INFORMATION TO USERS

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

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

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

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

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

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

UMI[®]

High Performance Web Servers

by

Sucheta Nadimpalli, B.E.

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

Ottawa-Carleton Institute for Electrical and Computer Engineering

Faculty of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

July 2000

© Copyright 2000, Sucheta Nadimpalli

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

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

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

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

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

0-612-57734-1

Canada

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

High Performance Web Servers

submitted by **Sucheta Nadimpalli, B.E.**

in partial fulfillment of the requirements for
the degree of Master of Engineering

Thesis Supervisor

Chair, Department of Systems and Computer Engineering

Carleton University

30th August 2000

A B S T R A C T

This thesis analyzes a number of performance improvement techniques to achieve high performance Web servers. These techniques include parallelization of Web page access and retrieving embedded objects from multiple servers concurrently, data replication and effective embedded object allocation, and load balancing. A performance prototype based on the Apache server is used for evaluating these techniques. The client perceived response time for a Web page request is chosen as the performance metric.

The effect of different workload characteristics on the benefit of using parallelism in Web page access have been investigated. One of the important observations is that parallelization of Web requests gives a significant improvement in performance. Replication of servers is observed to be beneficial especially when the embedded objects in a Web page are not evenly distributed across servers. Also load balancing at the dispatcher using a simple round robin policy for backend server selection gives a better performance compared to the default random policy used by the Apache server.

Acknowledgements

I would like to take this opportunity to express my thanks to Professor Majumdar for his guidance throughout the research. This thesis would not have seen the light of the day but for his continuous support and encouragement. Working with Professor Majumdar has been a valuable experience for me.

I wish to thank the faculty members of the RADS group for their help and support.

I sincerely acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), and Communications and Information Technology Ontario (CITO) for this research.

Finally, I would like to thank my family especially my husband and my daughter for their patience and understanding while I was pursuing my studies.

TABLE OF CONTENTS

List of Tables.....	vii
List of Figures.....	viii
1. INTRODUCTION.....	1
1.1. Motivation.....	1
1.2. Goals of the Thesis	3
1.3. Contributions and Publications.....	5
1.4. Thesis Organization	6
2. REVIEW OF EXISTING TECHNIQUES TO IMPROVE WORLD WIDE WEB PERFORMANCE.....	8
2.1. Workload Characterization of the World Wide Web.....	8
2.1.1. Benchmarks and Workload generators for testing Web servers	10
2.2. Hyper Text Transfer Protocol Enhancements	13
2.3. Study of Web Performance.....	15
2.3.1. Web Performance Measurement.....	16
2.4. OS Support and Resource Management for Scalable Web Servers.....	16
2.5. Caching in the Web.....	18
2.5.1. Caching models	19
2.5.2. Performance metrics	22
2.5.3. Cache replacement policies.....	22
2.5.4. Cache Coherency.....	25
2.5.5. Other Cache Design Issues.....	26
2.5.6. Prefetching	27
2.6. Replication	29
2.6.1. Data Dissemination.....	30
2.6.2. Replica Server Selection	31
2.7. Web Server Clustering	32
2.7.1. Request Distribution Schemes.....	32
2.7.2. Web Server Architectures for load balancing.....	34
2.8. Summary.....	36
3. SYSTEM ARCHITECTURE AND WORKLOAD FOR EVALUATING THE PROPOSED TECHNIQUES	39
3.1. Introduction	39
3.2. System Architecture.....	40
3.3. Workload and Instrumentation.....	43
3.3.1. Workload Generation Tool.....	44
3.3.2. Client setup	45
3.3.3. Workload Parameters.....	47
3.3.4. Performance Metrics.....	50

3.4.	System Configuration Parameters	52
3.4.1.	Server Configuration	52
3.4.2.	Rationale for choosing the Server Configuration Parameters.....	53
3.4.3.	System Configurations.....	55
3.5.	Summary.....	57
4.	PERFORMANCE STUDY OF SYSTEMS WITH SINGLE WEB PAGE CLASS.....	59
4.1.	Parallelization of Web page access.....	59
4.1.1.	Granularity of Parallelism.....	60
4.1.2.	Effect of increasing the number of back-end servers.....	61
4.2.	Request distribution policies at dispatcher	65
4.3.	Data replication and allocation.....	67
4.3.1.	Effect of data replication.....	68
4.3.2.	Effect of data allocation.....	69
4.3.3.	Impact of service demand	71
4.3.4.	Impact of number of embedded objects in a Web page	73
4.4.	Discussion.....	74
4.5.	Summary.....	76
5.	PERFORMANCE STUDY OF SYSTEMS WITH MULTIPLE WEB PAGE CLASSES	78
5.1.	Introduction	78
5.2.	Serial Vs Parallel Web page access	78
5.2.1.	Web page classes with no variabilities.....	79
5.2.2.	Intra-page variability	82
5.2.3.	Inter-page variability	84
5.2.4.	Intra-page and inter-page variability.....	86
5.3.	Performance study of a Popular Web site.....	87
5.4.	Discussion.....	90
5.5.	Summary.....	91
6.	CONCLUSIONS	93
6.1.	Summary.....	93
6.2.	Conclusions	95
6.3.	Future Research.....	98
	REFERENCES	100
	APPENDIX A	109
	APPENDIX B	110

List of Tables

3.1.	Test Parameters for performance study of systems with single Web page class	49
3.2.	Test Parameters for performance study of systems with two Web page classes	49
3.3.	Apache server configuration parameters	54
3.4.	Effect of server configuration parameters	55
4.1.	Granularity of Parallelism	60
5.1.	The Effect of Parallelism with a higher number of embedded objects in a Web page	81
5.2.	Popular Web site hosting Web pages that can be classified into different Web page classes based on the number of embedded objects	87

List of Figures

1.1.	A typical Web page design	2
2.1.	Hierarchical Caching Architecture	20
2.2.	A DNS based Web server cluster	35
3.1.	System Architecture	41
4.1.	Performance of a single class system: Response time	62
4.2.	Performance of a single class system: Throughput	62
4.3.	Effect of the number of back-end servers on response time for different service times	63
4.4.	Effect of the number of back-end servers on response time for different number of embedded objects	64
4.5.	Comparison of Random and Round Robin request distribution policies at the dispatcher	66
4.6.	Effect of data replication on the response time	68
4.7.	Effect of embedded object allocation on the response time	70
4.8.	Effect of embedded object allocation on the response time for a higher number of clients	71
4.9	The impact of service demand on the performance of PR and P (Allocation = 25 %)	72
4.10.	The impact of service demand on the performance of PR and P (Allocation = 25 %) at a higher number of clients	72
4.11.	The impact of number of embedded objects in a Web page on performance	73
4.12.	The impact of number of embedded objects in a Web page on the response time at a higher number of clients	74
5.1.	Serial Vs Parallel for no variabilities among Web page classes	81
5.2.	The Impact of Intra-Page Variability	83
5.3.	The Effect of Inter-Page Variability	84
5.4 .	The Effect of Inter-Page Variability with equal access to both Web Page Classes	85
5.5.	Systems characterized by both Inter-Page and Intra-Page Variabilities	86
5.6.	Inter-page variability for a Popular Web site with multiple Web page classes	89

CHAPTER 1

1. INTRODUCTION

1.1. Motivation

Due to a continuous growth in the volume of information available on the Web the World Wide Web user population is growing rapidly. The growing user population has led to the proliferation of Web sites such as dot-coms, music stores, and news sites which cater to different classes of audience. The development of several application programming interfaces such as NSAPI, FastCGI etc. and secure socket layer protocol (SSL) has aided in the growth of e-commerce systems.

Web pages are becoming more and more intensive due to embedded graphic images, navigation buttons or Web ads used in an e-commerce site for example. Figure 1.1 shows a typical Web page which consists of a main HTML document and various embedded objects that include both graphic images and text files. An average of 17 embedded objects per Web page has been observed in a study of popular Web sites described in [ABH-00].

Existing Web workload characterization studies show that most of the requests are for static documents although the demand for dynamically generated Web pages is increasing. Dynamic data is observed to give rise to a heavy server demand on the 1998

Olympic Web site [CHA-98]. The Web site recorded a hit of 110,414 hits in a minute during the women's figure skating championship event. Clearly, robust high performance servers are necessary to handle such computational demands. Other applications that impose a heavy demand on a Web server include large compressed files, movie clips, extremely large audio and video files and dynamic pages generated through cgi and JavaScripts. Effective management of servers is crucial for achieving user satisfaction on such environments.

Web users perform browsing as well as financial transactions as in the case of an e-commerce server. E-commerce servers have to handle a great deal of both static and dynamic Web page requests. A number of resource management strategies for the performance improvement of e-commerce servers is discussed in [MEN-99]. High

performance servers are essential since they lead to higher revenue and profits for the Web store owners.

Earlier efforts have put more emphasis on improving the Web performance by solving the problems caused by network traffic. Several modifications have been proposed, for example, in the new version of the Hyper Text Transfer Protocol i.e. HTTP/1.1 [FIE-97]. However there is a continuous increase in modem speeds and the bandwidths of the backbone network that interconnect the servers with the clients. As the communication bandwidth available to a client increases, the size of Web documents will tend to increase and each client would generate more and more Web page requests to the server and also download extremely large files [ARL-99]. This pushes the performance bottleneck to the server system and leads to an increase in the client's perceived latency for a Web page request.

Clearly there is a need for effective server architectures that are scalable, robust and capable of handling large traffic volumes. In addition, effective resource management techniques are essential for achieving high performance Web servers.

1.2. Goals of the Thesis

This research is motivated by the need for techniques for achieving high performance Web servers. A number of techniques that include parallelization of Web pages, embedded object replication and load balancing are investigated. The thesis objectives

are summarized as follows.

- Investigate the performance benefit due to parallelization of individual Web page access and retrieval of data from multiple servers in parallel. This thesis considers systems characterized by both a single class and multiple classes of Web pages. Each Web page class is characterized by a given number of embedded objects and service demands.
- Study the effect of workload characteristics such as inter-page and intra-page variabilities in service demands of embedded objects in a Web page on the performance of the parallel Web server.
- Investigate the effectiveness of data replication on the client perceived latency for Web server systems hosting Web pages with identical workload characteristics. Identify effective ways of allocating embedded objects in a Web page to the back-end servers and study the resulting system performance.
- Compare the effect of different request distribution policies to the back-end servers at the dispatcher on the client perceived latency of a Web page request.

Some of these investigations are made for systems with a single Web page class while the others are made for systems with multiple Web page classes.

1.3. Contributions and Publications

Using a performance prototype that consists of re-engineered Apache servers and synthetic clients running on Pentium II PC's under Linux, this research has investigated a number of techniques for improving system performance. These include parallelization of Web page requests, embedded object replication and allocation as well as load balancing. The important contributions of this research to the state of the art are briefly summarized:

- A significant performance benefit can accrue from parallelizing a Web page request and retrieving the embedded objects in a Web page from multiple servers in parallel. This is observed for a variety of workloads that have been considered in this study.
- Parallelization of Web page requests yields a performance benefit over the serial configuration when the service demand of an embedded object is higher than a threshold.
- On systems with multiple Web page classes, higher performance benefits accrue from accessing embedded objects of a Web page in parallel when the Web page classes on a system exhibit large inter-page variability. The effect of intra-page variability on performance is less pronounced.
- In the absence of data replication, a proper embedded object allocation scheme in conjunction with parallelization of Web page requests can significantly improve system performance.

- Although lesser in impact, the choice of the load balancing policy does affect system performance significantly.

The results of this research are applicable primarily to systems in which the servers play a dominant role in determining system performance.

A part of this research is to be published in the Proceedings of the International Conference on Parallel Processing (ICPP-2000) [NAD-00].

1.4. Thesis Organization

Chapter 2 is a review of the existing techniques and approaches in literature for improving Web performance. Observations of the different workload characterization studies, as well as the protocol modifications to improve Web latency are presented. Several other Web performance improvement techniques such as caching and replication in the distributed Web environment, and Web server clustering are also discussed.

The system architecture and the experimental setup used for evaluating the proposed techniques is described in Chapter 3. A brief description of Apache server configuration and the rationale behind the values chosen for the different configuration parameters of the servers is presented. Httpperf, the workload generator chosen for the experiments and the instrumentation used for the measurement of performance are presented next.

Chapter 4 presents a measurement study of the system to understand the impact of the granularity of parallelism and the use of additional resources such as higher number of back-end servers on the performance. Web server systems with a single Web page class are considered for all the experiments discussed in this chapter. A discussion of the experiments conducted to investigate the effect of parallelization of Web page requests on the performance of the Web server system is presented. Effect of embedded object replication and allocation on client response time for different workload characteristics is also included. Finally different request distribution strategies at the dispatcher to achieve load balancing among the back-end servers are compared.

Chapter 5 considers Web server systems consisting of multiple Web page classes for all the experiments. A detailed discussion of the experiments conducted to understand the impact of various workload characteristics such as inter-page and intra-page variabilities in service demands of the embedded objects on performance is presented. Each experiment description is followed by a discussion of the experimental results.

Chapter 6 summarizes the conclusions and offers directions for future research.

CHAPTER 2

2. REVIEW OF EXISTING TECHNIQUES TO IMPROVE WORLD WIDE WEB PERFORMANCE

This chapter is a literature study of various existing techniques for improving World Wide Web performance. Each approach has its shortcomings. Section 2.1 discusses the main observations of the Web workload characterization studies and presents a brief discussion of the benchmarks and workload generators for evaluating Web server performance. Several HTTP protocol enhancements are described in Section 2.2. Section 2.3 presents a discussion of studies that focus on Web server performance measurement. Section 2.4 discusses the OS and resource management strategies used for Web servers. Section 2.5 presents caching techniques for improving Web performance and related issues. Section 2.6 discusses data replication. Web server clustering is discussed in Section 2.7. Finally in Section 2.8 a brief summary is presented.

2.1. Workload Characterization of the World Wide Web

In order to be able to devise techniques for improving Web performance, a good understanding of WWW workload is the key. Some researchers have focussed their efforts to characterize the workload for Internet Web servers and to identify the workload invariants [ARL-96A, ARL-99]. Several proxy or server logs are analyzed, and also

static Web topology traversal is used for gathering the data required for these characterization studies.

Existing work on workload characterization shows that most of the traffic generated is for static Web pages. Although with growing e-commerce and increasing communication bandwidth availability to the user, the demand for dynamically generated pages and large files is also increasing. Other important observations of the earlier workload characterization study are listed below:

- Some of the files are more *popular* (i.e. more references are made to a small number of files and a large number of files are accessed only once). The active set of files changes over time although some files remain popular for extended period of time [ARL-99].
- Web file size is seen to have a *heavy tailed distribution*, i.e. there is a very large variability in the Web file sizes [ARL-96A].
- The network traffic of Web requests is shown to exhibit *self-similarities* [CRO-97]. This means that the nature of Web traffic is bursty across several time scales. This is attributed to the Web file sizes, the interarrival times of Web requests (which are shown to have a heavy tailed distribution as well), Web file popularity and the caching of Web documents. Self-similarity in traffic is shown to have a negative impact on network performance. This is because the distribution of queue lengths under self-similar traffic

decays more slowly [PAR-96].

- Clients are (more) willing to download extremely large files when access speeds are higher [ARL-99].

A recent study attempts to characterize Web server access patterns by using mathematical models [IYE-98]. Spectral analysis of observed data from real Web server systems is done to construct time-series processes. The authors claim that the mathematical representations derived make it possible to scale the request arrival rate at the Web server so that Web server performance can be examined at all traffic intensities of interest. This forms the basis for a new class of effective benchmarks which aid in evaluating Web server performance.

2.1.1. Benchmarks and Workload generators for testing Web servers

The benefit of workload characterization would be that it assists in developing benchmarks or workload generators for evaluating the Web server system's performance. Two types of workload generation techniques are currently being used for testing Web server performance namely: trace based and analytic approach based.

Trace based workload generation collects traces of busy Internet servers and proxies and replays them on the system under test. The argument for using traces is that workloads are too complicated to be described by a simple formula. The disadvantage of the trace

based approach is that they do not capture the changing behavior of a user or the behavior of a different user set. The analytic approach on the other hand uses various mathematical models for generating various workload characteristics and generates outputs that adhere to these models.

Workload of the seemingly ever evolving Web is different from traditional distributed systems due to unusual features such as highly variable demands [ARL-96A, CRO-97]. These are exhibited as variability in CPU demands and number of open connections. Web server performance study is essential in order to devise new performance improvement techniques and to evaluate them.

Performance measurement and synthetic workload generation tools such as SURGE [BAR-98], httpperf and popular Web server benchmarks such as SPECWeb [SPE-96], Webstone [WEB-00] are essential and play an important role in the study of Web server performance. A detailed discussion of the httpperf tool that has been used as the workload generator for this research is presented in Chapter 4.

WebStone and SPECWeb use load generator client processes on either a single node or multiple nodes to generate synthetic load. The workload supported by Webstone consists of HTML page retrieval and CGI requests. In the case of the SPECWeb benchmark, the workload generating client processes send HTTP GET requests to the Web server. The data files that are requested by the client processes are copied on to the server before the test. The data set that is provided with the benchmarks is developed by analyzing several

server logs. The Web server throughput and/or the request response time are typically measured and reported at the end of the test. The throughput metric is measured in bytes per second or in HTTP operations per second.

SURGE, the Scalable URL Reference Generator, generates workload that is based on empirical measurements of Web file size distribution, request size distribution, relative file popularity, embedded file references, temporal locality of reference and idle periods for individual users [BAR-98]. The Zipf distribution is used for modeling file popularity. Temporal locality is modeled by a lognormal distribution while a Pareto distribution is used for modeling request sizes and embedded references in a Web page. The data set is precomputed based on these distribution models and copied on to the server. SURGE uses multiple user equivalents (UE) to make requests for the files on the server. Each UE is implemented as a thread. In the experimental environment discussed in [BAR-98], a maximum of 50 UE threads were run on a SURGE host. A user equivalent is a program in an endless loop that alternates between times during which Web file requests are made (ON) and idle times (OFF). The two OFF periods namely Active OFF and Inactive OFF correspond to the requested file transfer time and user think time respectively. Lognormal and Pareto distributions are used for modeling the file size.

WAN-based Webs have network characteristics, such as the delay and packet loss, that differ from LANs on which most of the experiments and simulations are carried out. Hence evaluating Web server performance under overload and bursty traffic conditions becomes very important. In order to understand the impact of these network

characteristics on Web server performance, construction of a special set of client processes called scalable (S) clients has been proposed in [BAN-97]. An S-Client consists of a pair of processes – the connection establishment process and the connection handling process. WAN delays are simulated by a router that connects the client and server.

2.2. Hyper Text Transfer Protocol Enhancements

According to HTTP/1.0, the client needs to open up a new connection for each request [BER-98]. As most HTTP requests are for small files, the TCP connection establishment overhead is relatively higher and this leads to a slow start. Also the connection setup requires certain processing overhead at the server. A persistent single connection specified by using a *KeepAlive* header in HTTP/1.0, is used by the clients for transferring multiple files from the server. This way the connection establishing overhead is eliminated but the client still would make a request and wait for the response before making the next request. Opening parallel TCP connections could lead to increased congestion. Hence pipelining of requests on a persistent connection was proposed [PAD-94] which allows a client or Web browser to send a request to the server asking it to respond by sending all the associated embedded objects of a given Web page. These and several other improvements to the Hyper Text Transfer Protocol have been proposed in HTTP/1.1 [FIE-97].

A recent study [KRI-99] compares and highlights the differences between HTTP/1.0 and

HTTP/1.1. It discusses the protocol changes in various areas like extensibility, caching, bandwidth optimization, network connection management and security to name a few. Among the new headers introduced for caching are the *Etag* or the entity tag used for cache validation and cache control directives to be transmitted in requests and responses [DIN-96]. For bandwidth optimization, the *Range* header is specified which lets a user specify one or more contiguous ranges of bytes when the client needs only a part of the resource. Compression can be done either as content-coding or transfer-coding, a new addition in HTTP/1.1. The chunked transfer-encoding lets the sender break the message body into chunks of arbitrary length. This relieves the sender from having to buffer the entire message. The main observation of the comparison study in [KRI-99] is that although several useful new provisions are available in HTTP/1.1, the complexity of this new protocol increases the programming overhead for implementors of clients, servers and proxy caches.

A performance comparison of HTTP/1.0 and HTTP/1.1 is done in [BAR-99A]. The main observation of the authors is that when the server is not the bottleneck, the network demands of HTTP/1.1 are somewhat lower than those of HTTP/1.0 but when the server CPU is the bottleneck, there is no significant difference in performance between HTTP/1.0 and HTTP/1.1. The authors also observed that when memory on the server is limited and the disk system becomes the bottleneck, HTTP/1.1 performs significantly worse than HTTP/1.0. Based on these observations the authors propose a new connection management policy for HTTP/1.1 and term it as 'early close' where clients close their HTTP connections after the transfer of each Web page.

2.3. Study of Web Performance

Capacity planning and performance modeling is an important aspect of any environment. Capacity planning is an important aspect of research for distributed Web environments more so due to the burstiness and heavy-tailed characteristics of the workload. The dynamic nature of Web transactions makes capacity planning for e-commerce servers quite important.

Component-level performance models using queueing network theory have been proposed for the Web in [MEN-98]. The demands on system resources, waiting times, response times and throughputs are computed using open models for Web servers and closed models for Intranets.

Layered queueing models (LQM) for Web server capacity planning have been proposed in [DIL-98]. The layered queueing models are essentially extended queueing network models that consider contention for processes as well as physical resources such as processors and disks. These models aid in estimating queueing delays at servers. The various metrics considered in this study are the server's response time, client response time, server throughput and service demands. The use of LQM to consider tradeoffs in server configuration for several different network topologies is discussed. It is shown that an effective server configuration depends greatly on network characteristics.

2.3.1. Web Performance Measurement

Web server performance study for various workloads is important in order that the server provides the necessary service to its clients or users. WAN based Webs have network characteristics that differ from the LANs on which most Web servers are evaluated. WANs are characterized by high packet loss as well as long and variable delays due to congestion. These long and variable delays in the WANs cause long SYN-RCVD queues in the server's listening socket [BAN-97]. The Wide Area Web Measurement (WAWM) project uses an infrastructure that enables simultaneous measurements of Web client performance, network performance and Web server performance by the use of active and passive tools [BAR-99B]. WebMonitor, one of the tools used, measures activity and resource consumption both in the operating system and the Web server, with minimal overhead [ALM-97]. The proposed infrastructure is expected to be helpful in exploring the interaction between server and network behavior and in identifying cases in which the two do not inter-operate well.

2.4. OS Support and Resource Management for Scalable Web Servers

Apache, the most popular Web server's, multiprocess architecture discussed further in Chapter 3, suffers from large context switching overheads. A number of recent Web servers use a single process event-driven architecture [SQU-96, PAI-97]. These servers designed for multiprocessors use one thread or process per processor. An event driven server uses the *select* system call to simultaneously wait for events on all connections

being handled by the server. Another alternative Web server architecture is the single process multi-threaded architecture. Each connection is assigned to a dedicated thread. Since efficient support for huge pools of threads is not always available, a hybrid approach is used where a moderate-size pool of threads is multiplexed among many connections and events are used to control the assignment of connections to threads. The latter approach gets around the problem associated with the blocking disk I/O system calls of event-driven servers while keeping thread management overhead low.

Most existing operating systems provide inadequate support for large scale Internet server applications [BAN-98]. UNIX interfaces to improve scalability and fine-grained scheduling and resource management have been proposed in [BAN-98]. Often a process is the schedulable entity and the chargeable entity for the allocation of resources such as CPU time and memory. It is noted that in most operating systems the kernel does not account for resource consumption during the processing of network traffic, which can lead to scheduling anomalies and decreased throughput of the Internet servers. Resource container, an operating system abstraction, is defined in [BAN-98]. The resource container encompasses all system resources that the server uses to serve a particular connection. All kernel processing for a particular connection is charged to the appropriate resource container and scheduled at the priority of the container.

It has been emphasized that optimizations are necessary to build efficient and scalable Web servers and a framework called JAWS for building high performance servers over high speed networks is proposed in [HU-97]. Static and dynamic adaptations that can be

applied to develop high-performance Web servers are proposed.

The static adaptations include configuring an event dispatching model that is customized for OS platform-specific features. Common dynamic adaptations include prioritized request handling, caching strategies, and threading strategies. The authors adapted the proposed JAWS Web server framework to leverage the native asynchronous dispatching mechanisms provided by Windows NT. Their observations are that for small files, the synchronous Thread Pool strategy provides a better overall performance while under heavy loads and with large file transfers, the asynchronous model provides a more consistent quality of service. Thus, under Windows NT it is suggested that an optimal Web server should adapt itself dynamically to either an event dispatching or a file I/O model, depending on the server's workload.

In another study, connection scheduling based on a policy that gives preferential service to short connections has been proposed [CRO-99]. Comparison of different scheduling policies with the (Apache-like) connection size independent policy is made. The shortest connection first performed better than the default Apache server policy in terms of mean delay.

2.5. Caching in the Web

Caching has been observed to be one promising approach for improving the performance of the World Wide Web. The technical issues related to caching techniques

are discussed in [BAE-97, W3C-97]. One of the strong motivations for caching is the varying degree of popularity of Web documents. Some files are more popular than others. The more popular files can be cached closer to the client thus reducing the bandwidth consumption significantly. It also helps in reducing the load on the server and thus helps in improving the latency for uncached interactions

2.5.1. Caching models

Caching can be done at the client using a local disk and/or in-memory cache in Web browsers. Also caching can be done in the network as well as at the Web server. The client's local cache, the network cache and the Web server's cache thus form a hierarchy. Since the effectiveness of caching depends on the access frequency of a cached document, it was realized that sharing a cache among users would be beneficial, giving rise to caching proxies. Proxy caching has been observed to reduce the traffic at the server and also reduce the transfer latency due to the proximity of the proxy cache to the client.

Two different caching models have been proposed for scalability and manageability purposes. These models are described briefly below.

- Hierarchical caching model – It was realized that a group of caches can benefit from sharing another higher level cache just as two clients could benefit from sharing the proxy cache [CHA-96]. This forms the basis for the hierarchical caching model. Internet

Cache Protocol (ICP) [WES-97], a Web caching protocol, was developed for quick and efficient communication between the cache servers that are a part of the hierarchy. ICP is used by the cache servers to exchange hints about the existence of URLs in their cache.

The caches exchange lightweight ICP messages – queries and replies to gather information to use in selecting the appropriate location from which to retrieve an object. Each server can have a parent or sibling relationship with other servers in the hierarchy. Figure 2.1 shows the hierarchy of object caches. The caches on a stub network share a parent child relationship with the cache in the regional network. The caches in the regional network share a sibling relationship with each other whereas a parent child relationship with the cache in the backbone network.

The disadvantage of this approach is that client latency can be high if messages are exchanged with several siblings and if the parent is located at a greater distance away from the client. Only three levels in the hierarchy are possible as four levels add more delays. The topmost server has to have a huge cache and can become heavily loaded as it becomes responsible for more and more clients. SQUID, the most popular public domain caching software available [SQU-99], supports the hierarchical caching model.

- Cooperative caching model - The CRISP project [GAD-98] explored alternatives for the hierarchical cache architecture. Flat architecture or cooperative caching forms an alternative approach to hierarchical caching. In this architecture the caching server's local cache directory (i.e. the index of all documents the server holds locally) is supplemented by an independently maintained global directory that allows it to locate objects cached at other servers. This reduces the number of messages exchanged between two cache servers and thus reduces the communication overhead.

Caching can be demand driven (pull model): a Web object is cached when the client requests it from the server. The other approach called geographical push caching has been proposed for optimal use of global cache space [GWE-95]. In this case the data is distributed to cooperating servers by the origin server. Cooperating servers agree on serving the documents of other servers. The origin server monitors the request stream and decides when and where objects should be cached. It takes advantage of its understanding of the network topology, i.e. which hosts are in close network proximity, and the file's access history.

2.5.2. Performance metrics

The metrics used in the literature for analyzing the cache performance and evaluating the cache replacement policies are listed below.

- *Object hit rate* - is a ratio of the number of requested documents found in the cache to the total number of requests generated by the client. It expresses the percentage of times that the document can be retrieved from the cache without requesting the document from the origin server.
- *Byte hit rate* – is a ratio of the total number of bytes found in the cache to the total number of bytes requested by clients. Byte hit ratio is important as there is a large variability in the sizes of Web documents.
- *Response time or access time* - is the time it takes for a user to get a document.
- *Server CPU and I/O system Utilization* – It is the proportion of time for which the CPU and the I/O system is busy doing work related to cache management over a given period of time.

2.5.3. Cache replacement policies

There is a limit on the available cache size. Hence the cache contents need to be

replaced and updated based upon user access pattern frequently. Several cache replacement policies have been proposed (see [CAO-97, WIL-96] for example). These are based on several factors that take into account the document size, their access frequencies, time elapsed since last access to the document and also network latency. Trace driven simulations are used for evaluation of caching policies [ARL-96B]. The traces used are those of the proxy servers that are widely available. These policies are evaluated based on their resource consumption, bandwidth utilization and the cache hit rate.

Some of the cache replacement policies available in the literature are listed below. Several variations of these policies are possible (see [NIC-98] for example).

- LRU – This policy removes the least recently used or accessed document and replaces them with the new document requested by the users. It expects that the documents accessed in the recent past would be accessed in the future and thus is based on temporal locality of reference.
- LFU – The least frequently used documents are considered first to be replaced by the new documents requested by the user. A frequency count for each object in the cache is maintained. This count is updated on each access.
- LFU-Aging – This policy considers the frequency of access or popularity of the document in the recent past as well as the age of the document. The age of the document

is the current time minus the last modification time of the document. This policy is motivated by the observation that over time popular documents can become unpopular. The aging policy addresses the cache pollution that occurs due to this turnover in popular object set by using the additional parameter, the age of the document.

- **SIZE [WIL-96]** – This policy considers the size of the document while replacing the documents with the new ones. Documents with large sizes are removed first. This strategy tries to minimize the miss ratio by replacing one large object rather than many smaller ones.
- **Greedy Dual-Size [CAO-97]** – This policy replaces the documents with the lowest cost factor, H . H is associated with each document based on the downloading latency and the **SIZE**, where **SIZE** is the size of the document in bytes. When a page is brought into the cache H is set to cost function. A document with smallest H ($\min H$) is removed first and every document has its H reduced by $\min H$. If a document is accessed again, then its H is restored to its cost. By reducing H values over time and restoring them upon access, this policy integrates locality and cost concerns.

It has been observed that there is a difference in designing server caches to maximize cache hit rate with designing for maximizing savings in bytes transferred. Caching small popular objects increases the cache object hit rate while byte hit rate increases as large popular objects are cached. Caching a few but high download latency pages takes into account the cost associated with document retrieval.

Selecting a good cache replacement policy becomes important as the choice of the replacement policy affects network bandwidth demand and object hit rate which in turn affect page load time. Since the traffic generated due to HTTP requests accounts for a great portion of the Internet backbone traffic, the choice of cache replacement policy affects the global network traffic as well as local resource utilization.

2.5.4. Cache Coherency

The other problems related to caching in the distributed Web environment are concerned with cache coherency or cache consistency. The coherency required is not strict, although pages devoted to current news stories, catalogs, and product information for example need to be consistent with the server's version.

Cache consistency is achieved using HTTP headers and a time-to-live (TTL) parameter [DIN-96] or by using invalidations. Both of these commonly used approaches are discussed briefly below. In the former case, the document requested by the client is checked for its existence in the cache. If it is found in the cache the document is checked for its validity using the TTL associated with it. The document is *fresh* if TTL has not expired and *stale* otherwise. If the TTL has expired, the cache sends an *If-Modified-Since* message to the server to determine if the version held by the cache is still valid. If the document has been modified then it is fetched from the origin server.

Invalidation based consistency requires a server to maintain a copy of all the clients

interested in a document and send explicit invalidation signals to them when the copy gets updated [GWE-96]. For scalability purposes a multicast technique is used by the server for transmitting invalidations. A multicast group is assigned to each page and the clients join the groups associated with the pages they have accessed.

HTTP/1.1 has new headers for cache control to be transmitted in requests and responses [DIN-96]. These directives allow the client and the server to override the default caching algorithm. HTTP/1.1 also has special headers for consistency which allow the user to specify a maximum staleness for each request and thus let the client specify the consistency requirements.

2.5.5. Other Cache Design Issues

A study of the rate and nature of change of Web resources [FEL-97] shows that many resources change frequently. The document frequency of access, age since last modified, and frequency of modification were found to depend on several factors such as the content type and the Internet top-level domain (of document URL), but not on size. Web cache design has to be done with the consideration of this high rate of change in the Web.

Caching relies on the principle of locality of reference. Locality based cache replacement strategy, for example LRU, assumes that the most recently accessed data has the highest probability of being accessed again in the near future. In the study reported in [ROA-99],

the authors demonstrate that locality, which is cache specific, can be characterized with a single parameter which primarily varies with the position of the cache in the network. This information can be used to build accurate cache models

For large caching proxy servers, disk I/O is identified as a bottleneck. File system interaction of such Web servers, termed as *cache architecture*, is studied in [MAL-99]. It is observed that file system aspects of the workload characteristics generated by certain cache architectures can differ from usual workstation loads leading to a poor file system performance.

Caching helps in reducing the data transmission time. But document consistency or staleness checks made in the critical path of the request incur considerable latency. Also there is management overhead in terms of invalidations and lookup costs. Several recent studies suggest that the maximum hit rate achievable by any caching algorithm is just 40% to 50% due to presence of cookies, CGI scripts and limited locality of reference [W3C-97]. Also not all documents are cacheable. Uncacheable documents could be the ones generated dynamically or the documents with explicit headers or documents which require authorization.

2.5.6. Prefetching

Prefetching has been proposed as a complement to the already existing caching mechanisms [MAR-98]. But unlike caching it is a proactive approach. The client access

pattern is observed over time and accordingly files are prefetched during low network activity times. The time available for prefetching is termed as the lead time. It is the time interval measured from the instant one has all the information to identify the server to which one has to connect, to the time at which the actual request to the server arrives. By prefetching the documents, the latency for the first time request of a given document is attempted to be kept low. Prefetching is expected to help in increasing the cache hit rate. A prefetching cache needs to accurately predict future requests. Heuristics and prefetching algorithms help in this prediction. The effectiveness of these algorithms decides the potential benefit of prefetching. An approach which combines the server's active knowledge of its popular documents and the user access profiles for making prefetching decisions has been proposed in [MAR-98].

Prefetching is not favored by system administrators as it might overload the cache servers. Prefetching helps in reducing latency although at an expense of increased bandwidth usage. Some of the prefetched documents may not be accessed by the client at all.

Since the effectiveness of document prefetching is limited by accuracy of predictions and the availability of lead time and bandwidth, prefetching the '*means*' for document transfer has been proposed [COH-00]. The authors identify DNS query times, and TCP connection establishment as major causes of longer user perceived latency for requests than the actual data transfer times. Based on these observations, the following techniques are proposed in order to reduce user perceived latency:

- *pre-resolving* host names or performing DNS lookup beforehand.
- *pre-connecting* or prefetching TCP connections prior to issuance of HTTP requests.
- *pre-warming* or sending dummy HTTP HEAD request to Web servers.

The above techniques are proposed as alternatives to document prefetching and can be used with non-prefetchable URLs. Some of these techniques can be used in conjunction with document prefetching as well.

2.6. Replication

Replication as a complement to the caching technique, for reducing user-perceived latency and wide area network traffic, has been emphasized in [BAE-97]. Replication is similar to caching but is more active. Replicas are also called mirror sites. A replica site always holds the document whereas a cache may or may not hold the document. Replicas hold a permanent copy. Replica servers may hold the entire data set as that of the origin server or only a partial set which consists of the popular documents.

The use of replicated server systems for achieving the Quality of Service (QoS) required by users has been proposed in [CON-99]. The client browser implements its own load distribution strategy. In this strategy the DNS is assumed to maintain a list of the replica servers. The browser probes the replicas by sending a dummy request. The browser then selects the replica which provides the response in the desired value of response time.

In another study, parallel access for mirror sites in the Internet has been proposed [ROD-00]. Clients access multiple mirror sites in parallel to speedup document downloads while eliminating the problem of server selection. Clients connect to mirror sites using unicast TCP connections and dynamically request different pieces of a document from different sites. The observation is that even in a situation where clients are connected through modem lines, transmission rates at least as high as the fastest server are achieved.

Technical issues related to replication in the distributed Web environment namely: data distribution among the replicas, and replica selection for document retrieval by a client have been studied. Each of these is discussed in the following subsections.

2.6.1. Data Dissemination

Replicating most popular documents in advance based on geographical location of distributed servers is an intuitive approach. Replicating popular documents on servers closer to clients reduces network traffic as well as balances server loads. Distributed cooperative Web servers for example balance load by agreeing to serve the documents of other servers [GWE-95].

Web documents exhibit heterogeneity in terms of sizes, and popularity. Also the geographical location of clients accessing the Web documents and the frequency of document updates are different from one document to another. It is inferred that a universal policy may not be suitable for all documents. Hence the use of specialized

policies simultaneously has been proposed [PIE-99]. Depending on its characteristics, each document could be replicated with the best suited policy. A comparison of replication and caching, replication only, and caching only architectures with no replication is described in the paper.

2.6.2. Replica Server Selection

A client needs to locate a replica to which it can make requests and obtain the minimum transfer latency. These replica location or selection techniques can be divided into the following broad categories: network-layer server selection, application-layer selection, and the use of static metrics such as the number of hops between the client and the server. Network layer server selection is done by measuring the round trip time of a ping request. Application layer server selection is done by measuring the response time for an HTTP request. Several server or replica location techniques have been compared in [SAY-98].

Effective algorithms for server selection by the client need an understanding of the actual behavior of Internet mirror servers. Hence performance characteristics of mirror servers on the Internet have been studied [MYE-99]. The observations of the preliminary study are that a server's relative performance in comparison to other servers is stable over a period of time and that in order to achieve near-optimal performance a client needs to consider only a few servers of all mirrors of a particular site.

2.7. Web Server Clustering

Multiple Web server machines are configured to be used as a single high performance Web server or a cluster. Web server clusters can be comprised of servers which are local or distributed. In the former case, a front-end node in the cluster represents the server and distributes the incoming traffic among the multiple back-end nodes. The back-end nodes store and serve the data required by the clients.

2.7.1. Request Distribution Schemes

The request distribution scheme employed by the front-end node of a Web server cluster impacts the system performance. Locality of reference, caching, and load distribution are the factors considered in some of the request distribution schemes. These are discussed briefly.

- **Round-Robin** – The requests that arrive at the front-end are evenly distributed among the back-end servers 1 to S. That is if the current request is sent to server X the following request is sent to X+1 and so on. Once the server S is reached, the next request is sent to server 1. Every back-end server is expected to cache the contents of the Web site. The disadvantage of this policy is that there is unnecessary duplication of data in caches across several back end servers [KHA-99]. **Weighted Round Robin (WRR)** is a variation of Round Robin policy. WRR assigns a weight to each server based on the server's current load, and request distribution is based on these weights.

- File-based Request Distribution – The file space is partitioned and each partition is assigned to a particular back-end server. Locality of reference is exploited as the requests directed to a particular server are cached. This scheme is sensitive to the way files are being partitioned and client access pattern [KHA-99]. Inequality in the sizes of files being requested may lead to a poor load balance.
- Client-based Request Distribution – The client space is partitioned and each back-end server is assigned to serve requests from a partition. This scheme also ignores the server load and can lead to unequal load distribution. A variation of this scheme is proposed in [KHA-99]. A client oriented IP-level redirection protocol is used by the server which allows a client to be redirected to another server. The server adapts the TTL value over time based on its perceived importance of load balancing and locality.
- Locality Aware Request Distribution (LARD) [PAI-98] – This scheme exploits locality of reference by assigning a dynamic set of servers to each file. When a request arrives at the front-end server, it checks if any of the back-end servers is assigned to the file. If not, it assigns the least loaded server to serve the request. The partitioning is done on the fly and locality is not restricted to one server.

Trace based simulation studies of some of the above schemes are presented in [KHA-99].

Their main observation is that locality proved more important than load balancing.

The problem of organizing and serving the data among the back-end nodes has also been

studied [KHA-99]. In the case of disk-mirrored system, back-end servers contain identical data and can serve a request without having to access files on other machines. In disk-striped systems, the available disks divide the entire data set. Files may be partitioned such that portions of it may reside on every disk. The later can provide better access times and improve load balance across disks. Disk striping distributes the requests over several back-end servers and results in smaller disk queues. An interesting anomaly is that requests accessing same data could be served by the same set of servers, so that the cache-hit ratio improves but it leads to an imbalance in load.

Another interesting study [BUN-99] compares the random, round robin and server load based distribution policies. The main observations of this study are listed below:

- Current state (or load) information is necessary in achieving good load balance only when the achievable per-request bandwidth is not network or client limited.
- Load distribution based on a static hashed assignment of URL space yields similar performance to load distribution based on current cache contents.
- Current state information is not necessary for achieving good cache behavior.
- Special policies are required in order to achieve both good cache behavior (i.e. hit ratio) as well as load balance.

2.7.2. Web Server Architectures for load balancing

Various load balancing schemes for distributed Web server systems have been

summarized in [CAR-99]. These load balancing schemes can be classified into two broad categories namely: dynamic load balancing using distribution of information and load balancing using distributed Web server architectures. Content tree replication on a server's local disk, and use of distributed file systems help in achieving load balance by distribution of information or Web files. DNS-based and dispatcher-based architectures are commonly used for achieving load balance in the distributed Web environment. The DNS based approach is briefly discussed below.

As shown in Figure 2.2 a cluster DNS server translates the logical HTTP address (name) into the IP address of one of the Web servers in the clusters. This IP address is used by the clients to make further HTTP requests to the Web server. This approach poses problems due to the caching of DNS returned name-address mapping at the servers and

clients, as this leads to load imbalance. A fixed time duration called time to live (TTL), which is the validity period for caching name resolution has been used to resolve the above problem.

Various scheduling algorithms using the TTL, server load and client geographic location information have been studied in [COL-98]. The main observations of the study are that DNS policies based on detailed server state information do not balance the load well. Policies which use domain information and feedback from over utilized servers are better. Adapting the TTL values to the capacity of a server helps in balancing the load well for the distributed Web systems.

In the dispatcher based approach, a front end dispatcher is used to distribute the incoming requests among the cluster of servers. Packet rewriting or packet forwarding mechanisms are used by the dispatcher for dispatching the requests. A packet rewriting dispatcher is necessarily a TCP router which maintains a table containing mappings between connections and IP addresses. On the other hand a packet forwarding dispatcher rewrites the IP information header of a packet according to the mapping of session connection and a server.

2.8. Summary

This chapter presented a representative set of existing works on the workload characteristics of the World Wide Web. Several workload characteristics which set the

distributed Web environment apart from traditional distributed systems are listed. A brief description of popular benchmarks and workload generation tools is presented. A comparison between HTTP/1.0 and HTTP/1.1 is also described.

The discussion on capacity planning of Web servers is followed by a brief discussion on Web performance measurement. A discussion of several research studies presented shows the need for better operating system support including effective resource management strategies for achieving high performance Web servers.

The caching technique for improving the Web performance is presented. Several research studies which address issues related to caching such as cache coherency and cache replacement are presented. The prefetching technique which is an alternative/complement to the caching technique is also discussed.

Replication of Web servers is discussed next. Data dissemination and replica server selection techniques that have been proposed are presented. This is followed by the literature review on Web server clustering. The several request distribution schemes that have been proposed for a Web server cluster are presented followed by the discussion on the different Web server architectures proposed for load balancing.

This thesis focuses on high performance Web servers and the parallelization techniques for performance improvement. It also investigates the effect of request distribution strategies at the dispatcher for load balancing among the back-end servers. The next

chapter presents a description of the workloads chosen for the performance evaluation of these techniques.

CHAPTER 3

3. SYSTEM ARCHITECTURE AND WORKLOAD FOR EVALUATING THE PROPOSED TECHNIQUES

3.1. Introduction

As mentioned earlier in Chapter 1, existing research has put more emphasis on alleviating the problems in Web surfing due to the network. This thesis proposes techniques to improve client perceived latency for a Web page retrieval by improving Web server performance. This is important especially with a high bandwidth availability to the client and to the backbone. A performance prototype is used to evaluate the proposed techniques.

A description of the experimental setup as well as the workload chosen for the experimental study is presented in this chapter. Section 3.2 provides a high level description of the system architecture. Section 3.3 presents a discussion of workload generation and client setup. Also a description of the performance metrics and workload parameters is presented. Section 3.4 provides a description of the system configurations as well as the Apache server configuration parameters. The rationale for the chosen values of the Apache server configuration parameters is also presented. Finally a summary is provided in Section 3.5.

3.2. System Architecture

A description of the performance prototype that was developed using the most popular and widely used open source Web server called Apache [APA-00] is presented in this section. The system performance is measured on a network of PC's. This approach has been chosen because it does not rely on simplifying assumptions that often accompany analytic and simulation models and it captures the effect of real system overheads that is difficult to parameterize in model based approaches.

A re-engineered Apache server is used to achieve an architecture presented in Figure 3.1. A reverse proxy [ENG-98] configuration is used in the dispatcher. In this configuration, the incoming requests are internally redirected to the back-end server. This is achieved by using a *rewriting engine*, which uses the URL redirection feature of the HTTP protocol. The rewriting engine uses either a static file that consists of the rewrite map or a program for generating a rewrite map dynamically. The rewrite map is essentially a mapping of the requested URL to the back-end server. The rewriting engine looks up the rewrite map before redirecting the HTTP request to the back-end server.

Each back-end server as well as the dispatcher is an Apache server version 1.3.0 running under Linux 2.0.34 on a 266 MHz Pentium II PC with 64MB of RAM. Clients are processes produced by the httpperf workload generation tool running on similar Pentium PC's. Each Web page, as shown in Figure 1.1, consists of an html document and the embedded objects which can contain static as well as dynamic content. The client sends a

HTTP GET request for an html document to the dispatcher which forwards it to a back-end server. The dispatcher sends the response from the back-end server to the client. The client then makes concurrent requests for the embedded objects which the dispatcher distributes among the back-end servers. Clients, dispatcher and the back-end servers are connected by a 100MB Ethernet LAN. The PC's and the LAN form the “quiet alpha network” in the Real Time and Distributed Systems lab where an experiment runs without interference from other users.

As shown in Figure 3.1, the dispatcher is connected to a number of back-end servers that provide the response for satisfying the Web page requests. Static html files as well as static embedded objects in Web pages are stored on back-end servers. The static html file

and the embedded objects in the Web page are retrieved when a client requests that Web page. The server CPU's are used for simulating the generation of dynamic content or the embedded objects in Web pages. The disk access time, for retrieving the static file, and/or the dynamic content generation time is simulated using a delay incorporated through a for loop that is executed while serving a request on the back-end server. The duration of the for loop is a variable factor for the experiments. In the presence of caching a high cache hit rate of static content (at the server) leads to less file I/O. Systems in which dynamic content is more dominant than the static content are increasing in popularity. In such environments the inaccuracy from using only a for loop for generating the service demand would be minimal. In systems in which serving a request gives rise to a large file I/O but a negligible number of CPU operations, the for loop can be used to model the disk service time.

This experimental setup is based on HTTP/1.0. This is due to the limitation of the Apache reverse proxy, used as the dispatcher, which currently supports only HTTP/1.0. Concurrent client requests for the multiple embedded objects in the Web page accessed are generated. Client requests arrive at the dispatcher and are forwarded by the dispatcher to one of the back-end servers. The order in which the concurrent connections should be serviced is handled by the operating system. The back-end server serves the request by sending the requested information to the dispatcher which then sends the response it receives from the back-end server back to the client.

The above HTTP (or application) layer forwarding architecture has been chosen due to

it's simplicity as compared to other alternative architecture such as the packet rewriting based dispatcher which uses TCP layer forwarding. In the case of the TCP layer forwarding architecture mentioned in Section 2.7.2 of Chapter 2, the TCP router or dispatcher has to maintain a dynamically generated connection-to-back-end server mapping table, which can be cumbersome. The dispatcher in the system under study reads a static file, which specifies the URL-to-back-end server mapping, while dispatching a client request to the back-end server. Also no TCP or IP layer modifications are required in the current architecture.

This dispatcher in the above HTTP layer forwarding architecture, as in other dispatcher based architectures, can become a single point of failure and requires redundancy. This aspect is beyond the scope of this research. Also the dispatcher can become a performance bottleneck unless it's capacity is high enough to handle the incoming traffic. But due to it's simplicity it has been preferred over other alternative architectures as discussed above.

3.3. Workload and Instrumentation

A description of the workload generation tool and the instrumentation used for performance measurement are presented in the following subsection. This is followed by a description of the client setup and the performance metrics of interest in this study. The workload parameters are presented next.

3.3.1. Workload Generation Tool

A number of benchmarks such as SPECWeb, and WebStone are available for simulating a large user base of a Web site. The httpperf tool [MOS-97] has been used for workload generation in this study. One of the important reasons for its use is its ability to let the user define a call burst that corresponds to requests for the main HTML file and its constituent embedded objects. The user defined file consisting of the call bursts or Web page requests is used by the httpperf session request generator (see Appendix A). This feature helps in defining the workload that is suitable for understanding the benefits of parallelizing a Web page request.

Httpperf is composed of three components: the HTTP engine, the workload generator and the statistics collector. The HTTP engine is responsible for communication with the server and connection management. The statistics collector is responsible for measuring various performance metrics and producing the performance statistics. The two kinds of workload generators supported by this tool are *session request generators*, and *URL generators* that create the desired sequence of URLs to be accessed on the server. For this study the session request generator has been used which generates sessions of Web page requests by reading the session input file. A description of this file is provided in Appendix A.

The important characteristics of the workload that can be controlled by httpperf are described below:

- *Call burst (Web page)* - Each call burst consists of a fixed number of requests. It simulates the browser behavior that captures the first request for an HTML page selected through a user mouse click and the following requests for the embedded objects.
- *Burst-to-burst think time* – The think time of a user between successive call bursts. This is set to zero in all the experiments, the reasoning behind this is discussed further in Section 3.3.2.
- *Session* - A session consists of a number of successive call bursts. The length of a session is based on the confidence interval requirements associated with the performance measures.

Although the httpperf client processes in this study generate HTTP/1.0 requests, httpperf does support HTTP/1.1. Persistent connections with pipelining of requests could be used to fetch the embedded objects in a Web page while using HTTP/1.1 clients. As mentioned earlier in Section 3.2 HTTP/1.0 clients had been used in this study as the Apache reverse proxy, used as the dispatcher, does not support HTTP/1.1.

3.3.2. Client setup

One of the clients acts as the master and runs a control program for the experiment and starts up the httpperf workload generator on each client simultaneously,

using the *gettimeofday* system call. Each client runs on a separate machine. This limitation of one *httperf* process per machine is due to the fact that with an increase in simulated clients per machine, the client side CPU and memory contention arises [MOS-97]. The client can become the performance bottleneck thus making it difficult to evaluate the different techniques for server performance improvement. The *httperf* tool produces the response time values of each call burst as output.

Each client makes several call bursts or Web page requests during a session (see Appendix A). Each call burst starts with a request for the html document in a Web page. After receiving the reply the client makes concurrent requests for the embedded objects in accordance with the HTTP/1.0 protocol. Note that a real client “thinks” for a duration of time before initiating another call burst. In order to comply with *httperf* requirements we used one client per PC. We had 9 PC’s to simulate clients in our experimental system. In order to produce a significant load on the system we had to set client think time to zero. Note that in real life scenario a portion of the requests generated by the client base are handled by the proxy cache servers, thus reducing the number of requests that are handled by the dispatcher or the back-end servers. So the traffic produced by each client in our system corresponds to the aggregate of the traffic produced by multiple Web clients on the real system. Several call bursts in a session are generated by *httperf*. Finally the control program computes the mean response time reading for each client by using the output generated by *httperf*. Overall mean response time is the mean of all mean client response times.

A HTTP/1.1 compliant client browser could use persistent connection and pipelining of the embedded object requests of a Web page.

3.3.3. Workload Parameters

By using a workload generation tool we have used a synthetic workload in this research. The synthetic workload allows us to effectively control the different workload parameters and observe their effect on the effectiveness of the performance enhancement techniques. A number of experiments were conducted on the system. Each experiment is characterized by a set of workload and system parameters. Following is a listing of the workload parameters.

- N - is the number of clients. The maximum number of clients simulated is 9, each client being a httpperf process running on a separate machine.
- B - is the number of back-end servers used for serving the requests.
- ts_i - The service demand associated with the main html file or an embedded object inside a Web page. The processing of an object is simulated by executing a for loop on a back-end server the execution time of which is generated based on a mean ts_i and a given distribution (deterministic or exponential). Processing of each object is completed by a server before starting the processing of another object.

- D_i - The distribution of the service demand associated with the main html file or an embedded object inside a Web page.
- E_i - is the number of embedded objects in a Web page. A Web page consists of a main html file and E_i embedded objects. An embedded object may contain static or dynamic content or a combination of both.

The maximum number of concurrent HTTP requests generated by the clients at the server would be a product of the number of embedded objects in a Web page and the number of clients generating these requests. Thus with 9 clients and 16 embedded objects in a Web page, used in one of the experiments discussed in Section 5.2, the maximum number of simultaneous requests at the dispatcher is 144.

- WP_i - represents the Web page class i ($i = 1, 2, \dots$). Each Web page class is characterized by the number of embedded objects it is composed of as well as the mean value and distribution of the service demand for processing each embedded object in the Web page. Thus each Web page class is denoted as WP_i indicating that a Web page of class WP_i has a service demand that has a mean value of ts_i and a distribution given by D_i , and that the Web page has E_i embedded objects.
- $Pr[WP_i]$ - is the probability with which a client makes a request for a page in the Web page class WP_i . This parameter can be used to model the Web page popularity. The home page of a company, for example, can be more popular (higher $Pr[WP_i]$) than any

other Web page on the server as it is accessed more.

The default parameter values used in the experiments that are conducted for systems with single Web page class and two Web page classes are shown in Table 3.1 and Table 3.2 respectively. The choice of the default value for service demand is based on the performance measurement study [DIL-98], which describes the workload parameters

Table 3.1: Test Parameters for performance study of systems with a single Web page class

Parameter	Default Value
E_1	4
S	2
ts_1	20 msec
D_1	Deterministic
Burst-to-burst think time	0 sec

Table 3.2: Test Parameters for performance study of systems with two Web page classes

Parameter	Default Value
E_1	4
E_2	8
S	2
ts_1	20 msec
ts_2	40 msec
D_1	Deterministic
D_2	Deterministic
Burst-to-burst think time	0 sec

collected from large-scale, commercial Internet and Intranet Web servers. A factor-at-a-time approach is used for investigating the impact of a given parameter on performance. One of the parameters is varied while the others are held at their default levels in a given experiment.

3.3.4. Performance Metrics

Benchmarks, as discussed earlier, stress the Web servers and measure the throughput that can be achieved. Although the throughput is an important performance metric, often the user is more interested in experiencing a low latency for requests made while browsing the Web.

The mean response time is related to the throughput by Little's law: $N = XR$, where N is the number of clients in the system, X is the system throughput in Web pages/sec and R is the mean response time of a Web page request. $X = \sum_i (Xg_i / (E_i + 1))$, where Xg_i is the number of HTTP GETs per second generated for retrieving the HTML file and the embedded objects in Web page class i . Note that E_i is the number of embedded objects in the Web page class i as defined in Section 3.3.3. For a single class system $X = Xg_1 / E_1 + 1$.

The file sizes of all the HTML as well as embedded objects in a Web page class are identical and equal to 3Kbytes for all the experiments discussed in Chapter 4. The throughput in bytes transferred per second due to serving of requests of a Web page class can be obtained from the measured value of Xg_i : $Xb_i = (Xg_i * 3\text{kbytes})$. Directly measuring throughput in bytes/sec becomes important when the file sizes for the different embedded objects are different from one another.

Clients request Web pages that may be graphically intensive. The user has to wait for retrieving all the embedded objects in order to view the complete Web page. Often due to

network latency and delays caused by busy servers, clients interrupt the Web page transfer after many embedded objects have already been retrieved. This leads to a wastage of bandwidth as well as the resources on the server side. Hence it is important to consider the latency for an entire Web page retrieval as an important performance metric.

The performance metrics of interest in this research are described as follows:

- *Response time (for a Web page request) R* - is the time elapsed between *the issuing of the first request* for the html document of a Web page by the client and the *receipt of the first byte of the last embedded object* of the same Web page by the client. The sizes of the files requested by the clients were small (3 kbytes) and the data transfer times due to network latency were negligible compared to the service demands of the requests at the server. Hence the response time is not very sensitive to the choice of the byte in the last embedded object, the receipt of which is used for terminating the response time measurement.

Response time readings for the call bursts in a session are averaged to produce the mean response time for the session. As described in Section 3.3.2 all the client processes are started simultaneously. The last request performed by each client may not occur at the same time. This results in a time period during which the value of N does not remain fixed. The sessions are chosen to be long compared to the duration of this time interval and the results are not affected by the variable value of N during this period. The number of call bursts in a session is chosen to achieve an interval that is less than $\pm 5\%$ of the

mean response time for most experiments at a confidence level of 95%. A confidence interval less than +/- 10% is achieved for the other experiments. Each client session consists of 8192 Web page requests. Appendix B shows a set of sample confidence interval values obtained for an experiment described in Chapter 4.

- *Throughput (at the dispatcher) X_g* - is the rate at which the Web server (in this case the dispatcher) serves the requests. It is measured in terms of HTTP GETs/sec at the dispatcher. The results reported in Chapter 4 are measured using 'pbvalsrvr', which is present in the SURGE tool [BAR-98]. This throughput measurement program is used to analyze the server logs generated by the dispatcher. It reports the mean value of GETS per second for the duration of the test.

3.4. System Configuration Parameters

In this section a discussion on Apache server and its configuration parameters is presented first. This is followed by a discussion of the rationale behind the chosen level of server configuration parameters. Finally the three different system configurations that have been considered in this thesis are described.

3.4.1. Server Configuration

Apache on Linux uses a pre-forking model of multiple server processes. That is instead of forking a process to handle a HTTP connection after a request has arrived, a set

of server processes are forked prior to request arrival and these pre-forked processes call *accept()* directly to accept new connection requests. This eliminates the process forking overhead to handle each HTTP connection. Requests for embedded objects are maintained in a central queue accessible to all the processes. Whenever a process completes its current request it extracts a request that may have arrived at the server for processing.

Initially the parent process forks a number of server processes equal to the `StartServers` directive in the configuration file. Apache dynamically adapts to the load by maintaining enough server processes to handle the current load. A few spare servers are used to handle transient load spikes [GAU-00]. If required, the parent process may spawn server processes in addition to the maximum spare servers to handle the incoming traffic of requests. The configuration parameters for both the back-end server and the dispatcher used in our experiments are described in Table 3.3. The rationale behind the chosen levels of these configuration parameters is further discussed in the following subsection. The caching feature both at the back-end servers and the dispatcher had been disabled. This was necessary in order to analyze the impact of the intrinsic properties of the proposed performance improvement techniques in the absence of caching.

3.4.2. Rationale for choosing the Server Configuration Parameters

The rationale for the chosen levels of Apache server parameters shown in Table 3.3 is presented here.

MinSpareServers and MaxSpareServers form the lower and upper limit on the number of spare server processes. The number of processes waiting for requests is compared with these values by Apache to maintain enough server processes that are adequate for handling the current load. MaxRequestsPerChild is the number of requests each child process is allowed to process before the child dies and a new process is created to replace it. This is set to an appropriate value to ensure that there are no memory leaks [APA-98].

In case of the dispatcher (or reverse proxy) the number of server processes or MaxSpareServers are dependent on the number of requests arriving per second and the time required to process a request [ENG-98]. For our purpose we used a measurement based study to identify the value of MaxSpareServers for the dispatcher. This measurement based study is described below.

Table 3.3: Apache server configuration parameters

Apache Configuration Parameter	Dispatcher	Back-end server
StartServers	20	5
MinSpareServers	20	5
MaxSpareServers	25	10
MaxRequestsPerChild	10	30

The response time readings at the client were observed for $N = 2$, $ts_1 = 40$ msec, $E_1 = 4$ and $B = 2$ as the test parameters that are discussed in detail in section 3.3.3. These values were chosen as they were to be the default values in most of the experiments and we wanted to ensure that for these test parameters the server's maximum capacity of handling the incoming requests is not exceeded. As seen in Table 3.4, there is a difference of about 20% between the response time readings for MinSpareServers,

MaxSpareServers values of [5,10], and [20, 25] for the dispatcher. Increasing these values from [20, 25] to [30, 35] increases the overhead of server maintenance.

For the back-end servers on the other hand there is little difference when these parameters are varied from 5 to 20, and 10 to 25 respectively. MaxRequestsPerChild had to be fixed at 30 for the back-end servers to avoid memory leaks as specified in [APA-98] and had to be fixed to a lower value i.e. 10 for the dispatcher. We had encountered frequent segmentation faults when MaxRequestsPerChild was set to a value higher than 10 for the dispatcher. Note that the Apache server spawns additional processes to handle the arrival of bursts of high number of Web requests. Once these large number of requests in the burst are completed the additional processes do not serve any useful purpose. Restricting the life of each process by setting MaxRequestsPerChild to a low value leads to the reduction of the number of idle processes and lowers the server maintenance overhead.

Table 3.4: Effect of server configuration parameters

MinSpareServers	MaxSpareServers	Mean Response Time (msec)
5	10	291.70
10	15	239.29
15	20	221.56
20	25	221.39
30	35	221.41

3.4.3. System Configurations

Our experimental system can be configured in three following ways.

- *S (Serial)* – A single back-end server essentially serves the Web page request of a client by serving the constituent embedded objects serially. There is no replication of data and hence the requests for a given Web page can be served by a specific back-end server. That is, in this configuration all the embedded objects in Web pages WP_i reside on a single back-end server.
- *P (Parallelism but no Replication)* – This configuration corresponds to using multiple back-end servers for serving the embedded object requests in a Web page in parallel. There is no replication and a request for a given embedded object can be served by only a specific back-end server. The division of embedded objects among the back-end servers is denoted by allocation, a definition of which is as follows.

Allocation – is the percentage of the total number of embedded objects in a Web page that reside on a back-end server. If there are five back-end servers (i.e. $B = 5$) an allocation of (10%, 20%, 10%, 40%, 20%) means that 10% of the total number of embedded objects are available on first server, 20% on the second, 10% on the third server and so on. In case of only two back-end servers when the objects are equally divided between the two servers, allocation is (50%, 50%). But if one of the servers has $X\%$ of embedded objects, the second back-end server has $(100 - X)\%$ of the total embedded objects. Thus a shorter notation has been used. In the case of equal division of embedded objects allocation would be equal to 50%. For other divisions of the embedded objects the smaller percentage of the objects available on a server is reported. For

example, an allocation of 25% means one fourth of the total number of embedded objects are available on one server and the rest on the other.

- *PR (Parallelism with Replication)* – Multiple back-end servers are used for retrieving the embedded objects in a Web page in parallel and each object is available on every server. A request for any object can thus be served by any back-end server.

S and P configurations are used to study the benefit of parallelization of Web page requests. P and PR configurations are used to understand the benefit of data replication and the effect of embedded object allocation on the back-end servers.

3.5. Summary

This chapter presented a description of the system architecture that has been used for evaluating the performance improvement techniques proposed. A functional description of the system is presented. The reason behind choosing the architecture for performance evaluation is also explained.

The workload generation tool chosen for generating the workload is presented next. This is followed by a description of the client setup. A description of several workload parameters is presented. The Web page classification based on two parameters namely the service demand (and the distribution) of the constituent embedded object requests at the back-end servers, and the number of such embedded objects in the Web page is made.

The performance metrics used for evaluating the techniques are defined. The importance of considering the Web page response time as the main performance metric is stressed.

A description of the Apache server model is presented next. The important server configuration parameters are described. The rationale behind the chosen levels of these configuration parameters is presented. Several system configurations which help in evaluating the performance benefit of parallelization of Web page request, data replication as well as data allocation are presented.

The next chapter presents a performance study of systems with single Web page class. Multiple Web page class systems are considered in Chapter 5.

CHAPTER 4

4. PERFORMANCE STUDY OF SYSTEMS WITH A SINGLE WEB PAGE CLASS

This chapter presents a description of the various experiments conducted using a single Web page class. Section 4.1 describes the experiments and the various insights gained into system behavior when parallelization of Web page access is employed. Section 4.2 presents a comparison of different request distribution policies at the dispatcher for achieving load balance. Section 4.3 presents the experiments conducted to explore the benefits of data replication and embedded object allocation. A discussion of the effect of the number of embedded objects in a Web page and the service demand of an embedded object on the relative performance of P and PR is also presented in this section. Section 4.4 presents a discussion of the various insights gained into system behavior. Finally a brief summary is provided in Section 4.5.

4.1. Parallelization of Web page access

This section describes the various experiments conducted to identify the important test parameters and gain an insight into the system behavior when parallelization of a Web page access is employed by the clients. A single Web page class has been used in the experiments except for the case of Section 4.1.1 where two identical Web page

classes have been used. The impact of the granularity of parallelism on the benefit of parallelism has been explored using the Serial (S) and Parallel (P) configurations (see Section 4.1.1). The benefit of using multiple back-end servers to serve a request in parallel is analyzed in Section 4.1.2.

4.1.1. Granularity of Parallelism

For the system architecture under study, a comparison between Serial (S) and Parallel (P) configurations for different values of t_s is presented in Table 4.1. Note that two identical Web page classes are considered here. This is because for the parallel (P) configuration two back-end servers are required so that a Web page can be retrieved in parallel. For the serial configuration, however, the Web pages in a class would reside only on one back-end server. With two Web page classes, Web pages in a class reside on one back-end server while those in the other class would reside on the other back-end server.

Table 4.1 Granularity of Parallelism ($B = 2$, $N = 5$, $\Pr[WP_1] = 0.5$, $WP_1(t_s = t_s, D_1 = \text{deterministic}, E_1 = 4)$, $WP_2(t_s = t_s, D_2 = \text{deterministic}, E_2 = 4)$).

Mean Response Time (msec)							
$t_s = 4$ msec		$t_s = 10$ msec		$t_s = 20$ msec		$t_s = 40$ msec	
S	P	S	P	S	P	S	P
212	237	285	254	326	304	811	697

At $t_s = 4$ msec, the Parallel configuration demonstrates an inferior performance in comparison to the Serial configuration. As t_s increases the performance of P becomes more and more superior in comparison to the performance of S (see Table 4.1). The

dispatching overhead at the dispatcher is relatively higher when the service demand or the processing time of a request at the back-end server is low than that when the service demand is higher. In the subsequent experiments described in this chapter and later in Chapter 5, a t_s of 20 msec has been used as the default value for the service demand.

4.1.2. Effect of increasing the number of back-end servers

In order to understand the impact of additional resources such as higher number of back-end servers to serve a Web page request of an individual client, a set of experiments were conducted for $B = 1, 2$ and 4 . The number of clients is varied from $N = 1$ to 9 for a given number of back-end servers. $B = 1$ corresponds to “no parallelism” or the serial case. Each request is processed serially. When there are two or more back-end servers the request of a client can be processed in parallel and the embedded objects in a Web page can be retrieved concurrently. This corresponds to the PR system configuration.

The response times and throughput are plotted in Figure 4.1 and Figure 4.2 respectively. As mentioned in Chapter 3 the throughput has been measured using the Surge tool, in number of GETs/sec which is the rate at which the requests are served, at the dispatcher. As shown in Figure 4.1 for $B = 1$, the response time increases sharply with an increase in N . But it shows a more gradual rise with the addition of more servers. For $N = 1$, the response time reduces by 31% as B is increased from 1 to 2 while a reduction of 15% is observed when the back-end servers increases from 2 to 4. For $N = 9$, the response time

Figure 4.1: Performance of a single class system: Response time (WP_1 ($ts_1 = 20$, $D_1 = \text{deterministic}$, $E_1 = 4$)).

Figure 4.2: Performance of a single class system: Throughput (WP_1 ($ts_1 = 20$, $D_1 = \text{deterministic}$, $E_1 = 4$)).

reduces by 46% as B is increased from 1 to 2 while a reduction of 41% is observed when the back-end servers increases from 2 to 4. This confirms the intuition that the performance benefit due to additional resources is more significant at higher number of clients. As shown in Figure 4.2, the throughput at the dispatcher increases with addition of back-end servers.

The effect of different workload characteristics on the performance benefit accrued due to additional back end servers is shown in Figure 4.3 and Figure 4.4. In both cases the number of clients is fixed, i.e. $N = 2$. As expected, Figure 4.3 demonstrates a significant

reduction in the response time when the number of back-end servers increases. The response time versus the number of back-end servers graph is characterized by a knee at $B = 2$. For a service demand of 20 milliseconds, as the number of servers is increased

from 1 to 2, a reduction of approximately 30% in response time occurs whereas this reduction is only 10% as the number of servers is further increased to 4. A reduction in the mean response time of approximately 40% and 48% is observed for service demands of 40 and 80 milliseconds respectively as the number of servers is increased from 1 to 2. Thus as expected the benefits of parallelization and the deployment of multiple servers are observed to be more prominent for systems with large service demands.

Similar observations are made for different number of embedded objects in a Web page (see Figure 4.4). A sharp decrease in response time is observed as B is increased from 1 to 2. A smaller improvement in performance occurs by further increasing the number of back-end servers. The performance benefits of using more servers and accessing the

embedded objects in a Web page in parallel is observed to increase with an increase in E.

4.2. Request distribution policies at dispatcher

This section presents a comparison of two request distribution policies at the dispatcher for distributing the incoming requests to the back-end servers. The effectiveness of using a Round Robin (RR) request distribution policy at the dispatcher for load balancing among the back-end servers is compared with the default Random policy used by Apache [APA-99].

In all the experiments discussed in the next section, RR policy for routing a request for an embedded object to one of the back-end servers has been used at the dispatcher in the PR (Parallel with replication) system configuration. Consider for example an embedded object that is replicated on B servers. Subsequent requests for the objects are spread evenly across all the servers 1 to B. That is, if the current request is sent to server X the following request is sent to X+1 and so on. Once server B is reached the dispatcher uses server 1 when the following request for the same object arrives. The default routing policy used by Apache is Random. When a request for a replicated object arrives the back-end server is chosen randomly. Note that each Apache server process at the dispatcher employs the random policy of back-end server selection independently. This happens to be the simplest policy from implementation perspective.

For achieving RR the dispatcher's back-end server selection policy is modified. This is

done using a shared memory variable which is set to the appropriate value identifying the back-end server to which the next request should be forwarded. The variable is updated after each dispatching of request to a back-end server. Note that unlike the random policy in which selection of a back-end server by each Apache server process is independent, coordination of server processes is required via shared memory in the case of the Round Robin policy. As shown in Figure 4.5, the Round Robin policy does provide an improvement of approximately 17% ($N = 2$) and 8% ($N = 5$) in performance compared to the Random policy.

With RR, the file requests generated by the experimental workload are evenly spread on the servers. This is observed to produce a better load balancing than PR with Random for

which there is a possibility that one of the servers is selected consecutively while the other is idle.

The superior performance of Round Robin over the default Apache policy indicates the potential for performance improvement and motivates the investigation of better load balancing policies that forms an important direction for future research.

One of the main reasons for the inferior performance of Random is the possibility of multiple requests being routed by the independent server process to a back-end server while the other back-end servers may be idle. Since each back-end server is selected by a server process with a probability of $1/B$, the probability that multiple server processes at the dispatcher will select the same back-end server for forwarding multiple consecutive requests at the dispatcher queue, decreases with an increase in B . Thus the performance of Random is expected to improve with an increase in B . More experiments need to be conducted with different number of back-end servers for comparing the Random and Round Robin request distribution policies.

4.3. Data replication and allocation

This section discusses the experiments conducted to understand the impact of data replication and the embedded object allocation on response time. P (Parallel with no replication) and PR (Parallel with Replication) configurations have been used for comparison in all the experiments. Each experiment focuses on the effect of a particular

system/workload parameter on performance.

4.3.1. Effect of data replication

The mean response time achieved with various allocations for P are compared with the response times achieved with PR. The number of clients, N , is varied from 1 to 9 while the back-end servers is fixed at two. A single Web page class with a service demand of 20 milliseconds and 4 embedded objects is used.

As shown in Figure 4.6, as the number of clients increases PR exhibits a higher

performance in comparison to P with an allocation of 25%. With such an allocation, 25%

of the embedded objects in a Web page reside on one of the servers while the remaining 75% are allocated to the other server. With data replication and a Round Robin policy for load balancing, simultaneous requests for the same object have a higher probability of being sent to different servers. Note that such a situation arises when multiple clients request the same embedded object. Thus PR gives rise to a lower queueing delay and performs better than P. There is little difference between the response time readings for PR and P with an allocation of 50% i.e. one half of the set of objects reside on the first server and the other half on the second. This situation is discussed further in the next subsection.

It is interesting to compare the results achieved with P (allocation = 50%) presented in Figure 4.6 with those captured in Figure 4.5. Since the performance of P is comparable to that of PR (RR), data replication on the default Apache server (that employs the Random policy) can produce an inferior performance in comparison to a system that does not replicate data but allocates embedded objects evenly across the available servers.

4.3.2. Effect of data allocation

Mean response times achieved with various allocations for P are compared with the response times achieved with PR. The number of back-end servers is fixed at two. The number of clients is fixed at two and five for the results shown in Figure 4.7 and Figure 4.8 respectively. Note that allocation is a parameter for the P configuration only. As expected, when the allocation of embedded objects on the two servers is unbalanced

PR performs significantly better than P (see Figure 4.7). The unbalanced allocation gives rise to larger queueing delays at one of the servers as a result of which P performs poorly.

However as percentage allocation increases the difference between PR and P is observed to diminish. It is interesting to note that with the embedded objects distributed evenly between the two servers (allocation = 50%) P performs comparably with PR. This shows the importance of using appropriate allocation strategies on systems with multiple servers.

Similar observation is made for the system when the number of clients making Web page requests is 5. As shown in Figure 4.8 the performance benefit of PR with respect to P with an unbalanced allocation is much higher compared to the earlier case with $N = 2$,

emphasizing the need to use proper allocation strategies for systems having a larger user base accessing the data content.

4.3.3. Impact of service demand

As mentioned earlier the server demand models either the disk access time and the CPU time corresponding to the fetching of a file, or the processing of a dynamic page request. The effect of service demand on client response time is captured in Figure 4.9 and Figure 4.10. The number of embedded objects in a Web page and the number of back-end servers is fixed at 4 and 2 respectively. The effect of service demand for $N = 2$ and $N = 5$ clients is presented in Figure 4.9 and Figure 4.10 respectively.

Figure 4.9: The impact of service demand on the performance of PR and P (Allocation = 25 %) ($B = 2$, $N = 2$, WP_1 ($ts_1 = ts$, $D_1 = \text{deterministic}$, $E_1 = 4$)).

Figure 4.10: The impact of service demand on the performance of PR and P (Allocation = 25 %) at a higher number of clients ($B = 2$, $N = 5$, WP_1 ($ts_1 = ts$, $D_1 = \text{deterministic}$, $E_1 = 4$)).

The mean response time is observed to increase almost linearly with ts_1 . The performance difference between PR and P increases with an increase in service demand for a given number of clients. This is because the impact of the additional queueing delay experienced on a system using P (allocation = 25%) is more pronounced at higher values of ts_1 . Although the difference between the mean response times achieved with PR and P is smaller at $ts_1 = 20\text{msec}$, it is observed to increase by 46% (for $N = 2$) and by 39% (for $N = 5$) at $ts_1 = 80\text{ msec}$.

4.3.4. Impact of number of embedded objects in a Web page

This section presents a comparison between PR and P (allocation = 25%) for various number of embedded objects in a Web page. The service time associated with the

servicing of a request for an embedded object in a Web page is fixed at 20 msec, and the number of back-end servers is fixed at 2.

As the number of embedded objects in a given Web page increases the improvement produced by PR over P with allocation of 25% increases (see Figure 4.11 and Figure 4.12). A 20% reduction in response time readings is observed due to data replication on both servers for both 2 and 5 clients when the Web page consists of 4 embedded objects. With more embedded objects per page more queueing occurs in case of P (allocation = 25%) and it exhibits an inferior performance in comparison to PR.

4.4. Discussion

Several insights gained into system behavior are discussed here.

Accessing the embedded objects in a page in parallel, by using additional back-end servers, is observed to give a substantial improvement in performance. As the number of back-end servers, B , is increased from 1 to 2 a large improvement in response time is observed. A small improvement in performance is observed for a further increase in B . The granularity of parallelism (or the service demand of a request), for which parallelization of Web page requests gives a performance improvement, is dependent on the request dispatching overhead at the dispatcher relative to embedded object service time.

The Round Robin strategy for dispatching requests to the back-end servers on a replicated system is observed to outperform the default Random strategy used by Apache due to better load balance among the back-end servers.

The performance improvement of a replicated system is larger in comparison to a non-replicated system in which the allocation of embedded objects is not balanced across multiple servers. Such an unbalanced allocation of embedded objects may be an outcome of dedication of each of the back-end servers to a specific set of companies in an e-commerce system that hosts multiple electronic shops or due to the existence of a set of hot files that are accessed more frequently than the others. Unbalanced allocation of embedded objects can also be due the fact that the back-end servers have different amounts of available disk space. Often for a given Web site a Web page may share some of the content (i.e. embedded objects) with other Web pages which could also lead to

unbalanced allocations of embedded objects. The comparable performance of PR and P (allocation = 50%) indicates that for the given set of workload parameters the parallelism in a Web page access improves system performance substantially whereas data replication has only a secondary effect on performance.

Embedded object allocation to back-end servers can have a significant impact on performance. It is interesting to note that with a good object allocation even a non-replicated system can perform close to a replicated system that uses a Round Robin dispatching strategy and better than a replicated system in which a Random dispatching strategy is used. Note that allocation is also an indicator of the degree of parallelism. When allocation is 0, all the Web pages reside on the same server and the constituent embedded objects are accessed serially. When allocation is higher, a better parallelism in page access is possible.

Both the average service time, t_s , and the number of embedded objects in a page, E , have a strong impact on performance. An increase in both of these parameters increases mean response time. The performance difference between a replicated (PR) and a non-replicated (P (Allocation = 25%)) systems tends to increase with an increase in any of these two parameters.

4.5. Summary

This chapter described the experiments conducted for Web server systems with a

single Web page class. It described the various insights gained into system behavior.

The experiment conducted to understand the impact of granularity of parallelism on the performance benefit of parallelization of Web page requests is described first. This is followed by a description of the experiments conducted to understand the benefit of parallelization of Web page requests using additional resources or back-end servers.

A description of the experiments conducted for comparing the different request distribution policies at the dispatcher for achieving load balance is also presented. The need for better request distribution policies is emphasized. The experimental study to understand the benefit of data replication and embedded object allocation is presented next. The impact of workload characteristics such as number of embedded objects and service demand of a request on the benefit of parallel configuration are also discussed.

The last section presented a discussion of the various insights gained into system behavior. The next chapter presents a performance study of the different system configurations for multiple Web page class Web servers. The effect of various workload characteristics is further explored.

CHAPTER 5

5. PERFORMANCE STUDY OF SYSTEMS WITH MULTIPLE WEB PAGE CLASSES

5.1. Introduction

It is observed in the earlier chapter that parallelization of Web page requests has a prominent effect on performance. This is further explored in this chapter. A comparison between serial and parallel Web page access is made. Web server systems with multiple Web page classes are considered here. The effect of different workload characteristics on the benefit of parallelism is presented in Section 5.2. The performance study based on the workload characteristics of a popular Web site is presented in Section 5.3. This is followed by a discussion in Section 5.4. Section 5.5 provides a brief summary of the results presented in this chapter.

5.2. Serial vs Parallel Web Page Access

This section presents the different experiments conducted to understand the effect of different workload characteristics on the benefit of parallelism. A comparison of serial and parallel Web page access for Web page classes with no variabilities is presented first. Two Web page classes are considered here. This is because for the parallel (P)

configuration two back-end servers are required so that a Web page can be retrieved in parallel. For the serial configuration, however, the Web pages in a class would reside only on one back-end server. With two Web page classes, in the serial configuration, Web pages in a class reside on one back-end server while those in the other class would reside on the other back-end server. Also a two Web page class system would be the simplest case for studying inter-page variability. The effect of the existence of either intra-page variability or inter page variability in the workload on the performance benefit of parallelism is discussed next. Finally the effect of the existence of both inter-page as well as intra-page variability simultaneously is discussed. Systems with more than two Web page classes are discussed in Section 5.3.

5.2.1. Web page classes with no variabilities

Systems with no variabilities in the Web pages accessed by the clients are considered first. Two classes of Web pages WP_1 and WP_2 are used. All the Web pages are characterized by identical values of E and t_s . A deterministic distribution, D_i , of service demand means that the value of service demand is fixed at a constant value. Web pages in each class are accessed with an equal probability. The Web page to be accessed at any time during a user session is chosen randomly based on the value of a uniformly distributed random variable. If the value of the random variable is between 0 and 0.5, WP_1 is accessed and if the value is between 0.5 and 1, WP_2 is accessed. Note that with the absence of variabilities among Web page classes the system is reduced to a single class system. Such a system, with no variabilities in Web page classes, is considered first

in order to get a comparative analysis for the other cases discussed in the following subsections.

For the serial (S) configuration the embedded objects in WP_1 are resident on one back-end server whereas the embedded objects in WP_2 are resident on the other back-end server. On the other hand for the parallel, P (allocation = 50%), configuration 50% of the embedded objects in each Web page class reside on a server while the remaining 50% on the other back-end server. For P (allocation = 25%), 25% of the embedded objects in each Web page class reside on one back-end server whereas the remaining 75% of the embedded objects reside on the other back-end server. For the PR system configuration, all the embedded objects in WP_1 and WP_2 reside on both of the back-end servers.

A 27 % improvement is observed (see Figure 5.1) to accrue from using parallelism in Web page access when the number of clients and the number of back-end servers are 1 and 2 respectively. Since multiple embedded objects are accessed concurrently a substantial benefit is achieved. Benefits of parallelism are reduced at higher values of N. Since the request for each embedded object is enqueued independently at the dispatcher, there is no guarantee that the two embedded objects accessed concurrently will belong to the same client request and the benefits of parallelism are reduced at higher values of N. The benefit in performance changes from 27% to 5% as N is varied from 1 to 9.

P (allocation = 25%) leads to an inferior performance when compared to even the serial configuration. The performance of PR is comparable to that of P (allocation = 50%) as in

the earlier cases of a single Web page class discussed in Chapter 4. In the experiments discussed in the following sections the serial (S) and the parallel P configurations are considered for comparison purposes.

Table 5.1: The Effect of Parallelism with a higher number of embedded objects in a Web page ($B = 2$, $\Pr[WP_1] = 0.5$, WP_1 ($ts_1 = 20$, $D_1 = \text{deterministic}$, $E_1 = 16$), WP_2 ($ts_2 = 20$, $D_2 = \text{deterministic}$, $E_2 = 16$)).

Clients (N)	Mean Response Time (msec)	
	Serial (S)	P (Allocation=50%)
1	400	269
9	1850	1670

A similar observation is made for a larger number of embedded objects E (see Table 5.1). However the improvement in performance from parallel Web page access is observed to be slightly higher: 32% at $N = 1$ and 9% at $N = 9$. To reduce the length of experiments

only S and P configurations are considered here. A comparable performance of PR with P (Allocation = 50%) is expected, as in Figure 5.1.

5.2.2. Intra-page variability

Embedded objects in a Web page may have different service times associated with them. Systems characterized by Web pages with internal variability in service demands are analyzed in this section. The system contains two classes of Web pages, each with 4 embedded objects. The service time required for the processing of each embedded object in a Web page in class $i = 1$ is characterized by an exponential distribution with mean ts_1 . Similarly the service time required for the processing of each embedded object in a Web page in class $i = 2$ is characterized by an exponential distribution with mean ts_2 . Each Web page class is accessed with an equal probability. An exponential distribution is chosen for modeling variability in service demands because it is well known for its use in performance modeling. This is adequate for the high level nature of the questions addressed in this thesis. Using workloads that match the service demand characteristics of real systems is important. Detailed Web server access patterns with information about the service demands are essential. However, to the best of our knowledge such a description of real workload is not available in the literature.

As shown in Figure 5.2 a similar relationship between the response time and the number of clients for S and P (Allocation = 50%) configurations, as captured in Figure 5.1, is observed: the performance enhancement that accrues from using parallelism in Web page

access decreases as the number of clients (N) increases. The performance improvement due to parallelism for a given N is lower with an exponential distribution of service time than that compared to deterministic distributions of service times of embedded objects in a Web page.

A Web page access is complete when all of its embedded objects are processed. With an exponentially distributed service demand, the existence of service demands larger than the mean tends to inflate the mean response time. The effect of an unbalanced allocation is captured in P (Allocation = 25%) in which 25% of the embedded objects are allocated on one back-end server and the remaining 75% on the other. Systems with an unbalanced allocation are observed to give rise to a performance that is even inferior to that of serial.

5.2.3. Inter-page variability

A system with variability among Web pages is considered in this section. There are two classes of Web pages: WP_1 with 4 embedded objects and WP_2 with 8 embedded objects. The service time associated with each object within a Web page is fixed at the same value. Results of experiments with a system in which each client accesses WP_1 Web pages with a probability of 0.25 and WP_2 Web pages with a probability of 0.75 are displayed in Figure 5.3.

Benefits of using parallelism in Web page access is observed to increase with inter-page variability. A performance difference of 28% at $N = 1$ and of 17% at $N = 9$ are observed

between S and P with allocation = 50%. With S, all the embedded objects in a Web page are accessed in sequence from the same server even though the other server is idle. The impact of this load imbalance gets magnified and the performance of parallel processing improves when larger service demands are associated with a Web page. As a result the performance of Serial configuration is inferior to that of Parallel.

An unbalanced allocation produces results comparable to that of S at lower values of N and worse than S at higher values of N. A similar performance difference between S and P is observed when both classes of Web pages are accessed with the same probability. In comparison to the data captured in Figure 5.3 the performance improvement that accrues from parallelism in Web page access is observed to increase especially at higher values of

N (see Figure 5.4).

5.2.4. Intra-page and inter-page variability

A system characterized by both intra-page and inter-page variability are considered in this subsection. There are two classes of Web pages WP_1 and WP_2 . WP_1 contains 4 embedded objects and WP_2 contains 8 embedded objects. The service demand for an embedded object within a Web page in both WP_1 and WP_2 classes is characterized by an exponential distribution with a mean of 20 msec.

As shown in Figure 5.5 the ranking among the different system configurations in the

presence of both inter-page and intra-page variabilities, among the Web page classes, is the same as that observed with Figure 5.4. P (Allocation = 50%) achieves a 20% improvement in performance in comparison to S at $N = 1$ and a 18% improvement in performance at $N = 9$.

5.3. Performance Study of a Popular Web site

The observation from the various experiments described in Section 5.2 is that for a Web site with two Web page classes, the existence of inter-page variability leads to a better performance benefit of parallelization of Web page access. A similar experiment is conducted with the characteristics of popular Web pages which was studied in [ABH-00]. Table 5.2 shows the information about the popular Web pages that is reported by the site 100hot.com in the month of March 99. Web pages are classified into 30 different classes based on the number of embedded objects that are present in them. The Web page class, the number of embedded objects in the Web page class and the probability of each Web page class being accessed are listed in Table 5.2.

Table 5.2 Popular Web pages that can be classified into different Web page classes based on the number of embedded objects.

i	E_i	$\text{Pr}[\text{WP}_i]$
1	1	0.018
2	2	0.018
3	3	0.072
4	5	0.018
5	6	0.054
6	8	0.018

i	E_i	Pr[WP_i]
7	9	0.036
8	10	0.054
9	11	0.036
10	12	0.036
11	13	0.036
12	14	0.036
13	15	0.018
14	16	0.072
15	17	0.054
16	18	0.036
17	20	0.054
18	21	0.036
19	22	0.036
20	23	0.018
21	27	0.036
22	28	0.018
23	30	0.018
24	32	0.018
25	34	0.018
26	35	0.054
27	39	0.018
28	40	0.018
29	42	0.018
30	45	0.018

For the experiments conducted, an assumption was made that each Web page is being accessed once. Thus each Web page class has a different probability of being accessed depending on the number of Web pages in the class. For example the number of Web pages in class $i = 21$ are 2, each consisting of 27 embedded objects, and the probability of access is 0.036. Note that in the case of real Web sites these probabilities could be different and could change over time. Two back-end servers were used. In the Serial configuration, half of the total Web pages are available on one server and the other half on the other. A uniform distribution is used, as in the earlier cases, by the client deciding

which of the Web page classes is to be accessed next during the session. If the value of the random variable chosen from this distribution lies between 0 and 0.018 a request for WP_1 is made. If this value lies between 0.018 and 0.036 WP_2 is accessed and so on.

To reduce the length of experiments two system configurations of interest are considered: the serial (S) and parallel (P) configurations. For the serial configuration, the embedded objects of every other Web page class reside on one server whereas the embedded objects of the remaining Web page classes reside on the second server. That is embedded objects of $WP_1, WP_3, \dots$ reside on the first back-end server while the embedded objects in $WP_2, WP_4, \dots$ reside on other back-end server. For the parallel configuration, an allocation of 50% of the embedded objects to either of the back-end servers is made for Web page

classes with even number of embedded objects. For other classes the embedded objects on one of the back-end servers is equal to $\text{ceil}(E/2)$ while the rest of the embedded objects are available on the other back-end server.

As shown in Figure 5.6, a performance benefit of 38.42% is observed due to parallelization of Web page requests when a single client is accessing the Web pages. When the number of clients increases to $N = 9$, the performance benefit reduces to 14%. As mentioned before in Section 5.2.1, the request for each embedded object is enqueued independently at the dispatcher. Hence, as the number of clients accessing the Web pages increases, there is no guarantee that the two embedded objects accessed concurrently will belong to the same client request and the benefits of parallelism are reduced.

5.4. Discussion

A Web page is composed of a number of embedded objects. By distributing the objects on a Web page on multiple servers they can be processed in parallel to reduce client response times. A significant improvement in system performance is observed by distributing the embedded objects in a Web page on two back-end servers. At low system load ($N=1$) a performance benefit of approximately 30% is observed in a number of test cases. The benefit is observed to be reduced at higher system loads.

Intra-page variability is concerned with the difference in service demands associated with

the embedded objects inside a Web page whereas inter-page variability refers to the service demand variation from Web page to Web page. Intra-page variability is observed to reduce the benefits of parallelism in Web page access. On the other hand higher performance benefits accrue from accessing embedded objects in parallel when a larger inter-page variability exists on the system. This observation has been confirmed for the case of a popular Web site with multiple Web page classes.

Thus a Web page design should attempt to produce embedded objects with comparable service demands. Parallelization of Web page access is expected to produce a significant benefit on systems that host a variety of different Web pages.

5.5. Summary

This chapter described the experiments conducted for Web server systems with multiple Web page classes.

Several experiments that were conducted for achieving high performance through parallelization of Web page accesses for systems with two Web page classes are presented first. The experiments that were conducted to understand the impact of inter-page, intra-page variability are discussed. The applicability of the results of one of these experiments, mainly the effect of inter-page variability, is verified for a popular Web site consisting of 30 Web page classes.

The next chapter provides a summary and conclusions as well as offers directions for future research.

CHAPTER 6

6. CONCLUSIONS

Section 6.1 provides a summary of the work done in this thesis. Section 6.2 presents the conclusions that are drawn based on the experimental results. Finally Section 6.3 offers directions for future research.

6.1. Summary

This thesis presents a number of performance enhancement techniques for achieving high performance Web server systems. These techniques include parallelization of Web page requests using multiple servers in parallel to serve the requests, replication of data in conjunction with proper data allocation policies and effective request distribution strategies at the dispatcher.

By using a performance prototype based on the Apache Web server and a synthetic workload, this thesis has analyzed the effect of the proposed performance enhancement techniques on system performance. This prototyping and measurement based approach is selected as it captures the effect of system overheads that is difficult to capture accurately in analytic or simulation models. The performance prototype is based on the popular open source Apache servers running on a network of Pentium PC's. The system consists of a

dispatcher and multiple back-end servers. The dispatcher is essentially an Apache server configured as a reverse proxy using the rproxy module.

One of the advantages of this architecture is its simplicity. No hardware modifications are required as the forwarding of requests is done at the application level (or HTTP layer). The redirection or forwarding overheads for each individual request at the application layer would be slightly higher than that done at the TCP layer in the packet forwarding architectures. An actual comparison study of the two architectures would give more insight in to the performance benefit of one architecture over the other.

Httpperf is a tool that simulates the workload produced by a client browser and does the performance measurements for experiments. The session generator of this tool generates the client sessions where each session consists of several requests for Web pages. The session generator generates the sessions by reading the user defined input file which consists of the individual Web page requests. The httpperf tool is extended for measuring the response time of a Web page request which is essentially the time required for retrieving the main html document as well as the embedded objects. Most of the response time measurements have an interval that is less than +/- 5% of the mean response time at a confidence level of 95%.

The system is configured in several different ways based on the style of Web page access by the client and the use of data replication. Web pages are characterized based on the number of embedded objects they contain and the service demands of the embedded

objects.

Comparison of S (Serial) and P (Parallel) configurations reveals the benefits due to parallelization of Web page access. The variation of the workload characteristics of Web pages identifies the impact of the workload parameters on the performance benefit due to parallelization of Web page access. P and PR (Parallel with replication) system configurations help in identifying the benefit of data replication as well as the impact of embedded object allocation to the back-end servers on performance.

The next section summarizes the conclusions derived from the results of the experiments reported in the thesis.

6.2. Conclusions

Using parallelism in Web page access is observed to have a very significant impact on performance. A large improvement in response time can be expected by accessing multiple embedded objects in a Web page concurrently from multiple back-end servers. This improvement is dependent on the granularity of parallelism.

Certain Web pages may be accessed more heavily than the others. Parallelization of Web page access is observed to provide a similar performance benefit irrespective of the relative popularity of different classes of Web pages as long as the objects within a Web page are distributed uniformly across the available back-end servers.

Intra-page variability is concerned with the difference in service demands associated with the embedded objects inside a Web page where as inter-page variability refers to the service demand variation from one Web page class to another. Intra-page variability is observed to reduce the benefits of parallelism in Web page access. On the other hand higher performance benefits accrue from accessing embedded objects in parallel when a larger inter-page variability exists on the system. Thus parallelization of Web page access is expected to produce a significant benefit on systems that host a variety of different Web page classes. Examples of such systems include e-commerce sites with multiple virtual shops and systems with a set of hot pages that are accessed more frequently than the others.

Most of the experiments are performed with two back-end servers. Increasing the number of back-end servers beyond two on systems in which data is replicated on all servers does produce a further improvement in performance. The addition of back-end servers to the system improves performance sharply at first; however a knee is reached and an increase in the number of servers beyond this knee produces a smaller improvement in performance.

A balanced distribution of embedded objects within a Web page on the back-end servers is important for harvesting the benefits of parallelization in Web page access. If object allocation gives rise to an unequal distribution of service demand across the back-end servers, system performance may degrade significantly. Balanced distribution of

embedded objects becomes more important as the workload parameters such as service demand and number of embedded objects in a Web page increase.

Replicating each object on each of the available servers alleviates the problem due to unbalanced allocation of embedded objects. Twice the storage capacity is required however for such a replication on a two-back-end server system. Also data consistency policies would be required in the latter case when Web page content changes.

We have experimented with two load balancing policies at the dispatcher: Random and Round Robin. Although the load balancing policy is observed to have a secondary effect on system performance, the experimental results show that using a simple Round Robin strategy can produce a higher performance than the default Random strategy employed by the Apache server.

In the investigation of the performance improvement techniques we have used synthetic workload models and structural data for Web pages from real environments. We have used deterministic and exponential service times for embedded objects. For both of these distributions parallelization of Web page requests are observed to improve system performance. Investigating the performance improvement techniques proposed in this thesis for service times with a high variability that can be modeled by a hyper exponential distribution for example, forms an important direction for future research. The conclusions that are outlined above may depend on the underlying assumptions of the service demand distributions used in the different experiments. The results of this

research are applicable primarily to systems in which servers contribute to overall system performance in a significant way. Depending on the system one or more of the proposed techniques can be combined to achieve high system performance.

The next section suggests directions for future research.

6.3. Future Research

This section identifies directions for future research to extend the techniques for achieving high performance Web servers and to investigate the impact of workload characteristics on the performance benefit accrued from the proposed techniques.

The performance impact of several workload characteristics needs to be investigated further. The service demands of embedded objects for the experiments discussed in this thesis have been characterized by deterministic and exponential distributions. Using other distributions characterized by larger variabilities forms an important direction for further research.

The Apache server software distribution, Apache 2.0, to be available in the future is designed to offer numerous enhancements, improvements and performance boosts over the 1.3 codebase that has been used in this thesis. The most visible and noteworthy addition is the ability to run Apache in a hybrid thread/process mode on Linux, the platform on which the PC's in the experimental setup were running. These additions will

help in improving the scalability of the Apache HTTPD server. Extending the approach of parallelization of Web page access and other performance improvement techniques proposed in this thesis to the new version of Apache and investigating the effect on performance forms another direction for future research.

A comparison of two data distribution policies at the dispatcher, namely the default Apache Random policy and the Round Robin policy is done in this thesis. Caching of documents at the back-end servers as well as at the dispatcher had been disabled. In the presence of caching and a high cache hit rate, the service time variability for the static components of a Web page is expected to diminish. Smarter load distribution policies which help in achieving load balance as well as the contributions of caching on performance need to be studied in future.

REFERENCES

- [ABH-00] A. Abhari, S. Dandamudi, S. Majumdar, "Characterization of Web objects in Popular Web Documents," (to appear) In Proceedings of 13th International Conference on Parallel and Distributed Computing Systems (ISCA), Las Vegas, August 2000.
- [ALM-97] V. A. F. Almeida, J.M. Almeida, C. S. Muta, "Performance Analysis of a World Wide Web Server," In Proceedings of the 23rd National Computer Measurement Group (CMG) Conference, Orlando, December 1997, pp. 829-837.
- [APA-98] Apache Software Foundation, Apache Manual, <http://www.apache.org/docs>.
- [APA-99] Apache Software Foundation, Apache web server, <http://www.apache.org/>.
- [ARL-96A] M. F. Arlitt, C. L. Williamson, "Web server workload characterization: the search for invariants," In Proceedings of the ACM SIGMETRICS '96 Conference, Philadelphia, April 1996, pp. 126-137.
- [ARL-96B] M. F Arlitt, C. Williamson, "Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers," Technical Report, Dept. of Computer Science, University of Saskatchewan, Canada, 1996.
- [ARL-99] M. Arlitt, R. Friedrich, T. Jin, "Workload Characterization of a Web Proxy in a Cable Modem Environment," HPL-1999-48, HP Labs Technical Report, Palo Alto, CA, 1999.
- [BAE-97] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, P. Sturm, "Enhancing the

Web's Infrastructure: From Caching to Replication," IEEE Internet Computing, Vol. 1, No. 2, March 1997, pp. 18-27.

[BAN-97] G. Banga, P. Druschel, "Measuring the capacity of a web server," In Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS), Monterey, December 1997, pp. 61-71.

[BAN-98] G. Banga, P. Druschel, J.C. Mogul, "Better operating system features for faster network servers," In Proceedings of the Workshop on Internet Server Performance (WISP'98), Madison, June 1998, pp. 69-79.

[BAR-98] P. Barford, M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," In Proceedings of ACM SIGMETRIC International Conference on Measurement and Modeling of Computer Systems, Madison, June 1998, pp. 151-160.

[BAR-99A] P. Barford, M. Crovella, "A performance evaluation of Hyper Text Transfer Protocols," In Proceedings of ACM SIGMETRICS '99, Atlanta, Georgia, May 1999, pp. 188-217.

[BAR-99B] P. Barford, M. E. Crovella. "Measuring Web performance in the wide area," Technical Report TR-99-004, Boston University, CS Dept, April 1999.

[BER-98] T. Berners-Lee, R. Fielding, H. Frystck, Hypertext Transfer Protocol – HTTP/1.0, RFC 1945, HTTP Working Group, May 1998.

[BUN-99] R.B. Bunt, D.L. Eager, G. Oster, C.L. Williamson, "Achieving Load Balance

and Effective Caching in Clustered Web Servers,” In Proceedings of 4th International Web Caching Workshop, San Diego, CA, March 1999.

[CAO-97] P. Cao, S. Irani, “Cost-Aware WWW Proxy Caching Algorithms,” In Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS), Monterey, CA, December 1997, pp. 193-206.

[CAR-99] V. Cardellini, M. Colajanni, P. S. Yu, “Dynamic Load Balancing on Web server systems,” IEEE Internet Computing, Vol. 3, No. 3, May-June 1999.

[CHA-96] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, K. Worrel, “A hierarchical Internet object cache,” In Proceedings of USENIX Conference, San Diego, January 1996, pp. 153-163.

[CHA-98] J. Challenger, P. Dantzic, A. Iyengar, “A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites,” IBM Research T. J. Watson Research Center, NY, 1998.

[COH-00] E. Cohen, H. Kaplan, “Prefetching the Means for Document Transfer: A New Approach for reducing Web latency,” In Proceedings of 19th annual conference on Computer Communications, IEEE INFOCOM 2000, Tel-Aviv, Israel, March 2000.

[COL-98] M. Colajanni, P. S. Yu, D. M. Dias, “Analysis of Task Assignment Policies in Scalable Distributed Web server systems,” IEEE Transactions on Parallel and Distributed Systems, Vol. 9, No. 6, June 1998, pp. 585-600.

[CON-99] M. Conti, E Gregori, F Panzieri, “Load Distribution among Replicated Web

Servers: A QOS-based Approach,” In Proceedings of the 2nd Workshop on Internet Server Performance (WISP’99), Atlanta, May 1999.

[CRO-99] M. E. Crovella, R. Frangioso, M. Harchol-Balter, “Connection scheduling in web servers,” Technical Report TR-99-003, Boston University, CS Dept, April 1999.

[CRO-97] M. Crovella, A. Bestavros, “Self-Similarity in World-Wide Web Traffic and Possible Causes,” IEEE/ACM Transactions on Networking, Vol. 5, No. 6, December 1997, pp. 835-346.

[DIA-96] D. Dias, W. Kish, R. Mukherjee, R. Tewari, “A Scalable and Highly Available Web Server,” In Proceedings of the IEEE Computer Conference (COMPCON), San Jose, February 1996.

[DIL-98] J. Dilley, R. Friedrich, T. Jin, J. Rolia, “Web server performance measurement and modeling Techniques,” Performance Evaluation Journal, Vol. 33, Issue 1, June 1998, pp. 5-26.

[DIN-96] A. Dingle, “Cache consistency in the HTTP 1.1 proposed standard,” In Proceedings of the ICM Workshop on Web Caching, Warsaw, Poland, September 1996.

[ENG-98] R. S. Engelschall, “Load Balancing your Web site,” Web Techniques Magazine, Vol. 3, Issue 5, May 1998.

[FEL-97] D. Feldman, A. Krishnamurthy, J. Mogul, “Rate of change and other metrics: a live study of the World Wide Web,” In Proceedings of USENIX Symposium on Internet Technologies and Systems (USISTS), Monterey, CA, December, 1997, pp. 147-158.

- [FIE-97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, et al., "RFC 2068: Hypertext transfer protocol – HTTP/1.1," January 1997.
- [GAD-98] S. Gadde, J. Chase, M. Rabinovich, "A taste of Crispy Squid," In Proceedings of the Workshop on Internet Server Performance (WISP '98), Madison, June 1998, pp. 129-135.
- [GAU-00] D. Gaudet, "Apache Performance Notes", <http://www.apache.org/docs/misc/perf-tuning.html>.
- [GOL-98] Hunt, Goldszmidt, King, Mukherjee, "Network Dispatcher: a Connection Router for Scalable Internet Services" In Proceedings of the 7th World Wide Web Conference, Brisbane, Australia, April 1998.
- [GWE-95] J. Gwertzman, M. Seltzer, "The case for geographical push-caching," In Proceedings of the HotOS '95 Workshop, Orcas Island, May 1995.
- [HU-97] J. Hu, I. Pyarali, D. Schmidt, "Measuring the Impact of Event Dispatching and Concurrency Models on Web server Performance over high-speed Networks," In Proceedings of GLOBECOM '97, Phoenix, November 1997.
- [ING-00] D. B. Ingham, S. K. Shrivastava, F. Panzieri, "Constructing Dependable Web Services," IEEE Internet Computing, Vol. 4, No. 1, Jan 2000, pp. 25-33.
- [IYE-97] A. Iyengar, E. MacNair, T. Nguyen, "An Analysis of Web Server Performance," In Proceedings of GLOBECOM '97, Phoenix, November 1997.

- [IYE-98] A. Iyengar, E. MacNair, M. Squillante, L. Zhang, "A General Methodology for Characterizing Access Patterns and Analyzing Web Server Performance," In Proceedings of the 6th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '98), Montreal, Canada, July 1998.
- [JAI-91] R. Jain, "The Art of Computer Systems Performance Analysis," John Wiley and Sons, 1991.
- [KAT-94] E. D. Katz, M. Butler, R. McGrath, "A Scalable http server: The NCSA prototype," Computer Networks and ISDN Systems, Vol. 27, Issue 2, November 1994, pp. 155-164.
- [KHA-99] A. Khaleel, A. L. Narasimha Reddy, "Evaluation of Data and Request Distribution Policies in Clustered Servers," In Proceedings of 6th International Conference on High Performance Computing – HiPC '99, Calcutta, India, December 1999, pp. 55-60.
- [KRI-98] D. Krishnamurthy, J. Rolia, "The Internet Vs E-Commerce Servers: When Will Server Performance Matter?," In Proceedings of CASCON '98, Toronto, Nov 1998.
- [KRI-99] B. Krishnamurthy, J. C. Mogul, D. M. Kristol, "Key differences between http/1.0 and http/1.1," In Proceedings of 8th International WWW Conference, Toronto, May 1999, pp. 659-673.
- [MAL-99] C. Maltzahn, K. Richardson, D. Grunwald, "Reducing the disk I/O of Web

proxy server caches,” In Proceedings of the USENIX Annual Technical Conference, Monterey, CA, June 1999.

[MAR-98] E. P. Markatos, C. E. Chronaki. “A top-10 approach to prefetching the Web,” In Proceedings of the INET '98 conference, Geneva, Switzerland, July 1998. Available at http://www.isoc.org/inet98/proceedings/1i/li_2.htm.

[MEN-98] D. Menasce, V. Almeida, “Capacity planning for Web performance,” Prentice Hall, June 1998.

[MEN-99] D. Menasce, V. Almeida, R. Fonseca, M. Mendes, “Resource Management Policies for E-commerce Servers,” In Proceedings of the 2nd Workshop on Internet Server Performance (WISP'99), Atlanta, May 1999.

[MOS-97] D. Mosberger, T. Jin, “httperf – A tool to measure Web server performance,” In Proceedings of USENIX Symposium on Internet Technologies and Systems (USITS), Monterey, CA, December 1997, pp. 59-67.

[MYE-99] A. Myers, P. Dinda, H. Zhang, “Performance characteristics of mirror servers on the Internet,” In Proceedings of the IEEE INFOCOM '99 Conference, New York, March 1999.

[NAD-00] S. Nadimpalli, S. Majumdar, “Techniques for Achieving High Performance Web Servers,” (to appear) In Proceedings of the International Conference on Parallel Processing (ICPP 2000), Toronto, Canada, August 2000.

[NIC-98] N. Niclausse, Z. Liu, P. Nain. “A new efficient caching policy for the World

Wide Web,” In Proceedings of the Workshop on Internet Server Performance (WISP'98), Madison, June 1998, pp. 119-128.

[PAD-94] V.N. Padmanabhan, J.C. Mogul, “Improving http latency,” In Proceedings of the Second International WWW Conference, Chicago, IL, October 1994.

[PAI-98] V. Pai et al, “Locality-aware request distribution in cluster based network servers,” In Proceedings of the 8th ASPLOS, San Jose, October 1998, pp. 205-216.

[PAR-96] K. Park, G. Kim, M. E. Crovella, “On the relationship between file sizes, transport protocols, and self-similar network traffic,” In Proceedings of the Fourth International Conference on Network Protocols (ICNP'96), October 1996, pp. 171-180.

[PIE-99] G. Pierre, I. Kuz, M. Steen, A. Tanenbaum, “Differentiated Strategies for Replicating Web Documents,” IR-467, Vrije Universiteit, Dept of Math and Computer Science, November 1999.

[ROA-99] C. Roadknight, I. Marshall, D. Vearer, “File Popularity Characterization,” In Proceedings of the 2nd Workshop on Internet Server Performance (WISP'99), Atlanta, May 1999.

[ROD-00] P. Rodriguez, A. Kirpal, E. Biersack, “Parallel Access for Mirror sites in the Internet,” In Proceedings of 19th annual conference on Computer Communications, IEEE INFOCOM 2000, Tel-Aviv, Israel, March 2000.

[SAY-98] M. Sayal, Y. Breitbart, P. Scheuermann, R. Vingralek, “Selection Algorithms for Replicated Web Servers,” In Proceedings of the Workshop on Internet Server

Performance (WISP'98), Madison, June 1998, pp. 137-144.

[SCH-96] H. Schulzrinne, "World Wide Web: Whence, Whither, What Next ?," IEEE Network, March 1996, Vol. 10, No. 2, pp. 10-17.

[SPE-96] Standard Performance Evaluation Corporation. SPECWeb96 Benchmark, Available at "<http://www.spec.org/osg/web96>".

[SQU-99] "Squid Internet Object Cache," Available at <http://www.squid-cache.org/>

[W3C-97] World Wide Web Consortium, "Replication and Caching Position Statement," August 1997. Available at "<http://www.w3.org/Propagation/activity.html>".

[WEB-00] Mindcraft Inc, World Wide Web Server Benchmarking, Available at <http://www.mindcraft.com/webstone>".

[WES-97] D. Wessels, K. Claffy, "Internet Cache Protocol (ICP), version 2," Internet Draft, May 1997. Available at <http://ircache.nlanr.net/Cache/ICP>.

[WIL-96] S. Williams, M. Abrams, C. Standridge, G. Abdulla, E. Fox, "Removal policies in network caches for world-wide web documents," In Proceedings of the ACM SIGCOMM '96, Stanford University, CA, August 1996.

APPENDIX A

Input file for httpperf session request generator

#session 1

/foo.html

/foo2.html

/pict5.gif

/pict6.gif

/pict7.gif

/pict8.gif

The above description specifies an example session consisting of two call bursts. Each call burst in this example corresponds to the access of a Web page consisting of one html document and four embedded objects. The client session starts with a request for /foo.html. When the /foo.html response comes back, a burst of 4 requests will follow (/pict1.gif, /pict2.gif, /pict3.gif and /pict4.gif). When the last of those responses is received, the next request of /foo2.html is issued. The /foo2.html response is followed by a burst of requests for /pict5.gif, /pict6.gif, /pict7.gif and /pict8.gif which concludes this session.

APPENDIX B

Table B1: Sample of confidence interval values for response times ($WP_i(ts_i = 20\text{msec}, D_i = \text{deterministic}, E_i = 4), B = 2)$)

N	Mean Response time (msec)	Confidence Interval +/- (%)
1	95.69	0.85
3	196.59	0.79
5	305.86	0.44
7	401.40	0.49
9	502.49	0.61