0.156	0.002	[0.156,0.156]	0.157

Table 5.26: LR—RMF (n=100)—Mean Queue Size

Devices	No. of Customers	Simulated Output—E(N_q)			Theoretical Value—E(N_q)
		Mean	SD	CI	
CPU	10k	0.633	0.039	[0.639,0.627]	0.641
	20k	0.640	0.028	[0.644,0.636]	0.641
	40k	0.640	0.021	[0.643,0.637]	0.641
	60k	0.641	0.018	[0.644,0.638]	0.641
	80k	0.642	0.015	[0.644,0.640]	0.641
	100k	0.642	0.013	[0.644,0.640]	0.641
	500k	0.642	0.006	[0.643,0.641]	0.641
I/O_1	10k	0.157	0.017	[0.159,0.155]	0.157
	20k	0.157	0.011	[0.159,0.155]	0.157
	40k	0.157	0.008	[0.158,0.156]	0.157
	60k	0.157	0.007	[0.158,0.156]	0.157
	80k	0.157	0.006	[0.158,0.156]	0.157
	100k	0.157	0.005	[0.158,0.156]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_2	10k	0.157	0.015	[0.159,0.155]	0.157
	20k	0.157	0.012	[0.159,0.155]	0.157
	40k	0.156	0.008	[0.157,0.155]	0.157
	60k	0.157	0.006	[0.158,0.156]	0.157
	80k	0.157	0.005	[0.158,0.156]	0.157
	100k	0.156	0.005	[0.158,0.156]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_3	10k	0.157	0.016	[0.159,0.155]	0.157
	20k	0.157	0.012	[0.159,0.155]	0.157
	40k	0.156	0.008	[0.157,0.155]	0.157
	60k	0.156	0.006	[0.157,0.155]	0.157
	80k	0.156	0.006	[0.157,0.155]	0.157
	100k	0.156	0.005	[0.157,0.155]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_4	10k	0.157	0.016	[0.159,0.155]	0.157
	20k	0.157	0.011	[0.159,0.155]	0.157
	40k	0.157	0.009	[0.158,0.156]	0.157
	60k	0.156	0.007	[0.157,0.155]	0.157
	80k	0.156	0.005	[0.157,0.155]	0.157
	100k	0.157	0.005	[0.158,0.156]	0.157
	500k	0.157	0.002	[0.156,0.157]	0.157

Table 5.27: LR—RMF (n=200)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(N_q)**			**Theoretical Value—E(N_q)**
		Mean	**SD**	**CI**	
CPU	10k	0.639	0.039	[0.642,0.636]	0.641
	20k	0.639	0.028	[0.642,0.636]	0.641
	40k	0.642	0.021	[0.644,0.640]	0.641
	60k	0.642	0.017	[0.644,0.640]	0.641
	80k	0.642	0.015	[0.643,0.641]	0.641
	100k	0.642	0.013	[0.643,0.641]	0.641
	500k	0.641	0.006	[0.642,0.640]	0.641
I/O_1	10k	0.157	0.016	[0.158,0.156]	0.157
	20k	0.157	0.011	[0.158,0.156]	0.157
	40k	0.157	0.008	[0.158,0.156]	0.157
	60k	0.157	0.007	[0.158,0.156]	0.157
	80k	0.157	0.006	[0.158,0.156]	0.157
	100k	0.157	0.005	[0.157,0.157]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_2	10k	0.157	0.016	[0.158,0.156]	0.157
	20k	0.156	0.011	[0.157,0.155]	0.157
	40k	0.157	0.008	[0.158,0.156]	0.157
	60k	0.157	0.007	[0.158,0.156]	0.157
	80k	0.157	0.006	[0.158,0.156]	0.157
	100k	0.157	0.005	[0.157,0.157]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_3	10k	0.157	0.016	[0.158,0.156]	0.157
	20k	0.156	0.012	[0.157,0.155]	0.157
	40k	0.156	0.008	[0.157,0.155]	0.157
	60k	0.156	0.006	[0.157,0.155]	0.157
	80k	0.157	0.006	[0.158,0.156]	0.157
	100k	0.157	0.005	[0.157,0.157]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157
I/O_4	10k	0.157	0.016	[0.158,0.156]	0.157
	20k	0.157	0.011	[0.158,0.156]	0.157
	40k	0.157	0.008	[0.158,0.156]	0.157
	60k	0.157	0.007	[0.158,0.156]	0.157
	80k	0.156	0.005	[0.156,0.156]	0.157
	100k	0.156	0.005	[0.156,0.156]	0.157
	500k	0.157	0.002	[0.157,0.157]	0.157

Table 5.28: LR—RMF (n=500)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.777	0.039	[0.785,0.769]	0.789
	20k	0.780	0.003	[0.781,0.779]	0.789
	40k	0.787	0.023	[0.792,0.782]	0.789
	60k	0.787	0.020	[0.791,0.783]	0.789
	80k	0.787	0.017	[0.790,0.784]	0.789
	100k	0.786	0.015	[0.789,0.783]	0.789
	500k	0.790	0.007	[0.791,0.789]	0.789
I/O_1	10k	0.970	0.096	[0.989,0.951]	0.964
	20k	0.968	0.097	[0.987,0.949]	0.964
	40k	0.964	0.046	[0.973,0.955]	0.964
	60k	0.964	0.034	[0.971,0.957]	0.964
	80k	0.964	0.032	[0.970,0.958]	0.964
	100k	0.965	0.030	[0.971,0.959]	0.964
	500k	0.966	0.013	[0.969,0.963]	0.964
I/O_2	10k	0.983	0.090	[1.001,0.965]	0.964
	20k	0.967	0.062	[0.979,0.955]	0.964
	40k	0.964	0.047	[0.973,0.955]	0.964
	60k	0.965	0.037	[0.972,0.958]	0.964
	80k	0.962	0.031	[0.968,0.956]	0.964
	100k	0.963	0.029	[0.969,0.957]	0.964
	500k	0.965	0.012	[0.967,0.963]	0.964
I/O_3	10k	0.963	0.085	[0.980,0.946]	0.964
	20k	0.964	0.070	[0.978,0.950]	0.964
	40k	0.964	0.048	[0.974,0.954]	0.964
	60k	0.959	0.035	[0.966,0.952]	0.964
	80k	0.962	0.032	[0.968,0.956]	0.964
	100k	0.963	0.029	[0.969,0.957]	0.964
	500k	0.964	0.011	[0.966,0.962]	0.964
I/O_4	10k	0.956	0.094	[0.975,0.937]	0.964
	20k	0.964	0.061	[0.976,0.952]	0.964
	40k	0.964	0.048	[0.974,0.954]	0.964
	60k	0.959	0.038	[0.967,0.951]	0.964
	80k	0.962	0.033	[0.969,0.955]	0.964
	100k	0.963	0.029	[0.969,0.957]	0.964
	500k	0.962	0.014	[0.965,0.959]	0.964

Table 5.29: LR—RMF (n=100)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.780	0.044	[0.786,0.769]	0.789
	20k	0.787	0.031	[0.791,0.783]	0.789
	40k	0.787	0.024	[0.790,0.784]	0.789
	60k	0.788	0.020	[0.791,0.785]	0.789
	80k	0.789	0.016	[0.791,0.787]	0.789
	100k	0.789	0.015	[0.791,0.787]	0.789
	500k	0.789	0.007	[0.790,0.788]	0.789
I/O_1	10k	0.968	0.099	[0.982,0.954]	0.964
	20k	0.964	0.064	[0.973,0.955]	0.964
	40k	0.964	0.046	[0.971,0.957]	0.964
	60k	0.965	0.036	[0.970,0.960]	0.964
	80k	0.967	0.032	[0.972,0.962]	0.964
	100k	0.966	0.030	[0.970,0.962]	0.964
	500k	0.965	0.013	[0.967,0.963]	0.964
I/O_2	10k	0.967	0.086	[0.979,0.955]	0.964
	20k	0.964	0.066	[0.973,0.955]	0.964
	40k	0.962	0.045	[0.968,0.956]	0.964
	60k	0.965	0.036	[0.970,0.960]	0.964
	80k	0.965	0.030	[0.969,0.961]	0.964
	100k	0.965	0.029	[0.969,0.961]	0.964
	500k	0.963	0.013	[0.965,0.961]	0.964
I/O_3	10k	0.964	0.091	[0.977,0.951]	0.964
	20k	0.965	0.068	[0.975,0.955]	0.964
	40k	0.961	0.047	[0.968,0.954]	0.964
	60k	0.960	0.035	[0.965,0.955]	0.964
	80k	0.962	0.032	[0.967,0.957]	0.964
	100k	0.961	0.030	[0.965,0.957]	0.964
	500k	0.964	0.012	[0.966,0.962]	0.964
I/O_4	10k	0.963	0.092	[0.976,0.950]	0.964
	20k	0.964	0.062	[0.973,0.955]	0.964
	40k	0.962	0.048	[0.969,0.955]	0.964
	60k	0.961	0.037	[0.966,0.956]	0.964
	80k	0.961	0.031	[0.965,0.957]	0.964
	100k	0.963	0.027	[0.967,0.959]	0.964
	500k	0.963	0.013	[0.965,0.961]	0.964

Table 5.30: LR—RMF (n=200)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.786	0.044	[0.790,0.782]	0.789
	20k	0.786	0.031	[0.789,0.783]	0.789
	40k	0.789	0.023	[0.791,0.787]	0.789
	60k	0.789	0.019	[0.791,0.787]	0.789
	80k	0.790	0.016	[0.791,0.789]	0.789
	100k	0.790	0.015	[0.791,0.789]	0.789
	500k	0.789	0.007	[0.790,0.788]	0.789
I/O_1	10k	0.963	0.092	[0.971,0.955]	0.964
	20k	0.964	0.064	[0.970,0.958]	0.964
	40k	0.966	0.045	[0.970,0.962]	0.964
	60k	0.966	0.038	[0.969,0.963]	0.964
	80k	0.966	0.033	[0.969,0.963]	0.964
	100k	0.966	0.031	[0.969,0.963]	0.964
	500k	0.965	0.014	[0.966,0.964]	0.964
I/O_2	10k	0.963	0.091	[0.971,0.955]	0.964
	20k	0.962	0.064	[0.968,0.956]	0.964
	40k	0.965	0.045	[0.969,0.961]	0.964
	60k	0.966	0.037	[0.969,0.963]	0.964
	80k	0.965	0.031	[0.968,0.962]	0.964
	100k	0.965	0.029	[0.968,0.962]	0.964
	500k	0.963	0.013	[0.964,0.962]	0.964
I/O_3	10k	0.964	0.091	[0.972,0.956]	0.964
	20k	0.958	0.065	[0.964,0.952]	0.964
	40k	0.961	0.047	[0.965,0.957]	0.964
	60k	0.962	0.036	[0.965,0.959]	0.964
	80k	0.963	0.033	[0.966,0.960]	0.964
	100k	0.964	0.029	[0.967,0.961]	0.964
	500k	0.964	0.012	[0.965,0.963]	0.964
I/O_4	10k	0.962	0.091	[0.970,0.954]	0.964
	20k	0.962	0.064	[0.968,0.956]	0.964
	40k	0.963	0.045	[0.967,0.959]	0.964
	60k	0.963	0.038	[0.966,0.960]	0.964
	80k	0.962	0.031	[0.965,0.959]	0.964
	100k	0.962	0.029	[0.965,0.959]	0.964
	500k	0.963	0.013	[0.964,0.962]	0.964

Table 5.31: LR—RMF (n=500)—Mean Waiting Time

Appendix E

Steady State Performance—Long Runs (LR)

Model:

- *RoundRobin Policy Model With Feedback* (RRMF)

Replications:

- n = 100, 200 and 500

Performance Metrics:

- Mean Queue Size
- Mean Waiting Time

Input Parameters:

- Ranging from 10,000 to 500,000 input requests

Devices	**No. of Customers**	**Simulated Output—E(N_q)**			**Theoretical Value—E(N_q)**
		Mean	**SD**	**CI**	
CPU	10k	0.635	0.036	[0.642,0.628]	0.641
	20k	0.636	0.029	[0.642,0.630]	0.641
	40k	0.641	0.020	[0.645,0.637]	0.641
	60k	0.641	0.016	[0.644,0.638]	0.641
	80k	0.640	0.012	[0.642,0.638]	0.641
	100k	0.640	0.011	[0.642,0.638]	0.641
	500k	0.642	0.005	[0.643,0.641]	0.641
I/O_1	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.049	0.004	[0.050,0.048]	0.048
	40k	0.049	0.003	[0.050,0.048]	0.048
	60k	0.049	0.003	[0.050,0.048]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_2	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.048	0.004	[0.049,0.047]	0.048
	40k	0.049	0.003	[0.050,0.048]	0.048
	60k	0.048	0.002	[0.048,0.048]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_3	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.049	0.004	[0.050,0.048]	0.048
	40k	0.049	0.003	[0.050,0.048]	0.048
	60k	0.049	0.003	[0.050,0.048]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_4	10k	0.048	0.006	[0.049,0.047]	0.048
	20k	0.048	0.004	[0.049,0.047]	0.048
	40k	0.049	0.003	[0.050,0.048]	0.048
	60k	0.049	0.003	[0.050,0.048]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048

Table 5.32: LR—RRMF (n=100)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(N$_q$)**			**Theoretical Value—E(N$_q$)**
		Mean	**SD**	**CI**	
CPU	10k	0.636	0.037	[0.641,0.631]	0.641
	20k	0.641	0.027	[0.645,0.637]	0.641
	40k	0.640	0.019	[0.643,0.637]	0.641
	60k	0.641	0.016	[0.643,0.639]	0.641
	80k	0.642	0.014	[0.644,0.640]	0.641
	100k	0.642	0.012	[0.644,0.640]	0.641
	500k	0.641	0.005	[0.642,0.640]	0.641
I/O_1	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.049	0.004	[0.050,0.048]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.003	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_2	10k	0.048	0.006	[0.049,0.047]	0.048
	20k	0.049	0.004	[0.050,0.048]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.002	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_3	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.049	0.005	[0.050,0.048]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.003	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_4	10k	0.048	0.006	[0.049,0.047]	0.048
	20k	0.049	0.005	[0.050,0.048]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.003	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048

Table 5.33: LR—RRMF (n=200)—Mean Queue Size

Devices	No. of Customers	Simulated Output—E(N_q)			Theoretical Value—E(N_q)
		Mean	**SD**	**CI**	
CPU	10k	0.640	0.037	[0.643,0.637]	0.641
	20k	0.640	0.027	[0.642,0.638]	0.641
	40k	0.641	0.019	[0.643,0.639]	0.641
	60k	0.641	0.016	[0.642,0.640]	0.641
	80k	0.642	0.014	[0.643,0.641]	0.641
	100k	0.642	0.012	[0.643,0.641]	0.641
	500k	0.641	0.005	[0.641,0.641]	0.641
I/O_1	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.049	0.005	[0.050,0.048]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.002	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_2	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.049	0.004	[0.049,0.049]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.003	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_3	10k	0.049	0.006	[0.050,0.048]	0.048
	20k	0.049	0.005	[0.049,0.049]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.003	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048
I/O_4	10k	0.049	0.007	[0.050,0.048]	0.048
	20k	0.049	0.004	[0.049,0.049]	0.048
	40k	0.049	0.003	[0.049,0.049]	0.048
	60k	0.049	0.002	[0.049,0.049]	0.048
	80k	0.049	0.002	[0.049,0.049]	0.048
	100k	0.049	0.002	[0.049,0.049]	0.048
	500k	0.049	0.001	[0.049,0.049]	0.048

Table 5.34: LR—RRMF (n=500)—Mean Queue Size

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.782	0.042	[0.790,0.774]	0.789
	20k	0.783	0.033	[0.790,0.776]	0.789
	40k	0.788	0.022	[0.792,0.784]	0.789
	60k	0.788	0.018	[0.792,0.784]	0.789
	80k	0.787	0.014	[0.790,0.784]	0.789
	100k	0.787	0.013	[0.790,0.784]	0.789
	500k	0.789	0.005	[0.790,0.788]	0.789
I/O_1	10k	0.299	0.034	[0.306,0.292]	0.297
	20k	0.300	0.026	[0.305,0.295]	0.297
	40k	0.301	0.045	[0.305,0.297]	0.297
	60k	0.300	0.038	[0.303,0.297]	0.297
	80k	0.299	0.033	[0.302,0.296]	0.297
	100k	0.299	0.031	[0.301,0.297]	0.297
	500k	0.299	0.014	[0.300,0.298]	0.297
I/O_2	10k	0.299	0.037	[0.306,0.292]	0.297
	20k	0.296	0.026	[0.301,0.291]	0.297
	40k	0.299	0.018	[0.303,0.295]	0.297
	60k	0.298	0.014	[0.301,0.295]	0.297
	80k	0.299	0.012	[0.301,0.297]	0.297
	100k	0.300	0.011	[0.302,0.298]	0.297
	500k	0.300	0.005	[0.301,0.299]	0.297
I/O_3	10k	0.302	0.036	[0.309,0.295]	0.297
	20k	0.299	0.026	[0.304,0.294]	0.297
	40k	0.300	0.020	[0.304,0.296]	0.297
	60k	0.299	0.015	[0.302,0.296]	0.297
	80k	0.299	0.013	[0.302,0.296]	0.297
	100k	0.299	0.012	[0.301,0.297]	0.297
	500k	0.299	0.004	[0.300,0.298]	0.297
I/O_4	10k	0.295	0.036	[0.302,0.288]	0.297
	20k	0.298	0.026	[0.303,0.293]	0.297
	40k	0.300	0.020	[0.304,0.296]	0.297
	60k	0.301	0.016	[0.304,0.298]	0.297
	80k	0.299	0.013	[0.302,0.296]	0.297
	100k	0.299	0.013	[0.302,0.296]	0.297
	500k	0.299	0.005	[0.300,0.298]	0.297

Table 5.35: LR—RRMF (n=100)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.783	0.042	[0.789,0.777]	0.789
	20k	0.788	0.032	[0.793,0.783]	0.789
	40k	0.787	0.022	[0.790,0.784]	0.789
	60k	0.788	0.018	[0.791,0.785]	0.789
	80k	0.789	0.015	[0.791,0.787]	0.789
	100k	0.789	0.014	[0.791,0.787]	0.789
	500k	0.789	0.006	[0.790,0.788]	0.789
I/O_1	10k	0.300	0.037	[0.305,0.295]	0.297
	20k	0.301	0.025	[0.305,0.297]	0.297
	40k	0.299	0.020	[0.302,0.296]	0.297
	60k	0.299	0.016	[0.301,0.297]	0.297
	80k	0.300	0.013	[0.302,0.298]	0.297
	100k	0.300	0.012	[0.302,0.298]	0.297
	500k	0.299	0.005	[0.300,0.298]	0.297
I/O_2	10k	0.296	0.036	[0.301,0.291]	0.297
	20k	0.299	0.026	[0.303,0.295]	0.297
	40k	0.299	0.017	[0.301,0.297]	0.297
	60k	0.301	0.015	[0.303,0.299]	0.297
	80k	0.300	0.013	[0.302,0.298]	0.297
	100k	0.300	0.011	[0.302,0.298]	0.297
	500k	0.299	0.005	[0.300,0.298]	0.297
I/O_3	10k	0.299	0.035	[0.304,0.294]	0.297
	20k	0.300	0.027	[0.304,0.296]	0.297
	40k	0.299	0.019	[0.302,0.296]	0.297
	60k	0.299	0.015	[0.301,0.297]	0.297
	80k	0.299	0.013	[0.301,0.297]	0.297
	100k	0.299	0.012	[0.301,0.297]	0.297
	500k	0.300	0.004	[0.300,0.299]	0.297
I/O_4	10k	0.298	0.038	[0.303,0.293]	0.297
	20k	0.300	0.026	[0.304,0.296]	0.297
	40k	0.299	0.020	[0.302,0.296]	0.297
	60k	0.299	0.016	[0.301,0.297]	0.297
	80k	0.298	0.013	[0.300,0.296]	0.297
	100k	0.299	0.013	[0.301,0.297]	0.297
	500k	0.299	0.005	[0.300,0.298]	0.297

Table 5.36: LR—RRMF (n=200)—Mean Waiting Time

Devices	**No. of Customers**	**Simulated Output—E(W)**			**Theoretical Value—E(W)**
		Mean	**SD**	**CI**	
CPU	10k	0.788	0.042	[0.792,0.784]	0.789
	20k	0.787	0.030	[0.790,0.784]	0.789
	40k	0.789	0.021	[0.791,0.787]	0.789
	60k	0.789	0.017	[0.791,0.787]	0.789
	80k	0.790	0.016	[0.791,0.789]	0.789
	100k	0.789	0.014	[0.791,0.787]	0.789
	500k	0.789	0.006	[0.790,0.788]	0.789
I/O_1	10k	0.299	0.036	[0.302,0.296]	0.297
	20k	0.299	0.027	[0.301,0.297]	0.297
	40k	0.299	0.019	[0.301,0.297]	0.297
	60k	0.300	0.015	[0.301,0.299]	0.297
	80k	0.299	0.013	[0.300,0.298]	0.297
	100k	0.299	0.012	[0.300,0.298]	0.297
	500k	0.299	0.005	[0.300,0.299]	0.297
I/O_2	10k	0.298	0.035	[0.301,0.295]	0.297
	20k	0.300	0.026	[0.302,0.298]	0.297
	40k	0.300	0.018	[0.302,0.298]	0.297
	60k	0.300	0.015	[0.301,0.299]	0.297
	80k	0.300	0.013	[0.301,0.299]	0.297
	100k	0.300	0.011	[0.301,0.299]	0.297
	500k	0.299	0.005	[0.300,0.299]	0.297
I/O_3	10k	0.299	0.037	[0.302,0.296]	0.297
	20k	0.299	0.027	[0.301,0.297]	0.297
	40k	0.299	0.019	[0.301,0.297]	0.297
	60k	0.300	0.016	[0.301,0.299]	0.297
	80k	0.300	0.013	[0.301,0.299]	0.297
	100k	0.299	0.011	[0.300,0.298]	0.297
	500k	0.300	0.005	[0.300,0.299]	0.297
I/O_4	10k	0.301	0.039	[0.304,0.298]	0.297
	20k	0.299	0.026	[0.301,0.297]	0.297
	40k	0.299	0.019	[0.301,0.297]	0.297
	60k	0.299	0.015	[0.300,0.298]	0.297
	80k	0.299	0.012	[0.300,0.298]	0.297
	100k	0.299	0.012	[0.300,0.298]	0.297
	500k	0.299	0.005	[0.300,0.299]	0.297

Table 5.37: LR—RRMF (n=500)—Mean Waiting Time

Bibliography

[AB00] L. Aversa and A. Bestavros. Load Balancing a Cluster of Web Servers: Using Distributed Packet Rewriting. In *Proceedings of the 2000 IEEE International Conference on Performance, Computing and Communications (IPCCC 2000)*, pages 24–29, February 2000.

[ADZ99] M. Aron, P. Druschel, and Z. Zwaenepoel. Efficient Support for P-HTTP in Cluster-Based Web Servers. In *Proceedings of the 1999 USENIX Annual Technical Conference*, pages 185–198, June 1999.

[Aka99] Akamai Technologies, Inc. Akamai. http://www.akamai.com, 1999.

[All90] A. Allen. *Probability, statistics, and queueing theory with computer science applications*. Academic Press Professional, Inc., 1990.

[APE96] E. Anderson, D. Patterson, and E.Brewer. The Magicrouter, an Application of Fast Packet Interposing. http://www.cs.berkeley.edu/ eanders/projects/magicrouter, May 1996.

[App04] Apple. Quick Time VR. http://www.apple.com/quicktime/qtvr, 2004.

[AVI04] AVI. Audio Video Interleave. http://www.audio-video-affair.com/audiovideointerleave.html, 2004.

[AYHI96] D. Andersen, T. Yang, V. Holmedahl, and O. Ibarra. SWEB: Towards a Scalable World Wide Web Server on Multicomputers. In *Proceedings of the 10th International Conference of Parallel Processing Symposium (IPPS)*, pages 850–856, April 1996.

[BGMT98] G. Bolch, S. Greiner, H. Meer, and K. Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience, 1998.

[Bla00] U. Black. *Internet Architecture*. Prentice Hall PTR, 2000.

[BM98] M. Beck and T. Moore. The Internet2 Distributed Storage Infrastructure Project: An Architecture for Internet Content Channels. *Computer Networks and ISDN Systems*, 30(22–23):2141–2148, November 1998.

[CC01] E. Casallicchio and M. Colajanni. A Client-aware Dispatching Algorithm for Web Clusters Providing Multiple Services. In *Proceedings of the 10th International World Wide Web Conference (WWW10)*, pages 535–544, May 2001.

[CCCY02] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Yu. The State of the Art in Locally Distributed Web-server Systems. *ACM Computing Surveys*, 34(3):263–311, June 2002.

[CCY99] V. Cardellini, M. Colajanni, and P. Yu. Dynamic Load Balancing on Web Server Systems . *IEEE Internet Computing*, 3(3):28–39, May–Jun 1999.

[Cis00] Cisco Systems, Inc. LocalDirector. http://www.cisco.com, 2000.

[Cis02] Cisco Systems, Inc. DistributedDirector. http://www.cisco.com, 2002.

[CYD98] M. Colajanni, P. Yu, and D. Dias. Analysis of Task Assignment Policies in Scalable Distributed Web-server Systems. *IEEE Transactions on Parallel and Distributed Systems*, 9(6):585–600, June 1998.

[DCH$^+$97] O. P. Damant, P. E. Chung, Y. Huang, C. Kintala, and Y-M. Wang. ONE-IP: Techniques for Hosting a Service on a Cluster of Machines. *Computer Networks and ISDN Systems*, 29(8–13):1019–1027, April 1997.

[DGLS99] B. Devlin, J. Gray, B. Laing, and G. Spix. Scalability terminology: Farms, clones, partitions and pack: Racs and raps. Technical Report MS–TR–99–85, Microsoft Research, 1999.

[DKMT96] M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A Scalable and Highly Available Web Server. In *Proceedings of the 41st IEEE International Computer Conference (COMPCON96)*, pages 85–92, March 1996.

[DSL04] DSL. DSL. http://www.dsl.com, 2004.

[Eri95] Eric Meyer. Introduction to HTML. http://www.cwru.edu/help/introHTML/toc.html, 1995.

[F5N00] F5Networks, Inc. BIG-IP. http://www.f5labs.com, 2000.

[Fou02] Foundry Networks', Inc. ServerIron. http://www.foundrynet.com/products/webswitches/serveriron, 2002.

[GEP02] M. Ganeshan, R. Eskicioglu, and R. Peters. A taxonomy of commercial and prototype web server systems. Technical Report MS–TR–99–85, University of Manitoba, 2002.

[HKM98] G. Hunt, G. King, and R. Mukherjee. Network Dispatcher: A Connection Router for Scalable Internet Services. In *Proceedings of the 7th International World Wide Web Conference (WWW7)*, pages 347–357, April 1998.

[IBM01] IBM. IBM WebSphere Edge Server. http://www.ibm.com/software/webservers/edgeserver, 2001.

[Jac63] J. Jackson. JobShop-Like Queueing Systems. *Management Science*, 10:131–142, 1963.

[Jai91] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[Joi04] Joint Photographic Experts Group, JPEG. JPEG Committee. http://www.jpeg.org, 2004.

[KBM94] E. Katz, M. Butler, and R. McGrath. A Scalable HTTP Server: The NCSA Prototype. *Computer Networks and ISDN Systems*, 27:155–164, November 1994.

[Ken00] Ken Holman. XSLT. http://www.xml.com/pub/a/2000/08/holman, 2000.

[Ken04] Ken Holman. XSL. http://www.w3.org/Style/XSL, 2004.

[KR03] J. Kurose and K. Ross. *Computer Networking*. Pearson Education, Inc., 2003.

[LK91] A. Law and W. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, Inc., 1991.

[LMV02] P. L'Ecuyer, L. Meliani, and J. Vaucher. Ssj: A framework for stochastic simulation in java. In *Proceedings of the 2002 Winter Simulation Conference*, pages 234–242, December 2002.

[LVS98] LVS Project. Virtual Server Project. http://www.linuxvirtualserver.org, 1998.

[MA02] D. Menasce and V. Almeida. *Capacity Planning for Web Services*. Prentice Hall PTR, 2002.

[Mac87] M. MacDougall. *Simulating Computer Systems Techniques and Tools*. MIT Press, 1987.

[Mac04] MacroMedia. MacroMedia. http://www.macromedia.com, 2004.

[MB98] D. Maltz and P. Bhagwat. Application Layer Proxy Performance using TCP Splice. Technical Report RC–21139, IBM T.J. Watson Research Center, 1998.

[Mic00] Microsoft Corporation. Network Load Balancing. http://www.microsoft.com, 2000.

[Mov04] Movie Picture Experts Group. Movie Picture Experts Group. http://www.chiariglione.org/mpeg, 2004.

[NBK02] E. Nahum, T. Barzilai, and D. Kandlur. Performance issues in www servers. *IEEE/ACM Transactions on Networking (TON)*, 10:2–11, Feb 2002.

[NM00] S. Nadimpalli and S. Majumdar. Techniques for Achieving High Performance Web Servers. In *Proceedings of the 2000 International Conference on Parallel Processing (ICPP)*, pages 233–241, August 2000.

[Nor98] Norman Walsh. XML. http://www.xml.com/pub/a/98/10/guide0.html, 1998.

[Nor00] Nortel Networks, Ltd. Alteon Web Switch. http://www.nortelnetworks.com/products/01/alteon/webswitch, 2000.

[Nor02] Nortel Networks, Ltd. Alteon Web OS. http://www.nortelnetworks.com/products/01/alteon, 2002.

[PAB$^+$98] V. Pai, M. Aron, G. Banga, G. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution on Cluster-Based Network Servers. In *Proceedings of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, November 1998.

[Res00] Resonate, Inc. Central Dispatch. http://www.resonate.com, 2000.

[SGR00] T. Schroeder, S. Goddard, and B. Ramamurthy. Scalable Web Server Clustering technologies. *IEEE Network*, 14(3):38–45, May–Jun 2000.

[SHB00] Bianca Schroeder and Mor Harchol-Balter. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. In *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, page 211, 2000.

[TA01] Y. Teo and R. Ayani. Comparison of Load Balancing Strategies on Cluster-based Web Servers. *Transactions of the Society for Modeling and Simulation*, pages 185–195, Nov–Dec 2001.

[Tri02] K. Trivedi. *Probability and Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley and Sons, Inc., 2002.

[ZBCS99] X. Zhang, M. Barrientos, J. B. Chen, and M. Seltzer. HACC An Architecture for Cluster-Based Web Servers. In *Proceedings of 3rd Usenix Windows NT Symposium*, pages 155–164, July 1999.

[Zeu02] Zeus Technology, Inc. Zeus Load Balancer. http://www.zeus.com, 2002.