

LIBRARY
Michigan State
University

This is to certify that the

dissertation entitled

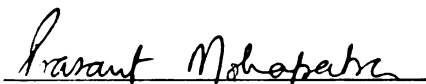
**A FRAMEWORK FOR SERVICE DIFFERENTIATING
INTERNET SERVERS**

presented by

Xiangping Chen

has been accepted towards fulfillment
of the requirements for

Doctoral degree in Computer Science
& Engineering


Major professor

Date 4/28/2000

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE
01 JAN 15 2004		

**A FRAMEWORK FOR SERVICE DIFFERENTIATING
INTERNET SERVERS**

By

Xiangping Chen

A DISSERTATION

**Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of**

DOCTOR OF PHILOSOPHY

Department of Computer Science and Engineering

2000

ABSTRACT

A FRAMEWORK FOR SERVICE DIFFERENTIATING INTERNET SERVERS

By

Xiangping Chen

The variances in application types and the service quality requirements have been increasing continuously. The current *best-effort* service model of the Internet and its servers might not be able to satisfy the evolving service quality demands of diverse applications. Therefore, *differentiated service* has been proposed as a potential solution to provide alternative quality of services (QoS) and is expected to be supported in the next generation Internet (NGI). The Service Differentiating Internet Server (SDIS) is proposed in this dissertation to provide service guarantee to prioritized client requests, which enforces predictable QoS by categorizing requests into different groups based on task types and client identifications and allocating resources based on the groups. Considering the popularity of the WWW on the Internet and its capability of providing a common interface to almost all the protocols on the Internet, Web server performance issues are emphasized in this study.

Four major steps are taken to conduct this study. First, the workload behavior of Web servers under different environments are studied. The size, access pattern, lifetime distribution, and behavior of modification of different types of documents in different classes of web sites are analyzed and compared. Results indicate significant differences in static as well as dynamic characteristics of documents types in different web site classes, which suggests that server workload characterization based on the application environments helps to improve the web server caching efficiency and thus improve the server throughput.

Second, the conceptual correctness of service differentiating Internet servers (SDIS) is examined and verified. Approaches of providing differentiated and QoS based web services, and modeling and analysis of web server scheduling under different workload situations are studied. Experimental studies and analyses prove that a SDIS provides significantly better services to high priority tasks compared to a traditional Internet server under high system utilization.

Third, the overload management versus QoS assurance in a web server is explored. The performance of a overloaded server becomes unstable and may eventually crash. To assure stable and predictable service, a simple and efficient admission control algorithm ACES is proposed. The ACES algorithm bounds response delay of requests by allocation of computational credits based on the estimation of task service times. Experimental study shows that the ACES algorithm provides effective control of response delay bounds under highly variant workload environments while maintaining the system throughput.

We further extend the research on performance assurance issues of SDIS, and present a novel and efficient admission control algorithm, PACERS, which provides services based on the server workload characteristics. Different levels of quality of services are assured by periodic allocation of system resources based on the estimation of request rate and service requirements of prioritized tasks. Theoretical analysis and experimental study show that the PACERS algorithm provides effective control of throughput and response delay boundary to the prioritized tasks, and preserves system throughput under various workload situations.

In summary, this research has the following contributions: traffic characterization which is essential for designing efficient and high performance Internet servers; the conceptualization of SDIS framework and its design to meet the service quality demands of evolving applications and services; development of efficient admission and overload control techniques aiming at performance assurance of Internet servers.

TABLE OF CONTENTS

LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 BACKGROUND	9
2.1 History and Growth of the Internet	10
2.2 Definition of the Internet	12
2.3 Service Model of the Internet	13
2.4 Classification of Internet Servers	15
2.4.1 Best-effort Internet Servers	15
2.4.2 Real-time Internet Servers	16
2.4.3 Hybrid Internet Servers	17
2.5 Performance Challenge of Web Servers	18
2.5.1 E-commerce Services	18
2.5.2 Multimedia Services	19
2.5.3 Service Path and Performance Bottlenecks	19
2.5.4 Optimization of Retrieval Behaviors	22
2.6 Improving Server Performance	23
2.6.1 High Performance Web Servers	24
2.6.2 Distributed Web Servers	25
2.6.3 Server QoS	27
2.6.4 Any Layer QoS Collaboration	29
2.6.5 Service Differentiating Internet Servers	31
CHAPTER 3 SERVER WORKLOAD CHARACTERIZATION . .	33
3.1 Overview	33
3.2 WWW Traces	35
3.3 Experimental Study	35
3.3.1 Summary of Traces	36
3.3.2 Document Types	37
3.3.3 Lifetime Calculation	39
3.4 Results	41
3.4.1 Access Frequency vs. File Size	41

3.4.2	Average Lifetime	42
3.4.3	Modification Distribution	43
3.4.4	Total Lifetime Distribution	47
3.5	Design Issues	49
3.5.1	Document Classification	49
3.5.2	A Two-State TTL Algorithm	50
3.5.3	Caching and Prefetching	53
3.6	Summary	55

CHAPTER 4 SERVICE DIFFERENTIATING INTERNET SERVERS 56

4.1	Overview	56
4.2	Service Differentiation	58
4.2.1	Prioritized Services	58
4.2.2	Customized Services	59
4.3	A Generalized Internet Server	60
4.3.1	Service Differentiating Internet Server Model	61
4.3.2	Server Processing	63
4.3.3	Admission Control	64
4.3.4	Process Scheduling	65
4.3.5	Task Assignment	65
4.3.6	System Overhead	66
4.4	Analysis of Waiting Time	67
4.4.1	Waiting Time in Non-prioritized Systems	67
4.4.2	Waiting Time of Prioritized Systems	69
4.5	Simulation	71
4.5.1	Workload Generation	72
4.5.2	Server Processing	73
4.5.3	Performance Metrics	74
4.6	Results	74
4.6.1	Effectiveness of Priority Based Scheduling	74
4.6.2	Maximum High Priority Ratio	77
4.6.3	Low Priority Task Performance	78
4.6.4	Task Assignment Schemes	79
4.6.5	Preferential Task Assignment	83
4.6.6	Admission Control Performance	84
4.7	Related Works	87
4.8	Summary	88

CHAPTER 5 BOUNDING RESPONSE DELAYS IN BUSY WEB SERVERS 90

5.1	Overview	90
5.2	Admission Control Algorithm	92
5.2.1	Overview of the algorithm	92
5.2.2	Delay Bounds Assurance	93
5.2.3	Admission Control and Service Time	94
5.2.4	The Double Queue Structure	95

5.3	Service Time Estimation	96
5.3.1	Web Object Distribution	96
5.3.2	Service Time Distribution	98
5.4	Simulation Methodology	101
5.5	Results	103
5.5.1	Throughput Comparison	104
5.5.2	Average Delay	105
5.5.3	Delay Bounds Miss Probability	105
5.6	Summary	107
 CHAPTER 6 AN EFFICIENT ADMISSION CONTROL ALGORITHM FOR SDIS		109
6.1	Overview	109
6.2	Workload Characterization	112
6.2.1	Access Distribution	112
6.2.2	Object type distribution	115
6.3	Admission Control Algorithm	116
6.3.1	Overview of the algorithm	116
6.3.2	Admission Control and Delay Bounds	117
6.3.3	Estimation of Request Rate	119
6.3.4	Estimation of Service Time	120
6.3.5	The Double Queue Structure	121
6.4	Response Delay Analysis	122
6.4.1	Ideal Case Delay Bounds	122
6.4.2	Waiting Time Estimation	125
6.5	Simulation Model and Parameters	126
6.6	Results and Evaluation	128
6.6.1	Throughput Performance	128
6.6.2	Delay Performance	129
6.6.3	Sensitivity Test	131
6.7	Related Works	133
6.8	Summary	134
 CHAPTER 7 CONCLUSIONS AND FUTURE WORK		136

LIST OF TABLES

3.1	Summary of logs from EDU class.	36
3.2	Summary of logs from COM class.	37
3.3	Summary of logs from NEWS class.	37
3.4	Types of Web documents.	38
3.5	Access pattern of different types in EDU class.	38
3.6	Access pattern of different types in COM class.	39
3.7	Access pattern of different file types in NEWS class.	39
3.8	Not-modified vs. Get retrieval.	41
3.9	Average Lifetime (unit days).	43
3.10	Highly mutable document ratio.	50
4.1	Trace Data Distribution.	72
4.2	Simulation Parameters.	73
5.1	Trace Data Distribution.	97
5.2	Simulation Configuration.	102
5.3	CQs allocated to each object types.	103
6.1	Traffic distribution vs. object type.	115
6.2	Simulation Configuration.	127
6.3	CQs requested by each object types.	127

LIST OF FIGURES

2.1	Growth of the Internet Hosts.	11
2.2	The path of web connections.	21
2.3	Request processing steps.	21
2.4	Mapping between the OSI Model and Internet infrastructure.	31
3.1	Traffic vs. doc. size of HTM.	42
3.2	Traffic vs. doc. size of GIF.	42
3.3	Modification distribution in EDU class.	44
3.4	Modification distribution in COM class.	45
3.5	Modification Distribution in NEWS class.	46
3.6	Total lifetime distribution of documents in the NEWS class.	48
3.7	The two-state TTL consistency algorithm.	51
3.8	Performance Comparison of three cache consistency algorithms.	52
4.1	Queuing Network Model for an Web Server.	62
4.2	Mean task response time vs. scheduling schemes.	75
4.3	Mean task slowdown vs. scheduling schemes.	75
4.4	95th percentile response time vs. scheduling schemes.	76
4.5	Mean response time vs. high priority ratio.	77
4.6	95th percentile response time vs. high priority ratio.	77
4.7	Mean response time vs. high priority ratio.	78
4.8	95th percentile response time vs. high priority ratio.	78
4.9	Mean slowdown vs. high priority ratio.	79
4.10	High priority task mean response time, priority ratio 1:1.	80
4.11	95th Percentile high priority task response time, priority ratio 1:1.	80
4.12	High priority task mean response time, priority ratio 4:1.	81
4.13	95th Percentile high priority task response time, priority ratio 4:1.	81
4.14	Low priority task mean response time, priority ratio 1:1.	82
4.15	95th percentile low priority task response time, priority ratio 1:1.	82
4.16	Low priority task mean response time, priority ratio 4:1.	83
4.17	95th percentile low priority task response time, priority ratio 4:1.	83
4.18	High priority task mean response time vs. PTA.	84
4.19	95th percentile high priority task response time vs. PTA.	84

4.20	Low priority task mean response time vs. PTA.	85
4.21	95th percentile low priority task response time vs. PTA. . . .	85
4.22	Reject ratio vs. task assignment schemes.	86
4.23	Reject and abort ratio vs. priority groups.	86
5.1	Mean response time of web objects.	99
5.2	Mean service time of web objects.	99
5.3	Web server system structure.	101
5.4	Throughput of stress test.	104
5.5	Throughput of sensitivity test.	104
5.6	Delay of stress test.	105
5.7	Delay of sensitivity test.	105
5.8	Dbm rate of stress test.	106
5.9	Dbm rate of sensitivity test.	106
6.1	Access distribution a day (1 sec).	113
6.2	Traffic distribution a day (1 sec).	113
6.3	Access distribution a day (1 min).	113
6.4	Traffic distribution a day (1 min).	113
6.5	Access trends in a week.	114
6.6	Traffic trends in a week.	114
6.7	Web server system structure.	122
6.8	Throughput using SAC.	129
6.9	Throughput using PACERS.	129
6.10	Performance comparison of low priority tasks.	130
6.11	Performance comparison of high priority tasks.	130
6.12	Throughput under fluctuating load.	131
6.13	Delay under fluctuating load.	131
6.14	Delay bound miss ratio under fluctuating load.	132
6.15	Delay bounds miss ratio under fluctuating load.	132
6.16	High Priority throughput high background traffic.	133
6.17	High Priority delay high background traffic.	133

CHAPTER 1 INTRODUCTION

With the exponential growth of the Internet, many of the latest developments in technology have been aimed at providing increasingly sophisticated information services on top of the basic Internet data communications. Much attention has been on the use of this global information infrastructure for support of commercial services, also well known as e-commerce. Due to its capability of providing quick and easy access to a large variety of information from sites all over the world, the World Wide Web (WWW or Web) has become the most widely used tool for access and dissemination of commercial, educational, and news information on the Internet. Efforts of using the Web in revenue generating activities is increasing at a fast rate [1] and is likely to maintain its growth rate in the future. It is becoming a typical behavior for popular web sites to receive tens of millions of retrieval requests per day.

One problem that is widely existing in the current web service is the unpredictability of the response time, which is not acceptable to time critical transactions that have limited tolerance to response delay. Although contemporary web servers are able to serve thousands of requests in one second, the response delay of a popular server can be several orders of magnitudes higher than the average value during high load periods, causing the *de facto* “denial-of-service” effects. It was estimated that in 1998 about 10~25% of e-commerce transactions were aborted owing to underlying long response delay, which translated to about 1.9 billion dollars lost of revenue [2]. Another challenge imposed on the web service is the increase of web pages with continuous media (CM) objects, such as audio or video clips. These streaming data types have

time constraints or deadlines that need to be met for their meaningful delivery. Sustained bandwidth assurance and low deadline miss ratio is more important than fast response.

The performance of end user perceivable Internet services are often measured as the round trip delay (RTT) of a request/response pair. The RTT is determined by two factors: the data transfer rate of the underling networks, and the Internet server processing capacity. For an Internet server, the performance is generally measured by the service time of each request, and the throughput, i.e., how many requests it can process per unit time. To maintain or improve the service quality of WWW retrieval, several efforts have been made, such as, shorten the retrieval path by client/proxy caching, decrease the retrieval delay by client side prefetching, reduce traffic size by delta transmission or filtering, and temporally or geographically distribute server load by server pushing/replication and server side caching. All of these approaches help to improve service performance from traditional web sites which contain mainly static information in the forms of small files. In web server environments with a significant amount of dynamic and CM objects which are deemed as “uncacheable” because of the frequent update or prohibitive large size, the performance gain from caching, prefetching or replication diminishes.

To improve the server throughput and response delay performance, web server vendors continue to upgrade server hardwares and optimize server software. Although it increases throughput and reduces response delay of a web server, the service migration from an existing infrastructure to a more advanced one is expensive and provides only temporary relief. Even the most powerful servers can be overwhelmed by the continuously increasing request rate, because the number of requests per second that a single server can handle is limited but the internet user population is potentially infinite. Even if high performance web servers or server clusters are available, response times can still hardly be assured at all time frames under fluctuating workloads. Be-

sides, service requirements such as sustained bandwidth or session protection cannot be fulfilled by simply speeding up the response delay.

Recent studies in web servers [3, 4, 5, 6] have addressed the technology of assuring service quality by prioritized task processing in a web server. By tasks prioritization, it is possible to provide high quality of services to high priority tasks during high load periods by blocking or dropping tasks with the lower priorities. We call this kind of servers as Service Differentiating Internet Servers (SDIS).

Similar efforts have been ongoing to introduce quality of services (QoS) to the Internet. The Internet integrated service framework (IntServ)[7] provides the capability for applications to choose among multiple, controlled levels of delivery service for their data packets. Resource reservation setup protocol such as RSVP communicates the requirements of applications to network elements along the path and to convey QoS management information between network elements and the application. In an integrated service enabled network, an application will be able to reserve resources along a route from the source to the destination. QoS aware routers will then schedule and prioritize packets to fulfill the reserved service requirements. One major drawback of the reservation scheme is poor scalability, since the number of reservation setup control messages processed by each router is proportional to the number of flows going through the router. Similarly the resource reservation approach is not appropriate for Internet servers because of poor scalability, low system utilization, and/or long setup delay.

The Internet differentiated service framework (DiffServ)[8] aims to implement scalable service differentiation in the Internet. It achieves scalability by aggregating traffic classification state which is conveyed by means of IP-layer packet marking and regulation of per-hop behavior (PHB). Data packets are classified and marked to receive particular PHB forwarding on network elements along their path. Sophisticated classification, marking, policing, and shaping operations need only be implemented at

network boundaries. Network resources are allocated to traffic streams by service provisioning policies which govern how traffic is marked and conditioned upon entry to a differentiated services capable network, and how that traffic is forwarded within that network. A wide variety of services can be implemented on top of these building blocks, or combined with resource reservation schemes. Differentiated service approach has been proposed as an efficient and scalable solution to provide better service for the next generation Internet (NGI) communication [9].

The objective of this dissertation is to design a service differentiation enabled Internet server which reciprocates QoS efforts from the Internet transmission using IntServ or DiffServ architectures. The server should be able to provide fast response to high priority tasks, timely delivery to real-time tasks, and minimal performance penalty to low priority tasks without degrading the overall system throughput. To summarize, the main motivation of the SDIS project can be itemized as follows:

- Several evolving applications use continuous media (CM) objects, such as Web based radio and TV channels. These data types have time constraints or deadlines that need to be met in order for any meaningful delivery. Thus, they may need service priorities over the non-real-time tasks in terms of isochronous processing, high storage capacity and sustained data delivery.
- The exploding e-commerce market supports several types of transactions and may need to classify services based on revenue and customer groups. The prioritized transactions may be necessary and facilitate e-commerce burstiness.
- The possible adoption of differentiated services in the next generation Internet will make the best-effort servers unacceptable and may defy the purpose of it. Thus, in order to provide end-to-end QoS guarantee, the Internet servers also need to provide differentiated services.

In this dissertation, we develop a model of service differentiating Internet servers, which can be implemented with various resource allocation algorithms including admission control, prioritized scheduling, task assignment and load balancing. Policies and feasibility of service differentiation are studied. Multiple schemes for quality of service assurance are tested and their performance are evaluated. This research shows that, traffic characterization is essential for designing efficient and high performance Internet servers. The SDIS framework is able to meet the different service quality demands of the next generation Internet, as well as evolving applications and services. Furthermore, it has been shown that the performance of an Internet server, such as response delay, drop rate and throughput, can be improved and assured through efficient admission control and overload management.

The main contribution of this dissertation can be summarized as follows.

1. **Web server workload characterization:** The study on the characteristics of web server workload helps to determine how the current web server can be improved to provide better services. This study characterizes the web server workload related to the web object distribution and server environments. Nine Web server traces are collected to represent three different classes of Web environments: educational, commercial and news. The size, access pattern, lifetime distribution, and modification behavior with regard to Web object types in different classes of Web sites are analyzed and compared. Results demonstrate significant differences in static as well as dynamic characteristics of object types in different Web site classes. The efficiency of server caching, thus server response performance, can be improved when caching priorities and Time-to-Live (TTL) preferences are given to certain types of Web objects. Based on the workload characterization study, a two-state cache consistency algorithm is proposed to improve the cache hit ratio and decrease unnecessary polling overhead. Preliminary results show satisfactory performance improvement compared to current

cache consistency algorithms. We also provide guidelines for the design and development of caching and prefetching techniques that exploit the Web workload characteristics in different environments.

2. **Service Differentiating Internet Servers (SDIS):** The conceptual design of SDIS is examined and analyzed. Approaches of providing differentiated and QoS based web services, and modeling and analysis of web server scheduling under different workload situations are studied. The workload include following major types: static small web objects, database workload (mainly dynamic data), multimedia workload (mainly CM data with bounded delay delivery constraints), and the combination of the three major workloads. Experimental study and analyses prove that under high system utilization, a service differentiating server provides significantly better services to high priority tasks compared to a traditional Internet server. Through analytical modeling and simulation study, we show the feasibility and performance benefits of the SDIS. Various aspects, such as admission control, scheduling, and task assignment schemes for SDIS are evaluated through real workload traces. The results of these studies are used as foundation for further studies on design and prototype development of the SDIS.
3. **An admission control algorithm based on Estimation of Service time (ACES):** The ACES algorithm is designed to provide user perceivable response delay bounds from busy web servers, and prevent servers from overloading. The ACES algorithm bounds response delay of requests by allocation of computational credits based on the estimation of task service times. The service times are estimated on the basis of the type of requests. A *double-queue* structure is implemented to diminish the inaccuracy of the estimation of service time and make use of spare capacity of the server, thus increasing the system throughput.

Experimental study shows that the ACES algorithm provides effective control of response delay bounds under highly variant workload environments while maintaining the system throughput.

4. **PACERS admission control algorithm for SDIS:** PACERS algorithm is proposed to provide different level of services based on the server workload characteristics. Alternative levels of QoS are assured by periodical allocation of system resources based on the estimation of request rate and service requirements of prioritized tasks. Admission of lower priority tasks is restricted during high load periods to prevent denial-of-services to high priority tasks. Response delays of most prioritized tasks are bounded by the length of the prediction period. Theoretical analysis and experimental study shows that the PACERS algorithm provides effective control of throughput and response delay boundary to the prioritized tasks, and preserves system throughput under various workload situations.

In brief, the objective of this study is to answer the following questions. First, why service differentiation is needed in an Internet server, what is the performance bottleneck in Internet services, and why not just build best-effort high performance servers? Second, what is the appropriate model of the service differentiating Internet servers (SDIS), and can the desired QoS be achieved in the SDIS? If it can, what is the cost of introducing QoS in Internet servers? Third, how to maintain the performance of a busy web server? Last, how to enforce QoS in the SDIS?

The remainder of this dissertation is organized as follows. Chapter 2 presents the background materials related to this work, including the definition and growth of Internet services, advances of contemporary Internet servers, and QoS researches on the Internet and its servers. In one word, *why service differentiation is needed in an Internet server?* Chapter 3 presents the the server workload characterization study,

which provides baseline of the SDIS design. The conceptual model of the SDIS is discussed in Chapter 4. The ACES algorithm and overload management of a server system is described in Chapter 5. Chapter 6 presents the PACERS algorithm and performance assurance methods in the SDIS. The conclusions and future work are discussed in Chapter 7.

CHAPTER 2 BACKGROUND

The Internet is a global network of thousands of computer networks connecting millions of computers across the world. A simple standard addressing system and communication protocol suite called TCP/IP (Transmission Control Protocol/Internet Protocol) provide an “open” infrastructure to connect different networks, which allows data exchange among various computers with the Internet connectivity. Since the beginning of 1990s, both traffic volume and capacity of the public Internet have experienced doubling growth each year [10]. The exponential growth of the Internet are driven by the continuing increase in number of Internet users as well as increasing innovative applications running on the Internet. Due to its fast growth and the capability of quick and easy access to a large variety of information from hosts all over the world, the Internet has become a widely used tool for global publication and exchange of information. Because of its success, it has been regarded as a revolution in the computer, communications, and media fields, which has been and continues to be a deep and broad influence on our daily life. The ever increasing volume and evolving Internet applications have been demanding enhanced services from the Internet infrastructure and servers. The objective of this dissertation is to explore the direction of the evolution of the Internet and its servers to support new applications and services with different service requirements. Understanding the Internet and its environments is essential in the performance study of the Internet and its servers. The following section briefly reviews the Internet history and growth.

2.1 History and Growth of the Internet

The Internet originated from the research on packet switching technique and the ARPANET project, which was a Cold War project since 1969, to create a communications network immune to a nuclear attack. The current Internet inherits the merits of scalability, reliability, and fault tolerance from its predecessors, but falls short in supporting real-time transmission and response capability due to the limitation of network transmission protocols. In the first few years, the ARPANET was primarily used to facilitate electronic mail (which was invented in 1971) services, well known as E-Mail these days. Later, new features and services gradually spawned, such as remote terminal emulation (first appeared in 1972, later known as TELNET), file transfer services (came out in 1973, later known as FTP), news group publication services (also came to exist since 1973, later were refined to NNTP), and eventually the World Wide Web (WWW, started around 1990).

During the early 1980s, the packet switching technique and the TCP/IP protocols matured and all the networks started using the TCP/IP protocols. The ARPANET became the backbone of the Internet. In 1986, the NSFNET was created, which led to an explosion of connections, especially from universities. The size (number of sites) and capacity (bandwidth) of the NSFNET continued to grow. It replaced the ARPANET as the backbone of the Internet in 1990. Since 1995, most of the Internet traffic in the United States have been routed through commercial network service providers, and since then the Internet has grown to be a commercial success with billions of dollars of annual investment.

Till the beginning of the 1990s, the access to the Internet was restricted to some research institutes and universities, mainly due to the limited backbone capacity, and poorly documented and hard to use services interfaces. The invention of the HTTP protocol, a Hyper-Text Transfer protocol, facilitated graphic information to be communicated and displayed throughout the Internet for the first time. New services

and new interfaces to the existing services were developed that are user friendly, which initiated the proliferation of a large number of the Internet users and hosts. The so called *cyber-community* is a direct result of the Internet success. Figure 2.1 plots a survey of hosts number on the Internet obtained from the *Hobbes' Internet Timeline v5.0* [11], which shows the exponential growth of the Internet hosts. In December 1969, the first prototype of the Internet was built connecting to 4 hosts. After 30 years, the current Internet consists of about 60 million computers around the world. The growth of the the Internet Domains, sub-Networks, WWW sites, and the bandwidth capacity yields similar trends.

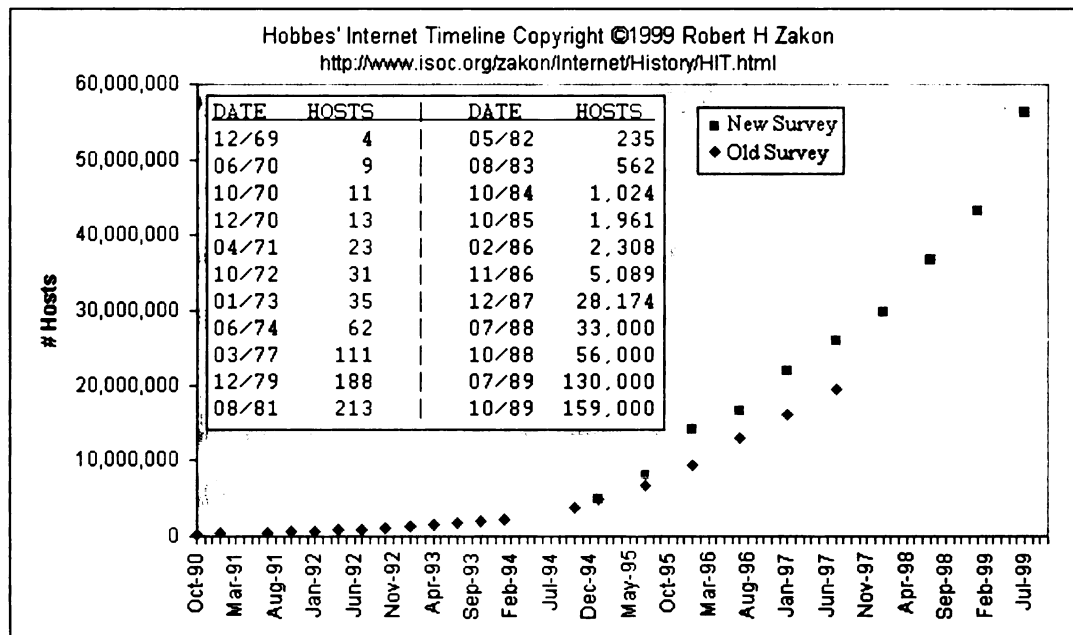


Figure 2.1 Growth of the Internet Hosts.

The Internet continues to change and evolve with the technical advances of the computer and communications industry. With the upgrade of network bandwidth, it is now evolving to provide such new services as real-time data transportation, for example, audio and video streams. The introduction of wireless/mobile access to the Internet, and development of affordable portable network access equipment such as

hand-held computers, is making possible the pervasion of nomadic computing and communications. The evolution of Internet applications introduces diverse workload characteristics and server requirements, thus demands much more sophisticated forms of service provisioning and billing policies, which in turn will drive further the evolution of the Internet.

2.2 Definition of the Internet

The deployment of the TCP/IP protocols and interconnectivity among different networks are essential components in the evolution of the Internet. To further study the Internet and its servers, we need to have an accurate definition of the term “Internet”. In 1995, the Federal Networking Council (FNC) passed a resolution defining the “Internet”:

The **Internet** refers to the global information system that – (i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons; (ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and (iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure described herein.

The IP protocol is a network-layer protocol (Layer 3 in the OSI reference model) that contains addressing and some control information that enables packets to be routed between hosts. Along with the TCP, IP represents the heart of the Internet protocols. It has two primary responsibilities: providing connectionless, best-effort delivery of datagrams through an internetwork; and providing fragmentation and re-assembly of datagrams to support data links with different maximum transmission

unit sizes. IP addressing scheme enables computers with different hardware configurations or using different operating systems to communicate with each other, such as personal computers running DOS, Windows 95/98 or Windows NT operating system, Macintosh computers, or UNIX-based workstations.

The TCP is a transport-layer protocol (Layer 4 in the OSI reference model) which provides reliable transmission of data in an IP environment. TCP delivers an unstructured stream of bytes identified by sequence numbers. It offers reliability by providing connection-oriented, end-to-end reliable packet delivery through an inter-network. It does this by sequencing bytes with a forwarding acknowledgment number that indicates to the destination the next byte the source expects to receive. Bytes not acknowledged within a specified time period are retransmitted. TCP offers efficient flow control, which means that, when sending acknowledgments back to the source, the receiving TCP process indicates the highest sequence number it can receive without overflowing its internal buffers.

The User Datagram Protocol (UDP) is another transport-layer protocol that belongs to the Internet protocol family, which is a connectionless protocol providing no reliability, flow-control, or error-recovery functions to IP. Because of simplicity, UDP headers contain fewer bytes and consume less network overhead than TCP. UDP is useful in situations where the reliability mechanisms of TCP are not necessary, such as in cases where a higher-layer protocol might provide error and flow control. TCP together with UDP provide an interface between IP and upper-layer applications.

2.3 Service Model of the Internet

The most popular network service architecture in the current generation Internet is the *client/server* architecture, which allows end users to share resources provided by the server in a simple and robust manner. The server is generally a powerful computer system that is dedicated to managing shared resources such as storage, network

bandwidth, or simply just computation time. Clients are generally less powerful computers, such as personal computers or workstations, to access resources provided by servers on behalf of the end users. A client sends out requests to a server throughout the Internet using a specific language (protocols above TCP/IP protocol) understood by the server, and the server sends back response using the same protocol. The client interprets and displays the response to end users.

Another type of network architecture, which is known as *peer-to-peer* architecture where each node has equivalent responsibility, is becoming widely used with the progress in computation power and storage capacity of personal computers. The *peer-to-peer* architecture is still limited by the lack of commercially-available object orientation analysis and design tools. To adopting to the *peer-to-peer* architecture will require more hardware in the to client hosts, and generally higher overhead in execution. The *client/server* architecture will still be popular and widely used in the Internet in the future. Our research of Internet services performance is thus based on the *client/server* architecture.

The end user perceivable performance in a *client/server* structured Internet services is often measured as the round trip delay of a request/response pair. The duration of the round trip delay is determined by the server performance as well as the underlying network bandwidth. The network should be able to transmit data fast and with no error. The server should be able to processes requests with little waiting time and service time.

How to deliver the information fast and reliably throughout the Internet, and how to provide quality of service in the network transmission is a rich and active research area, and lots of efforts have contributed to this field. Since the early 90s, the Internet bandwidth has doubled each year. The capacity of IP routers has also been increasing. New technology combining routing and switching techniques such as multiple protocol label switching (MPLS) [9] are beginning to be implemented to

simplify the computation complexity of routing and speed up the packet forwarding inter networks. Network level Quality of Service (QoS) provisioning have been investigated for a few years for implementation in the next generation Internet.

This dissertation is focused on the performance assurance of Internet servers, which is still quite limited in the literature reports. To further investigate the needs and future direction of the Internet servers, a brief review of current Internet servers in terms of service quality requirements is given out in the following section.

2.4 Classification of Internet Servers

Internet servers can be roughly classified into the four categories according to the requests QoS requirements: **Best-effort** servers, **Soft real-time** servers, **Hard real-time** servers, and **Hybrid** servers.

2.4.1 Best-effort Internet Servers

Best-effort servers are so far the most popular Internet servers. The major performance metrics of a best-effort server is the throughput. The server tries to serve as many client requests as possible in the *first-come-first-serve* (FCFS) manner. Usually there is no resource management technique is implemented in a **best-effort** server, and clients do not expect best-effort servers to always respond promptly. Response delay of several seconds, even minutes is acceptable, as long as most of the requests can be processed reliably.

An E-Mail server is a good example of a **best-effort** server. Messages sent through e-mails could arrive within a second, or as long as several hours, depending on the server load and network traffic. Interactive applications such as TELNET are not suitable for the **best-effort** paradigm, due to the unpredictability of response time which might not be able to meet the human desired interaction speed. However, TELNET servers are still implemented as **best-effort** servers in which fast response

is approximated by capacity planning of high performance servers. FTP servers are best-effort servers with preliminary admission control. A FTP server should be able provide acceptable bandwidth for bulk data transfer. Thus the number of active clients are limited to protect the transmission speed of existing sessions. Most of current HTTP servers, also called Web servers, are best-effort servers. However, temporary overload situations can degrade the server performance drastically, and in some cases, crash the server. One example of overloaded servers in the recent past is the traffic jam on the eBay on-line auction site, which froze out bidders for hours during the early summer of 1999 [12].

2.4.2 Real-time Internet Servers

Soft real-time and **Hard real-time** servers are emerging with the growth of multimedia applications on the Internet such as video conferencing, IP telephony, or Web radio/television broadcasting. Unlike **best-effort** servers, a **real-time** server should be able provide services within a certain time constraint, or known as deadline.

For **hard real-time** servers, responses which miss the deadline are meaningless and treated as failure. Most of earlier **hard real-time** services are ensured by using dedicated or private resources. In the current public Internet, hard real-time applications are still rare due to the best-effort nature of the Internet communication protocols. Assurance of the **hard real-time** constraints are emulated by forcing the server to work in a extremely low workload situation, or by using leased-lines for network connections. For **soft real-time** servers, occasional missing of deadline is tolerated, but not desired. Most of continuous media data transmission requires soft real-time delivery and transmission assurance.

There has been ongoing efforts in introducing real-time communications into the Internet, such as integrated network (IntServ) services using the Resource ReSer-Vation Protocol (RSVP), which provides hard real-time assurance of the round-trip

transmission delay by end-to-end channel reservation. Differentiated services (Diff-Serv) in the Internet Engineering Task Force (IETF) community, has been proposed as an efficient and scalable alternative for resource allocation to provide soft real-time assurance by defining the per-hop-behavior (PHB) of an IP packet, and building DiffServ capable routers. However, network layer QoS provisioning is not sufficient in guaranteeing user perceivable high performance if the server does not provide any means of service and performance assurances. Unexpected bursts of client demands may cause long queuing delay or even crash an overloaded web server. For example, a premium data flow with end-to-end QoS guarantee from the Internet transmission may still experience *denial-of-service* from an overloaded end server just like any best-effort data flow. Systematic research on real-time response assurance from an Internet server is needed to fulfill the dynamics of Internet applications.

2.4.3 Hybrid Internet Servers

Hybrid servers are able to serve requests with different service requirements at the same time. For example, Web servers should be able to serve request with *real-time* or *best-effort* requirements fairly. The World Wide Web comprises of Internet servers that supports hypertext to access almost every Internet protocol on a single interface, which include e-mail, file transfer, Telnet, News groups, and its own protocol of HTTP. Therefore, a web server should have the capability to accommodate applications with different service quality restrictions. Mostly due to the success of the Web technology, allowing users easy access to information linked throughout the globe, the Internet has now become almost a “community” service, standing beside telephone, radio, or television services. Due the capability of organizing almost all kinds of data formats into a single “page”, there has been an explosive increase of evolving applications that use the World Wide Web as a distributed information exchanging interface on the Internet.

To most home users, the Internet has been informally renamed as the Web. At the moment, most people use the term “Internet” to refer to the physical structure of the network, including computers and network elements that connect computers. They use the term “Web” to refer to the information space that can be accessed through the Internet. In the following discussions, the terms “Internet Server” and “Web Server” are used exchangeably to refer to the sites that provide Internet services and publish information on the Internet. In the next section, we discuss the current performance challenges of Internet servers.

2.5 Performance Challenge of Web Servers

It is common for a popular web site to receive tens of millions of retrieval requests a day. Many of the latest developments in technology have been aimed at providing increasingly sophisticated information services on top of the basic Internet data communications. For example, much of the latest attention has been on the use of this global information infrastructure for support of commercial services. Efforts to use the Web in revenue generating activities, or well known as e-commerce, is increasing at a fast rate [1] and likely to maintain its growth rate.

2.5.1 E-commerce Services

The e-commerce servers generally provide support for a variety of request types - browsing, products selection, billing arrangement, shipping agreement, and banking management. Some companies are highly successful in using the Web as their market routes, such as Amazon online bookstore and eBay online auction site. One problem that exists widely in contemporary web servers is, however, the unpredictability of the response time, which is not acceptable to time critical transactions which have low tolerance of response delay. The commercial usage of the Web has changed the data distributions in the web sites. More and more dynamic data are generated to provide

information sharing between different enterprise management systems and personalized services. This type of access requires more processing power and resources and thus adds to the server load and unpredictability.

Another performance challenge brought by the e-commerce is the wide use of secure transactions. The security mechanisms such as Secure Sockets Layer (SSL) or IPsec impose a very high load on the Internet servers due to the encryption, decryption and authentication processes. Some recent tests conducted by researchers at Networkshop Inc. indicate that Web servers capable of handling hundreds of transactions per second may be brought down to just a few transactions per second. Some server configurations suffered as much as a fifty-fold degradation in performance from SSL effects.

2.5.2 Multimedia Services

Continuous Media (CM) traffic, which includes audio and video streams, differs from traditional data in terms of resource and transmission requirements. These data types have additional needs in terms of temporal contiguity, high storage capacity, high bandwidth, and encoding and decoding support. They also require bounded delay guarantees in transmission, which contributes to the load.

The current web servers handle the CM request in the same manner as big files. As a result, non-CM traffic impairs the performance of CM traffic and vice versa. However, more and more CM objects are being used on the Web for audio-visual effects and revenue generation (Web TV), which puts additional burden on the server in the sense that it needs more elaborate resource management.

2.5.3 Service Path and Performance Bottlenecks

The current web servers are like enhanced file servers in an extremely large distributed system (i.e., the Internet). The difference is that “page”s, instead of files,

are managed by a server and accessed by clients across the world. A page is a web object in the HyperText Markup Language (HTML) format, which may contain links to other web objects or files. A page can be stored in the server as a disk file or generated on the fly in response to a request. The Unique Resource Locator (URL) protocol specifies how to locate a page stored in a Web site, providing naming and directory services to clients. The HTTP protocol allows pages downloading from and uploading to the server across the Internet. Web clients and servers communicate using the HTTP protocol on top of the TCP layer. Detailed description of a HTTP connection can be found in [13].

When a client makes a request to a particular web server, a TCP connection is established between the client and the server. However, if there is a caching proxy server sitting between the client and the server, the client requests might be able to be served by the proxy server, provided that the proxy server has a fresh copy of the requested URL. Once the connection is established, the client sends a request to the server. The server parses the request and issues a response. After the transmission of the response data completes, the TCP connection is closed. Figure 2.2 shows the path of web connections, where a domain represents a sub-network in the Internet.

Upon receiving a request, the server parses the URL and authenticates the user, then loads or computes the requested data. Once the server has obtained the requested content, it sends back a HTTP header followed by the content through the established TCP connection. A dynamic web object brings extra burden to the server because the response is generated by auxiliary applications running on the server, which consumes the computing power of the server. A server tries to process as many requests as possible in the FCFS order. Requests which exceed the server capacity are dropped. The use of real-time and streaming data delivery has yet to be considered in a traditional web server. Figure 2.3 depicts the basic sequential steps for serving a request.

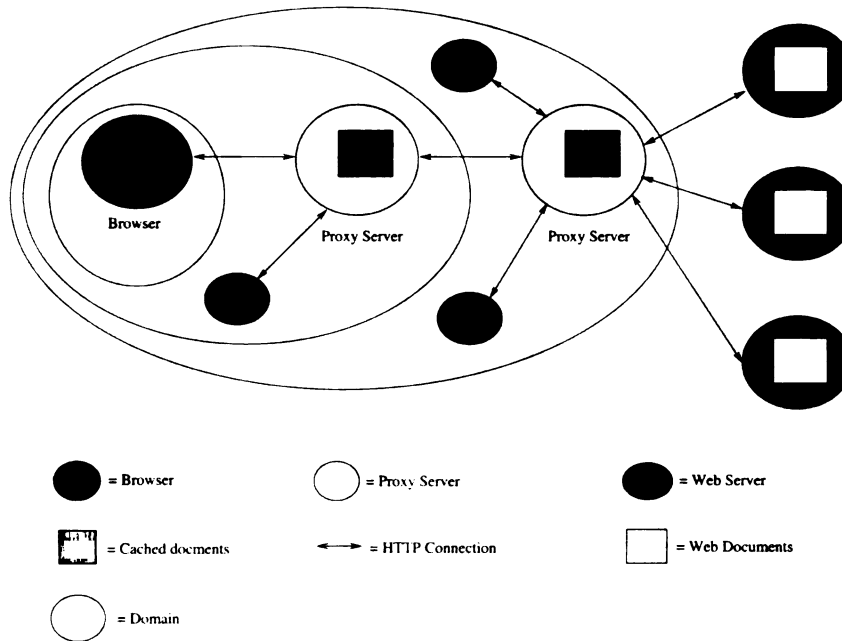


Figure 2.2 The path of web connections.

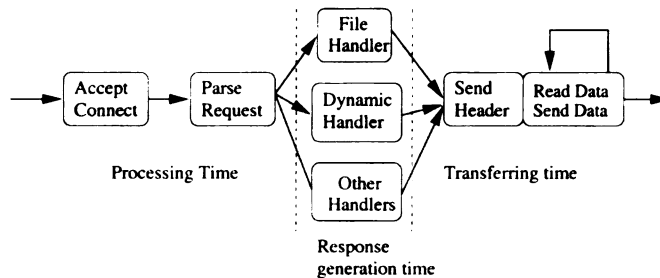


Figure 2.3 Request processing steps.

Possible performance bottlenecks which may contribute to long response delay of a web based transactions include server side CPU processing power and network bandwidth, and client side network bandwidth. Network backbone is generally not considered as a weak factor in web server performance research [14]. In a fast growing Internet, server CPU and network bandwidth might become bottlenecks of web services because of relatively slow capacity upgrade.

2.5.4 Optimization of Retrieval Behaviors

To improve the response performance of Web retrieval, several efforts have been made along the retrieval path. Caching data copies at client hosts or proxy servers [15, 16, 17, 18, 19] is regarded as a good technique for reducing both the network traffic and the server load by shortening the retrieval path thus improving retrieval latency of web pages. One problem comes with the caching technique is the maintenance of data consistency. If cached data copies are not consistent with data in the original web server, stale information will be sent back to web users, which impairs the value of web caching.

Another technique is called client side prefetching [20, 21]. Client side prefetching agencies retrieve data copies to local cache prior to client actually requesting it, thus reducing the retrieval latency by predicting the client browsing behaviors. However, misprediction introduces extra traffic and load on the servers and networks. Precise prediction algorithms are needed to balance the cost and effectiveness of the prefetching technique.

Server pushing/replication [26, 27] and caching [22, 23, 24] are used to temporally or geographically distribute server load. Temporal load distribution includes server side pushing of information during light load periods. Geographical distribution of server load includes mirror sites generation and redirection of requests during high load periods.

Delta transmission or filtering [25] is used to decrease the traffic volume between clients and servers, thus improves the retrieval performance. However, overhead are brought to both client and server side to compress and decompress data. Multiple version of data are needed for delta compression, which occupy more storage space.

All of the above approaches assume a traditional web site which contain mainly static information in the forms of small files. When dynamic objects requests have

high percentage of access rate, the effectiveness of above approaches decreases. Besides, the CM data is considered as cache inefficient, thus media services from the Web can hardly benefit from the above approaches.

2.6 Improving Server Performance

Although the current server processing model provides a simple and robust service paradigm. There are still some problems that restrict the wide usage of Internet servers as the e-commerce vehicle. First, a server has to drop client requests due to limited number of open connections and buffer space when the server processing speed can not keep up with the request rate. Dropping of requests causes retransmission from the client hosts, which puts additional load on the server as well as the network and thus causes a “snowball” effect. The “snowball” effect might lead the system to overload situation very quickly and even cause live-lock of a web server wherein no request can get service. Second, a server does not keep any state information for any session. It is quite possible that a nearly finished session is interrupted by a burst of incoming requests. In an online trading web site, success in completion of a session or a transaction is critical in bringing in profits. The stateless mode of server processing restricts the potential of revenue generation. Third, the existing best-effort Internet servers do not allocate scarce and expensive network and CPU resources during periods of high workload in terms of the importance of the requests or the expected response delay, which causes unpredictable response delay that is not acceptable to time critical and important transactions. A new service model and QoS aware server is needed to accommodate the requirements of emerging applications on the Internet.

Usually the Internet Service Providers (ISP) and Internet Content Providers (ICP) service all clients requests with the same level of performance. Extensive efforts have

been made both from both the academia and industry to scale up the server performance with the ever increasing server workload. However, work on at the servers has been limited. In recent years, increased usage of Internet has resulted in the shortage of network capacity and server processing power, compromising performance of traditional applications. At the same time, new applications have emerged with much improved service quality requirements. As a result, service and content providers are finding it necessary to offer alternative levels of service to meet customer demands for new applications. Differentiation of services allow service providers to increase their revenues through premium pricing policies by competitive service assurance. Thus the concept of QoS aware Web server has been proposed to compliment the service differentiation efforts from the network.

2.6.1 High Performance Web Servers

The most direct way to solve the server resource shortage with increasing demand is to upgrade server hardwares, such as increasing the CPU speed, using symmetric multiprocessing (SMP) or main-frame computer architectures , employment of high capacity cache and memory, or increasing the disk and network I/O capacity. The results published in [28] with the web server benchmark tests using SPECWEB99 provide a rough idea of hardware configuration of current commercial web servers. Although upgrading hardwares sounds simple and effective, it may be too too expensive to be acceptable as the upgrade requests might be too high for the entire spectrum of workload.

Optimization of the web server processing procedure is another ongoing effort and has been mainly focused on the following aspects. Reduction in connection establishment overhead can be achieved by persistent (or Keep-alive) connections being deployed in the HTTP 1.1 protocol [29]. Reduction of idle CPU overhead in processing a task can be achieved by multi-process, multi-threaded web servers [30],

because single-process architecture can not maintain good performance on disk-bound workloads that exceeds server cache capacity. Most current commercial web servers use multi-threaded structure which is faster than the multi-process server architecture, however, the multi-threaded model requires kernel level threads support from the operating system. Reduction of dynamic object processing overhead has encouraged efforts in Fast-CGI [31], dynamic data caching [32], and in-process dynamic processing such as NSAPI or ISAPI [33]. Other efforts include the wide usage of caching at all levels, and precomputing of URL and response headers.

The performance improvement from software optimization and hardware upgrade is comparatively high. The throughput of a single host web server in 1995 was at the most 200 static requests per second [34]. In 1998, most of the commercial web servers, e.g., Apache web server, Netscape Enterprise Server, and Microsoft Internet Information Server, have achieved benchmark throughput of 1000 to 4000 static requests per second [35], which are about 10 times faster than their predecessors.

Although adding hardware resources or buying a bigger server machine increases throughput and delay performance of a web server site, the replacement of an existing machine with a faster model is not cost-effective and provides only temporary relief. Even the most powerful server will be overwhelmed by the increasing request rate. Furthermore, the strain on Internet resources is likely to become more significant over the next several years as the number of commercial services and the size of the data objects being exchanged are likely to increase dramatically.

2.6.2 Distributed Web Servers

Besides building single high performance web servers, several studies have suggested to distribute load to multiple web servers. One approach to handle popular web sites might be the replication of information across a mirrored-sites network, which provides a list of independent URL sites that have to be manually selected by

the users. This solution relies on the user conjecture in load sharing, which is hardly effective in balancing the load among each mirror site. It also brings the overload of user “probing” behaviors [36].

The SWEB [37] project (University of California, Berkeley) of building scalable distributed web servers is based on the physical structure of Network of Workstations (NOW). A heavily loaded server replies with an *HTTP-redirection* code and provides a new URL to which the client can resubmit its request. However, by redirection, one access operation may require two or more connections to different web servers, thus this mechanism increases the response delay and network traffic in exchange of load sharing among a network of web servers. A variant of the HTTP-redirection approach is used in the *scalable web server architectures project* [38], where two levels of servers are used: redirection servers and normal HTTP servers. Data are partitioned according to their content and stored on different HTTP servers to achieve better caching efficiency. Redirection servers are used to distribute the users’ requests to the corresponding HTTP servers using the redirection mechanism supported by the HTTP protocol.

The Domain Name Server (DNS) approaches map one server host name to a set of server IP addresses [39], thus a cluster of HTTP servers appear to be a single server to clients. Round-Robin or Weighted Round-Robin DNS resolution is used to distribute Web requests across the cluster, and the distributed file system mechanism is employed to maintain a synchronized set of documents across the cluster. For example, there are seven IP addresses mapped to a single host name in the web site of *www.microsoft.com*. The problem with this approach is the temporary load imbalance caused by bursts of requests from a client, or caused by caching effects from name servers. The DWS [40, 41] project extends the DNS based load sharing architecture by adding some optimization criterion (e.g., load balancing, minimization of the system response time, minimization of overloaded servers, and client proximity). Another

variant of the DNS approach is to map a single server IP address into several hosts [42] in a Local Area Network (LAN). Temporary load imbalance caused by bursts of requests from one client is still remains unsolved.

Another project on building scalable distributed high performance web servers is being conducted at the IBM T. J. Watson research center. The reported techniques include server clustering, task distribution by TCP routing [43], and a server accelerator [44]. The TCP routing approach publishes the address of the server-side router. Every client request is sent to the router which then dispatches the request to an appropriate server based on the load situations. The dispatching is performed by changing the destination IP address of each incoming IP packet to the address of a selected server. The TCP routing approach provides better load balancing and more flexibility than the DNS approach. The server accelerator is a dedicated file server providing fast response to a small set of popular files. Other efforts of building high performance web servers over high speed networks include the JAWS project conducted at Washington University [45], where a prototype high performance web server that adapts to traffic patterns was built.

2.6.3 Server QoS

Even though high performance web servers or server clusters are available, response times can still hardly be assured at all times under fluctuating workloads. The “bursty” nature and the rapid growth in the volume of the Web traffic makes it expensive and inefficient to scale up network bandwidth and server capacity with the increasing demand. The third approach is to selectively provide resources and services based on the importance and resource requirement of a request, or by making the service aware of the Quality of Service (QoS) requests. Guarantee of service quality can be maintained by reservation and pre-allocation of resources for value desirable (premium) tasks. No matter how high the real workload is, those premium

tasks enjoy dedicated service resources, thus are immune to the high load influence imposed on servers. The analogous policy for the Internet transmission is end-to-end resource reservation through the Resource ReSerVation Protocol (RSVP) [46]. One major drawback of RSVP protocol is the poor scalability, since the number of RSVP control messages processed by each router is proportional to the number of QoS data flows going through the router. Similarly the resource reservation approach is not appropriate for Internet servers owing to poor scalability and low utilization problems.

Strict QoS enforcement is sometimes too pessimistic and expensive, especially to soft real-time applications wherein occasional degradation of service quality is tolerable. Statistical QoS instead can be used in the servers, where strict deadline guarantee is not necessary. Differentiated service (*DiffServ*) [47] has been proposed as an efficient and scalable solution to alternate the resource reservation approach discussed above, which is expected to be supported in the next generation Internet (NGI) communications [9]. *DiffServ* creates an “express-way” for high priority traffic by selective reallocation of network resources during peak workload periods. Similarly, prioritized services can be exploited from a service differentiated server to provide predictable services with statistical significance.

Recent studies on QoS support in web servers [3, 4, 6, 48] have addressed the technology of prioritized task processing in a web server and related performance issues. Bhatti and Friedrich [3] addressed the importance of server QoS mechanisms to support tiered service levels and overload management. A Web-QoS architecture prototype was developed by adding connection manager module to the Apache [49] Web server. Admission control and scheduling schemes to assure tiered services were also discussed.

Almeida et. al. [4] attempted to augment a web hosting server implementation with differentiated QoS features. They explored priority-based request scheduling at

both user and kernel levels, and found that by controlling the numbers of processes, one can improve the response time of high-priority requests notably while preserving the system throughput. They also reported that the kernel-level approach tends to penalize low-priority requests less significantly than the user-level approach, while improving the performance of high-priority requests.

Eggert and Heidemann [6] evaluated application level mechanisms to provide two different levels of web services, including limiting process pool size, lowering process priorities, limiting transmission rate, etc. Their results show less than 17% of high priority tasks are downgraded in all the cases by limiting the sending rate of background responses, even without the support from operating systems and networks. Pandey et. al. [48] described a distributed HTTP server which enables QoS by prioritizing pages on a web site and allocating server resources based on the priorities.

The above research indicates that by prioritization of tasks, it is possible to maintain high responsiveness of high priority tasks in high server load periods by blocking or dropping tasks with lower priorities. In fact, Hewlett Packard recently introduced a web server helper product called WebQoS [50], which already has the preliminary functionality of differentiated services on servers.

Other QoS assurance efforts on Web servers include QoS aware file content transcoding [51] and adaptive content delivery [52], which try to improve server throughput by downgrading the web objects quality during high load periods.

2.6.4 Any Layer QoS Collaboration

The idea of SDIS is stimulated by the concept of “any layer QoS”. The Open System Interconnection (OSI) model defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer at one site, proceeding to the bottom layer, over the communication channel to the next site and back up the hierarchy. Most of the computer communications systems have comparable layer structures with the OSI model, although two or

three OSI layers may be incorporated into one. Figure 2.4 maps the current Internet infrastructure with the OSI model.

ATM (Asynchronous Transfer Mode), is a protocol that transmits data as fixed sized packets/cells (see <http://www.atmforum.com/atmforum/index.html> for details). It was designed to make Broadband-ISDN (B-ISDN) a reality. B-ISDN was aimed to function as a communication network that can provide integrated broadband services such as high-speed-data service, video phone, video conferencing, CATV services. It is now mostly deployed at Datalink layer to provide intermediate fast switching for the Network backbone. IP protocol works at Network Layer, TCP/UDP protocols implement the functionality of Transport Layer. ATM is the first network protocol which defines classes of services (COS) and quantifies QoS. Numerous studies on ATM QoS have been reported on the literature. Interaction between TCP/IP and ATM with respect to QoS assurance is limited, however. Efforts on QoS guarantee from ATM switch sometimes counteract the effects of DiffServ or IntServ. Consider a scenario in which a high priority IP packet with low loss rate requirement is segmented into several ATM cells in an ATM network. If one of the cells is dropped by an ATM switch since it has no knowledge of the priority status of the IP packet, the entire packet has to be discarded at the destination due to incomplete transmission. There is a need to ensure that characteristics of IP services based on Differentiated Services (DS) architecture are maintained end to end across intermediate ATM networks [53].

Application layer QoS study, although still in its infant stage, is desirable. Applications have better understanding of data flow characteristics. It can provide finer control over resource allocation and distribution. For similar reason, application layer of QoS should be able to collaborate with network layer QoS effort to assure the user perceivable performance. For example, when a request comes to the network interface of a web server, it might carry QoS markers from lower network layer protocols, such as *DiffServ* or RSVP. The network interface should be able to respect the QoS markers and pass the QoS requirement to the higher application layer. The IP address of

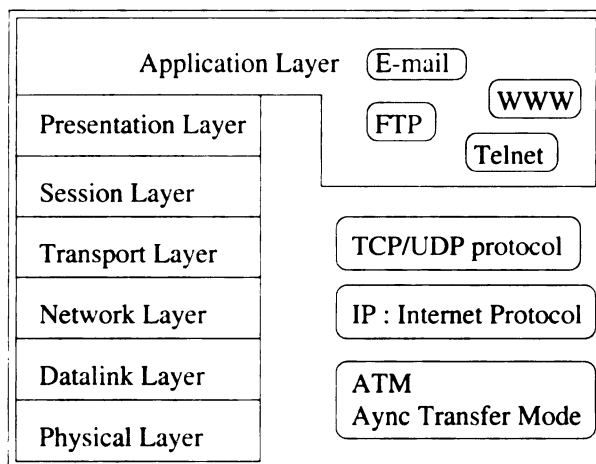


Figure 2.4 Mapping between the OSI Model and Internet infrastructure.

a client can also be served on the basis of their priority, although security issues need to be addressed to prevent forgery of IP addresses.

2.6.5 Service Differentiating Internet Servers

With the explosive growth of the Internet users and capacity, the diversity in the Internet service types and their resource requirements have been also increasing. New applications have emerged with much improved response time requirements. The demand for near real-time response are increasing, which can hardly be assured from a best-effort internet server. The contemporary egalitarian service model of the Internet and its server seems to be inadequate for the evolving demands. As the Internet grows it becomes obvious that various Internet applications can benefit from quality of service provisioning and differentiation. The Service Differentiating Internet Server (SDIS) is proposed to complement the efforts of Internet QoS provisioning and accommodate the service requirements of evolving Web applications. The QoS can be assured through prioritized processing, admission control, and scheduling schemes. During congestion SDIS should provide much better services to high-priority requests with minimal impact on the low priority requests.

When a request is sent to the server, the HTTP header provides further information of how to deal with the request, e.g., the URL of an object may provide hints of priority. The processing of a request in the SDIS can be classified as *real-time* with low rates of *denial-of-services*, *real-time* with delay bounds, and *best-effort* requests. Resource distribution and scheduling of each task can be done with respect to its classification. Client side operating system and browser information are also exposed in the HTTP header, which provides hints of client side network capacity. A SDIS server should be able to tailor the response information a client requested to meet the characteristics of the client connectivity. The ability of tailoring a web page is especially cherished in wireless web browsers and low bandwidth dial-up users. When a response begins to be delivered to the client side, the server should consider the task priority, delay and delay jitter requirements as well as client side bandwidth. The careful smoothing of transmission eliminates the congestion control overhead from lower network layers.

This chapter introduces the related works in the Internet QoS and server performances. It also pointed out the shortcoming of current *best-effort* service model of Internet servers. With the advances of Internet applications, more and more services can benefit from the QoS provisioning from the Internet and its servers. However, what is the appropriate model of the service differentiating Internet servers is still unclear. In the next chapter, we study the workload characteristics of web servers. The characterization of server workload helps clarify the emphasizes of a QoS aware server.

CHAPTER 3 SERVER WORKLOAD CHARACTERIZATION

3.1 Overview

The success of the web is largely due to its capability of providing quick and easy accesses to a large variety of information from sites all over the world. The variety of information and its behaviors however cause the unpredictability of retrieval latency, which agonizes web users. Characterization of server workload is essential to study the optimization and improvement of services. Several studies have been done on characterizing web traffic, especially the static behavior of web objects, including file size distribution, traffic distribution, file type distribution, etc. Pitkow [54] did an excellent survey on workload characterization study in recent years. However, little work on dynamic characteristics such as lifetime behavior and modification behaviors of web objects has been reported in the literature. Chankhunthod et. al. [18] periodically sampled 4600 web objects distributed in 2000 web sites and calculated the mean lifetime of these objects. Although they indicated that lifetime values of web objects varying widely, the lifetime distribution was not given in the paper. Gwertzman and Seltzer [55] mixed the statistical results of web objects from educational and commercial web sites, which concealed possible influences from the web environments. The rate of change of web objects in two corporate communities was studied in [56]. Such aggregated results might not be a true reflection of the characteristics of either kind of sites.

Workload characteristic study in terms of the server environments is also needed. We believe that different performance optimization should be conducted in different web server environment due to their different service objective and information characteristics. For example, caching of data copies along the retrieval path is regarded as a good technique for reducing both the network traffic and the server load, thus improving retrieval latency of web objects. One problem existing in caching techniques is data consistency. If cached data copies are not consistent with data in the original web server, stale information will be sent back to web users, which impairs the value of web caching. Thus the modification behavior of web information should be studied to improve the data consistency. However, the dynamic behavior of web objects are largely depend on the context and environment of a web site.

In the following research, we study the relationship between the characteristics of web objects in different classes of web sites with respect to object types, and the relationship between static and dynamic characteristics of web objects. Characterization of the lifetime and modification of web documents could provide reference data in the design process of caching and prefetching techniques. An efficient cache consistency policy may reduce unnecessary client polling, thereby reducing server load, network traffic, and retrieval latency. This research shows that the lifetime and modification behavior of a web document is not only influenced by its type, but also by the usage environment of the web site where it is placed.

The rest of the chapter is organized as follows. Web traces architecture used in the study is introduced in Section 3.2. Section 3.3 summarized the web traces used in the study and data analysis methods. Section 3.4 compares and analyzes the web objects characterization results. The inferences derived from the study and their impact on the design issues are outlined in Section 3.5, followed by the concluding remarks in Section 3.6.

3.2 WWW Traces

Access logs of nine web servers and proxy servers are used as traces in this research. Each line in an access log contains information of a single request for a document. A common log entry is in the form of:

HostName - - (*UserName*)¹ [*Timestamp*] “ *Request_Method URL*
(*HTTP_Version*) ” *Status_Code Size*

HostName is the name of the remote computer sending out requests on behalf on the user. *UserName* is an authenticated user name when the **User Authentication** scheme is used. Followed that is the timestamp of the connection. There are three *Request_Methods* in the access logs: **GET**, **POST**, and **HEAD**. A **GET** request asks a document from the server or triggers the execution of a program which produces the output to be sent to the client, which is used for standard page requests. The **POST** method is used to send out formatted input data to a program in the server. The execution of the program produces output that sent back to the client. The **HEAD** method is used for testing. *Status_Code* is a three digital number which describes the retrieval status of a transaction. Success codes are in the 200s; redirect codes are in the 300s; failure codes are in the 400s; and server error codes are in the 500s. *Size* refers to the size of the response content.

3.3 Experimental Study

We have collected trace files from three classes of web servers: educational (EDU), commercial (COM) and news (NEWS). Our objective in this study is to find out influence of the usage environment on the static and dynamic characteristics of different types of web objects.

¹*Contents in () are optional.*

3.3.1 Summary of Traces

One educational trace is the access logs from the Department of Electrical and Computer Engineering of Iowa State University. Two other educational traces come from Internet Traffic Archive (ITA) [57]. The traces from ITA have been used earlier for web server workload characterization in [58]. Commercial and news traces are extracted from access logs of a proxy server in the National Laboratory for Applied Network Research (NLNR) [59].

EDU class: The three traces used for the EDU class are: *Calgary* (access logs from a Computer Science departmental web server), *Saska* (access logs from a university web server), and *ISU* (access logs from an Electrical and Computer Engineering Departmental web server). Table 3.1 summarizes traces in the EDU class.

COM class: Due to the difficulty of collecting data from commercial web servers directly, the traces to three commercial web servers are extracted from NLNR proxy traces. *MS* (www.microsoft.com), *Compaq* (www.compaq.com) and *Disney* (www.disney.com) were chosen as representatives of commercial web servers. Table 3.2 summarizes traces in the COM class.

Table 3.1 Summary of logs from EDU class.

Item	<i>Calgary</i>	<i>Saska</i>	<i>ISU</i>
Log Duration	353 days	214 days	97 days
Start Date	10/24/94	06/01/95	02/03/98
Successful Req.	696,619	2,385,866	1,189,376
Avg. Req./Day	1,974	11,149	12,262
Distinct Req.	8,384	8,097	8,603
Traffic (MB)	7,758	12,637	7,338
Bytes/Day (MB)	21.98	59.05	75.65
Bytes/Req. (KB)	11.13	5.30	6.17

NEWS class: Traces used for the news class also come from the proxy traces of NLNR. *CNN* (www.cnn.com) and *Usa* (www.usatoday.com) are two popular on-line news web sites. *Weather* (www.weather.com) provides online national weather service. Table 3.3 summarizes traces in the NEWS class.

Table 3.2 Summary of logs from COM class.

Item	<i>MS</i>	<i>Compaq</i>	<i>Disney</i>
Log Duration	68 days	68 days	68 days
Start Date	03/14/98	03/14/98	03/14/98
Success Req.	347,589	103,486	91,397
Avg. Req./Day	5,112	1,522	1,344
Distinct Req.	30,716	22,726	17,514
Traffic (MB)	2,096	665	960
Bytes/Day (MB)	30.82	9.78	14.12
Bytes/Req. (KB)	6.03	6.43	10.51

Table 3.3 Summary of logs from NEWS class.

Item	<i>CNN</i>	<i>Usa</i>	<i>Weather</i>
Log Duration	68 days	68 days	68 days
Start Date	03/14/98	03/14/98	03/14/98
Success Req.	127,203	147,104	73,267
Avg. Req./Day	1,871	2,163	1,077
Distinct Req.	9,906	7,841	2,594
Traffic (MB)	947	834	571
Bytes/Day (MB)	13.93	12.26	8.40
Bytes/Req. (KB)	7.45	5.67	7.80

3.3.2 Document Types

The traces are categorized into 10 different types according to the nature of web documents. The type of a document is distinguished by its URL or the request method to it. For example, the suffix of a file name often tells the type of a file, and a **POST** request always indicates a *Form input*. The 10 types used in our study are listed in Table 3.4.

Tables 3.5, 3.6, and 3.7 list the proportion of accesses to different document types in three server classes. **Acc%** is the access percentage of each type of files, and **Doc%** is the percentage of unique URLs of each type of documents in a web server. **AccFrq** is the relative access frequency of a web document with respect to the average access frequency of HTM files. The reason for using relative measurement of **AccFrq** is to improve the comparability between a busy and a less busy web server, and get a fair

Table 3.4 Types of Web documents.

Index	Type	Explanation
0	TXT	Plain text
1	HTM	HTML files
2	ARC	Formatted txt files or archives
3	AUD	Audio files
4	GIF	GIF files ²
5	JPG	JPEG and other image files
6	FRM	Form input and responses
7	VDO	Video files
8	EXE	Executables and script files
9	OTH	Others

average result. **AvgTrf** is the average transfer size in kilobytes of each object type, and **AvgSz** is the average document size in kilobytes of each object type.

HTM, GIF, and JPG are the top three popular types. The document (a unique URL) percentage and the access percentage for the top three file types, however, are not the same in each web server class. Compared to the other two classes, there are more unique HTM documents in the EDU class, more unique GIF documents in the COM class, and the JPG documents in the NEWS class are much more than in other classes. The average HTM document size in the NEWS class is about twice of that in the COM class, which in turn is about twice the size of that in the EDU class.

Table 3.5 Access pattern of different types in EDU class.

Type	TXT	HTM	ARC	AUD	GIF	JPG	FRM	VDO	EXE	OTH
Acc %	0.3	41.5	0.6	0.0	45.6	5.4	5.7	0.0	0.1	0.7
Doc %	1.3	49.0	3.4	0.1	34.3	4.0	0.8	0.4	0.3	6.3
AccFrq	0.25	1.00	0.16	0.43	1.79	1.35	50.0	0.11	0.59	0.13
AvgTrf	43.67	5.66	246.90	72.82	6.28	20.86	2.23	621.82	5.97	58.76
AvgSz	25.37	4.63	278.60	170.56	14.51	43.09	4.72	751.39	46.66	58.76

These differences suggest that, even web documents of the same types have various average sizes in different server environments. In the following experiment, we also observe that the HTM files in the NEWS class are much more mutable than the other two classes. Here the document's size and lifetime characteristics indicate the same

Table 3.6 Access pattern of different types in COM class.

Type	TXT	HTM	ARC	AUD	GIF	JPG	FRM	VDO	EXE	OTH
Acc %	0.0	21.1	0.1	0.3	68.0	6.2	0.0	0.1	2.6	1.6
Doc %	0.1	36.6	0.3	0.5	50.6	8.8	0.1	0.2	2.2	1.2
AccFrq	0.66	1.00	0.43	0.65	2.44	1.22	0.22	0.56	1.79	2.05
AvgTrf	54.97	13.00	388.24	111.10	3.16	10.87	1.99	964.62	77.56	33.11
AvgSz	46.90	10.47	346.41	186.57	5.61	14.81	2.13	1320.05	239.58	76.49

Table 3.7 Access pattern of different file types in NEWS class.

Type	TXT	HTM	ARC	AUD	GIF	JPG	FRM	VDO	EXE	OTH
Acc %	1.6	16.5	0.0	0.0	65.0	13.5	0.1	0.1	1.9	1.4
Doc %	0.0	40.2	0.0	0.1	35.7	21.1	0.1	0.4	0.8	1.4
AccFrq	50.0	1.00	2.08	0.61	5.0	2.38	0.88	0.36	5.0	1.82
AvgTrf	1.05	19.04	38.52	265.11	4.59	10.16	8.22	904.66	31.25	13.72
AvgSz	1.49	18.10	34.02	235.40	7.93	9.77	7.48	1002.64	103.17	27.44

trend of caching gain and thus a size-based caching technique could be adopted [60]. However, our study on the size distribution in different classes of web sites show that this inference is not true for all the cases. The study reported here provides more insight to the caching and removal policies.

3.3.3 Lifetime Calculation

In the lifetime calculation procedure, an access request is treated as a client polling, and the corresponding access log records the result of the polling. A status code of **304** indicates that a requested document has not been modified since last time the same client requested it. A status code of **200** indicates that either the data copy in the client's local cache is stale or has been swapped out from the cache, or the client requests the document for the first time; and the document content is sent back to the client successfully. Since modification timestamp of a document is unavailable from standard access logs, it is difficult to tell directly whether a cached data copy is stale. File size is the only information about a web document in a log besides the URL. We rely on the assumption that most of the modifications result in a change of file sizes. To validate this assumption, modifications of web documents in a departmental

web server at ISU were monitored over one month period. The status of each web document was polled twice on a daily basis. Timestamps of last modification and file sizes were compared. Statistical results showed that more than 95% of modifications resulted in a change of file size. Using file size as an indicator of modification is thus a reasonable estimation.

If a web document is requested frequently, it is assumed that its size change could be indicated in access logs in a short period. This assumption has high credibility in the EDU class, since traces in the EDU class are access logs from original web servers, where the most frequent polling records are available. For traces in the COM and NEWS classes, high frequencies are ensured by the combination of multiple users' requests and selection of "hot" web servers. The assumption is based on the fact that the ratio of accesses with status *Not-Modified* (304) to accesses with status of *successful retrieval* (200) is generally high, especially in proxy trace files. High percentages of *Not-Modified* responses means that the access frequency and the default expiration time of web documents are much shorter than the real lifetime of them. In other words, any change of a web document could be found in a short period of time by frequent client polling. Table 3.8 shows the percentage of 304 responses to successful 200 responses. **Avg%** is the average 304/200 retrievals of web servers in the same class.

The lifetime of a web document is approximated by counting the intervals between two continuous retrievals of the same document name but with a different size. Here the same document refers to the documents having the same URL, and a lifetime of a document is an interval between two successive modifications of the same URL. The j th lifetime of a document i , denoted as LT_{ij} , can be obtained from, where MT_{ij} denotes the j th modification of document i .

$$LT_{ij} = MT_{(i+1)j} - MT_{ij} \quad (3.1)$$

A couple of factors appear to distort the results in our study. Some documents might never change during the period of trace recording. The longest observed lifetime of a document is made equal to the duration of trace files, which might be shorter than the real lifetime of a document. Average lifetime would thus be shortened by the limit of trace duration. However, the observed lifetime distribution is not affected by that limit. Another factor is that the accuracy of the results are influenced by the user reference patterns. The results of frequently accessed documents are more accurate than the less accessed ones. These two distorting factors are compensatory in nature. Furthermore, experimental results of the average lifetime of web documents in the EDU class are consistent with the previous findings in [55], which suggests that the influences of above two factors are negligible.

Table 3.8 Not-modified vs. Get retrieval.

	Server	304 #	200 #	304/200	Avg
E	Calg.	97,560	566,833	17.2 %	23%
D	Saska	151,607	1,032,966	14.7 %	
U	ISU	303,474	846,442	35.9 %	
C	Comp.	37,834	65,002	58.2 %	53%
O	MS	128,236	208,765	61.4 %	
M	Disn.	24,802	62,851	39.5 %	
N	CNN	54,835	67,045	81.8 %	56%
E	Usa	27,106	119,557	22.7 %	
W	Weath.	27,248	44,025	61.9 %	

3.4 Results

In this section, we characterize the nature of different types of web documents with respect to three classes: EDU, COM, and NEWS.

3.4.1 Access Frequency vs. File Size

Figure 3.1 shows the average transfer bytes vs. average size of HTML files in three classes. Very small difference exists between the average transfer bytes (average

of access percentage weighted document size) and the average document size for an HTML file in all the classes. It suggests that there is no obvious tendency of small HTML files being more likely to be accessed. To some extent, sizes of frequently accessed files are even bigger than the average HTML file size.

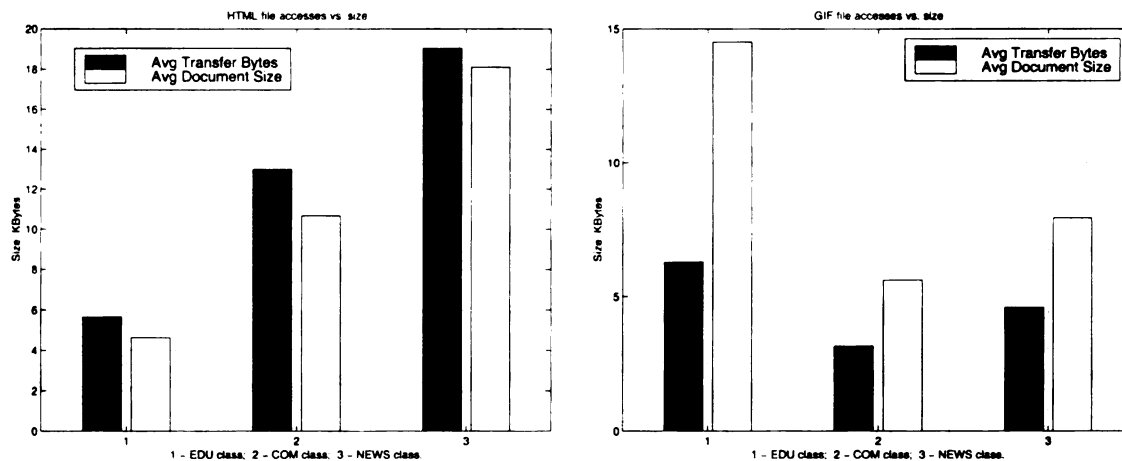


Figure 3.1 Traffic vs. doc. size of HTM. Figure 3.2 Traffic vs. doc. size of GIF.

Unlike HTML documents, frequently accessed GIF documents tend to be small, as observed in Figure 3.2 where the average transfer bytes are smaller than the average file sizes. This tendency is more obvious in the EDU class than in the other two classes. Although the average GIF document sizes are different in the three classes, the average transfer sizes of GIF documents are about the same. Our observations are consistent with [61], but different from [62], in which the author claimed that frequently accessed HTML documents also tend to be small. Considering the overhead of TCP connections, caching priority given to small GIF documents is a cost efficient policy in a limited sized web cache.

3.4.2 Average Lifetime

The average lifetime of the web documents in different classes were compared and summarized in Table 3.9. The results show that the documents in the EDU class are much more stable than the other two classes. The average lifetime of a document is

about 60 days in the EDU class. Web documents in the NEWS class are the most mutable, with only 2~3 days of average lifetime. Lifetime for a typical web document in the COM class is about 8~10 days.

Table 3.9 Average Lifetime (unit days).

File Type	TXT	HTM	ARC	AUD	GIF	JPG	FRM	VDO	EXE	OTH
EDU	56.9	61.5	21.8	13.6	63.4	55.6	N/A	10.3	29.7	45.6
COM	10.2	8.5	3.9	9.6	8.6	9.3	N/A	9.2	7.7	10.0
NEWS	8.5	2.9	2.8	2.2	4.4	2.6	N/A	2.0	7.6	4.8

There is no obvious difference of HTML and GIF average lifetime in the EDU and COM classes respectively. In the NEWS class, the average lifetime of GIF files is about 50% longer than that of HTML files. However, the average access frequency of a GIF file during its lifetime is 2 to 5 times of that of an HTML file. Usually the audio and video clips are expected more stable than other documents. Contrary to our expectations, results show that the lifetime characteristics of audio/video files do not differ significantly from other types of files.

3.4.3 Modification Distribution

The relationship between a document lifetime and its access frequency is examined. Average lifetime, modification times, and accesses during lifetime of *the most popular files* are collected. Web documents are sorted by their access frequency, *the most popular files* are defined as a small set of files that are responsible for 80% of retrieval requests. The distribution of modification times versus document percentage and access percentage are collected and shown in Figures 3.3, 3.4, and 3.5.

In each of these figures, there are four diagrams. The first one shows the document distribution of all file types; the second shows the percentage of documents with different modification frequency for each type of files; the third shows the access distribution of all file types; and the last one shows the percentage of accesses with dif-

ferent modification frequency for each file type. The figures indicate the distribution mode of each web object type.

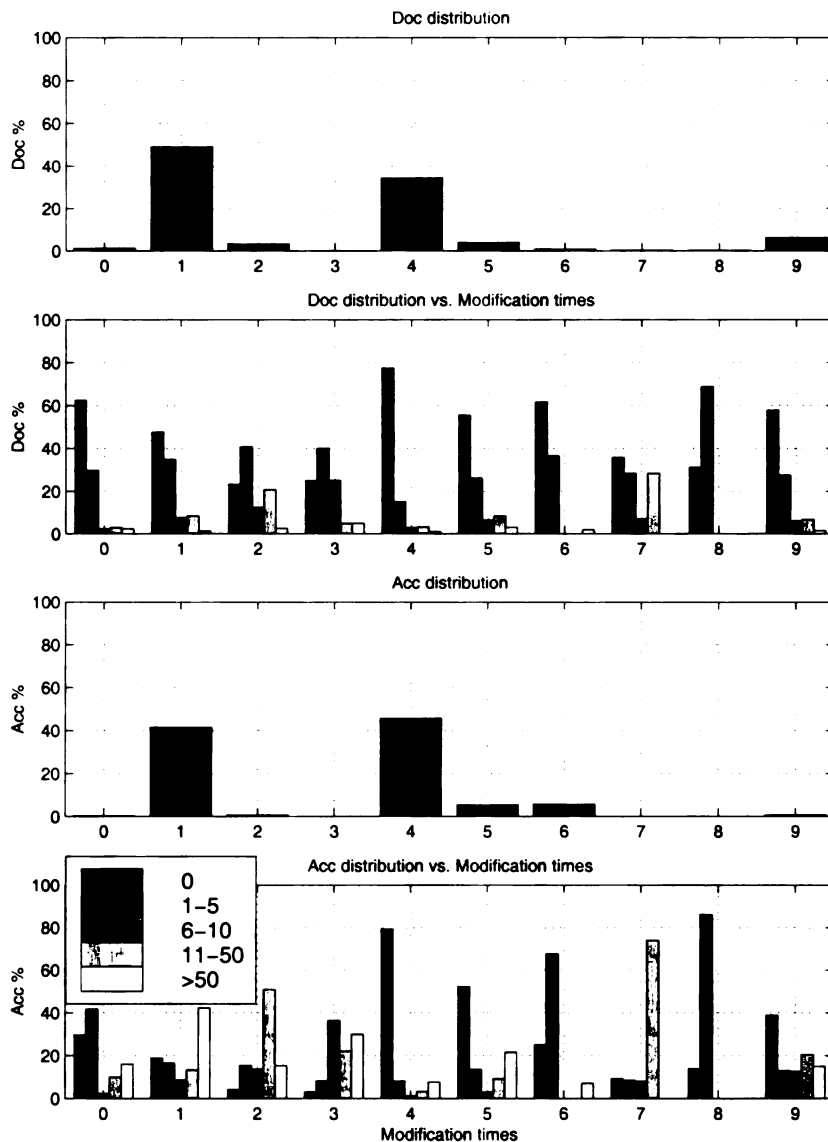


Figure 3.3 Modification distribution in EDU class.

Figure 3.3 suggests that some popular documents, especially those GIF documents, are seldom modified in the EDU class. Similar observation was reported by Bestavros in [63]. The average lifetime is about 2 months for a web document in the EDU class. On the other hand, 1.3% of the HTML files that are frequently updated accounted for 42% the of access requests, which suggested that the users have

tendency to access more frequently modified data.

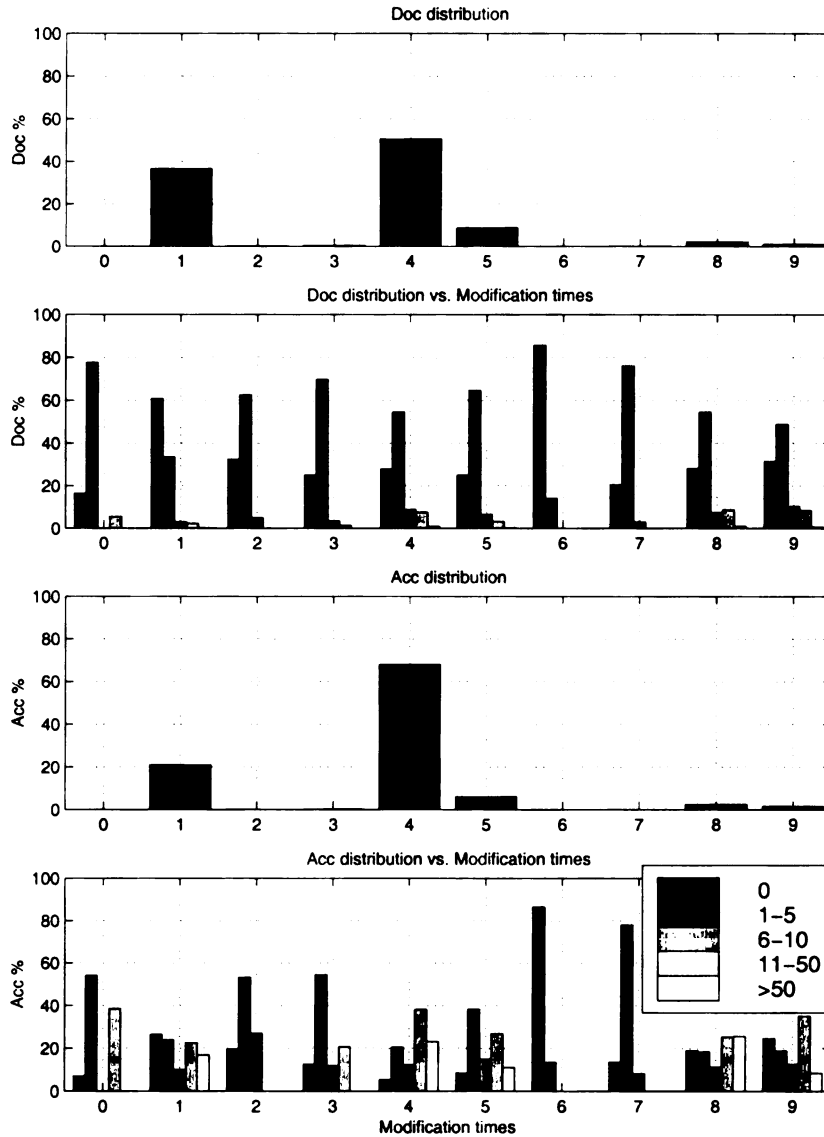


Figure 3.4 Modification distribution in COM class.

Figure 3.4 shows that more than 94% of HTML documents have average lifetime of more than 10 days in the COM class. However, the access percentage to those documents are less than 50%. On the other hand, 0.2% of HTML documents are modified in daily base, and the access requests to those documents are about 17% of the total accesses. This observation suggests that, Bestavros' conclusion [63] about the popularity of a document being inversely proportional to the possibility of its

modification is not valid for the HTML documents in the COM class.

GIF documents in the COM class also show that more popular documents were more mutable. The documents modified 11~50 times are responsible for 38% of access requests, and 1% of documents that are modified more than 50 times are responsible for 25% of total requests.

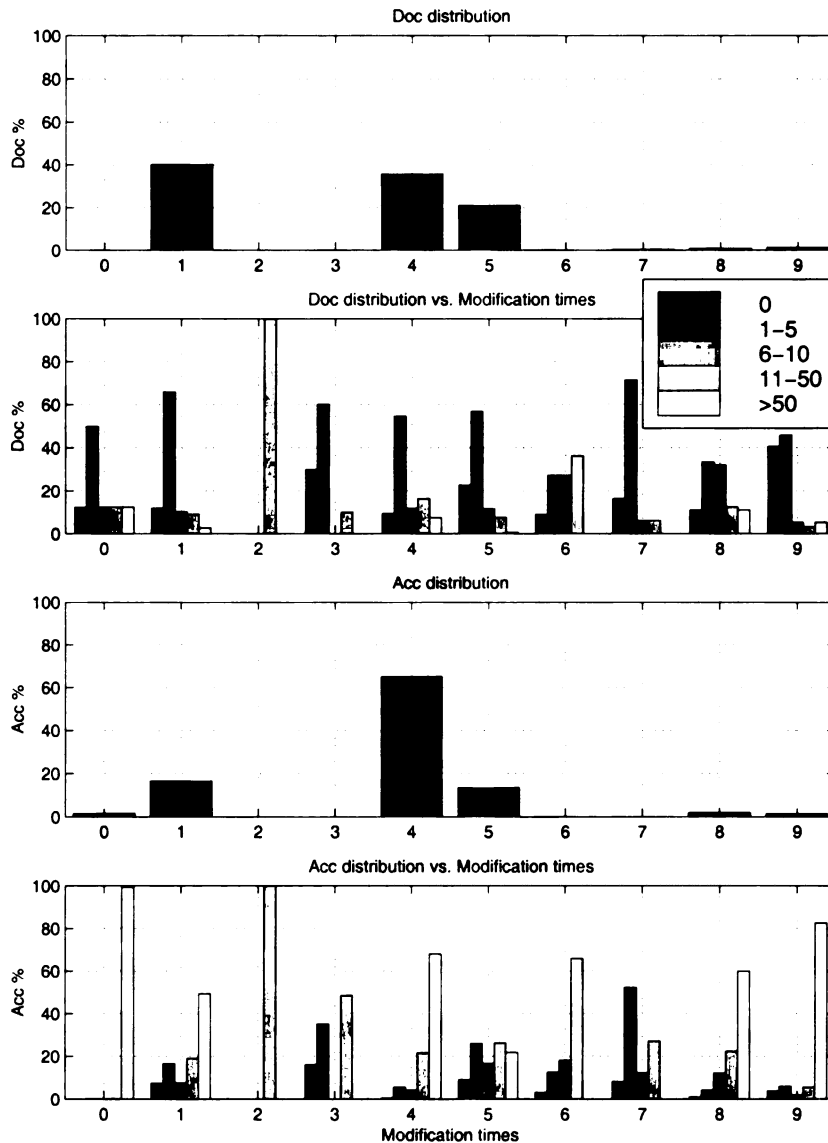


Figure 3.5 Modification Distribution in NEWS class.

Figure 3.5 shows that most references in the NEWS class are to the documents with modifications more than 50 times. Considering the total trace collection period of

68 days, The documents with lifetime of 1 day or less accounts for most of requests. This behavior is expected of the NEWS class. 2.7% of HTML documents in the NEWS class are modified more than 50 times, and they constitute 50% of the access requests. Although there are some rarely updated HTML documents, access times do not seem to increase with the lifetime of those documents. For the GIF documents in the NEWS class, the more popular a document is, the shorter lifetime it has. There is no GIF document which is never modified during the trace collection period.

3.4.4 Total Lifetime Distribution

In this research, documents with the same URL in trace files are considered as the same document, and the total lifetime of a web document i , TL_i , is defined as the duration between the first access, denoted as MT_{1i} , and the last access, denoted as MT_{ni} , to a the same URL. In other words, the total lifetime of a document is equal to the sum of the individual lifetimes of the document, where an individual lifetime is defined as duration between any two successive modifications of a document. The TL_i can be expressed as,

$$TL_i = MT_{in} - MT_{i1} = \sum_{j=1}^n LT_{ij}. \quad (3.2)$$

Figure 3.6 depicts the total lifetime distribution of three major types of documents (HTML, GIF and JPG) in the NEWS class. It can be observed from Figure 3.6 that the majority of web documents exist only a short period among the popular files. A specific kind of documents seems to be very popular in the NEWS class. The characteristic of this kind of documents is a long total lifetime with frequent modifications. Although the percentage of this kind of documents is not very high, they are responsible for high percentage of access requests. For example, during the trace collection period of 68 days, the front page of CNN — <http://www.cnn.com/> — was accessed 5,114 times, and was modified 4,867 times. To most of the individual

users, this is also a kind of “uncacheable” document due to its frequent modification behavior.

An interesting phenomenon presented in Figure 3.6 is that the total lifetime of documents and access requests are almost uniformly distributed except for shortly existed documents and frequently updated documents.

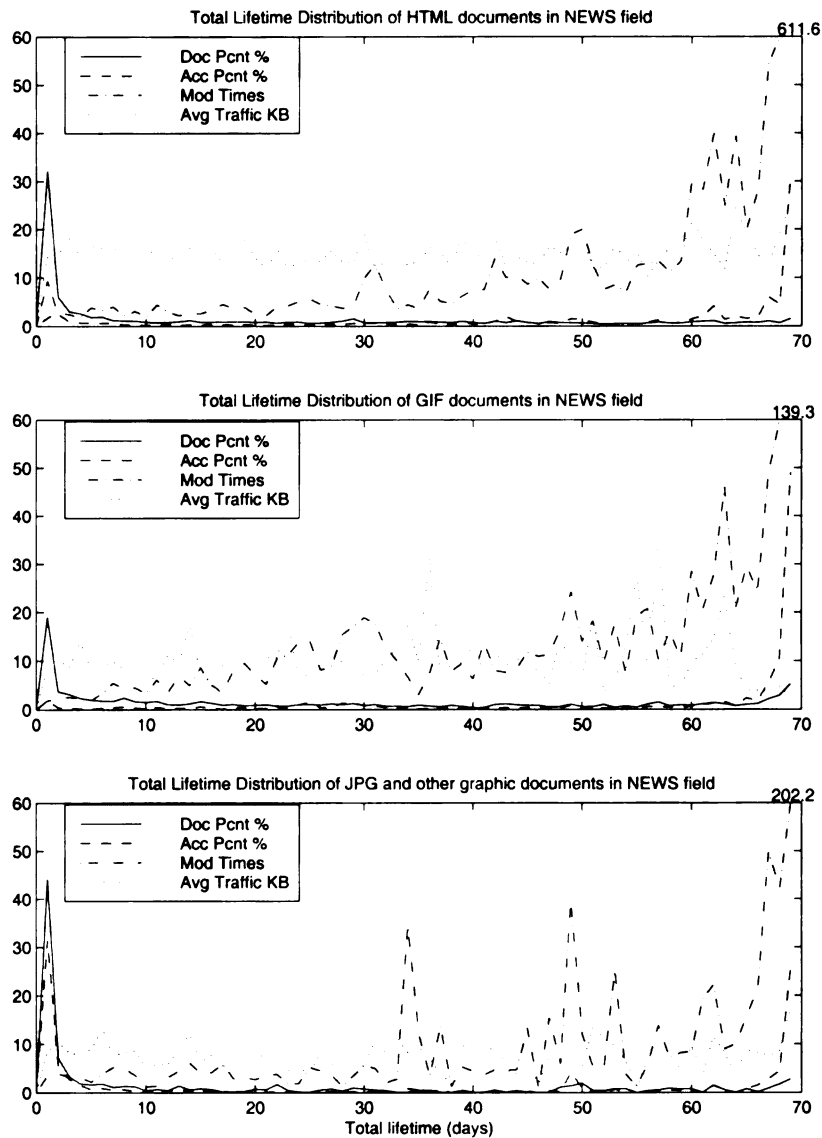


Figure 3.6 Total lifetime distribution of documents in the NEWS class.

However, modification times of a document tend to increase with its total lifetime TL_i . Considering the average lifetime of a document being equal to its total lifetime

divided by the modification times, we can conclude that the average lifetime and the access frequency of a document do not increase linearly with respect to its total lifetime. If the cache hit frequency is treated as the value of a cached document, the caching value of a document reduces with its age in the cache.

These analysis suggest that the traditional caching policy may not be suitable for retrieving web documents in the NEWS web sites. The total lifetime distribution of documents in the COM class is also analyzed (the results are not shown for space limitations). There are also shortly existed documents in the COM web sites, with a total lifetime close to 2 days, a day longer than that of the NEWS class. The document percentage and access percentage of the short-lived documents are significantly lower than the NEWS class. The same situation occurs in the frequently updated documents. The modification frequency and access percentage are also much lower than that of the NEWS class.

3.5 Design Issues

3.5.1 Document Classification

Based on the access, lifetime and modification characteristics, web documents can be classified into four categories: *highly mutable documents*, *stable documents*, *short life documents*, and *others*. The existence of the highly mutable documents and stable documents was also reported in [55].

A *highly mutable document* is defined as a document with frequent modifications, e.g., the front page of some web sites. They are usually accessed more frequently than other documents, see Table 10. Conventional caching policies such as LRU or LFU might be insensitive in distinguishing highly mutable documents, and lead to high percentage of stale data in caches. Instead, prefetching or server-pushing based data dissemination might be a suitable option for retrieving documents from web sites with high percentage of highly mutable documents.

Table 3.10 Highly mutable document ratio.

Environment	Doc %	Acc %
EDU	10	20
COM	3	25
NEWS	33	80

A *stable document* is in steady state with relatively long lifetime. Some of them are accessed frequently, while most of stable documents have relatively low access rate. In the EDU class, 44% of the HTML and 78% of the image documents never change during the whole trace period, 20% of the HTML file accesses and 80% of the image file accesses are to those stable documents. Nearly 30% of HTML documents in the COM class were never modified during the trace period and were responsible for 18% accesses.

Short life documents are accessed or existed in traces for only 1 or 2 days. Nearly 1/3 of the documents in the NEWS class are Short life documents. There are about 20% of short life documents in the COM class. The short life documents should be distinguished and removed from the local cache to save space for potentially useful data.

3.5.2 A Two-State TTL Algorithm

Keeping highly mutable and short life objects in a cache sometimes does not help to increase the cache hit rate, rather it could cause potentially useful (to be retrieved in the near future) data copies being deleted from the cache and thereby increase the number of requests to the original server. Most web caching policies fail to efficiently deal with all classes of web documents. For example, LFU is not suitable for distinguishing short life documents [64], and LRU is not effective for highly mutable documents.

A good cache consistency mechanism helps to prevent a cache from returning stale data as well as improve the cache efficiency. For instance, highly mutable and

short life documents are granted a short time-to-live (TTL). When the TTL expires, the document is treated as stale and needs to be retrieved again from the original server. In case of cache replacement, they are treated as expired data and discarded to save space for potentially useful data. This is especially important where clients are personal computers with limited cache space. At the same time, the TTL for stable documents should not be set too short to introduce bulks of “conditional get” requests, i.e., **GET** requests with **If-Modified-Since** fields.

Here, we present a two-state TTL consistency algorithm which is depicted in Figure 3.7. When a new data copy is cached for the first time, it is set in a *transient* state with a short TTL, say less than 1 hour. If the data copy is still valid after the short TTL, it is switched into a *steady* state with a relatively long TTL. This long TTL can be decided based on statistic measurement, or a portion of time elapsed since the last modification time.

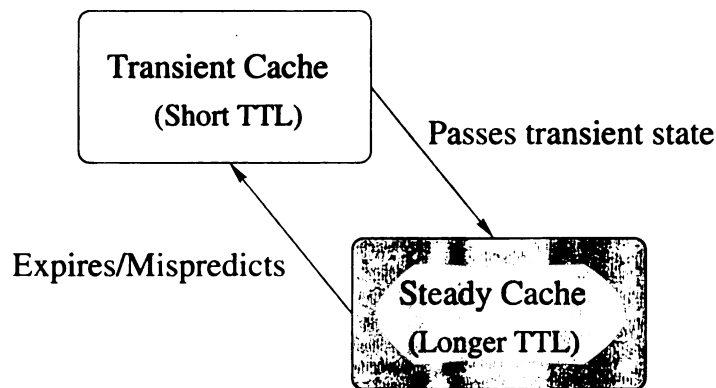


Figure 3.7 The two-state TTL consistency algorithm.

We set up a simulation using a cache simulator from Virginia Tech [60], and adapted it by extending three consistency, i.e., adaptive TTL, fixed TTL, and two-state TTL algorithms into caching policies. In the two-state TTL algorithm, the cache space is divided into two equal areas for transient and steady state documents respectively. New caching copies only replace documents in transient state, older steady state documents are replaced by newer steady state documents in cache re-

placement. The traces from EDU³, COM and NEWS⁴ classes are replayed to roughly check the effectiveness of the two-state TTL algorithm. Cache hit rate, conditional get requests with not-modified response, and the amount of stale data returned are collected and compared.

In the fixed TTL algorithm, the default TTL is set to be 1/4 of average lifetime in that class. In the adaptive TTL algorithm, TTL is set to be 1/2 of elapsed time since last modification. In the two-state TTL algorithm, transient TTL is set to be 15 minutes in NEWS class, 12 hours in COM class, 24 hours in EDU class, and adaptive TTL is used in steady states. The performance comparison of three cache consistency algorithms, fixed TTL, adaptive TTL, and two-state TTL, is shown in Figure 3.8

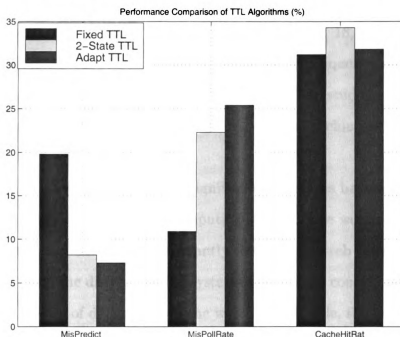


Figure 3.8 Performance Comparison of three cache consistency algorithms.

The results indicate that less than 1% of stale data are returned in the EDU class irrespective of the consistency policy. In the COM and NEWS class, using fixed TTL returns about 19.8% of stale data, and 10.9% of conditional get requests with

³We use the web client traces from Virginia Tech representing the EDU class.

⁴We use the NLNR proxy traces representing the COM and the NEWS classes.

not-modified response. Adaptive TTL performs the best in keeping cache consistency, returns about 7.3% of stale data, while 25.4% of unnecessary conditional get requests. Two-state TTL decreases 3.1% of unnecessary get requests from adaptive TTL algorithm at the cost of returning 0.9% more stale data. Furthermore, two-state TTL increases cache hit rate by average 2.8% compared to the other two TTL consistency algorithms, in a cache size of 5 MBytes. It proves that the two-state TTL algorithm do help to improve caching efficiency. Detailed performance evaluation can be done by varying the configuration and workload.

3.5.3 Caching and Prefetching

Due to the explosive growth of the Internet, lots of efforts have been directed towards implementation of client/proxy caching [15, 16, 17, 18, 19], sever caching [22, 23, 24], prefetching [20, 21], and replication [26, 27] techniques to reduce network latency and the server load. However, there has been limited studies reported on the integration of workload characterization and the design of caching techniques in the web environment.

In the context of caching, there exists significant differences between an web environment and a traditional distributed computing system. The workload characteristics in the two environments differ significantly. Most of the web caching algorithms are transplanted from the distributed file systems that do not consider the static and dynamic characteristics of documents on the web. For example, the number of documents on the web are much higher than traditional distributed file systems. A large quantity of short life documents exist on the web and accesses to the web documents tend to be bursty [65]. Caching algorithms should be developed targeting the web documents characteristics and for specific classes of environments.

Our study has shown that significant differences exist in both static and dynamic characteristics of web documents in different web environments. Caching algorithms should be adjusted with the characteristics to gain better performance.

The average lifetime of documents in the educational web sites are significantly higher than the commercial and news web sites. A high percentage of stable image files with high access rate exists on the educational web sites. They are the best candidates of the conventional caching algorithms. Special priority should be given to graphic documents, since more graphic documents are seldom changed compared to the other types of documents, and users' tendency of retrieving small graphic documents improves cache efficiency. In [60], the authors have presented a good evaluation of caching algorithms suited for documents in the educational web sites.

For documents in the commercial web sites, About 15% of the GIF documents and nearly 30% of the HTML files have short lifetime with high access frequencies. LRU combined with two-state expiration time filters out these documents effectively. In a commercial web site, there are also some frequently updated documents with very high access frequency. The assumption that the popular files are modified less frequently is not valid here. Prefetching with server invalidation should be adopted to deal with these highly mutable documents. Two possible types of prefetching schemes reported in [21] could be examined in these situations. The access frequency of documents on the commercial web sites tend to decrease with the total lifetime for most stable documents. In such cases, LRU might outperform LFU replacement schemes, which can explain why some trace driven simulation results are not consistent in web caching studies [64, 60, 19].

Most documents in the news web sites are highly mutable and many are accessed only in a short period. Except for large proxy servers, news documents are "not worthing caching" for personal web users. Instead, server initiated push caching as in [27] might be used to gain better performance. For the less mutable documents in the news web sites, their TTL should be set significantly shorter than which from the other classes of web sites.

3.6 Summary

Different types of web objects from different sites have different static and dynamic characteristics. In this study, we have reported an extensive study on the size, access, lifetime and modification behavior of web documents from various web sites, and tried to find out some rules in improving caching schemes with the knowledge of web environments dynamics. Some of the major observations derived from our study are itemized as follows.

- The document size, access pattern and modification behavior are dependent on the document type as well as the server environments. The TTL estimation algorithm should be made adopted to the document type and/or the server environment. A two-state TTL weak consistency algorithm based on multi-mode characteristics of web documents introduced in this study might outperform current fixed TTL or adaptive TTL algorithms.
- A generalized technique for caching and prefetching technique may not be effective for a diverse WWW environment. Rather, server-specific or data-specific caching and prefetching techniques should be adopted. Caching consistency policy should be taken into consideration in caching schemes to improve the cache efficiency.
- Several additional interesting behavior of different types of web documents in different classes of servers have been reported in the study. In COM and NEWS classes, there is a strong trend that users access highly mutable documents more frequently than other documents.

The results suggested that the performance of an optimization scheme is influenced by different classes of web servers as well as by different types of web documents. Thus, caching, prefetching, server initiated replication, and other service optimization techniques should be designed such that they exploit these varied characteristics.

CHAPTER 4 SERVICE DIFFERENTIATING INTERNET SERVERS

4.1 Overview

The existing best-effort with *First Come First Serve* (FCFS) scheduling service model of the Internet servers leads to mis-allocation of scarce and expensive network and CPU resources during heavy load periods, thus causes unpredictable response delay, which is not acceptable to time critical transactions. The problem of unpredictability in response time becomes extremely serious during information retrieval from the Web, since more and more applications with time constraints are using the Web as their distributed interfaces. Upgrading hardware to faster CPU, more efficient OS, and broader I/O bandwidth are always solutions for improving the response time and throughput of Internet servers. However, the “bursty” nature and the rapid growth of the Internet traffic volume make it expensive and also inefficient to scale up network bandwidth and server capacity with the increasing peak demand. Previous studies [58, 62, 66] have shown that the peak workload of a web server may exceeds its average load by orders of magnitudes. Thus the server utilization would be very low if we design the system to satisfy requirements of the workload during peak periods.

Another approach of assuring service quality is to pre-allocate resources for value desirable tasks. No matter how high the real workload is, those cherished tasks enjoy dedicated system resources. The analogous policy for the Internet transmission

is the end-to-end resource reservation through the Resource ReSerVation Protocol (RSVP) [46] scheme. One major drawback of the RSVP scheme is poor scalability, since the number of RSVP control messages processed by each router is proportional to the number of flows going through the router. Similarly the resource reservation approach is not appropriate for Internet servers because of poor scalability, low system utilization, and/or long setup delay.

Differentiated services approach, known as *DiffServ* [47] in the IETF community, has been proposed as an efficient and scalable solution to provide better service for the next generation Internet (NGI) communication [9]. *DiffServ* creates an “express-way” for high priority traffic by selective reallocation of network resources during peak workload periods. Similarly, prioritized services can be exploited by a Service Differentiating Internet Server (SDIS) to provide predictable services with statistical guarantees. Especially, the response delay to mission critical requests can be bounded by allocating CPU and I/O resources with respect to their priorities. Our research objective is to design Internet servers that can provide fast response to high priority tasks, minimize the performance penalty of low priority tasks caused by service differentiation without degrading the overall system throughput. In this study, we present a model of SDIS and evaluate its performance through analysis and simulation. During congestion a SDIS provides much better services to high-priority requests with minimal impact on the low priority requests. We have also analyzed several features of SDIS, such as admission control, task scheduling, degree of priorities, and the overheads. Impact of some of these issues on SDIS performance are quantified in our study. The results demonstrate the feasibility and performance advantages of SDIS.

The remainder of this chapter is organized as follows. Section 4.2 describes policies for providing differentiated services from an Internet Server. The server model, admission control and scheduling issues are discussed in Section 4.3. The simulation environment and results are discussed in Sections 4.4 and 4.5, respectively. Section

4.6 analyses the results, Section 4.7 describes the related works, followed by the concluding remarks in Section 4.8.

4.2 Service Differentiation

A traditional Internet server processes requests in FCFS manner. During a high load period, each task has to wait in a queue for a long time before getting services. Overhead from tasks competing for limited resources, such as open connections and network bandwidth, is increased. “Retry” from impatient clients worsen the load situation and cause the “snowball” effect. More elaborate resource allocation schemes rather than the best-effort model need to be adopted to provide predictable services in high load periods. In this section policies of differentiating services that could be deployed by an Internet server, especially a web server, are discussed.

4.2.1 Prioritized Services

In the *DiffServ* model for the Internet, different priorities are assigned to packets, high priority packets enjoy timely processing and assurance of services. Similarly at the server, an incoming request can be assigned to a priority group on the basis of the *client*, *network*, *content*, or *owner* of the requested object. It can also inherit a performance level defined by low level network protocols such as the priority assigned by the *DiffServ* model.

Client based service prioritization scheme provides means for a client to purchase premium services at a certain cost. A client can be authenticated by combination of cookies, client host ID, session ID provided by the server, or by other client profile information in a server. More detailed user authentication gives the users more flexibility for QoS selection, whilst increases the load of the status information maintenance at the server.

Network based service differentiation is determined by the underlying network level protocols. Clients specify priority of packets which carry request information. The server extracts the packets priorities and sends reply at the matching service levels.

Content based service differentiation is determined by the “value” or “importance” of the web object being requested. In an on-line stock exchange center, priority can be granted to purchase/sell requests over queries. Similarly, in an on-line shopping web site, the transactions of someone with items in the shopping cart or someone entering payment information should have higher priority than the general browsing transactions.

Owner based prioritization of service provides selective QoS in a web hosting environment. A web host can grant priorities to requests made to web pages belonging to owner organizations based on the prices they paid.

The latter two service differentiations are based on the URL or request methods, GET or POST, of a web object, which is defined in the HTTP protocol set. Several open issues associated with prioritized service and their studies and results are reported in the following sections. How many levels of priorities can be set up to provide both flexibility and efficiency of the system is an important metric but is considered beyond the scope of this work. Similarly, quantification of QoS and pricing issues for different level of services also need further investigation and analysis, and are not emphasised in this study.

4.2.2 Customized Services

The resources consumed by an Internet server include CPU processing power, network bandwidth, disk I/O, etc. Different types of tasks have various resource requirements. Serving a dynamic web page is much more CPU intensive than retrieving a static web page. Consequently, it is not uncommon that the response times of such

dynamic requests are orders of magnitude higher than the response times of static requests. In case the CPU load becomes high, low priority dynamic requests should be restricted to prevent CPU from overload and preserve system throughput. Some tasks are I/O intensive and consume large volume of disk and network bandwidth for a sustained duration. Most of CM tasks fall into this category. Low priority CM requests are restricted to prevent too many open connections blocking I/O channels. High priority CM tasks should be carefully scheduled to meet the bounded delay requirements. However, most contemporary web servers drop requests indiscriminately in case of overload or I/O congestion, which causes further burst of request flows and service degradation of high priority requests.

If the network supports *IntServ* [7] architecture, an Internet server can be enhanced to honor QoS information carried by each data flow. The data flow response delay and dropping rate requirements of an admitted request should be met, and the rest of system resources can be allocated fairly across “best-effort” or “better-than-best-effort” traffic flows.

In order to provide customized service, available resources in each system component should be monitored in time. Overload avoidance and detection algorithms should be explored to maintain system response time and throughput. Informed dropping mechanism should be adopted to prevent client from retrying during congestion. Admission control and process scheduling issues that help in service differentiation and overload control are discussed in the following sections.

4.3 A Generalized Internet Server

In this section, we have presented a model of a generalized Internet server and have analyzed the feasibility of service differentiation. Although other models of Internet servers exist, the goal of the proposed model is to study the issues involved in service differentiation, not the types of Internet servers.

4.3.1 Service Differentiating Internet Server Model

Figure 4.1 shows a queuing model of a generalized QoS aware Internet server. The server architecture is extracted from Apache [49] web server source code, which is the most popular server type in current Internet. QoS aware components are derived from a web server design design in [43]. Another kind of distributed web server model can be found in [37]. No task dispatcher is needed in the latter model, load sharing and balancing is achieved through distributed negotiation and request forwarding. Apparently, the latter architecture can be regarded as a group of single server providing prioritized service.

The system consists of four major logical components, an task initiator T_I , a task dispatcher T_D , a task server pool $S_i(i = 1...N)$, and a communication channel N_S . A represents the requests sent from clients. dA represents the dropping probability of incoming requests. A' are requests that got served. The initiator T_I maintains open connections of a system. Incoming requests are queued awaiting acceptance by the T_I . Each accepted request is assigned a task, and each task is granted an appropriate priority level based on system settings. The task dispatcher T_D assigns tasks to a task server, and each task server schedules and processes tasks according to their priorities. Responses are sent back to clients through the communication channel N_S . To simplify the model, we assume that the server connects to the client through a high speed network. We have ignored blocking and flow control from client side network connection as they are beyond the scope of the proposed study.

The task initiator T_I picks up requests from the incoming request queue. The queue length is limited by the operating system constraints and allowable open connections in the server. Incoming requests are tail-dropped if the queue is full. With the cooperation of intelligent network interface, network layer priority setting of a request can be passed on to the application layer. T_I can be used to collect priority information from low level network protocols and convey it to the task dispatcher

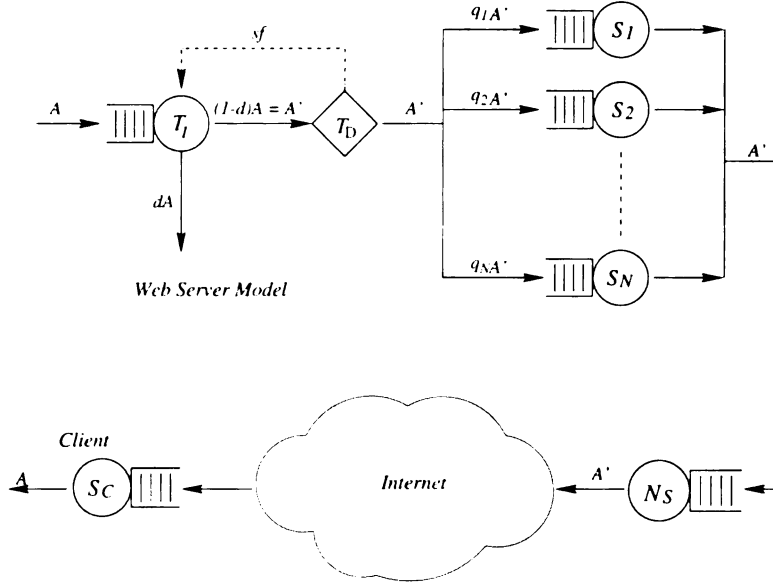


Figure 4.1 Queuing Network Model for a Web Server.

T_D . Otherwise, T_D determines the task priority based on the implementation of the service differentiation algorithm.

After determining the priority of each task, T_D selects a task server to process the task. Priority is assigned based on the criteria discussed in Section 4.2. The task dispatcher identifies the clients and checks profile of the requested web object. At the same time, it monitors the popularity of web objects and the load of system components. Statistic information of the popularity of web objects is used for server caching and prefetching optimization, and the load information is used for the task assignment procedure.

Each task server, $S_i (i = 1 \dots N)$, represents a processing unit that takes care of processing a task. For a static request, the task server translates the URL into the corresponding file path, finds the file, copies the file contents into memory, and sends back reply to the client through the Internet. For a dynamic request, the server invokes a gateway interface. A web page is generated “on the fly” based on the result returned from the gateway interface and sent back to the client. A task server is an abstract concept in the sense that it can be a child process in a multi-process

server, a thread in a multi-threaded server, a processor in a multi-processor server, or a host in a server cluster. Based on the system environment, each task server share or has an independent waiting queue, memory, cache space, and other resources. Task servers can be symmetric, i.e., with the same hardware configuration, or can be asymmetric with different hardware configurations. The task dispatcher provides a bridge to seamlessly integrate new hardware equipment into the server system. In an asymmetric task server environment, response time should be predicted when a task is assigned to a specific server.

The capacity of the communication channel N_S is determined by the bandwidth of the server network access point. In the current Internet infrastructure, a server generally has T1 (192 KB/s) or T3 (5.4 MB/s) connection. An Intranet web server generally connects to clients through high-speed local area networks such as 10 Mb or 100 Mb Ethernet. Data throughput is determined by the channel bandwidth and network utilization.

4.3.2 Server Processing

To process each web task, resources are consumed in I/O interrupt handling, TCP checksum computing, URL parsing, user authentication, file location, reply transmission, and logging. We design a server queuing model based on the estimated cost for processing each request using the following formula:

$$T_s = T_{init} + T_{cpu} + T_{data} + T_{net} \quad (4.1)$$

T_s is the sum of service time of each system component. T_{init} is the time to bring a request from network interface to the application server. T_{cpu} is the CPU time to handle an incoming request, plus the computational cost of dynamic tasks. T_{net} is the cost to send processing results back to the client. T_{data} is the cost to load the data from disks into server cache. If the requested data is already in the server cache,

T_{data} is zero. If the requested data needs to be loaded from the disks, the value of T_{data} is approximated by a disk model with fixed seek time and disk bandwidth.

The system model can be constructed as a queuing network. The waiting time of a task is determined by the time it spends on every network component, and the service time it gets is the sum of the service time received from each component. Most components are sensitive to system overload [67, 68]. The system becomes unstable and response time increases drastically when the request access rate and resource consumption exceed certain threshold. QoS can be assured by maintaining the system load below the threshold, which can be achieved by admission control, scheduling, and efficient task assignment schemes (if multiple servers are available).

4.3.3 Admission Control

Admission control can be implemented using the following two steps: The first step of admission control is provided by the initiator T_I . Incoming requests are rejected in case of access rates exceeding the processing capacity of the system. In Figure 4.1, sf provides feedback of system capacity to the initiator. To avoid bulk rejection, early overload detection can be adopted and two levels of thresholds can be used similar to the Random Early Detection (RED) scheme [69]. Low priority packets are dropped with increasing probability after the system load exceeds the first threshold. All the packets are dropped when the system load exceeds the second level of threshold. Average queue length at the initiator can be used as the threshold indicator. The value of the thresholds can be manually configured, or dynamically adjusted depending on the application environment or on the variety of incoming requests.

The second step of admission control can be provided by the task dispatcher T_D . The task dispatcher monitors the task server load and makes decision for dropping requests to assure that the system works in acceptable load condition. Contrary to

the first step of uninformed dropping, the task dispatcher sends back brief explanation to clients stating why it cannot fulfill the request. The informed dropping decreases the “retry” attempts from clients. When one task server is overloaded, tasks are redirected to the other task servers. If all the task servers are overloaded, low priority tasks experience “informed dropping” to relieve the overload situation. In case of communication channel overload, low priority large size tasks are first discarded, continuous media data requests are discarded when bounded delay QoS can not be assured. Available bandwidth is used as a metric of channel load.

4.3.4 Process Scheduling

In a task server, priority based scheduling is adapted to provide faster processing of some tasks than others. In this study, we have used strict priority scheduling policy, i.e., low priority tasks do not get service if higher priority tasks are waiting, whilst some of the other scheduling schemes such as weighted fair scheduling can also be considered. Tasks in the same priority group are serviced in FCFS order. Preemptive scheduling allows high priority tasks to preempt low priority tasks, and thus eliminates the need for tasks to wait for completion of service of lower priority tasks. It does introduce the complexity of context switching and status preservation of preempted tasks. We consider a non-preemptive scheduling approach in our study to avoid the complexity caused by preemption.

Scheduling in the communication channel also follows priority based queuing. Unlike the task server, the data that is being sent out is restricted by the page size of cache memory and network protocols. The scheduling in the communication channel is similar to prioritized process sharing.

4.3.5 Task Assignment

A distributed server system introduces flexibility of selecting a server for a specific task. Selection can be based on task priority, task type, spatial and temporal locality

of web objects, and hardware configuration.

One or more high performance web servers can be reserved for high priority tasks to make sure that the high priority tasks always encounter short queue length. Another approach of task assignment could be the assignment of task type to different servers. Same type of tasks often consume about the same amount of processing time, while tasks with different types may consume processing time with very high variance. Task assignment schemes which lead towards lower variance of processing time help to improve response time and throughput. CM data require streaming delivery for a sustained time period, for which process sharing performs better than FCFS. In addition, task assignment schemes considering temporal locality has the potential of improving caching efficiency.

Task assignment scheme can be combined with scheduling. If each task server has the facility of prioritized processing, then tasks can be sent to the corresponding server right away to alleviate the queuing load at the task dispatcher. Otherwise, the task dispatcher blocks low priority tasks unless there are no high priority tasks waiting to be served at a server.

4.3.6 System Overhead

Prioritized servicing inevitably brings overhead in two aspects: the extra CPU cycles for arbitrating the appropriate priority for each task and kernel overhead in priority-based scheduling. The overhead brought by priority assignment depends on the priority assignment criteria. If the priority is inherited from the lower network layers, for example, client based priority assignment and *DiffServ* enabled packet classification, the arbitrating overhead is trivial since we only need to pass a QoS parameter from the network interface. If the priority assignment is based on the URLs of web objects, the priority arbitration can be combined into the URL parsing step of a conventional web server. By combining URL parsing with priority assignment

functionality, the overhead is reduced. However, part of the task processing cannot benefit from prioritized services. Other methods of decreasing overhead include separation of coding space of cookies, directory caching, and separation of naming space based on the ownership of objects.

4.4 Analysis of Waiting Time

We first present an analysis of the comparison of waiting time estimates between prioritized and non-prioritized system. The analysis is based on simplistic queuing model assumption and is meant for studying the impact of various workload parameters on the waiting time. A detailed performance evaluation is done in the next section.

4.4.1 Waiting Time in Non-prioritized Systems

First we consider a general non-preemptive queueing system with FCFS discipline assuming infinite queue space. A task waiting time w can be decomposed into two parts: the residual life of the task being serviced upon its arrival w_1 , and the delay it experiences due to tasks enqueued upon its arrival w_2 . The waiting time, w , can be expressed as:

$$w = w_1 + w_2. \quad (4.2)$$

Let n_q be the number of queued tasks when a new task arrives, N_q be the expected number of queue length of the system, $x(i)$ be the service time of the i th task in the queue, and X be the expected system capacity expressed in time units. The waiting time of queued tasks, w_2 , and the expected value of w_2 , denoted as W_2 , equal to:

$$w_2 = \sum_{i=1}^{n_q} x(i) \quad (4.3)$$

$$W_2 = E\{n_q\} * E\{x\} = N_q * X. \quad (4.4)$$

Let λ be the mean task arrival rate, and W_1 be the expected residual life of the task being served when a new task arrives. According to Little's Law, the expected waiting time of a task, W , can be represented as:

$$W = \frac{N_q}{\lambda} = W_1 + N_q * X = \frac{W_1}{1 - X * \lambda}. \quad (4.5)$$

Equation 4.5 indicates that the expected waiting time of a task can be expressed as a function of inter-arrival rate, service time and the residual life of a task. When the task arrival is an independent and identically distributed random process, the Pollaczek-Khinchin (P-K) mean value formula can be used to find the average waiting time in the queue as:

$$W = \frac{\lambda * (X^2 + E\{x^2\})}{2(1 - X * \lambda)}. \quad (4.6)$$

However, given the existence of self-similarity (or long-term dependence) characteristics in the Internet traffic, a more general arrival distribution estimator should be used. A precise estimation of waiting time in a G/G/1 queueing system is hard to obtain, while we can get an approximation of waiting time in a heavy traffic situation [70]. The value can also be served as an upper bound of mean waiting time in a general queueing system, which is expressed as:

$$W \approx \frac{\lambda * (\sigma_a + \sigma_x)}{2(1 - X * \lambda)}, \quad (4.7)$$

where σ_a and σ_x are the variance of the inter-arrival rate and service time, respectively. It can be observed from Equation 4.7 that the mean waiting time increases nearly linearly with the variance of inter-arrival and service time during heavy load period.

4.4.2 Waiting Time of Prioritized Systems

Considering a non-preemptive queueing system with prioritized services, where tasks are queued and processed according to priority groups. The waiting time of a task with priority p ($p = 1, \dots, P$) is decomposed into three parts: $w_{1,p}$ is the residual life of the executing task when a new job arrives; $w_{2,p}$ is the service time of queued tasks with equal or higher priority when the task enters the system; and $w_{3,p}$ the delay due to higher priority tasks arriving during its waiting period. For the highest priority tasks, the last part of waiting time needs not to be considered. Let λ_p ($p = 1, \dots, P$) be the task inter-arrival rate of the p th priority group (P is the highest priority), and $\lambda = \sum_{p=1}^P \lambda_p$. The expected waiting time of the tasks in the p th priority group can be expressed as:

$$\begin{aligned}
 E\{w_p\} &= E\{w_{1,p}\} + E\{w_{2,p}\} + E\{w_{3,p}\} \\
 &= W_1 + \sum_{i=p}^P \sum_{j=1}^{n_i} x_{i,j} + \sum_{i=p+1}^P E\{w_p\} \lambda_i E\{x_i\} \\
 &= W_1 + \sum_{i=p}^P E\{w_i\} \lambda_i E\{x_i\} + E\{w_p\} \sum_{i=p+1}^P \lambda_i E\{x_i\}. \tag{4.8}
 \end{aligned}$$

$$N_q = W * \lambda = \sum_{p=1}^P \lambda_p E\{w_p\}. \tag{4.9}$$

W_1 is the mean residual life of an executing task, which is unrelated to the priority distribution due to non-preemptive discipline of the system. Assume the mean service time of each priority group is X_p , ($p = 1, \dots, P$). Then, combining Equations 4.5, 4.8, and 4.9, we get the expression for W_p as a function of X_i , λ_i , ($i = 1, \dots, P$) and W as:

$$W_p = \frac{W_1}{(1 - \sum_{i=p}^P \lambda_i X_i)(1 - \sum_{i=p+1}^P \lambda_i X_i)} \tag{4.10}$$

$$= \frac{W(1 - \sum_{i=1}^P \lambda_i X_i)}{(1 - \sum_{i=p}^P \lambda_i X_i)(1 - \sum_{i=p+1}^P \lambda_i X_i)}. \tag{4.11}$$

Equation 4.10 illustrates the waiting time relationship between different priority groups. The overhead with several priorities may be high and it may not be worthwhile to support a large number of priorities from a cost/performance standpoint. A few classes of priorities may be enough to provide adequate service differentiation. For the sake of simplicity, let us consider a system with two priorities assigned to tasks, high and low priority tasks. Let W_h be the expected waiting time of high priority tasks, and W_l the expected waiting time of low priority tasks. ρ, ρ_h, ρ_l are the utilization factors of the system, high priority group, and low priority group, respectively. The waiting times can be derived as:

$$W_h = \frac{W_1}{1 - \rho_h} = W \frac{(1 - \rho)}{1 - \rho_h}, \text{ and} \quad (4.12)$$

$$W_l = \frac{W_1}{(1 - \rho_h)(1 - \rho)} = W \frac{1}{1 - \rho_h}. \quad (4.13)$$

From the discussions in the previous subsection, we know that the average waiting time of a task is determined by the distribution of inter-arrival rate and the service time. These distributions are rather stable for an existing system. The average residual life of a task, W_1 , in the system has an upper bound of X , which occurs when the system utilization approaches 1. Given the average waiting time of a system, which can be easily derived from the statistics of response time of tasks, we can estimate the waiting time of different priority groups versus their utilization factors. For example, assume the service time distribution of different priority groups are the same, i.e., $X = X_l = X_h$. P_h and P_l are the probability of a task in the high priority group and the low priority group. Equations 4.12 and 4.13 can be rewritten as:

$$W_h = \frac{W_1}{1 - P_h * \rho} \leq \frac{X}{1 - P_h * X * \lambda} \quad (4.14)$$

$$W_l = \frac{W_1}{(1 - P_h * X * \lambda)(1 - X * \lambda)} = W \frac{1}{1 - P_h * X * \lambda} \quad (4.15)$$

The value of W_h increases inversely with respect to $(1 - P_h * \rho)$. Even the system operates close to its full capacity, the expected waiting time of high priority tasks can still be controlled by adjusting the value of P_h . The value of W_l increases inversely with respect to the product of $(1 - P_h * \rho)$ and $(1 - \rho)$. If the value of $(1 - P_h * \rho)$ remains constant, e.g., $C = 1 - P_h * \rho, 0 < C < 1$, W_l follows the changes of W linearly against various system load situations.

The above analysis provides an average estimation of response time of tasks with different priorities. In reality, the variance of response time can be quite high, and a conservative system design is needed to assure the response time of high priority tasks. A rule of thumb is to keep high priority utilization factors no more than the acceptable system utilization factor. For example, if the delay begins to increase sharply from the server utilization of 0.5, then the system can provide acceptable service to high priority requests with $\lambda_h = 0.5X$. The analysis presented in this section is based on the time independent identical arrival pattern and service time. This analysis is useful for obtaining estimates of waiting time in prioritized and non-prioritized systems. The results could be used to study the impact of the workload parameters such as inter-arrival rate, high priority task ratio, service time, and server utilization on the waiting times. The comparison of the magnitude of impact of these parameters on the high priority and low priority tasks can be also examined.

4.5 Simulation

The request arrival patterns at Internet servers are known to exhibit high self-similarity and long-range dependencies. Thus it is difficult to build a good analytical model that can provide us an in-depth view of the performance estimation. In this section, we have used a simulation model and traces from a real web server as workload to examine the performance of SDIS. We have implemented an event driven simulator for the experimental study of the server model proposed in Section 3. The simulator

model is built using the CSIM [71] simulation package. Response time and slowdown performance under different workload along with the scheduling and task assignment schemes have been examined and compared. Other aspects, including admission control and server side caching algorithms, have been studied as well.

4.5.1 Workload Generation

Previous works [66, 61, 72] have suggested apparent self-similarity and long-term dependency of the WWW traffic pattern. However, accurate synthesis of self-similar traffic remains an open problem [68, 73]. In this study, we propose to generate workload from real trace files. We monitored the logs from a departmental web server in the Computer Science department at Michigan State University. The trace files contain 866,587 requests in one-week period. The access logs provide the request timestamp, client ID, object URL, service status, and reply size of each request. The referrer logs complement burst and session information. Request type distribution from a week access log is list in Table 4.1. Coefficient of Variance (CoV) of request inter-arrival rate is 3.59 during the observation period.

Table 4.1 Trace Data Distribution.

Item	HTML	Image	Audio	Video	Dynamic	Other	Total
Req. Ratio (%)	19.2	68.8	0.2	0.1	3.9	7.8	100
Traffic Ratio (%)	15.0	49.2	1.6	6.7	5.4	20.2	100
Mean Trans. Size(KB)	5.76	4.98	579.9	2503.9	3.84	19.0	7.39
Transfer Size CoV	1.90	2.46	1.76	1.56	1.33	7.90	14.41

Request replayers take data extracted from the traces, regenerate and send requests to the server. The number of independent request replayers is changed to generate different workload intensity. “Burst” of the request flows is well preserved by using multiple independent request replayers. Timestamps have 1-second resolution, and requests with the same timestamp are assumed to be distributed exponentially in a one second time period. Data from each day time period, 9 AM to 9 PM, are used as input of the simulator.

A study based on the traces obtained from ClarkNet [74] was reported by us in [5]. Most of the trends in the inferences obtained here are similar to that obtained through the ClarkNet traces. We have used traces from Michigan State University here to reflect the behavior of most recent traffic patterns.

4.5.2 Server Processing

The parameters for task processing behavior are derived from [75, 45], and by monitoring the network traffic to and from our departmental web server. Studies in [76, 77] have shown that smaller files tend to be more frequently accessed, and caching file copies decreases disk accesses and thereby improves response time. Caching hit ratio data is selected based on the study in [76, 78]. Cache size of each server is set to be 32 MB for static objects with size less than 32 KB. Dynamic and big, i.e., sizes equal or larger than 32 KB, web objects are treated as uncacheable. For a dynamic object request, the service time is dominated by the CPU computation time; and for a large size file request, it is dominated by I/O processing time. The simulation parameters are shown in Table 4.2.

Table 4.2 Simulation Parameters.

Parameter	value
Number of Task Servers	4
System Capacity for Static Objects	1000 req./sec
Task dispatcher Capacity	4000 req./sec
Disk Bandwidth	10 MBps
Disk Seeking Overhead	1 ms
Network Bandwidth	100 Mbps
Outbound Network BW	80 Mbps
Cache Size	32M bytes/server
Caching Threshold	32 KB
Dynamic Objects Processing Overhead	10 ms
Priority Level	2

4.5.3 Performance Metrics

The effectiveness of a scheduling scheme is measured in terms of mean slowdown, mean response time, as well as the 95th percentile response time. Mean response time is defined as the time between the acceptance of a request and the completion of the reply, which is the sum of the waiting time and service time. Mean response time indicates the average time a task stays in the system. The 95th percentile response time shows the response time that majority of tasks experience, which gives out statistical predictability of system responsiveness. Service time includes the service time incurred in the scheduler, in a task server, and in the network interface. Slowdown of a task is the ratio of its response time to its service time. Slowdown of a task gives out a metrics of user tolerance. A user is often willing to wait longer time for a “big” task, which is reflected by the slowdown parameter.

4.6 Results

In this section, we present and analyze the results obtained from our simulator using the real workload traces. We have examined the impact of priority-based scheduling, task assignment policies, and admission control schemes.

4.6.1 Effectiveness of Priority Based Scheduling

In the Internet environment, both access interval and service time distribution are significantly different from the widely used synthetically modeled workload. Therefore, we have used real workload traces for performance evaluation. The high variance of the inter-arrival time and service rate degrades system performance [79], and forces the web server to operate in low utilization state. Figure 4.2 records the mean response time of tasks under different scheduling schemes. Figure 4.3 shows the slowdown of tasks under different scheduling schemes, and Figure 4.4 shows the 95th percentile response time of tasks.

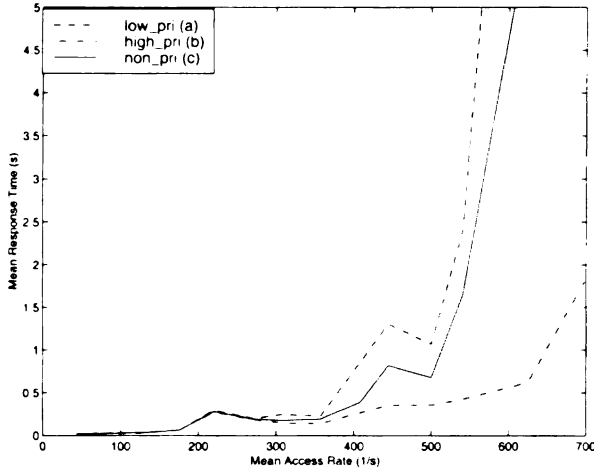


Figure 4.2 Mean task response time vs. scheduling schemes.

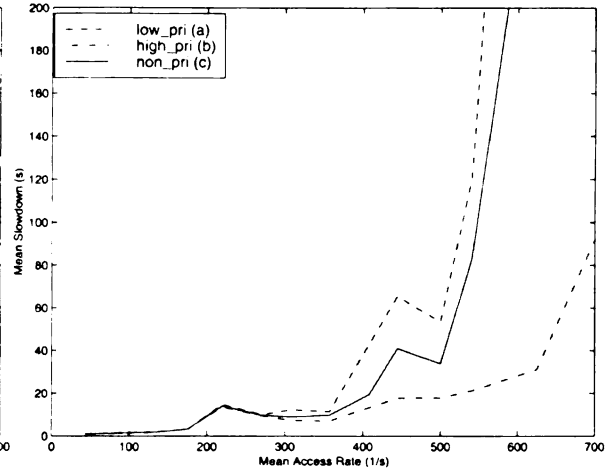


Figure 4.3 Mean task slowdown vs. scheduling schemes.

Curve (c) in each figure indicates the performance trends of non-prioritized processing of tasks. As we can see from Figures 4.2, 4.3 and 4.4, the response time increases sharply with respect to the system utilization. Curves (a) and (b) in each figure are the response time and mean slowdown curves of low priority requests and high priority requests, respectively. High priority tags are assigned to half of the requests randomly, and the rest of requests are marked as low priority tasks. Thus the ratio of high priority to low priority tasks is 1 to 1, and both types of tasks are randomly distributed in the whole arrival sequence. Tasks are assigned to each task server using *RoundRobin* scheduling irrespective of their priorities. Tasks queued at each server are served based on their priority. Specific scheduling and task assignment approaches are discussed later in this section. Performance degradation of the high priority task group happens at a much higher utilization compared to the non-priority-based model. On the other hand, the performance curve of low priority task is fairly close to the performance curve without priority differentiation.

We only consider the stable states of the system in the study, i.e., the task response time and slow down before the sharp performance degradation, or the “knee” of the

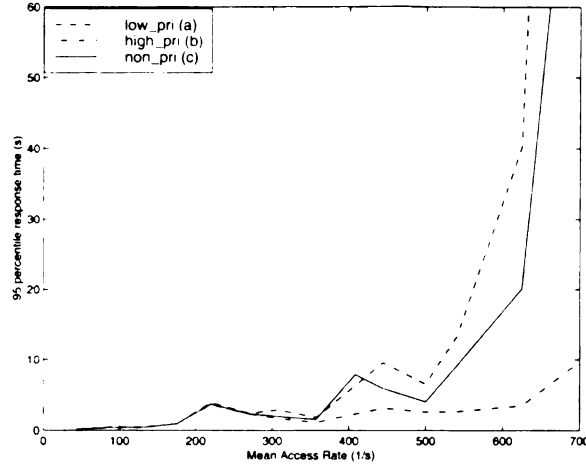


Figure 4.4 95th percentile response time vs. scheduling schemes.

performance curve. From the preceding results, we observe that, the delay is bounded in an acceptable range for high priority tasks in a system with “performance knee” at about 50% of the system capacity for static objects with 50% or less high priority tasks. A steep rise in response time of low priority requests occurs at about 50% of the system capacity, which is about the same as response degradation point without priority differentiation. High priority requests incur average low delay, i.e., less than 2 seconds, even when the system approaches full utilization. Note that the experiment could not achieve the full system capacity due to the overhead incurred by dynamic objects. The 95th percentile response delay of high priority requests is also controlled within a reasonable range, which is less than 10 seconds in the experiment.

The irregularities in the curve at low server utilization are caused by load imbalance among servers while serving continuous media objects. *RoundRobin* scheduling treat each task indifferently, which fails to balance load among servers since continuous media tasks consume much more disk and network bandwidth than other tasks. In the following subsection, we have varied the high priority task proportions from 0.5 used in preceding experiment to 0.9, corresponding to the portions of low priority task from 0.5 to 0.1. In reality the proportion of high priority tasks would be lower

than that of the low priority tasks. The proportions that we have used as rather conservative and may be considered as the worst case scenarios.

4.6.2 Maximum High Priority Ratio

Next, we examine the relationship between the high priority task ratio and the “knee of the curves” in the system. The simulation setup is kept the same as in the last experiment, the only difference is that the high priority task ratio varies from 0.5 to 0.9.

Figures 4.5 and 4.6 show response delay curves of high priority tasks with high priority ratio varying from 0.5 to 0.9. As high priority task ratio increases, the response time curve gets closer to the non-prioritized system response time curve, and the benefit margin obtained from differentiating services diminishes. Figure 4.5 shows that the “knee”s of the mean response time of high priority tasks move closer to that of the non-prioritized system with the increase in high priority ratio. However, by controlling the proportion of high priority requests, we can obtain significant performance benefit from the SDIS.

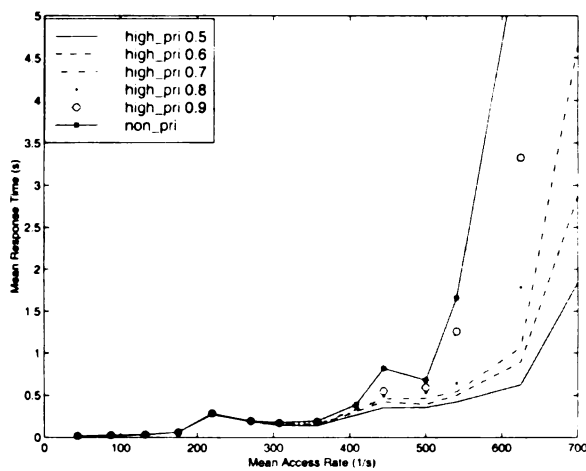


Figure 4.5 Mean response time vs. high priority ratio.

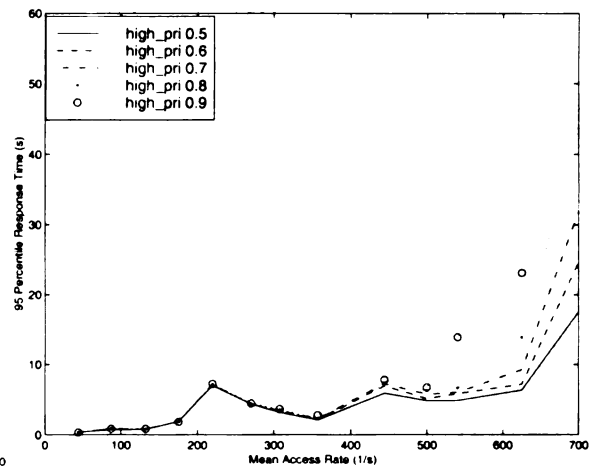


Figure 4.6 95th percentile response time vs. high priority ratio.

Figure 4.6 displays the 95th percentile response time of high priority tasks with high priority ratio ranging from 0.5 to 0.9. The monitored time frame is set to be 60 seconds, which is the default timeout period used in this study. It can be observed from the figure that the high priority tasks rarely gets timed out and retransmitted. Service availability of high priority tasks is much higher than that of low priority tasks.

4.6.3 Low Priority Task Performance

Figures 4.7 and 4.8 show the mean response time and 95th percentile response time curves of low priority tasks with the high priority ratio ranging from 0.5 to 0.9, i.e., low priority task ratio from 0.5 to 0.1, versus traffic intensity.

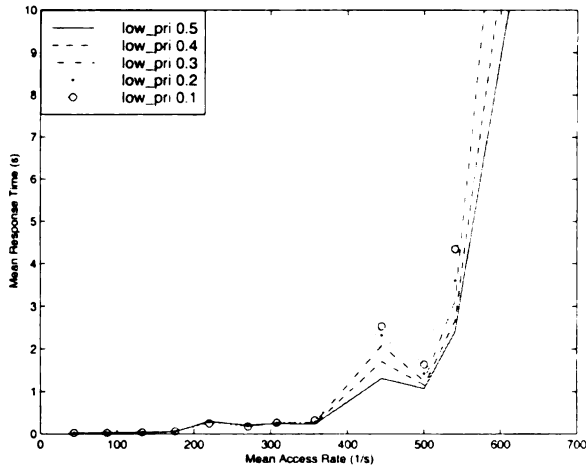


Figure 4.7 Mean response time vs. high priority ratio.

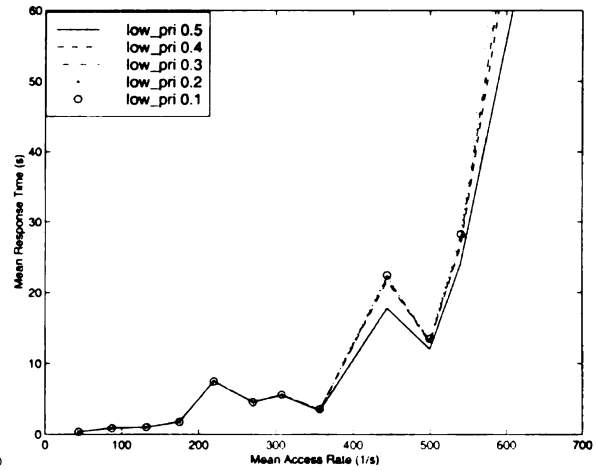


Figure 4.8 95th percentile response time vs. high priority ratio.

It can be observed that the response time of low priority tasks becomes worse with the increase in high priority task proportion, as expected. On the other hand, we find that the spectrum of the occurrence of the “performance knee” in low priority response time curves is relatively narrow, which reflects the minimal influence on low priority tasks with service differentiation. The range of the 95th percentile response

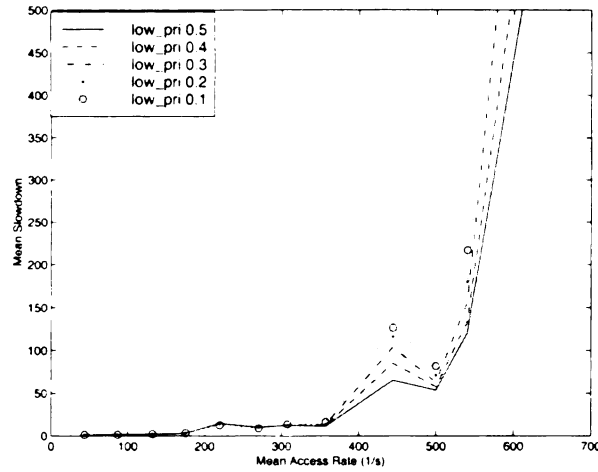


Figure 4.9 Mean slowdown vs. high priority ratio.

time of low priority tasks with varying high priority task ratio is narrower than the mean response time range under the same situation, as shown in Figure 4.8. The slowdown performance of low priority tasks is shown in Figure 4.9, which is rather consistent with the response time performances shown in Figures 4.7 and 4.8.

4.6.4 Task Assignment Schemes

In a distributed server environment, an appropriate task assignment scheme decreases the waiting time variance and thereby improves the system performance. We studied four types of task assignments, Round_Robin (**rr**), Shortest_Queue_First (**sqf**), Prioritized_Shortest_Queue_First (**psqf**), and Reserved_PSQF (**rpsq**), in the experiment. Tasks are assigned to task servers in rotational order when we use **rr** task assignment scheme. It is the simplest task assignment scheme. **sqf** task assignment scheme is based on load balancing techniques, which assigns tasks to the server with lowest number of active processes. The effectiveness of the **sqf** assignment scheme depends on the accuracy of system load information the dispatcher uses. In the experiment, we assume that the scheduler always gets the updated process number in each task server, and the overhead of system load monitoring is 10% of service

time. To adapt to the differentiated service environment, a **psqf** task assignment scheme is introduced in which a new task is assigned to the server with the least number of waiting tasks of equal or higher priority than the incoming task. We also tried a resource reservation scheme **rpsq** in assigning a task, i.e., some resources are reserved for high priority tasks. A high priority task waiting time is expected to decrease by reserving one or two servers exclusively serving high priority tasks.

The response time of high priority tasks are showed in Figures 4.10, 4.11, 4.12 and 4.13. Figures 4.14, 4.15, 4.16 and 4.17 illustrate the corresponding response time variations of low priority tasks. Experimental parameters are kept the same in each of the task assignment schemes. The ratio between high priority and low priority tasks is one to one in Figures 4.10, 4.11, 4.14, and 4.15 to illustrate normal load situation of high priority tasks, and four to one in Figures 4.12, 4.13, 4.16 and 4.17 to show heavy load performance of high priority tasks.

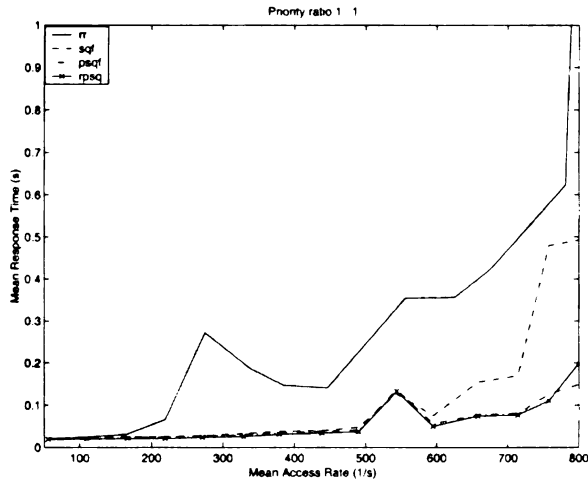


Figure 4.10 High priority task mean response time, priority ratio 1:1.

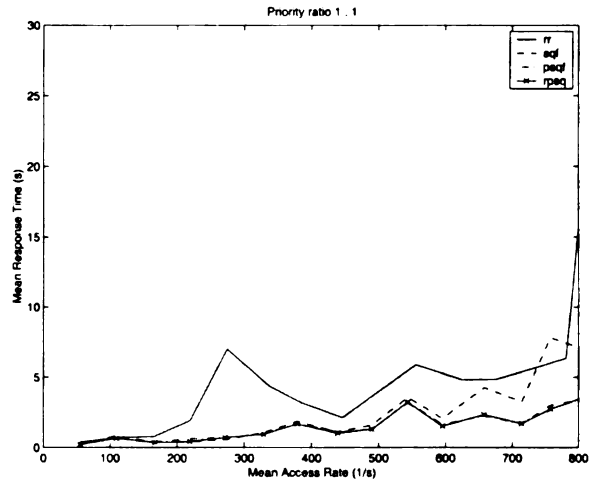


Figure 4.11 95th Percentile high priority task response time, priority ratio 1:1.

The **rr** task assignment scheme consumes the least system overhead, which assigns tasks in rotation order of task servers. There is no need to monitor task server

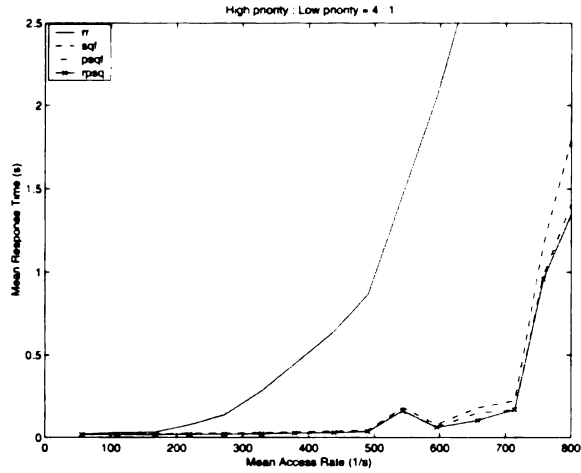


Figure 4.12 High priority task mean response time, priority ratio 4:1.

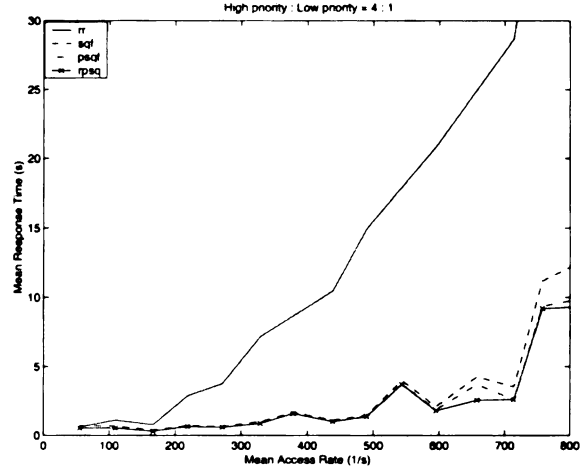


Figure 4.13 95th Percentile high priority task response time, priority ratio 4:1.

load dynamically to make the decision. However, one big task can cause temporary server overload and degrade the system responsibility and throughput drastically. As we can see from the Figures 4.10, 4.12, 4.14, 4.16, using **rr** scheme, both high priority and low priority tasks experience worse response time performance than other task assignment schemes irrespective the proportion of high priority tasks. On the other hand, introducing load balancing techniques in the system improves the system performance on the whole. The experimental results show improved response time performance in both high priority and low priority tasks using the **sqf** scheme compared to the **rr** task assignment scheme.

There are not much differences between the **sqf** and the **psqf** schemes in light load situation. However, the **psqf** scheme performs better in heavy load situation and keeps mean and the 95th percentile response time of high priority tasks considerably lower than the **sqf** scheme, see Figures 4.10 and 4.12. The mean and the 95th percentile response time of low priority tasks are about the same under the **sqf** and the **psqf** assignment schemes, see Figures 4.14 and 4.16.

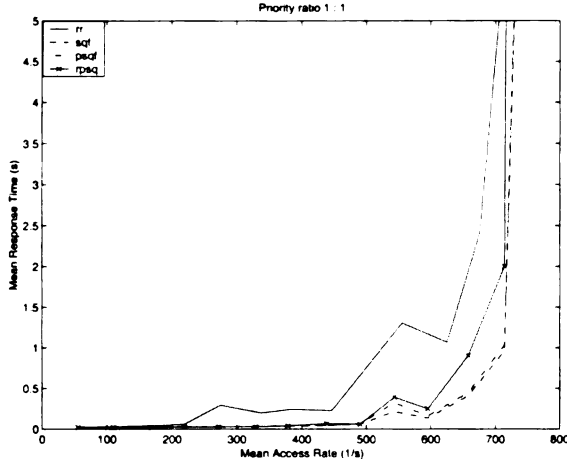


Figure 4.14 Low priority task mean response time, priority ratio 1:1.

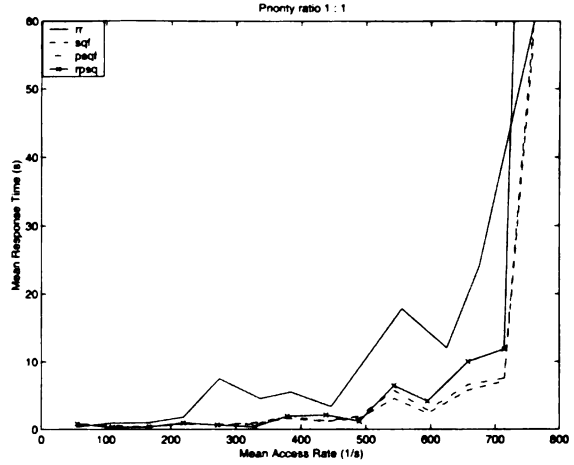


Figure 4.15 95th percentile low priority task response time, priority ratio 1:1.

The performance of high priority task under reservation based **sqf** and **psqf** schemes does not differ much from non-reservation based **sqf** and **psqf** schemes. On the contrary, reservation based **psqf**, denoted as **rpsq**, degrades system response time of low priority tasks when the priority ratio is one to one. The “curve knee” of low priority task response time moves closer to that of the **rr** scheme. Reserving some resources for high priority tasks is proved to be not as effective as load balancing based task assignment schemes.

Comparing the low priority task performance against different task assignment schemes with varied high priority ratio, we observe that there are not much differences in the occurrence of “curve knee”s under different task assignment schemes, although the slope of the curves are different after the “knee”. The results suggest that the increase of high priority task ratio causes low performance penalty to low priority tasks if the system load is well balanced.

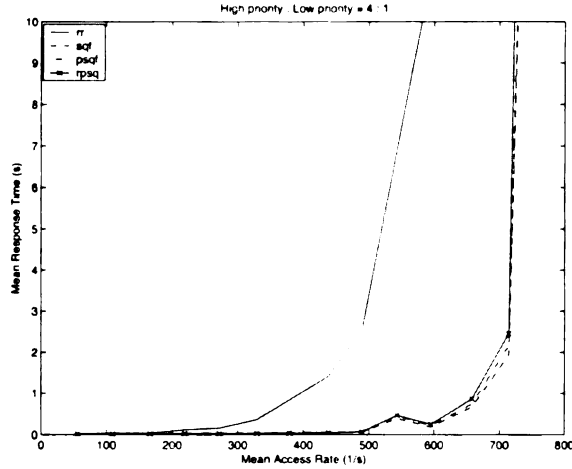


Figure 4.16 Low priority task mean response time, priority ratio 4:1.

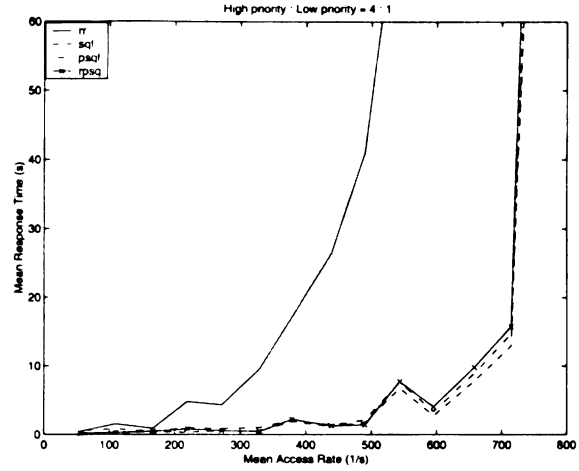


Figure 4.17 95th percentile low priority task response time, priority ratio 4:1.

4.6.5 Preferential Task Assignment

In the previous experiment we have tested the performance issues of different task assignment schemes based on the concept of load sharing and/or load balancing. The results indicates that a simple **rr** task assignment scheme is not efficient in balancing the load and improve system performance in diverse workload environments. Temporary system load imbalance caused by big tasks hurts the system responsiveness to a great extent. Next, we study a preferential task assignment (PTA) on the basis of memory and type affinity in a distributed server environment. The stateless task assignment schemes discussed earlier are enhanced by the stateful delivery. Each type of web objects has its preference of a primary server. The same or same type of web objects go to a specific server as far as the server load is below a threshold. Here we set the load threshold as 1.2 of mean load of task servers. Otherwise, tasks are sent to a secondary server of lighter load at a migration cost. The migration cost is set to be 5% of mean service time. Popular web objects are detected and duplicate cache copies are stored in primary and secondary servers. The copy number and location of cache

copy of a “hot” web object is determined by the object type and the popularity of the object. Continuous media objects are sent to the same server, and processing sharing is used in that particular server. Results are shown in the Figures 4.18 through 4.21. High priority ratio is set to be 0.8 to show the performance bounds of high priority tasks.

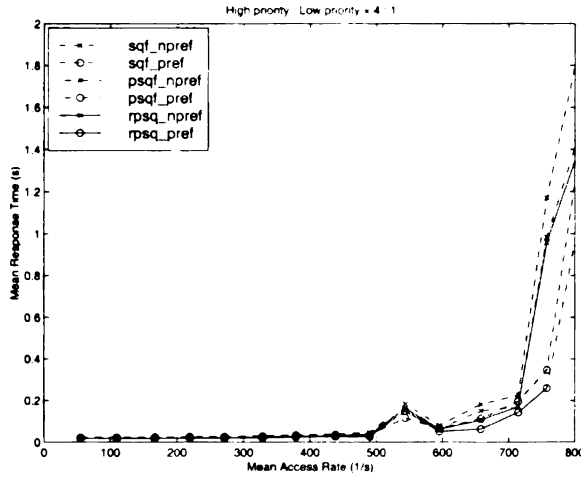


Figure 4.18 High priority task mean response time vs. PTA.

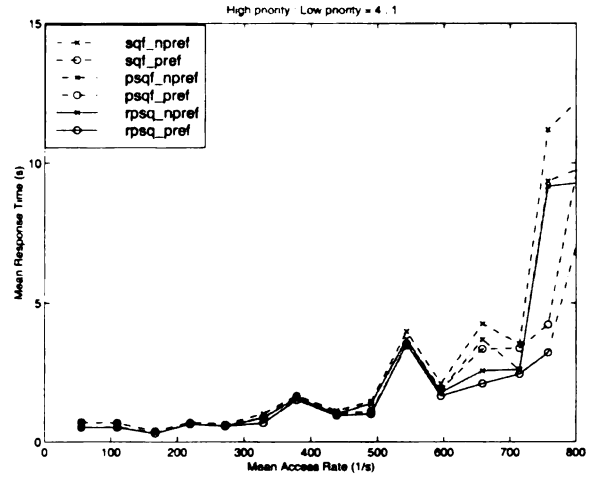


Figure 4.19 95th percentile high priority task response time vs. PTA.

Figures 4.18 and 4.19 compare the response time versus task arrival rate of stateless task assignment schemes and those of the preferential task assignment schemes. Preferential task assignments do help to move performance “knee” of high priority tasks from about 0.8 to 0.9 of the system utilization. The performance improvement comes from improved server caching hit ratio, and decreased service time variance. Performance changes in low priority tasks are negligible, as observed in Figures 4.20 and 4.21. The results indicate that stateful task assignments based on memory or type affinity improve the system performance in the whole.

4.6.6 Admission Control Performance

In the previous experiments, we assume that a processor queue space is unlimited, so is the lifetime of a task. In a realistic scenario, the number of open connections

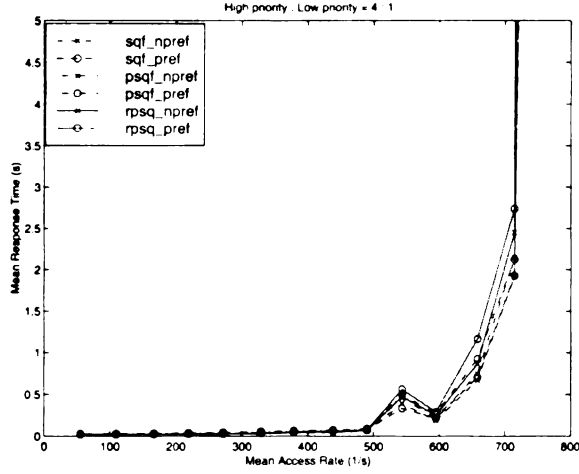


Figure 4.20 Low priority task mean response time vs. PTA.

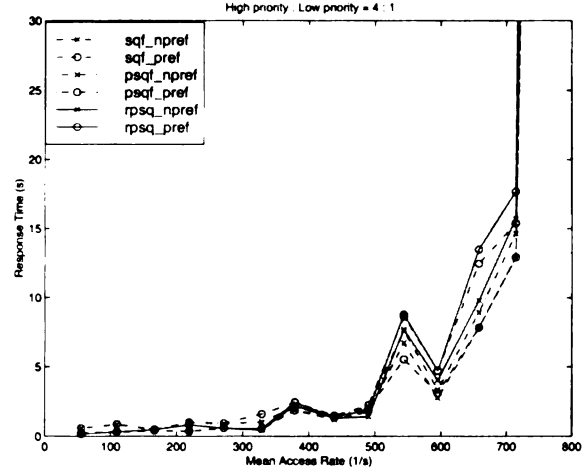


Figure 4.21 95th percentile low priority task response time vs. PTA.

and queue space are often restricted to system predefined values, which corresponds to the availability of resources. High priority tasks experience the same probability of denial of service in an overloaded system if there is no appropriate admission control scheme. In this study, we explore the effectiveness of reserving buffer space for high priority tasks by an admission control scheme and the early discard admission control (EDAC) scheme discussed in section 3. We find that EDAC is suited for light to medium workload. During heavy load period, EDAC is not effective in assuring high priority tasks to get system resources. The accepted low priority tasks starve a long time for service, and those waiting tasks occupy buffer space and cause denial of service to high priority tasks even when the access rate of high priority tasks is below the system capacity, which is shown in Figure 4.22. The ratio between high and low priority tasks is one to one, the early detection threshold of low priority tasks is set to be 0.5 of the buffer space, and the threshold to high priority tasks be the buffer space. We collected the rejection rate data of high and low priority tasks using **sqf**, and **psqf** task assignment schemes, which are the near optimal and optimal task assignment schemes as examined earlier. The slope of the reject rate of high priority

tasks is almost the same as the rejection rate of low priority tasks irrespective of the task assignment schemes. In the following experiment with timeout consideration, we only present rejection rate and abort rate data using **sqf** task assignment, since there are no noticeable differences in performance between **sqf** and **psqf** schemes.

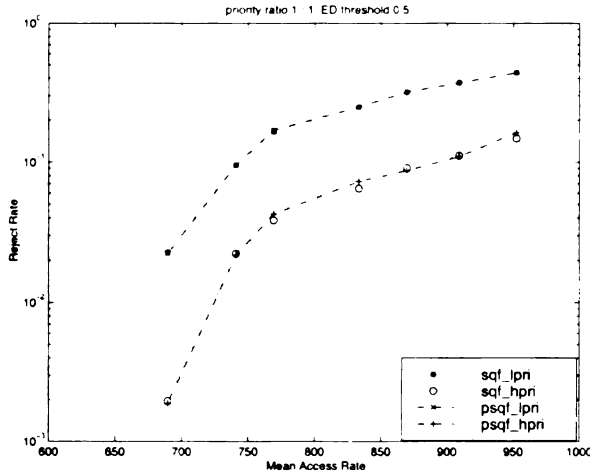


Figure 4.22 Reject ratio vs. task assignment schemes.

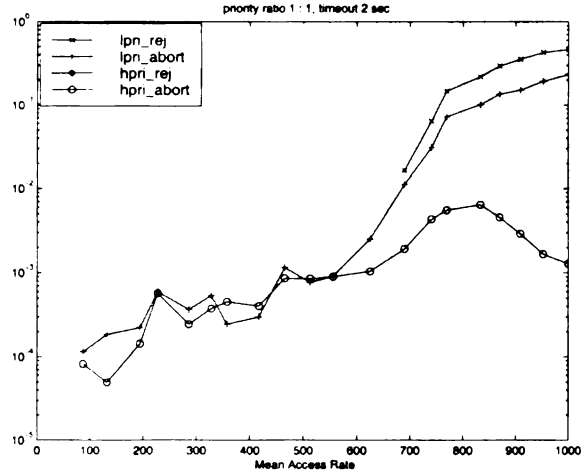


Figure 4.23 Reject and abort ratio vs. priority groups.

The experimental result shown in Figure 4.22 suggest that EDAC is not effective in keeping rejection rate of high priority tasks low under heavy load situations. The rejection rate of high priority tasks increases to more than 10% in high load situation which may be considered as high. We further consider adding timeout to release system resources from starved tasks, and any operations related to tasks that have expired timeout are aborted. The abort rate and rejection rate performance is collected and displayed in Figure 4.23. In results shown in Figure 4.23, a timeout of 2 seconds is added to release the system resources from stale low priority tasks. We repeat the experiment with increased high priority task traffic volume till peak access rate equals system capacity. The result shows that 99.5% percentage of high priority tasks are served within 2 seconds. No high priority task experience denial of services against various task inter-arrival rate. The abort rate of high priority tasks varies

from 0.05% to 0.5%, depends on the distribution of high priority task access rate. It is worth noticing that the abort rate of high priority requests decreases when the task access rate approaches system capacity, which is due to the increase of rejection rate of low priority tasks. The abort rate of low priority tasks is about the same as high priority tasks under light to medium load, and continue to increase with the workload. There is no significant difference between the rejection rate of low priority tasks with and without timeout setting. The results indicate that EDAC and timeout effectively reallocate system resources to high priority tasks during server overload periods.

4.7 Related Works

Although a significant amount of research have been done on the service differentiation at the network level, the work on service differentiation in Internet servers have been limited. Almeida et al. [65] have implemented a prototype of web server which can provide prioritized service in a web hosting environment and studied performance issues in both user space and kernel space. They compared performance of high priority tasks and low priority tasks using synthetic benchmarks. Their research results encouraged us to study prioritized service in a general web server environment. Our work is different from theirs in that we use empirical distribution input which represent the real workload of a web server. In addition, we have also analyzed several other issues of SDIS including task assignment, admission control, impact of memory affinity, etc. Bhatti and Friedrich in HP Labs [80] have built a prototype of task classifier and scheduler on an Apache web server, and studied response time and throughput issues of prioritized services on a web server. They used stress tests which might not be adequate in simulating high variance of web server workload environment. The pricing issues related to the QoS provision on the Internet was analyzed in [81]. They discussed several QoS classification options, which can be also provided by

our server model. We extend their work in terms of service differentiation and classification. The authors of [82] studied task assignment policies in a distributed server system model. Each host processes tasks in FCFS order and the task resource demand is known in advance. Their results suggested that a size-based policy performed the best in a environment of task processing time with high variance and linearly to reply size. However, they did not consider the QoS assurance issues. The authors of [37] investigated the issues of building a scalable web server system on workstation clusters. Load balancing was achieved by **redirection** mechanism provided by HTTP protocol. Extra round-trip delay was the drawback of their scheme. The authors of [43] introduced another type of scalable distributed web server, load sharing and balancing is achieved by combining DNS approach and router dispatching. They have not considered QoS issues in their studies.

4.8 Summary

The next generation Internet will demand differentiated services from Internet servers, which can be achieved through priority based service. Service differentiating Internet servers are needed to provide high quality of service to high priority tasks even under high system utilization. An incoming request can be assigned to a priority group on the basis of the *client*, *network*, *content*, or *owner* of the requested object, and current server load situations. It can also inherit a performance level defined by low level network protocols. Resources are allocated and processes are scheduled according to priority groups. In this study, we prove that under near-saturation of web server utilization, differentiated services provide significantly better services to high priority tasks compared to a traditional web server with minimal performance penalty on low priority tasks. We also present quantitative performance estimation of different levels of tasks.

A high priority task waiting time is determined by high priority arrival rate and the whole system utilization. To maintain a stable system response and throughput states, the arrival rate of high priority tasks shall not exceed the “knee” of the whole system performance curve. Thus we have provided a framework for the determination of the maximum acceptable rate of high priority tasks.

Task assignment schemes toward decreasing the variation of waiting time and service time shortens the perceivable average waiting time of each priority group. A distributed server environment and shortest queue first task assignment schemes help in balancing the load between each task servers, thus reducing the response time variance and average response time. The results suggest that system load monitoring and balancing improve performance for both high priority and low priority tasks. We also explore how to make use of state information in improving system performance. Corresponding preferential task assignment and caching schemes based on object behavior and types help to extend the stable states area in terms of system utilization. Reservation of system resources realized by early detection admission control and timeout schemes improve the system availability for high priority tasks as well as the system throughput.

The results from this study prove that the combination of selective early discard, timeout, and priority queuing is necessary and maybe sufficient to provide predictable response from the next generation Internet Servers. We can configure our server system so that the response time to high priority requests can be controlled irrespective of the volume of low priority requests.

CHAPTER 5 BOUNDING RESPONSE DELAYS IN BUSY WEB SERVERS

5.1 Overview

There has been an explosive increase of evolving applications that use the World Wide Web (WWW) as a distributed information exchange interface on the Internet. Efforts of using the WWW in revenue generating activities, or well known as e-commerce, are also increasing at a fast rate [1]. A widely existing problem in contemporary Web servers, however, is the unpredictability of response times, which is not acceptable to time critical transactions with response delay constraints. On the other hand, *one second* response time is expected from Web sites, which is appropriate to human response speed [83]. Although current Web servers can serve thousands of requests in one second, the average response delay of a popular server may be orders of magnitudes higher than expected during high load periods.

Traditional Web servers are designed more or less like file servers, wherein the majority of activities are retrieval of small static file objects such as HTML or image files. Evolving web servers are more like middlewares to emerging new applications which demand diverse and much improved service qualities. For example, the average CPU time for satisfying dynamic requests is 10 to 100 times higher than the average CPU time for static requests. Secure transactions on the Web, which is popular in E-commerce, consume much more CPU resources than regular transactions. The growth of the Internet traffic and dynamics of server processing impose continuous stress on

server load. With the increase in load, the performance of an Internet server may degrade up until a certain level. Beyond this level, which we call as overload point, the performance drops drastically and the server behavior becomes unstable and may eventually crash [5]. Because of the increase in response time during overloads, users become impatient and sometimes abort the retrievals, which causes waste of precious and expensive server resources. Effective admission control (AC) mechanisms can be used to reject the requests with high abort probability, preserve response delay and throughput of servers.

Most contemporary web servers use a rather naive admission control scheme, called *tail-dropping*, in which incoming requests are dropped when the request queue becomes full. The *tail-dropping* AC scheme requires careful system capacity planning and works well only in static workload situations. In a highly variable workload environment, more flexible AC schemes are needed to adapt to the dynamics of Web traffic.

In this study, we propose a simple and effective AC algorithm called ACES (Admission Control based on Estimation of Service time) to provide bounded response delay to incoming requests under highly variant workload and service processing environments. The ACES algorithm admits or rejects requests based on the estimation of task service times. The service time estimation for each task is determined by the task types. Admission of a request is decided by comparing the available computation power for a duration equivalent to the delay bound with the estimated service time of the request. If the task is admitted based on this constraint, then it is likely that the request will be served within the specific time bounds. A *double-queueing* structure is used to compromise the inaccuracy of estimation and make use of spare capacity of the system, while simplifying the queue management. Experimental study shows that the ACES algorithm provides effective control of response delay bounds to tasks, and preserves system throughput under various workload conditions.

The ACES algorithm also can be used for session management in a generic web server system. Session based predictive AC has been proven to be effective in a uniform workload situation [84], i.e., the resource requirement is regarded as the same in the experiment. The ACES algorithm is similar to the predictive AC proposed in their study. We extend the model by taking into consideration of reserving queue space for premium sessions and packets.

The rest of the study is organized as follows. Section 5.2 discusses in detail about the ACES algorithm. Section 5.3 answers the questions of how to make estimation by analyzing the workload characteristics of a server from real web server traces. The simulation experiment and performance evaluation of the algorithm are reported in Sections 5.4 and 5.5, respectively. Section 5.6 concludes the study.

5.2 Admission Control Algorithm

The goal of ACES admission control algorithm is to provide response delay bounds to incoming requests while preserving the system throughput of busy web servers.

5.2.1 Overview of the algorithm

Usually a queue of incoming requests is maintained in a web server awaiting to be processed. Using the *tail-dropping* AC scheme, incoming requests are put in the queue until the queue is full. Queue length is not always a good indicator of system overload, especially when the variance of processing time is high. Without effective admission control, the server response time and throughput deteriorate drastically when the aggregate request rate exceeds server capacity, indiscriminately affecting all clients. Abdelzaher and Bhatti [52] reported that as much as half of the web system's processing capacity is wasted on eventually aborted/rejected requests when the load is high.

To accommodate the high variance in service pattern of web server systems, we propose a simple and adaptive admission control algorithm, *Admission Control based on Estimation of Service time* (ACES), to provide assurance of bounded response delay, while preserving the system throughput. Service time of each task is estimated based on the request types. Inaccurate estimations are dynamically adjusted and bounded delay is achieved by a *double-queue* architecture.

5.2.2 Delay Bounds Assurance

In a stable system, i.e., if the server has the capacity to process all the requests in the steady state, a queue and the associated queuing delay are introduced to smooth out the short term variations in the job arrival rate. When the short term arrival rate is faster than the processing rate, newly arrived requests have to wait in the queue for service. If variations of job arrival rate and processing rate are high, more queueing buffer is needed, and therefore average queueing time and response delay would be high. One way to increase the response time predictability of a system is to limit the number of accepted requests in a given time period to no more than what the system can handle in that period of time. Borowsky [85] has provided a theoretical proof of short term response time verses arrival rate in a FCFS work-conserving system.

The theorem is described as below: Let $K(t)$ be the queue length at time t . Let $\sum_{n=0}^{K(t)} S_n$ be the sum of the service time for requests in queue at time t . S_0 is the residual service time of the request in service. Let $N(t, t')$ be the number of requests arrive in the period (t, t') , $\sum_{i=1}^{N(t, t')} S_i$ be the sum of service time required by requests $N(t, t')$. Therefore,

$$\sum_{n=0}^{K(t)} S_n \leq \sum_{i=1}^{N(t-T, t)} S_i \leq T. \quad (5.1)$$

The response delay required by pending jobs is no more than T , if the sum of service time required by arriving jobs at any duration T is bounded by time T .

The response time (or delay) of all requests can be bounded by T by restricting the workload arriving in every interval of length T .

Assume that the time is divided into slots $(0, T)$, $(T, 2T)$, ..., $(kT, (k+1)T)$ For simplicity, we use the begin point of each time period, kT , to represent the duration $(kT, (k+1)T)$. Let C_{unit} be the unit processing capacity of the system, $C(kT)$ be the predicted system processing capacity in period kT , $S(i, kT)$ be the service time needed by the i th task at period kT , and $n(kT)$ be the number of admitted tasks in period kT . If the server has a precise knowledge of service time needed by each task, the admission decision can be made based on the following equation:

$$C(kT) = C_{unit} * T \geq \sum_{i=1}^{n(kT)} S(i, kT). \quad (5.2)$$

If the expression is true, then the request is admitted, otherwise it is rejected. The maximum response delay of each task is bounded by the value of T if the service time of each task is known a priori of admission decision. In other words, the admission control manager should have knowledge of the service time of incoming tasks.

5.2.3 Admission Control and Service Time

In deciding to accept or reject a request, the admission control manager should ensure that the sum of service time of accepted tasks does not exceed the system capacity. In reality, it is not possible to know the service time $S(i, kT)$ in advance. High variance in resource requirement is a widely recognized characteristic of web server workload. As indicated in the previous section, however, the service time and bandwidth requirement of the same type of requests are more or less consistent. Service time of web objects can be thus estimated based on the request type distribution of the server access pattern. We approximate the service time of each task by using weighted *computation quantum* (CQ) matching the CPU processing time of different type of tasks.

When a new request arrives, the admission control manager checks if there are enough CQ available to grant to the request. Only requests that are granted enough CQ can be enqueued and served eventually. The number of CQ issued to each task is determined by the resource requirement of the request. For example, the CQ granted to a dynamic request might be 10 times more than the CQ granted to a static request. Here we denote n as the number of types of requests to a web server, which can be derived from the studies of type characteristics of a web server. Let $N_i(kT)$ be the number of request of i type in period kT , CQ_i be the weighted CQ matching the CPU processing time for type i tasks. Then Equation (5.2) can be approximated as:

$$C(kT) \geq \sum_{i=1}^n CQ_i * N_i(kT). \quad (5.3)$$

As in the case of Equation (5.2), a request is admitted if Equation (5.3) holds true, otherwise is rejected.

5.2.4 The Double Queue Structure

Since Equation (5.3) is based on the estimation of service time, we should be careful to adjust the accumulate delay influence of possible over-allocation during a time period. One simple solution is to discard unfinished tasks at the end of each period, although this solution may result in a lot of wastage of resources that have been consumed by the discarded requests. Similarly, a part of the processing power of the system could be wasted during under-allocation situations. To handle the over/under allocation problems, we propose to use a *double-queue* structure. A primary queue is used as the incoming task queue, and a secondary queue is added for the backed up requests (we call this as backup queue). Incoming requests are put into the primary queue, if admitted, otherwise they are dropped. At the beginning of each period, all unfinished tasks in the primary queue are moved to the backup queue. Tasks in the backup queue cannot be served if there are tasks in the primary queue.

Whenever the primary queue becomes empty, tasks in the backup queue get served. When the backup queue becomes full, requests in the backup queue are discarded. By using a *double-queue* structure, newly accepted requests need not wait in the queue for a long time to get service, thus bounded delay is achieved for most of the requests. More details about the double queueing scheme follows in the next section.

Similar algorithm can be expanded to I/O admission control. Because the transmission of a sustained CM object may extend for several sample periods, a *double-queue* structure can not be used directly. The available bandwidth varies in different periods. Let B_{unit} be the unit time processing capacity of the system. Let $B(kT)$ be the effective bandwidth in period kT , wb_i the bandwidth weight of each type of tasks. Let $B_r(nT)$ be the sum of bandwidth needed by unfinished tasks at the end of $((n - 1)T, nT)$. Let m be the maximum number of periods a task can extend. The decision of if a web object get fully serviced can be made based on the equation below:

$$B(kT) \geq \sum_{i=1}^{n(kT)} B(i, kT) \approx \sum_{i=1}^{Tn} wb_i N_i(kT), \quad (5.4)$$

$$B(kT) = B_{unit} * T - \sum_{i=k-m}^k B_r(kT). \quad (5.5)$$

5.3 Service Time Estimation

5.3.1 Web Object Distribution

To explore a simple and effective to estimate the service time of web objects, we analyzed the traces of a busy web server to track trends and characteristics of resource requirements of the requests. The trace data is collected for a week from the web server of the Department of Computer Science and Engineering at Michigan State University, which encountered about a million requests during the period.

Table 5.1 lists the requested Web object types and corresponding traffic distribution of the traces. Web object types are categorized on the same basis as reported in [86]. The first and second rows of the Table show the request ratio of each major type of web objects. It can be observed that requests for small and static web objects, i.e., HTML and image files still dominate the incoming requests and Web traffic. The third row shows the mean response size in kilobytes of each object type. The fourth row shows the coefficient of variance (CoV) of the reply size of each web object type. The data show that the CoV of web traffic is as low as 1.2 for each type of web objects. However, the CoV of the aggregated web traffic can be as high as 14.41.

Table 5.1 Trace Data Distribution.

Item	HTML	Image	Audio	Video	Dynamic	Other	Total
Req. Ratio (%)	19.2	68.8	0.2	0.1	4.9	6.8	100
Traffic Ratio (%)	15.0	49.2	2.6	6.7	4.4	20.2	100
Mean Traffic	5.76	4.98	579.9	2503.9	6.84	19.0	7.39
Traffic CoV	1.90	2.46	1.76	1.56	1.33	7.90	14.41

There are non-negligible percentage of dynamic objects. This type of access requires more processing power, and thus increases the server load and the unpredictability of response delay. Previous studies [14] have estimated that a dynamic object requires 10 to 100 times of service time than a typical static object. The processing time difference was confirmed by recent benchmark tests of popular commercial web servers [35] and the following service time characteristic study. The throughput of serving dynamic tasks are usually lower than serving static tasks by a factor of 10. Continuous media (CM) objects (audio and video clips) are now being used more extensively in the web environments (compared to the previous results in [86]). These data types requires high storage capacity (thus uncacheable), high density I/O processing, and sustained bandwidth with bounded delay constraints. The average CPU time for serving a CM object is also much higher than the average CPU time for serving small static objects. Significant differences in size of different

type of web objects, high volume of CM objects, and computation-intensive dynamic requests suggest that the service time variance of the requested web objects should be considered in estimating the server capacity and response time. Classification of Web objects can be used to approximate and regulate the server processing, thus decrease the variance in server processing and improve the delay performance.

5.3.2 Service Time Distribution

Previous studies [52, 82, 87] suggested that, the service time of retrieving static web objects such as HTML and image files can be divided into two parts: a rather fixed URL processing time, and the content transfer time which increases linearly with the file sizes. To collect the service time attributes, we set up an experiment environment consisting of one Apache (version 1.3.12) web server and three WebStone (version 2.5) clients, connected through a 10 Mb Ethernet. The web server hardware is based on a Pentium II 233MHZ CPU with 128 MB memory. The operating system used in the server is Linux 2.2.9. Three clients are installed in Sun UltraSparc 10 with 128 MB memory, running Solaris 2.6.

The CPU cycles are used as timing scaled to obtain a precise time resolution, since the typical service time of a request is much less than one second. Even millisecond resolution is not sufficient to provide an accurate picture of the service time distribution in different processing procedures. The *performance monitoring counters* provided by the Intel P6 family processors are used to record the elapsed busy CPU cycles of the current process.

The response time is defined as the duration from the time that the server accepts a client request to the time that the response transmission finishes. The response time include CPU busy times and idle times waiting for resources. Figure 5.1 plots the response time of requests versus file sizes and maximum process numbers.

It can be observed from the figure that the response times are rather constant if the files are smaller than 64 Kbytes. If the files size exceeds 64 Kbytes, the mean

response times are determined by network bandwidth. For example, the effective bandwidth of the TCP traffic in the Ethernet is 1 Mbytes per second. The response time of a 1 Mbyte file request is around one second under single process situation. The response time of 1 Mbyte file request is around ten seconds if the maximum process number is 10.

The service time is defined as the CPU cycles consumed between the time that the server accepts a client request, and the time that the server finishes sending the response. Figure 5.2 depicts the mean service time of static objects versus requested file size under different *maximum process number* (MPN) in the server. It is well known that the MPN has direct influence in service time distribution, which are discussed in detail in following paragraphs. CPU cycles have been converted into seconds in the figure. The CoV of obtained service times vary between 0.09 to 0.11.

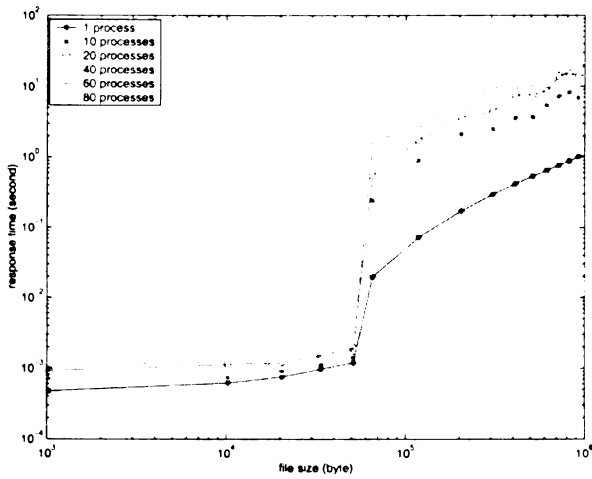


Figure 5.1 Mean response time of web objects.

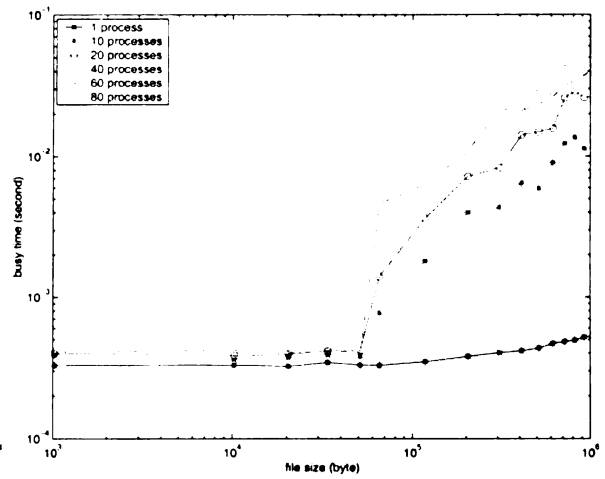


Figure 5.2 Mean service time of web objects.

It can be observed from the figures that the curves have two phases. If the files are smaller than 64 Kbytes, the service times with the same MPN are rather constant, and the service times of $MPN > 1$ is about twice as the service time of $MPN = 1$. If the file sizes exceed 64 Kbytes, the mean service times increase linearly with the file sizes and the MPN value. We call this phenomenon the *64 KB leap*. The *64 KB leap*

is mainly due to the restriction by the IP protocol. Since the maximum IP packet number is 64 KB, any response bigger than 64 KB can not be transferred in one packet. Memory contention overhead and context switching overhead are introduced due to the network I/O blocking between packets. Some other factors also contribute to the *64 KB leap*. For example, asynchronized disk I/O is widely used in current UNIX operating systems, and the default size for *read-ahead* operation is 64 KB. At most 64 KB can be loaded from hard disk in one I/O operation.

The slopes of service time increase linearly with the number of maximum processes, because the increase of process number caused higher probability of page faults, higher context switching, and synchronization overhead. Based on the observation from Figure 5.2, the service time of a task $T(s, n)$ can be estimated by the file size s KB and MPN value of n .

$$T(s, n) = a + [s/64] * (b + c * n), \quad (5.6)$$

where a is the service time for small static web objects, i.e., requests for files smaller than 64 KB. b is the data transferring time factor, and c is the context switching overhead factor. Using linear regression, we get the relative value for a, b , and c as: $a : b : c = 1 : 0.06 : 0.18$.

Rechecking the file size distributions, we find that less than 1% of HTML and image files are larger than 64 KB. Thus the service time of most HTML and image files are rather uniform. Service times of dynamic objects, mainly CGI requests, are depend on the computation complexity of the URL processing instead of response size. Using the testing CGI scripts provided by WebStone 2.5 test set, the average service time of a CGI request is around one order of magnitude higher than the service times of static objects with a file size less than 64 KB. The experiment results indicate that the object type is a good indicator of CPU time needed, which can be derived from the requested URL easily. Besides, classification of object types introduces less overhead than retrieving file size information, which requires one *fstat()* system call in UNIX operating systems.

5.4 Simulation Methodology

To test the effectiveness of the ACES algorithm, we develop an event driven simulator for the web server using empirical workload from real trace files. The reference locality, size and object type distribution extracted from the logs of the Computer Science departmental web server at Michigan State University are used for the workload generation.

The simulated system structure is shown in Figure 5.3. An incoming request is first sent to the admission control manager **AC**. The **AC** classifies the request type and decides if the request can be enqueued based on the AC algorithm. Enqueued requests wait to be served in the primary queue Q_p . At the end of each period or the beginning of the next period, unfinished requests are sent to the back up queue Q_b . When Q_b gets full, it is cleared up and all the tasks are dropped. The task scheduler **TS** picks up requests in the queues and sends to the server pool. No request in the backup queue is serviced unless the primary queue is empty. Replies are sent back through the server network interface **Nb**. The system configuration is set based on the average configuration of current popular web servers as in the Table 5.4.

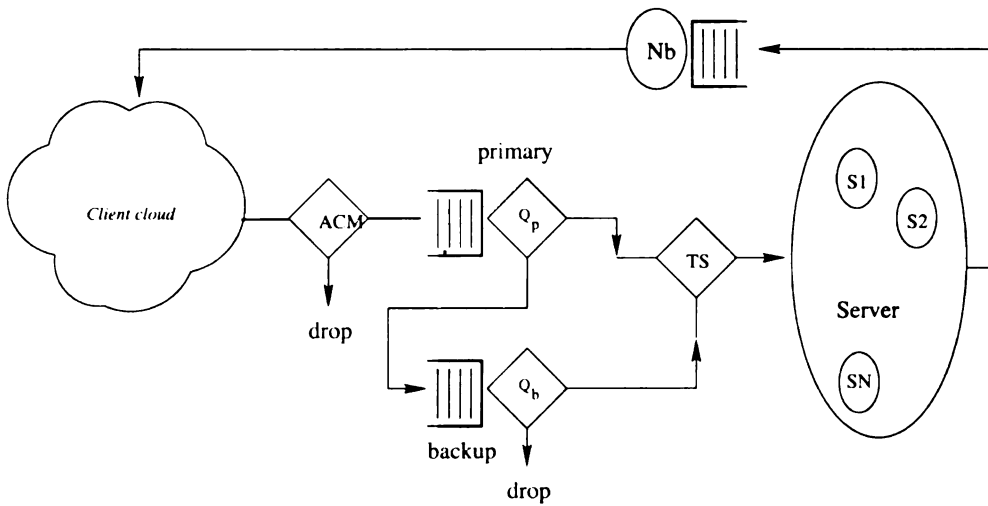


Figure 5.3 Web server system structure.

Table 5.2 Simulation Configuration.

Parameter	value
Priority Level	2
Scheduling period	1 second
System Capacity for Static Objects	1000 req./sec
Network Bandwidth	50 Mbps
Disk Bandwidth	10 Mbps
Caching hit ratio	0.7
Dynamic Objects Processing Overhead	10 ms
Maximum open connection	1000
Total queue length	1000
Response delay bounds	1 second

Three performance metrics are used in the simulation: server throughput, mean response time, and response delay bounds miss rate. System throughput indicates the server capability and measures the rate of request served. Mean response time and delay bounds miss rate quantify the service qualities. The delay bounds miss rate is measured in terms of the ratio of the number of requests with response delay of more than one second to the total number of admitted requests.

Three kinds of admission control algorithms are implemented to examine the effectiveness of ACES algorithm in terms of providing bounded delay and high system throughput. The first admission control algorithm, we call it *simple admission control* (SAC) algorithm, is analogous to the leaky bucket [88] algorithm used in traffic engineering in the Network transmissions. Using the SAC scheme, each admitted request is allocate one CQ irrespective of the request type, thus there is no estimation of service time. The SAC scheme performs better than the tail-dropping scheme, since it smoothes out the web server traffic and provides preliminary overload protection.

Another admission control algorithm used for comparison purpose is the *conservative admission control* (CAC) scheme. The CAC scheme is a hypothetical scheme, in which the server is assumed to have precise knowledge of service time a request needs. Using the CAC scheme, the allocated CQ matches exactly to the service time require-

ments. Although implementation of the CAC is unrealistic, we use it to compare the performance of ACES scheme.

Based on the data collected, web objects are classified into 4 types: static, dynamic, audio and video objects. The CQs consumed by each object type in the ACES scheme are listed in the Table 5.3.

Table 5.3 CQs allocated to each object types.

Object Types	Static	Dynamic	Audio	Video
CQs	1	10	20	100

The service time and corresponding CQs assigned to audio and video files are calculated based on their average size. The maxium process number is set as 30, which is the default configuration for Apache web server.

5.5 Results

The three admission control algorithms as noted in the previous section are deployed in the web server simulator. Two kinds of workloads are used in the experiment. The first is the stress test workload, in which traffic intensity increases continuously till 2.5 times of the system capacity. We try to explore the capability of the three algorithms in preserving the system throughput under extremely high load by stress test. The second kind of workload is aimed to examine the sensitivity of the three algorithms under fluctuating workload. There are two modes in the workload series used in the sensitivity test; One is sustained lightload or overload, and the other is occasional lightload or overload. The maximum workload is 2 times of the system capacity. The occasional overload duration is 2 time units, and the sustained overload duration is 10 time units. Each time unit is 100 times of the observation period, i.e., 100 seconds.

5.5.1 Throughput Comparison

Figures 5.4 and 5.5 plots the throughput performance of the three admission control algorithms under the two tests. In Figure 5.4, X-axis is the aggregate load of the server, and Y-axis is the normalized throughput. In Figure 5.5, X-axis is the time series of the load input, Y-axis is the the normalized system throughput, and the dotted line is the load intensity.

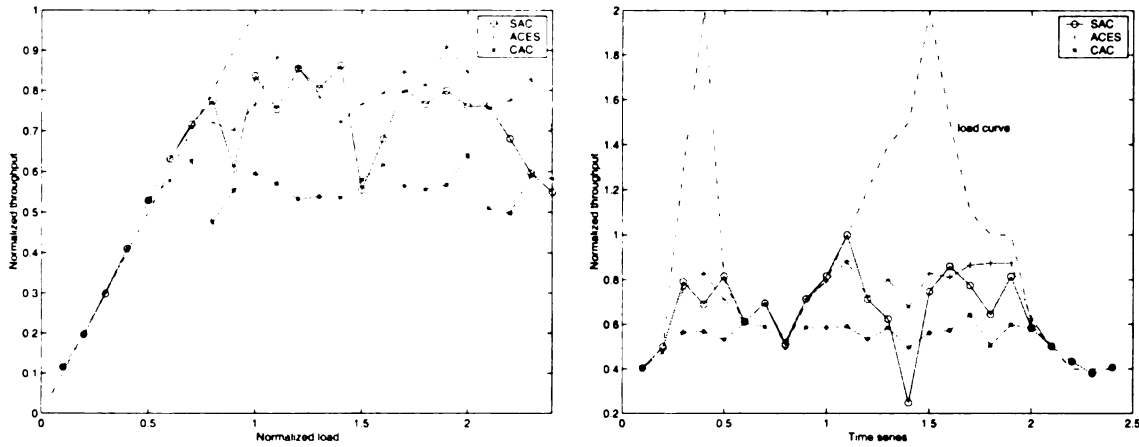


Figure 5.4 Throughput of stress test. Figure 5.5 Throughput of sensitivity test.

It can be observed that the throughput of the system increases linearly with the workload during low load periods (lower than 0.5 of system capacity) irrespective of the admission control scheme. With the increase in load, the throughput of the CAC scheme stays behind the other two schemes. The reason is that the CAC scheme wastes system resources when the inter-arrival rate of requests is temporarily lower than the system capacity. When the system load is around the system capacity, the throughput performance the ACES scheme is very close to the SAC scheme. Under even higher load situation, the ACES scheme outperforms the SAC scheme because of the variable CQ allocations based on the estimation of service times. Another interesting phenomena showed in the figures is that, the throughput of the system tends to be more stable and not influenced by the fluctuation of the workload when using ACES compared to SAC.

5.5.2 Average Delay

Figures 5.6 and 5.7 depicts the average delay performance of the three admission control algorithms under the stress test and sensitivity test. The mean response delay of the ACES scheme is fairly close to that of the CAC scheme, which is about one tenth of the delay bounds during overload situations. The average delay is slightly higher in using the SAC scheme than the other two schemes. However, the average delay of the SAC scheme is one order of magnitude higher than the other two schemes during overload periods. Since nearly all the tasks need to wait from the tail of queue for service under high load situations, the average delay of the SAC scheme under high load periods is determined by the system total queue length. The experiment proves that the system responsiveness of ACES is close to the ideal case. Based on the queue length and waiting time relationship per Little's Law, the average delay differences between the three AC schemes suggest that the required queue length might be much shorter for ACES compared to that of SAC.

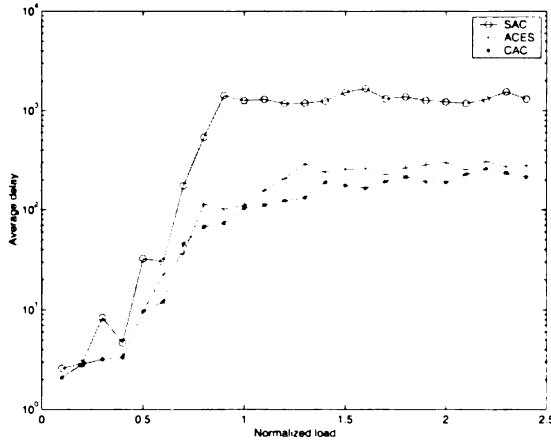


Figure 5.6 Delay of stress test.

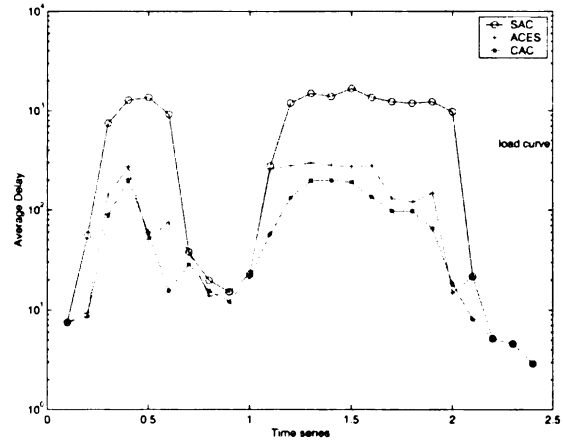


Figure 5.7 Delay of sensitivity test.

5.5.3 Delay Bounds Miss Probability

Figures 5.8 and 5.9 depicts the delay bounds miss ratio of the three admission control algorithms under two tests. As expected, delay bounds miss rate (**Dbm** rate)

for CAC is zero, since the system stops admitting tasks when it cannot serve them within the delay bounds. Using the SAC scheme, on the contrary, the system fails to meet the delay bounds for nearly all the tasks in high load periods. The reason is that the SAC scheme fails to catch up with the changes of system resource consumption and tends to over-admit tasks. The over admission leads to the formation of long queue, thus introducing long waiting time for all admitted tasks. However, under low or medium load situation (about less than 0.7 of the system capacity), the SAC scheme performs as well as the other two schemes. The performance of ACES is good under reasonably high load (less than 1.7 of the system capacity), and nearly no delay bounds miss is incurred at this load. Under extremely high load situation, the ACES scheme has about 10% of delay bound miss ratio which is not too high.

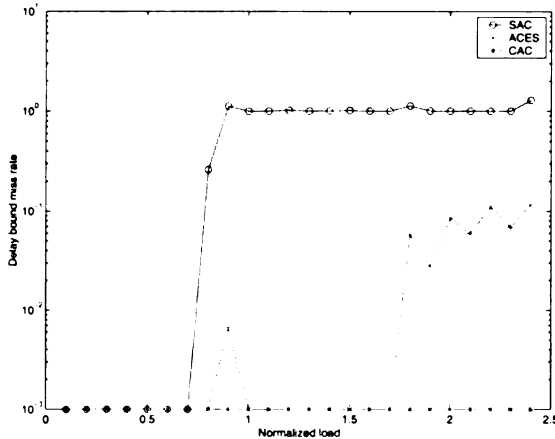


Figure 5.8 Dbm rate of stress test.

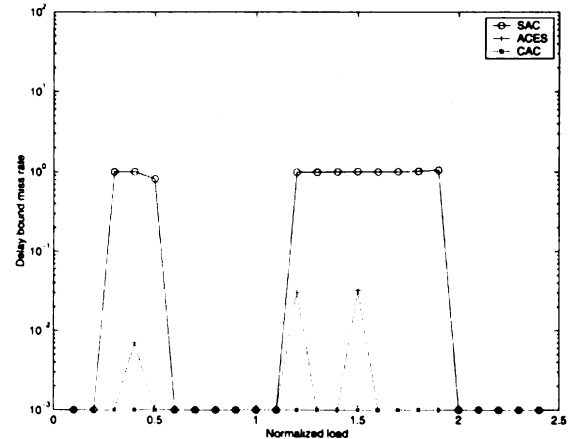


Figure 5.9 Dbm rate of sensitivity test.

The two tests prove that the ACES algorithm is effective in providing delay assurance as well as in preserving system throughput under diverse load and overload situation in a web server system. Its throughput performance is closer to the SAC scheme, and the response delay performance is closer to the CAC scheme.

The CAC scheme is very powerful in controlling the deadline misses of tasks, as it does not incur any bound miss. However, the throughput of CAC is much lower than the other two schemes. The low throughput is partly due to the strict admission

control. The server does not make full use of computing resources when the variance of request rate is high. The CAC scheme is suited for the hard real-time services where deadline guarantee is critical.

The SAC scheme is suitable for a system with uniform workload, as it does not work well in highly diverse workload environments such as a web server. The SAC scheme performs well during low or medium load periods. During high or overload periods, the SAC scheme tends to over allocate system resources, which causes long response delay. Accumulative effect of overload has long term influence in degradation of system responsiveness and throughput.

5.6 Summary

The growth of the Internet and WWW applications have imposed continuous challenge to the Internet server performance and quality of service assurance in terms of predictable delay and throughput. However, the prevalent “bursty” nature of the server access patterns makes it difficult and expensive to maintain fast response at all times even by a high performance server. The peak workload of an Internet server may exceeds the average workload by orders of magnitudes. In this study, we present a simple and effective admission control algorithm, ACES, to adapt to the highly variant processing times of web server environments. Tasks are admitted based on the estimation service time. A double-queue structure is deployed to compromise the inaccuracy of service time estimation, prevent accumulation of response delay, and improve the system throughput. A detained experimental measurement of service time distribution of web objects is conducted to provide foundations of service time estimation. Simulation results demonstrate that the ACES algorithm is able to provide assurance of response time while maintain its throughput under various workload situations.

Results also show that the ACES algorithm outperforms existing *tail-dropping* admission control scheme by up to 10 times delay improvement. It exceeds the throughput performance of conservative admission control scheme by about 30% on medium to high load. Although this study is based on the UNIX environment with high speed connection, it can be easily extended to other environments with adaption of different service time computation quantum.

CHAPTER 6 AN EFFICIENT ADMISSION CONTROL ALGORITHM FOR SDIS

6.1 Overview

Since the beginning of 1990s, Internet and Internet services have experienced exponential growth. Owing to its universal accessibility and low cost, the use of the web in revenue generating activities, well known as e-commerce, is also increasing at a fast rate [1]. A widely existing problem in contemporary web servers, however, is the unpredictability of response time, which is caused by the FCFS service model and the “bursty” workload behaviors. Usually, *one second* response time is desired from web sites, which is appropriate to the human response speed [83]. Long response delay frustrates user interest in interaction with servers, thus devalues the web service quality. Although current web servers are able to serve thousands of requests per second, the average response delay of a popular server can be several orders of magnitudes higher than that expected during high load periods, causing the *de facto* “denial-of-service” effects.

The response delay of Internet service is determined by two factors: the quality (delay, delay jitter, and loss rate, etc.) of network transmission, and the processing capacity of the server. Study of quality of service (QoS) in network transmission are active in recent years, including efforts of building *Integrated Services* (IntServ) architecture [7] with end-to-end QoS guarantee, and *Differentiated Service* (DiffServ) [8] architecture with alternative levels of services provisioning. However, network

layer QoS is not sufficient in providing user perceivable high performance if the server does not offer any service and performance assurances. Unexpected bursts of client demands may cause long queuing delay or even crash an overloaded web server. Even premium data flow with end-to-end QoS guarantee may still experience service rejection when the server is overloaded. On the other hand, session/transaction-based service differentiation can only be provided from application layer servers. With the boost of resource requirements of web based applications, the web is expected to evolve from “free” to “paid” web in the near future. Value added services will emerge with competitive differentiation of service offerings based on profits instead of the *best-effort* service discipline.

Recent studies on QoS support in web servers have addressed the technology of prioritized processing in a web server and related performance issues [3, 4, 5, 6]. By prioritization of tasks, it is possible to maintain low response delay for high priority tasks during high server load periods by blocking or dropping low priority tasks. We call this kind of servers as service differentiating Internet servers (SDIS), which reciprocates QoS efforts in the Internet transmission. The basic ideas of SDIS include classification of client requests into groups with different service requirements, resource allocations based on the task groups, and prioritized scheduling and task assignments schemes. A detailed study on the concept and performance evaluation of SDIS is reported in [89].

Prioritized scheduling of a web server has been proven effective in providing significantly better delay performance to high priority tasks at relatively low cost to lower priority tasks in previous studies. However, it is still possible that a significant amount of high priority tasks are dropped under extremely high load situations and user acceptable response delay cannot be ensured. Effective admission control (AC) mechanisms are needed to assure the drop rate and the delay bounds of tasks.

Most contemporary web servers use a rather naive AC scheme, namely *tail-dropping* AC, in which incoming requests are dropped when the number of tasks

awaiting exceeds a predefined threshold. The *tail-dropping* AC scheme requires careful system capacity planning and works well only in steady workload situations. In a highly variable workload environment, more flexible AC schemes are needed to adapt to the dynamics of traffic. For example, secure transactions on the Web, which is popular in E-commerce, consume much more CPU resources than regular transactions due to encryption/decryption and multiple handshaking overheads. The AC algorithm should be designed to consider the characteristics of a wide variety of requests.

In this study we propose a simple and effective AC algorithm, PACERS (*Periodical Admission Control based on Estimation of Request rate and Service time*), to provide bounds on response delay for incoming requests under highly variant workload environments. The PACERS algorithm dynamically adjusts system capacity for each priority group by estimating the request rate of tasks. Admission of a request is decided by comparing the available computation power for a duration equivalent to the predetermined delay bound with the estimated service time of the request. If the task is admitted based on this constraint, then it is very likely that the request will be served within the specific time bounds. The service time estimation is done on the basis of the request types. A *double-queueing* organization is used to diminish the inaccuracy of estimation and exploit spare capacity of the system, while simplifying the task management. Theoretical analysis and experimental study show that the PACERS algorithm bounds the response delay for tasks in different priority groups, assures the service availability to high priority tasks even under high load, and preserves system throughput under various workload.

The rest of the chapter is organized as follows. Section 6.2 answers how to estimate request rate and service times by analyzing the workload characteristics of a real web server. Section 6.3 discusses the PACERS algorithm in detail and its implementation issues. Section 6.4 provides a theoretical proof of the delay performance of

the PACERS algorithm for each priority group. Simulation results and performance evaluation of the algorithm are reported in Sections 6.5 and 6.6, respectively. Related works are discussed in section 6.7. Section 6.8 concludes the study.

6.2 Workload Characterization

Admission control policies are used extensively to provide congestion control and to enforce desirable performance in computer and communication systems. The workload on web servers are different from that of the network traffic, thus existing network-level AC algorithms might not suit well in web server environments. Apparent self-similarity and long-term dependency are prevalent in the WWW traffic pattern [66, 61, 72]. On the other hand, processing of web requests are much more complicated than the handling of network packets. To explore a simple and effective admission control algorithm which fits to the web server environments, we analyze the traces of a busy web server to track trends and characteristics of resource requirements of the requests. The trace data is collected for a week from the web server of the Department of Computer Science and Engineering at Michigan State University, which encountered about a million requests during the period.

6.2.1 Access Distribution

The access logs provide the request timestamp, client ID, object URL, service status, and reply size of each request. The referrer logs complement burst and session information. Coefficient of Variance (CoV) of request inter-arrival rate is 3.59 during the observation period. High CoV means high variances of access rates, which are demonstrated in Figures 6.1, 6.2, 6.3, 6.4, 6.5 and 6.6. Figures 6.1 and 6.2 plot the number of requests and web traffic volume (reply size in bytes) of the web server per second in a day. Figures 6.3 and 6.4 show the number of requests and web traffic volume of a web server per minute in a day. The requested web traffic volume have

even higher CoV value of 14.4. Note that the traffic volume is calculated according to the access logs of the web server, in which the reply size of each request is recorded. The access logs record the time point when a request is finished instead of the traffic transmission duration. The real network traffic should be smoother than those indicated in the diagram due to throttling effects of network transmission.

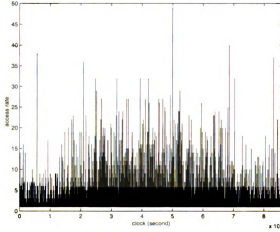


Figure 6.1 Access distribution a day (1 sec).

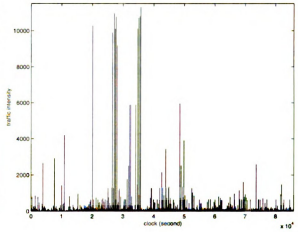


Figure 6.2 Traffic distribution a day (1 sec).

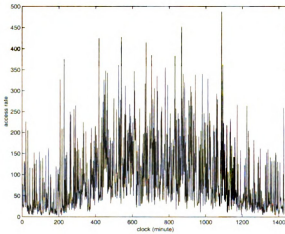


Figure 6.3 Access distribution a day (1 min).

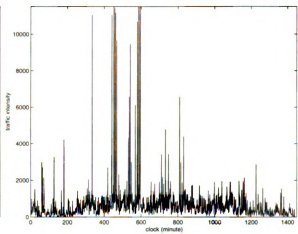


Figure 6.4 Traffic distribution a day (1 min).

The Coefficient of Variance (CoV) of request inter-arrival rate with 1 second res-

olution is 3.59 during the observation period, which by large is due to the cyclic behavior of the server access pattern. Figures 6.5 and 6.6 show the request intensity and traffic volume per hour in a week. Distinct seasonal/periodic behaviors according to the time series can be observed from the figures. For example, the access intensity continues to increase from early morning, at 10AM reaches the high intensity area, and stays high till late afternoon, then decreases slowly to the lowest point in a day, which is about 3AM in the next day. Similar traffic pattern repeats around the same time each day, which suggests that both short and long term history should be used as predictors of future workloads.

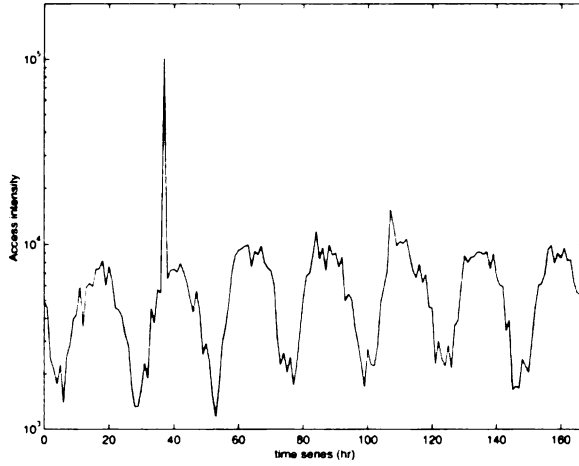


Figure 6.5 Access trends in a week.

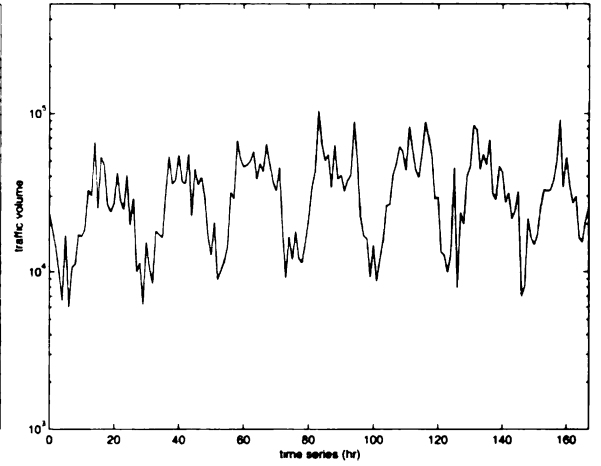


Figure 6.6 Traffic trends in a week.

The fluctuation in traffic intensity can be several orders of magnitudes, which causes the high variances in access rate and bandwidth requirements, thus resulting in high average waiting time of tasks in a system. Care should be taken to eliminate the effects of variances. An interesting phenomenon is that the access pattern of the web server tends to be consistent in a short time window. For example, the CoV of access rates decrease to around 1 when they are measured on an hourly basis. The decrease for CoV is even more obvious during busy hours. Similar observations have been reported in [91, 92, 77], which suggest that a multi-state Modulated Markov

Poisson Process (MMPP) can be used to approximate or predict the burstiness of the aggregate input of the web server, which is discussed in more detail in Section 6.3.3.

Note that during a week's observation period, there is one overload point which causes request access rate of about 10 times higher than the common peak load of the day. We examined the logs and found that the abnormal overload was caused by a group of careless CGI based requests. Those requests directly caused the web server failing to respond to any other request for 21 minutes. This is one of the examples which justify the need for appropriate AC to prevent system resources being wasted in careless or malicious request attempts.

6.2.2 Object type distribution

Table 6.1 lists the requested Web object types and corresponding traffic distribution of the traces. Web object types are categorized on the same basis as reported in [86]. The first and second rows of the Table show the request ratio of each major type of web objects. It can be observed that requests for small and static web objects, i.e., HTML and image files still dominate the incoming requests and web traffic. The third row shows the mean reply size of each object type. The fourth row shows the CoV of the reply size of each web object type. The data show that the CoV of web traffic is as low as 1~2 for each type of web objects. However, the CoV of the aggregated web traffic can be as high as 14.41.

Table 6.1 Traffic distribution vs. object type.

Item	Req. (%)	Traffic (%)	Mean (KB)	CoV
HTML	19.2	15.0	5.76	1.90
Image	68.8	49.2	4.98	2.46
Audio	0.2	2.5	579.9	1.76
Video	0.1	6.7	2503.9	1.56
Dynamic	4.9	4.4	6.84	1.33
Other	6.8	20.2	19.0	7.90
Total	100	100	7.39	14.4

There are non-negligible percentage of dynamic objects. This type of access requires more processing power, and thus increases the server load and the unpredictability of response delay. Previous studies [14] have estimated that a dynamic object requires 10 to 100 times of service time than a typical static object. The processing time difference was confirmed by recent benchmark tests of popular commercial web servers [35] and the service time characteristic study in Chapter 5.3.2. The throughput of serving dynamic tasks are usually lower than serving static tasks by a factor of 10. Continuous media (CM) objects (audio and video clips) are now being used more extensively in the web environments (compared to the previous results in [86]). These data types requires high storage capacity (thus uncacheable), high density I/O processing, and sustained bandwidth with bounded delay constraints. The average CPU time for serving a CM object is also much higher than the average CPU time for serving small static objects. Significant differences in size of different type of web objects, high volume of CM objects, and computation-intensive dynamic requests suggest that the service time variance of the requested web objects should be considered in estimating the server capacity and response time.

6.3 Admission Control Algorithm

This section presents how the web server workload characteristics discussed in the previous section impacts the design of our AC algorithm. The goal of the PACERS algorithm is to provide response delay bounds to incoming requests while preserving the system throughput of busy web servers.

6.3.1 Overview of the algorithm

Usually a queue of incoming requests is maintained in a web server awaiting to be processed. Using the *tail-dropping* AC scheme, incoming requests are put in the queue until the queue is full. Queue length is not always a good indicator of system

load, especially when the variance of processing time is high. Without effective AC, the server response time and throughput deteriorate drastically when the aggregate request rate exceeds server capacity, indiscriminately affecting all clients. Abdelzaher and Bhatti [52] reported that as much as half of the web system's processing capacity is wasted on eventually aborted/rejected requests when the load is high.

To accommodate the high variance in request rate and service time of web servers, we propose a simple and adaptive AC algorithm, *Periodic AC based on Estimation of Request rate and Service time* (PACERS), to ensure the availability and delay performance of prioritized tasks. The predictive strategy estimates the periodic request rate of each priority group, and guarantees the performance of higher priority tasks by restricting admission of lower priority tasks. The request rate estimation is based on the history of access pattern. Service time of each task is estimated based on the request types which is more or less delineated by the size of the responses. Inaccurate estimations are dynamically adjusted by a *double-queue* architecture as described later.

6.3.2 Admission Control and Delay Bounds

We first examine a non-prioritized system to have a systematic understanding of the proposed algorithm. Assume that the time is divided into discrete slots $(0, T)$, $(T, 2T)$, ..., $(kT, (k+1)T)$, For simplicity, we use the beginning point of each time period, kT , to represent the duration $(kT, (k+1)T)$. Let c be the unit processing capacity of the system, $C(kT)$ be the predicted system processing capacity in period kT , $S(i, kT)$ be the service time needed by the i th task at period kT , and $n(kT)$ be the number of admitted tasks in period kT . If the server has a precise knowledge of service time needed by each task, the admission decision can be made based on the following expression.

$$C(kT) = c * T \geq \sum_{i=1}^{n(kT)} S(i, kT). \quad (6.1)$$

If expression 6.1 is true, then the request is admitted, otherwise it is rejected. The maximum response delay of each task is bounded by the value of T if the service time of each task is known prior to the admission decision phase.

In a prioritized system, the periodic system capacity seen by different priority groups changes with the prediction of resource requirements of each priority group. By adjusting the assigned system capacity to each priority group, the PACERS algorithm provides service quality assurance to prioritized tasks. There are two kinds of service disciplines that can be provided by the PACERS algorithm: *prioritized resource allocation* (PRA) and *weighted fair allocation* (WFA). PRA is implemented by assigning resources equal to the whole system capacity to the highest priority tasks (or premium tasks). Lower priority tasks get the remaining resources. The WFA is realized by setting shares of system resources in each priority group, where each priority group get at most/least their shares. In this study we only discuss the PRA control scheme. The WFA scheme can be easily extended from the PRA scheme by setting a minimum or maximum resource ratio for each priority group.

The objective of the server is to ensure QoS to high priority tasks whenever their arrival rate is lower than the system capacity. Thus, for high priority tasks, the available resources are equal to the system period capacity. For lower priority tasks, the available resources are the system capacity minus the predicted resource requirements of higher priority requests during a time period. Since the priority ratio and type distribution of incoming tasks vary over time, dynamic assignment of system capacity is needed to preserve the system throughput.

Assume all the requests are classified and are assigned a priority p , ($p = 1, \dots, P$), wherein P denotes the highest priority. Let $\lambda_i^{predicted}$, ($i = 1, \dots, P$) be the predicted inter-arrival rates of tasks for each priority group. The system capacity available to priority group p at time $kT \leq t \leq (k+1)T$, denoted as $C_p(kT, t)$, is:

$$C_p(kT, t) = C(kT) - \sum_{i=p+1}^P \lambda_i^{predicted}(kT) * T - \lambda_p(kT) * t. \quad (6.2)$$

A task with priority p is admitted if the service time is equal or less than available capacity $C_p(kT, t)$.

6.3.3 Estimation of Request Rate

The resources allocated to each priority group is based on the prediction of the request rate of incoming prioritized tasks. Apparent seasonal workload patterns corresponding to daily cycles discussed in the previous section can be used to predict current traffic intensity based on the workload history. On the other hand, reports in [92, 77] suggested that the aggregate web traffic tends to smooth out as Poisson traffic in short observation time windows. This fact was further proved by Morris and Lin in [91]. Based on the above published results, we decide to use Markov-Modulated Poisson Process (MMPP) described in [93] to capture the seasonal behavior of the aggregate workload of web servers, while preserving the tractability of modulated Poisson process. The request arrival process is described as a Markov process $M(i)$ with state space $1, 2, \dots, i, \dots, N$. State i has arrivals complying with Poisson process at rate λ_i . To follow the seasonal behavior, especially the day/night cyclic behavior of the web server load, the observed traffic data is chopped into subsets for each hour on a daily basis. The Markov transition matrix $Q(n) = [Q_{ij}(n)]$, $(n = 1, \dots, 24)$ for each hour can be easily calculated by quantizing the inter-arrival rate in each observation period, and calculating the frequency at which $M(n)$ is switched from state i to state j . The predicted access rate $\lambda^{predicted}(kT)$ can be expressed by the following equation:

$$\lambda^{predicted}(kT) = [\lambda((k-1)T)] \cdot Q(kT), \quad (6.3)$$

where $[\lambda((k-1)T)]$ is the state vector of measured inter-arrival rate in the previous period. $Q(kT)$ can be further adjusted by comparing the differences between predicted data and measured data, to catch up with the long term trends of a web

server load. In the experiment, we use three state $(p_{inc}, p_{same}, p_{dec})$ transition matrices to capture the traffic trends in each observation period. p_{inc} is the probability of increment request rate, p_{same} the probability of the same request rate, and p_{dec} the probability of decrement request rate. The δ value for increment and decrement is 10% of measured value. The experiment shows that there is not much difference in the capability of capturing the traffic trends in each observation period between a three state transition matrix and more complicated state transition matrices.

6.3.4 Estimation of Service Time

While deciding to accept a request, the system should ensure that the sum of service time of accepted tasks do not exceed the system capacity. In reality, it is not possible to know the service time $S(i, kT)$ in advance. High variance in resource requirement is a widely recognized characteristic of web server workload. As indicated in the previous section, however, the service time and bandwidth requirement of the same type of requests are more or less consistent. Service time of web objects can be thus estimated based on the request type distribution of the server access pattern. We approximate the service time of each task by using weighted *computation quantum* (CQ) matching the CPU processing time of different type of tasks.

When a new request arrives, the system checks if there are enough CQ available to grant to the request. Only requests that are granted enough CQ can be enqueued and served eventually. The number of CQ issued to each task is determined by the resource requirement of the request. Here we denote T_n as the number of types of requests to a web server, which can be derived from the workload characterization of a web server.

Let $N_i(kT)$ be the number of requests of type i in period kT , CQ_i be the weighted CQ matching the CPU processing time for type i tasks. Then Equations (6.1) and (6.2) can be

approximated as:

$$C(kT) \geq \sum_{i=1}^{Tn} CQ_i * N_i(kT) \quad (6.4)$$

$$C_p(kT) \geq \sum_{i=1}^{Tn} CQ_i * N_{i,p}(kT). \quad (6.5)$$

As in the case of Equations (6.1) and (6.2), a request is admitted if Equations (5.3) and (6.5) hold true, otherwise is rejected.

6.3.5 The Double Queue Structure

Since Equations (5.3) and (6.5) are approximations of Equations (6.1) and (6.2), care needs to be taken to amortize the accumulate delay influence of over-admission during a time period. Assume that a restricted prioritized processing is enforced inside a queue, i.e., no lower priority tasks get served if there is a higher priority task waiting, incoming requests are queued in the order from high priorities to low priorities. Requests in the same priority are queued in FCFS order. When over-admission happens, it is possible that low priority tasks stay in the queue for a long time awaiting services while high priority tasks get dropped due to lack of queue space. On the other hand, under-admission wastes system capacity. A *double-queue* structure is used to handle the over/under admission problems. A primary queue is used as the incoming task queue, and a secondary queue is added for the backed up requests (we call this as the backup queue). The system structure is shown in Figure 5.3.

An incoming request is first sent to the AC manager ACM. The ACM classifies the request priority and decides if the request can be enqueued. Enqueued requests wait to be served in the primary queue Q_p . At the beginning of the next period, unfinished requests are sent to the backup queue Q_b . When the Q_b becomes full, it is cleared up and queued tasks are dropped. Other methods for expunging tasks from Q_b can be explored. The task scheduler TS picks up requests in the queues

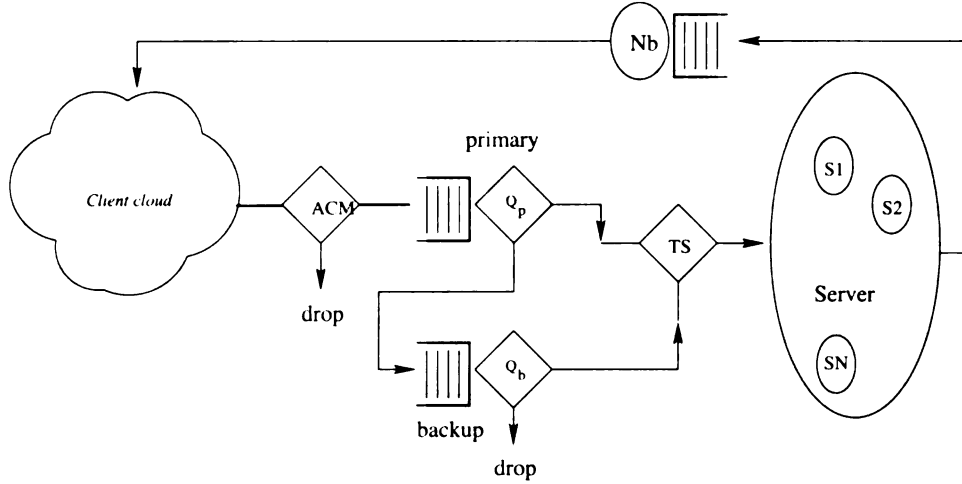


Figure 6.7 Web server system structure.

and sends to the server pool. No request in the Q_b can be picked up unless the Q_p is empty. Replies are sent back through the server network interface N_b . By using a double queue structure, newly accepted requests need not wait for a long time for service, thus bounded delay is achieved for most of the requests.

6.4 Response Delay Analysis

Service differentiation is desired to permit differentiated pricing of Internet services. These characteristics may be specified in quantitative or statistical terms of throughput, delay, and/or drop rate. AC is generally used to regulate the throughput of a specific service group. The PACERS algorithm provides delay bounds and average waiting time assurance as well as throughput regulation.

6.4.1 Ideal Case Delay Bounds

In a stable system, i.e., the server has the capacity to process all the requests in the steady state, the request queue helps in smoothing out the short term variations in the job arrival rate. When the short term arrival rate is faster than the processing rate, newly arrived requests have to wait in the queue for service. If the variations

of job arrival rate and processing rate are high, more queue space is needed, and the average queueing time and response delay tend to be high. If enough buffer space is not available, denial of service occurs when the queue is full.

One way to increase the response time predictability of a system is to limit the number of requests in a given “short time” period T to no more than what the system can handle in that period of time. Borowsky [85] provided a theoretical proof that the service time required by pending jobs is no more than T in a FCFS work-conserving system, if the sum of service time required by the arriving jobs at any duration T is bounded by time T . Thus the response time (or delay) of all requests can be bounded by T by restricting the workload arriving in every interval of length T time.

In a prioritized environment where each request is assigned a specific priority, the processing order and resource allocation are based on the priority of each request. The FCFS processing assumption in the above theorem is no longer valid. In [5], we proved that the mean response time of a request is bounded by the arrival rate with equal or higher priority and the service rate of the system, if requests with different priorities have the same service requirement distribution and a strict priority scheduling is used.

Let $p(1 \leq p \leq P)$ denote the priority of the requests and P be the total number of priorities. Let $s_p(i)$ be the service time requirement of task i belonging to priority p , and $N_p(t - T, t)$ be the number of requests of priority p arriving during the period T .

Lemma : In a prioritized preemptive work-conserving environment with no pending work, if $N(t - T, t)$ requests arrive belonging to a single priority level p , then their response time is bounded by $T_p(T_p < T)$ if $\sum_{i=1}^{N(t-T, t)} s_p(i) \leq T_p$ (proved in [85]).

Theorem : If the priority levels are in the increasing order of p , the response time of a task with priority p is bounded by $\sum_{q=p}^P T_q$. If $\sum_{q=1}^P T_q \leq T$, the response time of any task is bounded by T .

Proof : Let $R_{p,i}(n)$ be the response time of the i th task in the priority class p during period n , and $busy_time(n)$ be the amount of time in (n) that the system is

busy during period n . Let the request time series be partitioned into periods of n with length T .

For $n = 1, i.e., t = (0, T)$, without loss of generality, let all requests arriving during $(0, T)$ be reshuffled to be served in the decreasing order of p (for $p = 1, \dots, P$), then $R_{p,i}(1) \leq T_P$, for $i = 1, \dots, N_P(1)$, and $R_{p,i}(1) \leq \sum_{q=p+1}^P T_q + T_p \leq \sum_{q=p}^P T_q$, for $p = 1, \dots, P$.

Assume the above expressions hold true in the period $k - 1$, reshuffle the requests in queue in the decreasing order of p during period k . Let $Q_p(k)$ be the sum of the service times for the p th priority tasks, then

$$Q_p(k) = Q_p(k-1) + \sum_{q=p}^P \sum_{i=1}^{N_q(k-1)} s_q(i) - busy_time(k), \quad (6.6)$$

Clearly, $Q_p(k-1) \leq busy_time(k)$. By assumption, $\sum_{q=p}^P \sum_{i=1}^{N_q(k-1)} s_q(i) \leq \sum_{q=p}^P T_q$, so

$$Q_p(k) \leq \sum_{q=p}^P \sum_{i=1}^{N_q(k-1)} s_q(i) \leq \sum_{q=p}^P T_q. \quad (6.7)$$

Hence, by induction and from Equation (6.7), we get $R_{p,i}(k) \leq \sum_{q=p}^P T_q$ ($i = 1, \dots, N_p(k)$), for $p = 1, \dots, P$, and $k = 1, 2, \dots$, i.e., the response delay boundary of each prioritized task can be achieved by controlling the requested service time in each period of length T . As long as the sum of service times requested by high priority tasks does not exceed T , the system response time can be assured by restricting the number of admitted low priority tasks.

The granularity of T effects the system performance. Increasing the value of T smoothes out the access variance between adjacent periods and allows more requests to be admitted. However, larger value of T also increases the variance of response time and degrades the system responsiveness. Both user perceivable delay and system throughput should be considered in determining an appropriate value of T .

6.4.2 Waiting Time Estimation

The service time of a web object can be estimated from statistics of the same or similar type of web object with acceptable variance. The number of requests allowed by the system can be derived from the expected waiting time and the job arrival rate. Let $W_p(t)$ denote the expected waiting time of tasks with priority p at time t , $A_p(t)$ be the expected arrival rate of tasks with priority p at time t . Let $W(t)$ denote the expected waiting time of a task in the system at time t , $A(t)$ be the expected arrival rate of tasks of the system at time t . In the period k , the expected number of tasks in the system equals:

$$N(k) = W(k) * A(k) = \sum_{p=1}^P A_p(k) W_p(k). \quad (6.8)$$

According to Little's Law [70], the waiting times of each priority group equals:

$$W_p(k) = \frac{N(k) - \sum_{i=p+1}^P A_i(k) W_i(k)}{A_p(k)}. \quad (6.9)$$

On the other hand, the waiting time of a task in the p th priority group equals the residual life of the executing task W_0 plus the sum of service times of equal or higher priority tasks in the queue, and the sum of service times of higher priority tasks arriving while it waits in the queue. The mean service time of group i in period k is represented as $S_i(k)$. The waiting time is thus equal to :

$$W_p(k) = W_0 + \sum_{i=p}^P N_i(k) + \sum_{i=p+1}^P W_p(k) A_i(k) S_i(k). \quad (6.10)$$

Let $\rho_i(k) = A_i(k) S_i(k)$. By combining results of Equations (6.9) and (6.10) we can get,

$$W_p(k) = \frac{W_0}{(1 - \sum_{i=p}^P \rho_i(k))(1 - \sum_{i=p+1}^P \rho_i(k))}$$

$$\leq \frac{1}{c(1 - \sum_{i=p}^P \rho_i(k))(1 - \sum_{i=p+1}^P \rho_i(k))}. \quad (6.11)$$

c is the unit processing capacity of the system, its inverse value is the worst case expected residual life of an executing task, which happens when the utilization factor approaches to 1. Equation (6.11) shows the mean waiting time of prioritized tasks by using the estimation of the inter-arrival rate of equal or higher priority tasks.

Similarly, we can get the expected task inter-arrival rate and acceptable number of tasks in each period based on the expected waiting time. The result is shown as:

$$N_p(k) = A_p(k) * T = \frac{W_0 - W_p(k)(1 - \sum_{i=p+1}^P \rho_i(k))^2}{1 - W_p(k)S_p(k) \sum_{i=p+1}^P \rho_i(k)} T. \quad (6.12)$$

Note that the above analysis are based on the periodic data collection assumption. In fact, all the data used in the above equations can be easily obtained from a few counters which are reset periodically. The workload of each priority level is estimated at the end of each period. Expected number of tasks is determined based on the expected inter-arrival rate. During one period, if the number of incoming tasks of one priority level exceeds the expected number, then there is no need to accept new requests in the same priority level until the beginning of next period.

6.5 Simulation Model and Parameters

In this study, we simulate an event driven server model using empirical workload from real trace files. The reference locality, size and object type distribution are extracted from the logs of the Computer Science departmental web server at Michigan State University and are used for the workload generation. We only consider two priority levels to simplify the study since the primary concern is to examine the

Table 6.2 Simulation Configuration.

Parameter	value
Priority Level	2
Scheduling Period	1 sec.
Static Obj. Throughput	1000 req./sec
Dynamic Obj. Throughput	100 req./sec
Network Bandwidth	100 Mbps
Disk Bandwidth	100 Mbps
Caching hit ratio	0.7
Maximum open connections	1000
Maximum queue length	1000
Maximum server process number	30
Response delay bound	1 sec.

effectiveness of the PACERS algorithm. The system configuration is shown in Table 6.2.

Three performance metrics are used in the simulation: server throughput, mean response time, and response delay bound miss rate of each priority group. System throughput indicates the server capability. Mean response time and delay bound miss rate quantify the service qualities in each priority group. The delay bound miss rate presents the proportion of tasks whose response time exceed the bounded delay.

Based on the data reported in Section 6.2, web objects are classified into 4 types: static, dynamic, audio and video objects. The CQs consumed by each object types and their input percentage are listed in Table 6.3. The CQs allocated to each object type is based on the results in Section 5.3.2.

Table 6.3 CQs requested by each object types.

Object Types	Static	Dynamic	Audio	Video
Credits	1	10	20	100
Request Freq.	94.7	5	0.2	0.1

6.6 Results and Evaluation

Our experiments evaluate the performance of the PACERS algorithm as opposed to a simple admission control (SAC) algorithm, which is analogous to the leaky bucket [88] algorithm used in traffic engineering in network transmissions. The SAC algorithm produces CQ at a constant rate matching the server throughput; each admitted task consumes one CQ. Incoming tasks are dropped if there is no CQ available. However, estimation of the request rates is not implemented in the SAC algorithm. The SAC scheme outperforms the *tail-drop* scheme, since it smoothes out the web server traffic and also provides preliminary overload protection.

In the experiment, high priorities are assigned to 50% of the incoming requests randomly, and the remaining requests are marked as low priority. Thus the ratio of high priority to low priority tasks is 1 to 1, and both types of tasks are randomly distributed in the whole arrival sequence. The web server trace in Section 6.2 is used to generate request at various rate and the performance is recorded from the simulated server environment.

6.6.1 Throughput Performance

Figures 6.8 and 6.9 plot the throughput of stress test of two priority groups. Normalized load in the x-axis is the ratio of real load to the system capacity, and the normalized throughput is ratio of the throughput to the system capacity. In the stress test, the aggregate request rate increases continuously till 2.5 times of the system capacity. The objective of this experiment is to explore the capability of the algorithm in preserving the system throughput and delay performance under extremely high load by the stress test.

Figure 6.8 shows the throughput variation using the SAC algorithm. It can be observed that the throughput of each priority group is proportional to the traffic percentage of each group. High priority tasks suffer from the same low throughput

as low priority tasks when the aggregate workload is high. Predefined QoS profile strictly prioritized admission and scheduling, are not ensured.

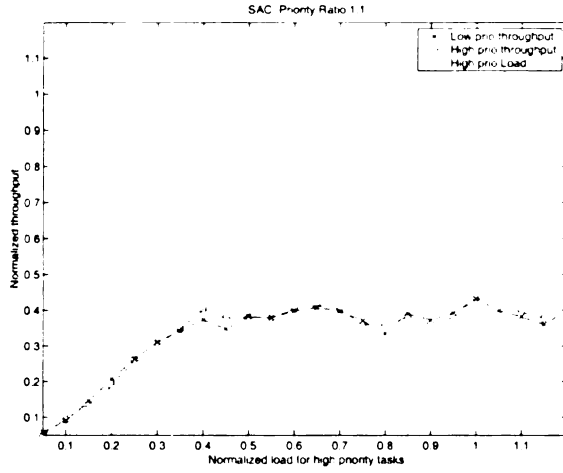


Figure 6.8 Throughput using SAC.

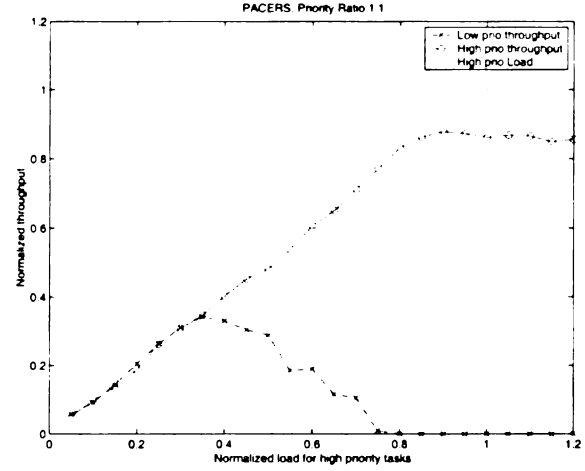


Figure 6.9 Throughput using PACERS.

Figure 6.9 shows the throughput of each priority group when the PACERS scheme is deployed. The system stops serving low priority tasks when the high priority load approaches about 80% of system capacity. On the other hand, the throughput of high priority tasks equals the request rate of high priority group until the system utilization reaches 0.8, which is the highest throughput of the system using the traces as input workload. The throughput of high priority tasks remain at the 80% level when the high priority tasks overload the system. It can also be observed that the aggregate throughput of the two AC schemes remains at the same level. It suggested that the prediction and reservation behaviors of the PACERS scheme do not degrade the system aggregate throughput.

6.6.2 Delay Performance

The average delay of each priority group is also well controlled under the PACERS scheme. As shown in Figures 6.10 and 6.11, the average delay of each priority group

is much lower than the predefined delay bounds of 1 second by using the PACERS algorithm.

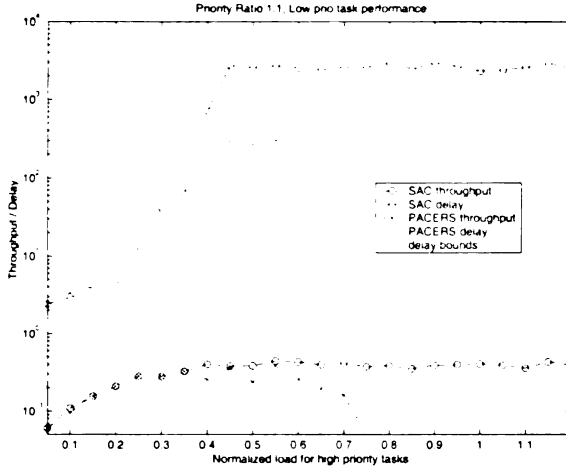


Figure 6.10 Performance comparison of low priority tasks.

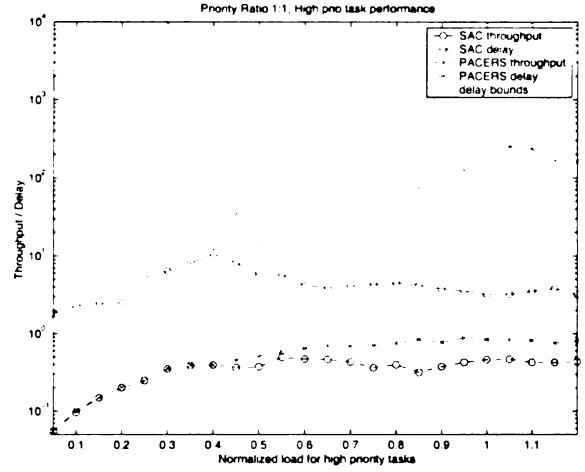


Figure 6.11 Performance comparison of high priority tasks.

Figure 6.10 plots the low priority task delay and throughput performance of the two AC algorithms. Using the SAC algorithm, low priority task throughput is proportional to the ratio of low priority traffic. However, extremely long delays are introduced when prioritized processing is used. On the contrary, the PACERS algorithm blocks admission of low priority tasks during high load periods in exchange of low delay bounds miss ratio. The prediction and CQ assignment behavior of the PACERS scheme avoids unnecessary waiting of low priority tasks, thus decrease the system overload during high load periods.

Figure 6.11 plots the high priority task delay and throughput performance of the two AC algorithms. Using the SAC algorithm, high priority task throughput is proportional to the ratio of high priority traffic to the total traffic volume, although the high priority task traffic is low. Delays of high priority tasks are kept low because prioritized processing is used. On the contrary, the PACERS algorithm preserves throughput performance of high priority tasks during high load periods with the cost of higher average response delay, but still within the delay bounds.

As we can see from the workload analysis, the traffic intensity varies with time periods. It is reasonable to believe that traffic intensity of each priority group also varies with time in service differentiating environments. Request rate estimation helps to preserve system resources for high priority tasks and reduce wastage of system resources. We test the performance of the PACERS algorithm under fluctuating load situation.

6.6.3 Sensitivity Test

Figures 6.12, 6.13 and 6.14 plot throughput and delay performance under fluctuating workload. The experiment is aimed at examining the sensitivity of the PACERS algorithm to the variation in the workload. The test set can be phased as sustained lightload or overload, and occasional lightload or overload. The occasional overload duration is 2 time units, and the sustained overload duration is 10 time units. Each time unit is 1000 times of the observation period.

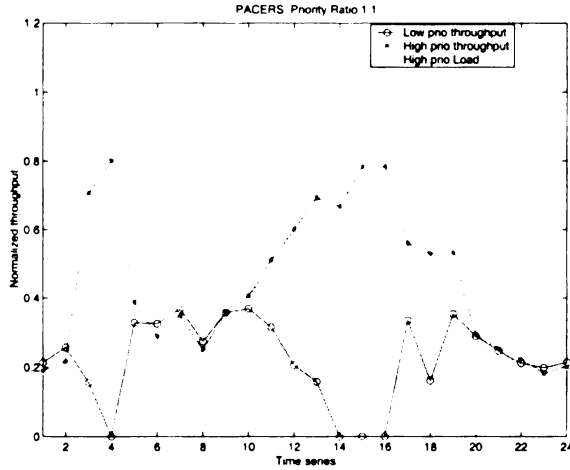


Figure 6.12 Throughput under fluctuating load.

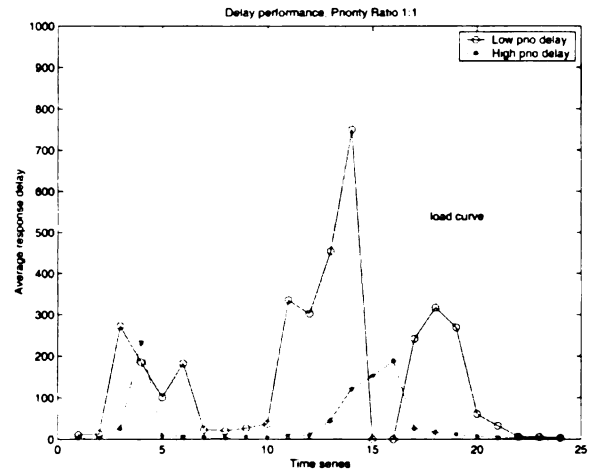


Figure 6.13 Delay under fluctuating load.

The maximum resource requirements of high priority requests equal the system capacity. It can be observed from Figure 6.12 that the PACERS scheme is very sensitive to the load fluctuation, which blocks the low priority tasks under high load

situation to preserve the throughput of high priority tasks, and resume service of low priority tasks during medium to low load situation. Figure 6.13 shows the delay performance of the two priority groups. The delay of high priority tasks follow the trends of incoming high priority workload, but the maximum value is only about one fifth of the delay bounds. The delay of low priority tasks is also well controlled under the delay bound. Figure 6.14 shows the delay bounds miss ratio of the two priority groups. High priority tasks experience zero delay bounds miss ratio under various load situations; delay bound miss ratio of low priority tasks occasionally exceeds zero.

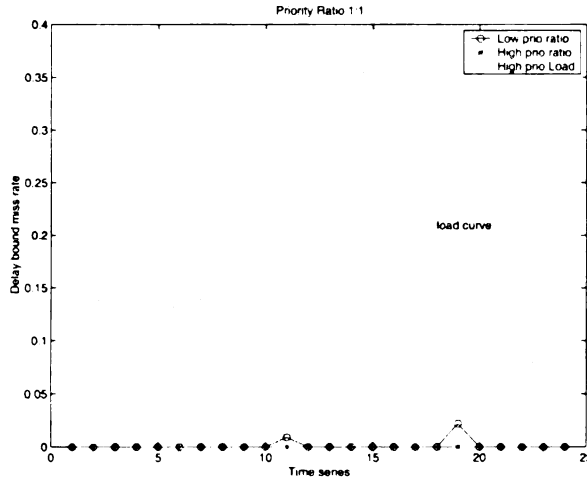


Figure 6.14 Delay bound miss ratio under fluctuating load.

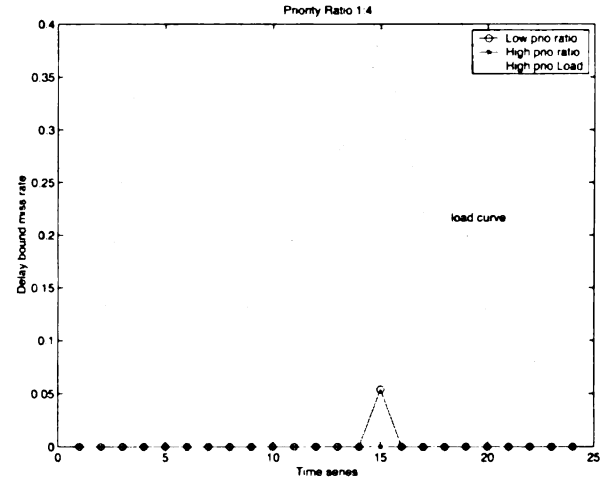


Figure 6.15 Delay bounds miss ratio under fluctuating load.

6.6.3.1 Performance Isolation of Priority Groups

Figures 6.15, 6.16, and 6.17 show the experiment results where high priority tasks traffic ratio is 20%. The objective of this experiment is to examine the performance impact when large quantity of low priority tasks is presented.

It can be observed that the throughput and delay performance of high priority tasks remain the same with or without high volume of low priority traffic. The experiment proves that a large amount of low priority traffic does not have any impact on

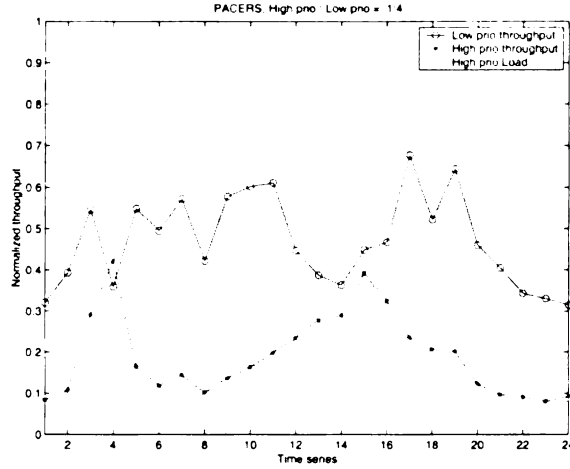


Figure 6.16 High Priority throughput high background traffic.

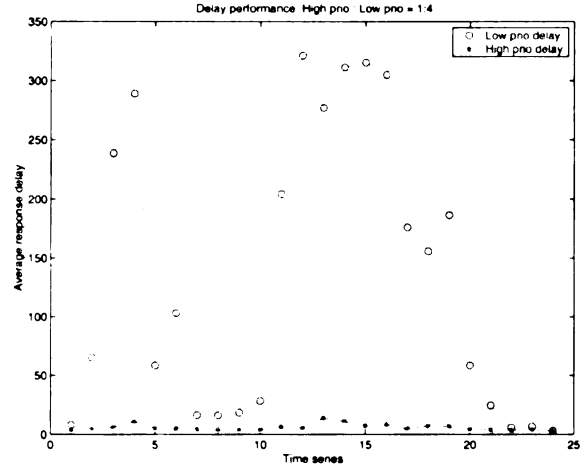


Figure 6.17 High Priority delay high background traffic.

the throughput or average delay performance of high priority tasks. Thus the PACERS scheme offers effective performance protection to high priority requests under various load situations.

6.7 Related Works

Several studies on QoS support in web servers have addressed the technology of prioritized task processing and admission control issues. Bhatti and Friedrich [3] addressed the importance of server QoS mechanisms to support tiered service levels and overload management. A Web-QoS architecture prototype was developed by adding connection manager module to the Apache [49] Web server. Admission control is implemented by blocking low priority tasks when the high priority waiting task number exceeds the threshold. Eggert and Heidemann [6] evaluated application level mechanisms to provide two different levels of web services. The admission control is implemented by limiting process pool size and response transmission rate to different priority groups. Pandey et. al. [48] described a distributed HTTP server which enables QoS by prioritizing pages on a web site. The admission control is realized by

assigning communication channel to prioritized pages. All of the admission control mechanisms mentioned above are based on a predefined “threshold”. Performance of high priority tasks is guaranteed by emulation of a fixed bandwidth “leased line”. However, it is expensive to satisfy the requirements of burst workload by “leased line” scheme, since peak loads are several orders of magnitude higher than average load.

6.8 Summary

When a server is unable to provide satisfactory service to all requests, selective resource allocation is a promising technique to assure service to requests which are more important to clients or servers. In this study, we present a novel and effective admission control algorithm, PACERS, to adapt to the highly variant access patterns and processing of web server environments. Tasks are admitted based on the estimation of periodical behavior of prioritized task groups and service times. The performance of high priority tasks are preserved by blocking the traffic of low priority tasks when the system load is high. Delay of most tasks are bounded by the algorithm processing period.

Theoretical proof and simulation results demonstrate that the PACERS algorithm is able to provide assurance of response time and throughput performance for each priority group under various workload situations. Its aggregate throughput performance is similar to the SAC algorithm, while the peak high priority task throughput is much higher than the SAC algorithm. The average delay is about 10 times lower than the predefined delay bound, and the delay bound miss ratio is 0 during most of the time periods. Compared to “threshold” based admission control such as *tail-dropping*, the PACERS algorithm adjusts to the burst in the workload dynamically, increasing throughput during low load periods while ensuring QoS during high load periods.

The algorithm can be expanded for web server session control if the session/connection establishment requests are assigned lower priority to protect existing sessions in the server. The estimation of the average response time of the server system can be used to determine the appropriate queue space, thus can be used for capacity planning.

CHAPTER 7 CONCLUSIONS AND FUTURE WORK

The major contribution of this dissertation can be summarized in the answers to the following questions:

Why service differentiation is needed in Internet servers? This question can be further divided into two sub-questions. First, why choose Internet servers as the research subject? The answer is that the Internet servers are becoming the performance bottlenecks of Internet services with increasing service volume and diversity of applications. Second, why service differentiation is needed? The reason is that the widespread of e-commerce activities in the Internet environment demands significant amount of dynamic data generation and secure transactions. To design a uniform high performance server to meet customer expectation at all times is impractical and cost-ineffective. The service differentiating Internet servers (SDIS) is designed to meet customer expectations for evolving diverse applications, such as CM data with time constraints that need to be met, and e-commerce transactions that need classified services based on revenue generation. The quality of services of SDIS are enforced by categorizing requests into different groups based on task types and client identifications, then allocating resources based on the priority groups.

What is the appropriate model for a SDIS? After extensive literature review and analysis of existing web server systems, including Apache web server and some popular load balancers used in Internet services, a queueing model which allows various resource allocation algorithms is designed to simulate the web server processing procedure. Modular framework of the web server model allows modeling of server

processing with various architectures. CPU processing capacity, network bandwidth, open connections and memory capacity are critical resources in the server model. Task classification policies, methods of task differentiation, and desired QoS in each class of tasks are discussed. An incoming request can be assigned to a priority group on the basis of the *client*, *network*, *content*, or *owner* of the requested object, and current server load situations. It can also inherit a performance level defined by low level network protocols.

Can desired performance being achieved in SDIS? To answer this question, we appraise the performance of a SDIS using trace input from real web servers. Experimental study and analyses prove that under high system utilization, a service differentiating server provides significantly better services to high priority tasks at minimal cost to low priority tasks. Through analytical modeling and simulation study, we show the feasibility and performance benefits of the SDIS. Admission control, prioritized scheduling, and task assignment policies are evaluated with respect to the performance assurances of different groups. A few limitations, such as overload effects of a web server and starvation problem of low priority tasks are also discussed in this dissertation.

How to maintain stable performance of a web server? The most important performance metrics of a web server is the response delay and throughput. Experimental results indicate that the response delay increases slowly with server load until the server load achieves the “overload threshold”. The response delay of an overloaded server is several orders of magnitude higher than normal value thus unacceptable to end users. Throughput of an overloaded server also decreases due to “snowball” effects and resources contention overhead. An admission control scheme, ACES, is designed to assure the user perceivable response performance and server throughput. Experimental evaluation proves the effectiveness of the ACES algorithm.

How to enforce quality of services in SDIS? Due to the diversity of request rates and applications in web servers, static resource allocation mechanisms fail to enforce

QoS of classified tasks with the dynamics of the server environment. Two characteristics of the server workload need to be handled in resource allocation: the cyclic fluctuation of traffic intensity, big variance of task processing time and relatively consistent service time of the same type of web objects. We propose to provide different level of services based on the server workload characteristics. Different levels of QoS are assured by periodical allocation of system resources based on the estimation of request rate and service requirements of prioritized tasks. Admission of low priority tasks is restricted or blocked during high load periods to maintain the throughput of high priority tasks and response delay boundary of all the accepted tasks. Theoretical analysis proves the feasibility of the algorithm. Experimental study shows that effective control of throughput and response delay boundary to the prioritized tasks is achieved under various workload situations.

While answering these questions, we present a framework of a service differentiating Internet server in this dissertation. SDIS is not simply another type of web server, it has much deeper and broader influence in the future of the Internet applications. Traditionally, the content published in the Internet is available for “free”, and the content providers have no obligation to ensure that the user retrieval process is a pleasant experience. However, the introduction of SDIS changes the egalitarian philosophy of the Internet and its servers. Alternative levels of service quality become available at certain costs. Users have more choices on how services are provided, and the service and content providers can increase their margin profit by allocating resources to value added transactions.

Today’s Internet suffers from its own success. Technology designed for a network of thousands of computers is laboring to serve a network with millions of hosts. The Next generation Internet (NGI) program was initiated to offer reliable, affordable, and secure information delivery at rates thousands of times faster than that of today. In the near future, the Internet will provide a powerful and flexible environment for

business, education, culture, and entertainment. People will use this environment to study, work, shop, entertain, and manage finances. The customer will be able to choose among different levels of services with varying prices.

One goal of the NGI program is to develop and test new network services and techniques. These will include advances in quality of service provisioning. Several techniques, in bringing in QoS into the network, have already existed as individual components for a few years, such as class of services (COS) in ATM networks, RSVP based integrated service framework, and PHB based differentiated service on the Internet. However, the application layer QoS is still a new field that needs to be explored. Substantial system integrations across layers are required for them to provide seamless support for QoS aware applications. The study of SDIS and any layer QoS collaboration strongly supports the objective of the NGI advances.

In this dissertation, we have built a general simulation model of SDIS and evaluated its performance under a set of admission control and scheduling schemes. There is still lots of work that needs to be done to complete the building of a framework of SDIS. When service differentiation is implemented in a real system, lots of factors need to be reconsidered, such as context switching overhead and policies, memory allocation, and influence from some optimization of the operating system. A prototype of SDIS will be implemented to compare the results among real system, simulation and theoretical derivation. As we stated earlier, to effectively control the resources allocated to tasks in a server, accurate estimation of traffic trends and service requirements are needed. More extensive workload characterization work should be conducted. For example, the service time and response time distribution versus server load and connection number would provide adequate guidelines for efficient admission and overload control algorithms. Study of QoS aware server management issues will be also investigated in future.

Bibliography

- [1] E. J. W. West, "Using the internet for business - web oriented routes to market and existing it infrastructures," *Computer Networks and ISDN Systems*, vol. 29, pp. 1769 – 1776, July 1997.
- [2] T. Wilson, "E-biz bucks lost under ssl strain," *Internet Week Online*, May 20 1999. <http://www.internetwk.com/lead/lead052099.htm>.
- [3] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, pp. 64–71, September/October 1999.
- [4] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing differentiated quality-of-service in web hosting services," in *1998 Workshop on Internet Server Performance*, June 1998. <http://www.cs.wisc.edu/~cao/publications.html>.
- [5] X. Chen and P. Mohapatra, "Providing differentiated service from an internet server," in *proceedings of IEEE Internet Conference on Computer Communications and Networks*, (Boston, MA), October 1999. <http://www.cse.msu.edu/rgroups/isal/pubs/conf/>.
- [6] L. Eggert and J. Heidemann, "Application-level differentiated services for web servers," *In World Wide Web Journal*, vol. 3, no. 2, pp. 133–142, 1999. <http://www.isi.edu/~larse/papers/index.html>.
- [7] *Integrated Services (intserv)*. <http://www.ietf.org/html.charters/intserv-charter.html>.
- [8] <http://www.ietf.org/html.charters/diffserv-charter.html>.
- [9] X. Xiao and L. M. Ni, "Internet qos : the big picture," *IEEE Network Magazine*, pp. 8–18, March/April 1999.
- [10] K. G. Coffman and A. M. Odlyzko, "The size and growth rate of the internet," *First Monday*, 1998. <http://www.firstmonday.dk/issues/issue3.10/coffman/index.html>.
- [11] R. H. Zakon, "Hobbes' internet timeline v3.3," 1998. <http://info.isoc.org/guest/zakon/Internet/History/HIT.html>.
- [12] <http://cnn.com/TECH/computing/9905/24/sitefall.idg/>.
- [13] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," in *ACM SIGCOMM Computer Communication Review*, July 1996. <http://http.cs.berkeley.edu/~padmanab/index.html#Publications>.
- [14] A. K. Iyengar, E. MacNair, and T. Nguyen, "An analysis web server performance," in *Proceedings of the IEEE 1997 Global Telecommunications Conference (GLOBECOM '97)*, (Phoenix, AZ), November 1997.

- [15] S. Glassman, "A caching relay for the world wide web," in *Proceedings of the First International Conference on the World Wide Web*, (Geneva, Switzerland), May 1994. <http://www1.cern.ch/PapersWWW94/steveg.ps>.
- [16] P. Cao and S. Irani, "Cost-aware www proxy caching algorithms," in *Proceedings of the 2nd Web Caching Workshop*, (Boulder, Colorado), June 1997.
- [17] M. A. R. Wooster, "Proxy caching that estimates page load delays," in *Proceedings of the Sixth International World Wide Web Conference*, (Santa Clara, CA), April 1997.
- [18] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell, "A hierarchical internet object cache," in *Proceedings of the 1996 USENIX Technical Conference*, (San Diego, CA), January 1996. <http://netweb.usc.edu/danzig/>.
- [19] A. Bestavros and et. al., "Application level document caching in the internet," in *Proceedings of IEEE SDNE'96, The Second International Workshop on Services in Distributed and Networked Environments*, (Whistler, British Columbia), June 1995.
- [20] A. Bestavros, "Using speculation to reduce server load and service time on the www," Tech. Rep. CS Tech Report BUCS-TR-95-006, Boston University, February 1995.
- [21] E. Markatos and C. E. Chronaki, "A top-10 approach to prefetching on the web," Tech. Rep. 173, ICS-FORTH, Heraklion, Crete, Greece, August 1996. <http://www.ccsf.caltech.edu/~markatos/avg/www.html>.
- [22] R. Tewari, H. Vin, and et. al., "Resource based caching for web servers," in *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN98)*, (San Jose, CA), 1998.
- [23] A. Reddy, "Evaluation of caching strategies for an internet server," in *Proceedings of the International Conference in Multimedia Computing and Systems*, (Ottawa, Ontario, Canada), June 1997.
- [24] E. Markatos, "Main memory caching of web documents," in *FProceedings of the Fifth International World Wide Web Conference*, (Paris, France), May 1996.
- [25] M. Kurcewicz, W. Sylwestrzak, and A. Wierzbicki, "A filtering algorithm for proxy caches," in *Proceedings of the 3rd International WWW Caching Workshop*, (Manchester, England), TERENA, Trans-European Research and Education Networking Association, June 1998.
- [26] M. Baentsch and et. al., "Introducing application-level replication and naming into today's web," in *Proceedings of the Fifth International World Wide Web Conference*, (Paris, France), May 1996.
- [27] A. Bestavros and C. Cunha, "Server-initiated document dissemination for the www," *IEEE Data Engineering Bulletin*, vol. 19, pp. 3–11, September 1996. <http://www.cs.bu.edu/faculty/best/res/papers/HomeDIS.html>.
- [28] <http://www.specbench.org/osg/web99/results/res99q4/>.
- [29] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, and C. Lilley, "Network performance effects of http/1.1, css1, and png," in *W3C, SIGCOMM'97*, 1997. <http://www.w3.org/People/Frystyk/#Refereed>.
- [30] <http://www.cs.wustl.edu/~jxh/research/research.html>.

- [31] G. Venkitachalam and T. cker Chiueh, "High performance common gateway interface invocation," in *Proceedings of the IEEE Workshop on Internet Applications (WIAPP) '99*, (San Jose, CA), July 1999.
- [32] A. Iyengar and J. Challenger, "Improving web server performance by caching dynamic data," in *USENIX Symposium on Internet Technologies and Systems*, (Monterey, CA), December 1997. <http://www.usenix.org/publications/library/proceedings/usits97/iyengar.html>.
- [33] J. C. Collins, "Beyond cgi: Using the apis," *PC Magazine*, April 1998. <http://www.zdnet.com/devhead/stories/articles/0,4413,1600169,00.html>.
- [34] T. Kwan, R. McGrath, and D. Reed, "Ncsa's world wide web server: Design and performance," *IEEE Computer Magazine*, pp. 68–74, November 1995.
- [35] G. Alwang, "Web servers benchmark tests," *PC Magazine*, May 1998. <http://www.zdnet.com/pcmag/features/webserver98/bench.html>.
- [36] C. Yoshikawa, B. Chun, P. Eastham, A. Vahdat, T. Anderson, and D. Culler, "Using smart clients to build scalable services," in *Proceedings of the 1997 USENIX Annual Technical Conference*, (Anaheim, California), 1997.
- [37] D. Andresen, T. Yang, and O. Ibarra, "Towards a scalable distributed www server on networked workstations," *The Journal of Parallel and Distributed Computing (JP DC)*, 1996. <http://alexandria.sdc.ucsb.edu/public-documents/annual-report/node52.html>.
- [38] A. Mourad and H. Liu, "Scalable web server architectures," in *Proceedings of the IEEE Symposium on Computers and Communications (ISCC '97)*, (Alexandria, Egypt), July 1997. <http://www.computer.org/conferen/proceed/7852abs.htm#E37E7>.
- [39] E. D. Katz, M. Butler, and R. McGrath, "A scalable http server: The ncsa prototype," in *First International World Wide Web Conference*, (Geneva, Switzerland), May 1994.
- [40] <http://traianus.ce.uniroma2.it/dws/>.
- [41] V. Cardellini, M. Colajanni, and P. S. Yu, "Dns dispatching algorithms with state estimators for scalable web-server clusters," *World Wide Web*, vol. 2, pp. 101–113, July 1999. <http://traianus.ce.uniroma2.it/dws/WWW99.ps>.
- [42] O. P. Daman, P. E. Chung, Y. Huang, C. Kintala, and Y.-M. Wang, "One-ip: Techniques for hosting a service on a cluster of machines," in *The sixth World Wide Web Conference*, 1996. <http://www.bell-labs.com/user/emerald/paper/oneip/oneip.html>.
- [43] D. Dias, W. Kish, R. Mukherjee, and R. Tewari, "A scalable and highly available server," in *Proceedings of the IEEE International Computer Conference, (COMP-CON'96)*, February 1996. <http://www.research.ibm.com/webvideo/pub.html>.
- [44] A. Iyengar, E. Levy, J. Song, and D. Dias, "Design and performance of a web server accelerator," in *Proceedings of IEEE INFOCOM'99*, (New York), March 1999. <http://www.research.ibm.com/people/i/iyengar/arun2.html>.
- [45] J. Hu, S. Mungee, and D. Schmidt, "Techniques for developing and measuring high-performance web servers over atm networks," in *INFOCOM'98*, 1998. <http://www.cs.wustl.edu/~jxh/research/research.html>.
- [46] R. Braden, L. Zhang, and S. B. et. al., "Resource reservation protocol (rsvp) – version 1." RFC 2205, Proposed Standard, September 1997. <http://www.isi.edu/div7/rsvp/pub.html>.

- [47] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," December 1998. RFC 2475.
- [48] R. Pandey, J. F. Barnes, and R. Olsson, "Supporting Quality Of Service in HTTP Servers," in *Proceedings of the Seventeenth Annual SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, (Puerto Vallarta, Mexico), pp. 247–256, ACM, June 1998.
- [49] "Apache server project." <http://www.apache.org>.
- [50] <http://www.internetsolutions.enterprise.hp.com/webqos/products/overview/index.html>.
- [51] S. Chandra, C. Ellis, and A. Vahdat, "Differentiated multimedia web services using quality aware transcoding," in *Proceedings of the IEEE Infocom 2000 Conference*, (Tel-Aviv, Israel), March 2000. <http://www.cs.duke.edu/surendar/infocom00.pdf>.
- [52] T. F. Abdelzaher and N. Bhatti, "Web server qos management by adaptive content delivery," in *IEEE Infocom 2000*, 2000. <http://www.ieee-infocom.org/2000/papers>.
- [53] S. Ayandeh, A. Krishnamu, and A. Malis, "Mapping to atm classes of service for differentiated services architecture." draft-ayandeh-diffserv-atm-00.txt, November 1999. <http://search.ietf.org/internet-drafts/draft-ayandeh-diffserv-atm-00.txt>.
- [54] J. Pitkow and M. Recker, "Summary of www characterization," *Computer Networks and ISDN Systems*, vol. 30, pp. 551–558, 1998.
- [55] J. Gwertzman and M. Seltzer, "World-wide web cache consistency," in *Proceedings of the 1996 Usenix Technical Conference*, (San Diego, CA), January 1996. <http://www.eecs.harvard.edu/~vino/web/usenix.196/>.
- [56] F. Dougli, A. Feldmann, and et. al., "Rate of change and other metrics: A live study of the world wide web," in *Proceedings of the USENIX Symp. on Internet Technologies and Systems*, December 1997.
- [57] <http://ita.ee.lbl.gov/html/traces.html>.
- [58] M. F. Arlitt and C. L. Williamson, "Internet web servers: Workload characterization and performance implications," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 631–645, October 1997.
- [59] <http://www.nlanr.net/NA/Learn/popular.html>.
- [60] S. Williams, M. Abrams, et al., "Removal policies in network caches for world-wide web documents," in *ACM SIGCOMM*, (Stanford, CA), pp. 293–305, August 1996. <http://www.cs.vt.edu/~chitra/pubs.html>.
- [61] T. T. Kwan, R. E. McGrath, and D. A. Reed, "User access patterns to ncsa's world wide web server," CS Tech Report UIUCDCS-R-95-1934, University of Illinois at Urbana-Champaign, February 1995. ftp://ftp.cs.uiuc.edu/pub/dept/tech_reports/1995/.
- [62] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of www client-based traces," CS Tech Report BU-CS-95-010, Boston University, April 1995. <http://www.cs.bu.edu/techreports/>.
- [63] A. Bestavros, "Demand-based document dissemination to reduce traffic and balance load in distributed information systems," in *Proceedings of the 1995 Seventh IEEE Symposium on Parallel and Distributed Processing*, (San Antonio, Texas), October 1995. <http://www.cs.bu.edu/groups/oceans/papers/Home.html>.

- [64] J. Pitkow and M. Recker, "Simple yet robust caching algorithm based on dynamic access patterns," in *Second International World Wide Web Conference*, (Chicago, IL), October 1994.
- [65] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing differentiated quality-of-service in web hosting services," in *1998 Workshop on Internet Server Performance*, June 1998.
<http://www.cs.wisc.edu/~cao/publications.html>.
- [66] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 835–846, December 1997.
- [67] L. P. Slothouber, "A model of web server performance," in *the Fifth International World Wide Web Conference*, (Paris, France), May 1996.
http://www.inria.fr/mistral/personnel/Zhen.Liu/web_lit.html.
- [68] P. Barford and M. E. Crovella, "Generating representative web workloads for network and server performance evaluation," in *Proceedings of Performance'98/ACM SIGMETRICS'98*, (Madison WI), pp. 151–160, July 1998.
<http://www.cs.bu.edu/faculty/crovella/papers.html>.
- [69] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.
<http://www.aciri.org/floyd/papers/red/red.html>.
- [70] L. Kleinrock, *Queueing Systems*. John Wiley & Sons, 1976.
- [71] H. Schwetman, "Object-oriented simulation modeling with c++/csim," in *Proceedings of 1995 Winter Simulation Conference*, (Washington, D.C.), pp. 529–533, 1995.
- [72] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing reference locality in the www," in *In Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems (PDIS 96)*, (Miami Beach, FL), IEEE, December 1996.
- [73] V. Paxson and S. Floyd, "Why we don't know how to simulate the internet," in *Proceedings of the 1997 Winter Simulation Conference*, December 1997.
<http://wwwnrg.ee.lbl.gov/floyd/papers.html>.
- [74] "<http://ita.ee.lbl.gov/html/contrib/clarknet-http.html>."
- [75] Y. Hu, A. Nanda, and Q. Yang, "Measurement, analysis and performance improvement of the apache web server," in *18th IEEE International Performance, Computing and Communications Conference (IPCCC'99)*, (Phoenix/Scottsdale, Arizona), February 1999. <http://www.ele.uri.edu/hu/research.html>.
- [76] E. P. Markatos, "Main memory caching of web documents," in *the Fifth International World Wide Web Conference*, (Paris, France), May 1996.
http://www5conf.inria.fr/fich_html/papers/P1/Overview.html.
- [77] J. C. Mogul, "Network behavior of a busy web server and its clients," Tech. Rep. Technical Report WRL 95/5, DEC Western Research Laboratory, Palo Alto, CA, October 1995.
- [78] P. Rodriguez, E. W. Biersack, and K. W. Ross, "Improving the latency in the web: Caching or multicast?," in *3rd International WWW Caching Workshop*, (Manchester, England), June 1998. <http://www.eurecom.fr/rodrigue/>.

- [79] A. Erramilli, O. Narayan, and W. Willinger, "Experimental queuing analysis with long-range dependent packet traffic," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 209–223, April 1996.
- [80] N. Bhatti and R. Friedrich, "Web server support for tiered services," *IEEE Network*, September/October 1999.
- [81] P. Fishburn and A. Odlyzko, "Dynamic behavior of differential pricing and quality of service options for the internet," in *ICE'98*, (Charleston, SC), 1998.
- [82] M. Harchol-Balter, M. E. Crovella, and C. D. Murta, "On choosing a task assignment policy for a distributed server system," in *Proceedings of Performance Tools '98, Lecture Notes in Computer Science*, vol. 1469, pp. 231–242, 1998.
- [83] *Usability Engineering*. Academic Press, 1993.
- [84] L. Cherkasova and P. Phaal, "Hybrid and predictive admission strategies to improve the performance of an overloaded web server," Tech. Rep. HPL-98-125R1, Hewlett-Packard Laboratories, 1998. <http://www.hpl.hp.com/techreports/98/HPL-98-125R1.html>.
- [85] E. Borowsky, R. Golding, P. Jacobson, A. Merchant, L. Schreier, M. Spasojevic, and J. Wilkes, "Capacity planning with phased workloads," in *Proc. WOSP'98*, (Santa Fe, NM), ACM, October 1998. <http://www.hpl.hp.com/research/itc/csl/ssp/papers/index.html>.
- [86] X. Chen and P. Mohapatra, "Lifetime behavior and its impact on web caching," in *Proceedings of the IEEE Workshop on Internet Applications*, (San Jose, CA), July 1999. <http://dlib.computer.org/conferen/wiapp/0197/pdf/01970054.pdf>.
- [87] K. Li and S. Jamin, "A measurement-based admission-controlled web server," in *Proceedings of the IEEE Infocom 2000 Conference*, (Tel-Aviv, Israel), March 2000.
- [88] K. Sohrawy and M. Sidi, "On the performance of bursty and correlated sources subject to leaky bucket rate-based access control schemes," in *Proceedings of the Conference on Computer Communications (IEEE Infocom)*, (Bal Harbour, Florida), pp. 426–434, April 1991.
- [89] X. Chen and P. Mohapatra, "Service differentiating Internet servers." submitted to *IEEE Transaction on Computers*, 2000. <http://http://www.cse.msu.edu/rgroups/isal/pubs/journal/>.
- [90] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM Transactions on Networking*, vol. 5, pp. 835–845, December 1997. <http://www.cs.bu.edu/groups/oceans/papers/Home.html>.
- [91] R. Morris and D. Lin, "Variance of aggregated web traffic," in *IEEE Infocom 2000*, 2000. <http://www.ieee-infocom.org/2000/papers>.
- [92] A. K. Iyengar, M. S. Squillante, and L. Zhang, "Analysis and characterization of large-scale web server access patterns and performance," *World Wide Web*, pp. 85–100, 1999.
- [93] V. S. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *IEEE Communications Magazine*, vol. 32, pp. 70–81, March 1994.

MICHIGAN STATE UNIVERSITY LIBRARY



3 1293 02088 06