An Analysis of Issues Facing World Wide Web Servers

A Thesis presented

by

Stephen Lee Manley

to

Computer Science

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

April 7, 1997

Abstract

The World Wide Web has captured the public's interest like no other computer application or tool. In response, businesses have attempted to capitalize on the Web's popularity. As a result propaganda, assumption, and unfounded theories have taken the place of facts, scientific analysis, and well-reasoned theories. As with all things, the Web's popularity comes with a price for the first time the computer industry must satisfy exponentially increasing demand. As the World Wide Web becomes the "World Wide Wait" and the Internet changes from the "Information Superhighway" to a "Giant Traffic Jam," the public demands improvements in Web performance.

The lack of cogent scientific analysis prevents true improvement in Web conditions. Nobody knows the source of the bottlenecks. Some assert that the server must be the problem. Others blame Internet congestion. Still others place the blame on modems or slow Local Area Networks. The Web's massive size and growth have made research difficult, but those same factors make such work indispensable.

This thesis examines issues facing the Web by focusing on traffic patterns on a variety of servers. The thesis presents a method of categorizing different Web site growth patterns. It then disproves the theory that CGI has become an important and varied tool on most Web sites. Most importantly, however, the thesis focuses on the source of latency on the Web. An in-depth examination of the data, leads to the conclusion that the server cannot be a primary source of latency on the World Wide Web.

The thesis then details the creation of a new realistic, self-configuring, scaling Web server benchmark. By using a site's Web server logs, the benchmark can create a model of the site's traffic. The model can be reduced by a series of abstractions, and scaled to predict future behavior. Finally, the thesis shows the benchmark models realistic Web server traffic, and can serve as a tool for scientific analysis of developments on the Web, and their effects on the server.

Table of Contents

Chapter 1

Introduction

Computer scientists have been looking at exponential growth patterns for years. Exponential growth in computing power has extended computer functionality in ways that few people thought possible. Only four years ago, an Intel 486, 33MHz processor was considered the top-of-the-line. Now, the Intel Pentium Pro runs at 200 MHz -- 6 times as fast. Four years ago, machines ran with 4 MB of RAM. Now, retailers sell standard PCs with 16 MB of RAM. The advances in computer power have made a desktop machine more powerful than the supercomputers of the past. Advances in technology enable someone to send text, pictures, movies, or sounds to another person thousands of miles away.

After years of enjoying exponential growth in computing power, the computer community now faces exponential growth in demand. The Internet and World Wide Web (WWW or Web) have captured the world's imagination like no other computer product or service. The Internet, nicknamed “The Information Superhighway” enables any two computers in the world to send information to one another. Popular World Wide Web browsers such as Netscape or Microsoft Internet Explorer give users a simple to use Graphical User Interface (GUI). With this simple but powerful interface, a user can download a file from another computer with only a click of the mouse. The simplicity of the Web promotes the construction of Web sites, groups of Web pages, for purposes ranging from Kellogg's advertising their cereals to high school and college students setting up their resumes. The WWW has enabled everybody to become a publisher, with the HyperText Markup Language (HTML) facilitating the layout of text, pictures, animated pictures, sounds, and movies onto one Web page. Moreover, advances in technology continue to extend the functionality of the Internet. The Common Gateway Interface (CGI) enables somebody making a Web site to create a program that behaves differently for each client that accesses the site. Java allows the user to download a program from the server to his/her own computer. The WWW is simple to use and is becoming increasingly powerful.

The simplicity, power, and ubiquity of the Web sparked, and continue to fuel, its tremendous growth. Table 1 shows the growth of the Internet and the much more rapid growth of the Web from December 1991 until March 1995 [8]. Perhaps it is the greatest testament to the growth of these services, that in the past two years nobody has even attempted to quantify the amount of traffic on the Internet or the Web. The best estimates on the number of Web pages and servers come from the Alta Vista search engine run by the Digital Electronic Corporation. Alta Vista claims to have indexed over thirty-one million WWW pages on over 476,000 WWW sites.

Date	Total Traffic (GB)	WWW Percentage
12/91	1900	0.00
12/92	4300	0.02
12/93	8200	2.21
12/94	16,000	15.98
3/95	20,329	23.88

Table 1: Internet and Web growth from 12/91 to 3/95
Perhaps the greatest indication of the Web's continued growth comes from the fact that nobody even attempts to determine the traffic currently on the Internet.

The exponentially increasing demand for services due to the World Wide Web has led to great concerns about scaling computing resources to meet future needs. America Online Users can scarcely login to their system. Most users experience tremendous latencies in using the WWW during the day, leading to the nickname - "World Wide Wait." As a result, companies and individuals continually ask, "How can the Internet and World Wide Web effectively scale with demand?" Microsoft and Netscape are trying to make their WWW server software increasingly powerful. Silicon Graphics, Incorporated and Sun claim that their only their machines have the requisite power needed to run a WWW server. Internet Service Providers buy hundreds of megabits of bandwidth to serve their clients' needs. Researchers continue to refine the HyperText Transfer Protocol (HTTP)[1], which enables servers to send Web pages to clients. While the developments are exciting, and seem to hold great promise, all of the proposed enhancements to the Web suffer from an identical flaw. How can anybody solve a problem, without knowing exactly what the problem is? Companies spend millions of dollars and thousands of work hours on improving WWW performance. Every hacker claims to know how to improve web performance, but nobody has even identified the source of WWW performance degradation.

The nature of the World Wide Web has made it an extremely difficult phenomenon to measure. The distributed and extensive nature of Web makes it difficult to understand all of the issues involved. Those who possess interesting information on client behavior refuse to distribute it. Users don't want their behavior and access patterns made public, for obvious reasons. Webmasters refuse to make access logs from their servers available, due to possible security holes as well as privacy concerns. Finally, by the time the information is released and the analysis completed, the conclusions have become obsolete. With a lack of current and widely available general information, one has a very difficult time analyzing the World Wide Web.

The goal of this work is to conduct scientific analysis on the effects of the server on WWW scaling. Servers create logs of data, known as server, access, or Web logs, that detail the requests sent to the server. Access logs can reveal the type of workload being placed on a server, as well as its ability to handle such a load. From a series of these logs, one can also see trends in the type of traffic on that server. Therefore, to understand the role of the server's effect on the handling of increased WWW traffic, as well as its limitations and assets in handling a greater workload, I analyze a variety of WWW logs. While an analysis of the traffic patterns handled by a server is useful, one would like to actively test the server, to monitor its behavior as it runs. Benchmarks were created for the very reason of simulating realistic behavior, in a controlled environment. To gather interesting data about the server, the benchmark must realistically model the type of load experienced by a server. To this end, I have constructed a self-configuring WWW benchmark. The benchmark analyzes a set of server logs to create a model of the site's traffic, which becomes the basis of the benchmark's activity.

Thesis Roadmap

Chapter 2 discusses other research into the access patterns to WWW servers and the history of Web server benchmarking. Chapter 3 presents an analysis of server traffic patterns for a variety of sites. Chapter 4 details the design of the self-configuring, scaling, WWW server benchmark. Chapter 5 explores the quality of the benchmark's approximation of actual WWW traffic patterns. Finally, Chapter 6 concludes the work, presenting the most important results, and their importance in the issue of improving the scalability of the Web.

Chapter 2

Related Work

A survey of the relevant work in measuring Web site traffic and server performance leads to one inescapable conclusion. No real analysis of the Web has been conducted in the past two years. Basing all of the current developments on traffic patterns from two years ago can be considered nothing short of disastrous. The work presented laid a good foundation for a more in-depth analysis of Web traffic patterns. Such work never occurred.

The chapter discusses the research into the fields that pertain to Web server performance. Section 2.1 examines statistical analyses of the traffic experienced by a variety of WWW servers. Section 2.2 considers current attempts at statistical modeling of WWW traffic. Section 2.3 covers research that measure the impact of new developments on Web server performance. Section 2.4 discusses the range of benchmarks currently available.

2.1 Web Server Traffic Analysis

Research into realistic traffic patterns has been extremely limited. The research efforts have been divided between proxy servers and primary Web servers. A proxy server acts as a mediator between primary Web servers and users. All user requests for a given area, such as Harvard University, can be funneled through a proxy server, which then passes the request to the intended primary server. Much of the analysis of servers has been conducted to help justify the viability and effectiveness of other research. The difficulty in conducting accurate and effective analyses have been limited by the difficulty in finding accessible Web logs, the enormous variety of issues to explore, and the rapidly changing nature of the Web.

Information about the behavior of users and servers is not freely available to researchers. The Web pages that any given user accesses are known only to that user. Only when user requests are directed through a proxy can one collect information on the behavior of real users. Even in these situations, users obviously do not want information about their Web surfing habits released. As a result, proxy server logs remain unavailable. On the other side of the connection, webmasters are unwilling to distribute information about a server. Some claim that access to server logs could lead to the discovery and exploitation of security holes. Others do not wish to identify those who access their site, as they feel their customers would feel violated and not return. Some ISPs don't want to release information on the number of requests and bytes of data they handle, as they fear it might make clients unwilling to subscribe to them for service. Furthermore, those who have gained access to logs, or have done research in the past, have seen the conditions of the Web change so rapidly, that their conclusions are outdated. Still, despite the difficulties inherent in analyzing Web traffic, other researchers have laid the groundwork, and posed interesting questions that should continue to be analyzed.

2.1.1 Caching

Pitkow and Recker used WWW logs from the Georgia Tech main university server, from January 1, 1994 to March 31, 1994 to test the effectiveness of caching schemes[12]. In the three months, the server handled approximately 305,000 requests, with an average of 3500 requests a day. The researchers were attempting to model the likelihood that a file would be accessed, based on the previous weeks' traffic. Their model, based on human memory patterns, could quite accurately predict the likelihood that a file would be accessed during the coming week. Less than a year later, this research was obsolete.

By 1995 researchers looking at WWW caching were examining ways to reduce the bandwidth consumption of the Internet, as traffic had increased exponentially in the time since the Georgia Tech research was completed. No longer were people concerned with *whether* a file would be accessed, but with *how often* it was accessed, and the concurrent bandwidth consumption. Bestavros and Gwertzman both examined the benefits of server-side caching. Their primary conclusion from WWW server logs was that, as users surf the Web, most requests download a small number of unique bytes. From this data, they advocate that a server can effectively cache its most popular files in a variety of proxy caches, thus reducing latency and bandwidth consumption on the Internet.

Bestavros and Gwertzman both use Web server access logs to detail the potential benefits of server-side caching. Bestavros examined a series of 1994 and 1995 logs from the Boston University(BU) Computer Science Department server. His support for the effectiveness of caching is enhanced by the conclusion that 0.5% of the files account for 69% of requests, and that 91% of requests download a mere 10% of unique bytes on the server. He contends that this behavior is indicative of the entire Internet because the logs of the Rolling Stones Fan Page, from Nov. 1, 1994 to Feb. 19, 1995 exhibit similar patterns[2]. Gwertzman derives similar results from the December 1994 logs of the NCSA site, the main Harvard University site, the Harvard Division of Engineering and Applied Sciences site, and a Harvard computer club server[8]. A small number of bytes and files account for the majority of sites' requests. Interestingly, Gwertzman's figures show a significant disparity between the most popular sites, and common sites. The Harvard sites actually handle less traffic than that seen by the Georgia Tech site - between 1400 and 2800 requests a day. The NCSA site processed over 600,000 requests a day - over half a year's traffic on the other servers listed by Gwertzman. Still despite the enormous disparity in the types of sites, Gwertzman and Bestavros both claim that their extremely focused analysis of a limited number of sites gives sufficient justification for their theories. My detailed examination of over ten sites in chapter 3 will discuss issues ignored by Gwertzman and Bestavros, as well as the current needs for caching.

An analysis of proxy server logs helps characterize user behavior. Proxy server logs reveal access patterns of a subset of users to all sites, rather than all users to one Web site, so that the entire behavior pattern of a variety of users can be seen. Duane Wessels set up a proxy server at the University of Colorado at Boulder and analyzed the traffic patterns from October 31, 1994 to December 31, 1994[16]. Over the 62 day period, almost 12,000 requests were generated - of which only 7.8% were handled by the proxy server's cache. This result surprised Wessels because he assumed that the people at one college would all tend to access the same pages more often. With the greater number of Web sites and users, this number it seems unlikely that this number would increase. The type of traffic was also detailed : 99% of requests were to GET files, while 1% were to POST data to the remote server. Most of the time, users merely downloading information, and did not send anything to the server. Of the requests, 42% were for HTML files, and another 23% of requests were for index files, HTML files or directory listings. Only 12.5% of requests were for image files, and another 3% of requests were due to CGI. Surprisingly, most requests from the University of Colorado resulted in text being returned. With the proliferation of images on the Web, this seemed unusual. Wessels does note that the University of Colorado used text-based browsers, which would decrease the quantity of image traffic to the site's users. Still, his analysis provides interesting data. First, while Gwertzman and Bestavros show that most Web sites show a significant skewing of requests for a small percentage of files, the same skewing cannot be seen in the requests emerging from a site. Moreover, over 65% of requests were for HTML and index files. The data show that in 1994, different sites handle different levels of traffic, that HTML was a primary source of traffic, and that requests to a site's Web server showed much greater skewing than requests through a proxy server. Wessels work into analyzing proxy server logs forms a model for my own analysis. He used the logs, not to further an agenda, but to assess the current types of Web traffic.

2.1.2 Network Behavior

Jeffrey Mogul conducted the most comprehensive analysis of server behavior, recording information on a set of busy servers as they ran[10]. Mogul's analysis of three servers used during the California elections of 1994 was facilitated by his instrumentation of servers to record the number of concurrently open HTTP requests, connection duration, PCB (process control block) statistics, and packet transmission statistics. The three servers handled 1.6 million requests in a 10 day period, with 1 million requests coming on one day. Combined, the servers handled as many as 60 requests per second. Some requests required more than 1000 seconds to complete, but most were handled in under one second. The PCB table, which stores information on each open connection, grew as large as 1297 entries, 1049 of which were in TIME_WAIT - a state to ensure proper closure of the connection. At this time, only 100 connections were alive. Over 90% of requests used fewer than 25 packets between client and server. The fact that most Web requests are small means that they can be handled more quickly than the infrastructure for the connection can be discarded. Nearly 70% of requests were for GIF formatted images, while only 30% of requests were for HTML files. Other types of traffic were negligible. In this data, one can see the more expected pattern of images dominating the types of requests. Mogul's in-depth analysis of a busy Internet server in 1994, yielded information about the need for maintaining PCB (Protocol Control Block -- the infrastructure of the connection) tables effectively, to cope with large numbers of open connections, and even larger numbers of closed, but not destroyed, connections. His work described the increasingly prominent role of images in Web traffic. While Mogul makes no claim that the site he analyzes is indicative, he still only looks to one site. Furthermore, like Wessels' work, the data and analysis suffer from over two years of aging. Therefore, in Chapter 3, I extend Mogul's analysis of connections, type of traffic, burstiness, and types of traffic on a server to current data, for a wide variety of sites. I also provide a means of characterizing sites to help alleviate the myth that all sites exhibit similar characteristics.

2.2 Modeling of WWW Traffic

Researchers have begun to statistically model the traffic patterns found on the World Wide Web. A current theory in the field is that Web traffic from a set of clients at a site, fits the model of self-similarity. By describing Web traffic as self-similar, one looks at the inherent burstiness of Web traffic. Burstiness is a characteristic by which many requests arrive at once, followed by stretches of time with few or no connections. The bursts, whether examined on a time scale of one second, one minute, or one hour, look similar. If Web traffic were self-similar, current burstiness would correlate to future burstiness -- long-range dependence. Crovella and Bestavros use the BU client traces detailed in Section 2.1.1 to show similar patterns of burstiness over four orders of magnitude of time - 1 to 1000 seconds[4]. They note the importance of user “think time” when they read the Web page they have downloaded. The work concludes that WWW traffic is self-similar, but it cannot explain the cause of the self-similarity. The authors conjecture that the self-similar behavior can be traced to the basic hierarchical structure of Web sites, and the effects on the user. For my benchmark, I use the idea of modeling Web traffic, but rather than using a mathematical model at the request level, I create a heuristic model on a user level. Still, the fundamental ideas of the importance of user idle time, and burstiness play a role in any accurate model of Web traffic. Furthermore, like Crovella and Bestavros, I contend that future WWW traffic patterns can be reasonably predicted, and use such methods to form the basis of a benchmark that scales its load to model the future.

2.3 Analysis of Recent Developments in Web Traffic

Other levels of research have gone beyond measurement and estimation of traffic patterns. In an attempt to identify performance limitations on systems, researchers have examined the effects of CGI traffic on a server, as well as the potential benefits of persistent connections. Some people have theorized that CGI traffic causes a server's performance to degrade because the server starts entirely new processes to execute the task. On the other hand, people have proclaimed that persistent connections can ease the burden placed on servers. Originally, Web browsers made a separate connection to the server for each downloaded file. As many Web pages contain tens of small files -inlined images- and the cost of setting up a connection is large, the performance impact of this process would seem to be noticeable to users. Moreover, as Mogul noted in his analysis of the California election server, the operating system needed to deal with thousands of closed, but not deleted, connections. Thus, researchers have attempted to quantify the benefits of persistent connections. Section 2.3.1 describes the work done to analyze the impact of CGI on server performance. Section 2.3.2 discusses the analyses of persistent connections.

2.3.1 CGI

The analysis of CGI focused on the process of handling one CGI request. Edward and Rees offer a quantitative analysis of a CGI request that details the amount of time spent in different stages of the request[5]. The work used a CGI script with trivial functionality to isolate the effects of a server handling CGI, rather than the script itself. Their platform of measurement was CERN httpd, one of the originally popular Web servers. Their data indicate that the most time-consuming operation is the transfer of request data to the CGI script. On the other hand, their conclusions focus on optimization of process creation - which consumed only one-tenth of the time required to do the IPC. The study's limited and contradictory nature, and the non-trivial amount of CGI traffic run on servers led to a further analysis of the Edwards and Rees work. In chapter 3, I emphasize the insignificance of CGI on most servers.

2.3.2 Persistent Connections

The original analysis of persistent connections examined the probability of reuse of open connections. Mogul ran the server access logs from the popular California election server, as well as that of a popular corporate site, through a simulator to estimate the effectiveness of persistent connections[9]. The logs revealed that persistent connections could significantly reduce the number of open connections, by reusing the open connections. The logs showed that over 50% of users retrieved at least 10 files per open connection. On average, one connection would support six requests. His results revealed that with a timeout of sixty seconds, connections were reused fairly often, while also enabling the maximum number of open connections to remain low. Mogul also focused on the importance of allowing users to only open one connection at a time, rather than the Netscape driven standard of four connections per user. The age of his data leads to concerns about the accuracy of the results. Furthermore, in the construction of the new HTTP protocol, browsers are allowed to open two persistent connections. Therefore, the work needs to be updated to examine the effects of such a decision.

Currently, the World Wide Web Consortium(W3C) is conducting an analysis of persistent connections[11]. Rather than using historical logs to simulate behavior, the W3C chose to implement a client and a server that implement persistent connections. They then analyzed the performance weaknesses of their implementations. At first, they discovered that the latency incurred with one persistent connection lagged significantly behind that of the standard type of request. They concluded that the parallelism of 4 open connections enabled the client to receive data more quickly than the one open connection - with future requests not being produced until after current requests. They concluded that only by pipelining requests can persistent connections achieve reasonable latency. By pipelining requests, servers receive multiple requests from the client before they have even fulfilled a request. In this way, the transportation of data is not serialized, and performance is enhanced. On the other hand, in validating a simple page with multiple files, persistent connections generated only one-tenth of the packets as non-persistent connections. Since validating pages usually results in a response that the file is unmodified, little data is sent to the user. This research validates the potential benefits of persistent connections, as well as detailing a potential performance limitation. However, the authors do not give evidence of the potential role of persistent connections in real traffic patterns. Their research is focuses on analysis of assumed situations. Thus, while their results detail the effects of an actual implementation of persistent connections, their tests suffer from a lack of realism. They refer to Mogul's research, but understand the limitations of the study. In chapter 3, I update the research that Mogul began. In chapter 4, I describe a benchmark that can use persistent connections.

2.4 Benchmarking

Properly designed and implemented, a benchmark enables researchers, such as the W3C, to accurately test new developments. Poorly constructed, a benchmark creates faulty impressions of needs in a given community. Web server research has suffered from just such a crisis. Benchmarks have either ignored the inherent differences in the traffic handled by a variety of sites, or have completely ignored the considerations of realism in generating traffic. Currently, developments in benchmarking technology have attempted to address some of the flaws of older benchmarks. Section 2.4.1 describes the first Web server benchmark - Silicon Graphics, Incorporated's WebStone. Section 2.4.2 looks at SPEC's venture into Web server analysis. Section 2.4.3 considers recent developments from Lotus and Harvard, with DBench.

2.4.1 WebStone

WebStone filled the complete void of Web server benchmarks, and despite its flaws, many of its features have rightfully become standards. The general approach of WebStone is to create a master process that spawns and gathers data from multiple client processes[15]. The master spawns multiple clients that repeatedly open one connection to the server and download one file. The key parameters of the benchmark are fileset, number of client processes, and the length of the test run. Since it is impossible to quantify WWW traffic in one way, these parameters are configurable. The theory, then, is that users can configure the benchmark to simulate the behavior experienced by their sites. Unfortunately, the lack of a well-researched and reasonable standardized load leads to difficulties in comparing WebStone numbers. In addition, there is no suitable means of deriving a suitable configuration for a particular site. As is often the case, individuals frequently have misconceptions about the inner workings of their systems. The configurations used and reported often generate traffic that has little to no resemblance of real WWW traffic. As an attempt to provide a standard for comparison and testing, WebStone attempted to offer a "standard load" with a fileset of five files ranging from 500 bytes to 5 megabytes, varying by an order of magnitude. The configuration requires running with 20 to 100 users, increasing by 10 users every 10 minutes. WebStone's final limitation arises from its focus on throughput. Even with an accurate portrayal of a site in the configuration logs, the behavior of the WebStone clients is so unlike real browser behavior that the model would still fail. Since WebStone focuses on the number of bytes that can be transferred, portraying accurate user behavior is not salient. Instead, the benchmark chooses to try to make as many requests as possible in a short time. Unfortunately, throughput can be as much a measure of the networking hardware of the system, as performance of the server software and operating system.

2.4.2 SPECWeb

SPECWeb '96 and '97[13,14] attempt to rectify the lack of comparability of statistics, while retaining some of the structural aspects of WebStone. SPECWeb also builds upon the client-master model, but rather than spawning off clients that randomly download files as rapidly as possible, SPEC designed the benchmark to scale the load it generates to stress test a server. The file set is determined by the goal for operations per second on the server. Furthermore, client process behavior is also constrained by this goal. Thus, clients will remain idle for some amount of time to ramp up the generated load over time. Thus, the SPEC benchmark attempts to show the results of increasing the load on a server, until it can handle no more. SPECWeb controls the load generation of its processes with a finer granularity than WebStone because it controls how clients behave, not just the number of clients. The metric reported by SPECWeb is a report of the number of operations (connections) that the server can execute per second. This metric helps show the ability of the server to handle requests, not merely its ability to send bytes over the network. While SPEC initially bases its definition of users from analysis of Web server logs, the definition degrades as users are controlled primarily by the load they generate, rather than approximation to the log. Like WebStone, the load quickly grows to be artificial. Moreover, since the load is based on analysis of a limited, and similar, group of servers, the benchmark does not provide indicative data for the entire set of benchmarks.

2.4.3 DBench

The most recent development in Web server benchmarking is DBench, from a group at Lotus and Professor Brad Chen of Harvard[3]. DBench, not yet available, attempts to model realistic traffic patterns, by extracting the clients' accesses from WWW server logs. Essentially, the benchmark separates user profiles by IP addresses from the access logs. Then, the benchmark chooses to replay users from the log. Then, as with SPECWeb, the number of each user type selected increases to put greater load on the server. By replaying the user profiles, the benchmark achieves a great deal of reality. It models different types of WWW traffic, ranging from conditional GET requests, to CGI traffic, and requests that generate errors. Of course, one of the benchmark's assets creates problems when trying to generate standard results. Using the benchmark requires complete access to the Web site from the logs, as well as access to the original logs. Since sites tend to change, and webmasters are notably hesitant to surrender WWW logs, the benchmark may not be able to produce standard loads. On the other hand, the system reports metrics beyond that of throughput or operations per second. The benchmark focuses on the latency experienced by users. Since end users are those for whom the Web site is managed and run, their perception is paramount. Thus, DBench will take further steps in benchmarking servers. Many of the flaws in DBench are addressed by hBench:Web, presented in chapter 4. This benchmark also bases its traffic on WWW server logs, but does so to create a freely redistributable configuration. Moreover, it scales its traffic not by reusing the same user profile, but by selecting from a set of profiles. Finally, my benchmark can increase its load by creating a new configuration that is based on historical patterns of growth, or allow the user to change the traffic patterns by increasing the probability that certain users will be run. As a result, the behavior of hBench:Web consistently maintains a link to real user behavior, and the tests can easily be redistributed.

Chapter 3

Analysis of Web Server Access Logs

The public's enthusiasm for the Web has been matched only by that of computer companies. Since money drives the industry, the emphasis from Microsoft to the smallest start-up has been on rapid technological development, rather than well reasoned scientific advancement. New Web tools make money, statistical analyses of the Web do not. Concurrently, the academic community has made little effort in researching the Web. As a result, the limited analyses are all outdated. All published statistics on server behavior comes from data gathered in late 1994. In the past two years, the nature of the Web has fundamentally changed. CGI has introduced server and user interaction. Java and animated GIF images have continued to find their way onto Web pages. The number of Web users has grown at an unknown, but likely exponential rate. With the tremendous growth and change in Web sites, users, and technology, a comprehensive analysis of real traffic can help focus research on the most instrumental issues facing the computing community.

The statistical analysis presented in this chapter focuses on traffic patterns observed on a variety of Web sites. Server logs can reveal an enormous amount of information about users, server behavior, changes in sites, and potential benefits of new technical developments. First, section 3.1 details the different types of sites surveyed. Section 3.2 presents a basic traffic analysis for the variety of Web sites. Section 3.3 presents a simple taxonomy for discussing different types of site growth. Section 3.4 shows that the role of CGI on servers does not match the enormous attention that accompanied its creation. Section 3.5 discusses the potential benefits of persistent connections and the means of reaching those benefits. Section 3.6 demonstrates that the traditional view of caching will not fit the Web, and presents a new perspective on Web caching. Section 3.7 concludes by proving that servers are not the primary source of latency when using the Web.

3.1 Site characterization

The Web sites that form the basis of the studies presented were selected by variety in content and purpose. Table 2 describes the sites by content, site suffix, service provider, server software, and dates for which logs are analyzed. The three academic sites(.edu) differ from all other types of sites because one webmaster does not generate all of the pages. The students ensure that there is no specific direction or purpose for the site. The Harvard University Faculty of Arts and Sciences (FAS) site presents information about Harvard University undergraduate life, and hosts all undergraduate WWW pages. This site can be considered indicative of academic sites, with its extensive breadth of Web page directories, all of which have little depth. All students can post Web pages, which leads to a great

Anonymized Site Name	Site Type	Content	Service Provider	Server Software	Time
Traditional Business	.com	Information on subject matter, advertisements	ISP	Apache 1.1.3	1/96 - 2/97
Harvard University Electrical Engineering and Computer Science (EECS)	.edu	Graduate student Web pages, department information	Harvard EECS	NCSA 1.4.2	4/96- 2/97
Harvard University Faculty Arts and Sciences (FAS)	.edu	Large academic institution, undergraduate Web pages	Harvard FAS	NCSA 1.4.2 Apache 1.1.3	10/94- 2/96
ISP company page	.com	Simple advertisement	ISP	Apache 1.1.3	9/96 - 2/97
Rice University Electrical and Computer Engineering (ECE)	.edu	Graduate student Web pages, department information	Rice ECE	NCSA 1.4.2 Netscape Netsite-Commerce 1.0	7/95- 12/96
Adult-Content	.com	Adult images, movies, chatrooms	ISP	Apache 1.1.3	3/96- 9/96
Organization for Members of same Profession	.org	Articles and images pertaining to field	ISP	Microsoft IIS/3.0	4/96- 2/97
Web site designer	.com	Samples of different sites, games	ISP	Apache 1.1.3	8/96- 2/97
Government Agency	.gov	Information on agency's actions	ISP	Apache 1.1.3	8/96- 2/97
Free Web Software Site	.com	Free, proprietary Web software	ISP	Apache 1.1.3	4/96- 1/97

Table 2: List of sites and their characteristics
The sites come from a variety of content categories, and the logs give the history of the site.

variety in pages and an extraordinary number of directories of students, but the 4 MB per user disk quota prevents any student from developing a very developed set of pages. The other academic logs come from the graduate computer science and engineering divisions of Harvard and Rice University. These sites tend to have fewer students making Web pages and the students' pages tend to be more extensive, but the model remains the same.

The sites from the ISP demonstrate the different purposes behind Web sites. The government site publishes information on its activities. The organization site produces content for a limited audience. The company sites include one of each of the major business models on the Web. One site comes from the set of ubiquitous adult-content sites. The traditional business uses its Web site as an advertisement. The Web software company makes its living by creating free Web software, that other companies and sites license. The final business model is that of the Web site designer, which uses its site as an advertisement for a Web product.

The server software run by the sites incorporated the major players in the Web server market. While the Apache 1.1.3 was the server software chosen for most sites, within the sites there are also versions of Microsoft's IIS/3.0, Netscape's Netsite-Commerce 1.0 and NCSA 1.4.2. While the companies that make servers try to differentiate their product from other servers on the market, the servers all generate the same type of access log. All of the servers, in default mode, list the IP address of the machine generating the request, the time when the request was received, the method or the request, the URL, the protocol version, the type of response, and the size of the body of the response. The ISP altered its servers to generate logs that record not only the starting time of the connection, but the amount of time necessary for the server to handle the request.

The lack of geographical diversity of the logs underscores the difficulty in convincing sites to release their logs. All of the sites, except Rice ECE, in Houston, Texas., run from the East Coast of the United States. This study includes no logs from other countries, or even from the West Coast of the U.S. Additionally, most of the logs come from one Internet Service Provider. The nature of the ISP still indicates that the data is indicative. Its traffic and setup meet or exceed that of most other large ISPs, and as a leader in its field, other ISPs follow their direction. Fortunately, the wide diversity of sites hosted by the ISP enabled the choice of a variety of sites.

The attitudes of people such as those who run the Harvard FAS site lead to the difficulties in finding Web logs. The FAS site barred all access to Web server logs (though not until after I had retrieved a year and a half of data). The system administrators' reasons for protecting the data bordered on ridiculous. First, they stated a fear that I could use the data to track students' Web surfing habits, and stalk them. Second, they feared that I could find and exploit security holes on the system. The difficulty in obtaining Web server logs plagues much of the research on the Web.

3.2 General Analysis

An analysis of Web server logs can be summed up with one word: "growth". Virtually all of the sites' traffic at least doubles every year. While each site has different types and sizes of files, with different methods, all experience exponential growth patterns.

Site Name	Request Growth (%)	Bytes Transferred Growth (%)	Number of files on Site Growth (%)	Number of bytes on Server Growth (%)
Traditional Business	60	105	37	67
Harvard EECS	28	18	19	16
Harvard FAS	27	31	33	33
ISP	-2	7	-2	9
Rice ECE	13	17	7	14
Adult Content*	-27	-29	1	-1
Organization	-21	-20	-23	-19
Web Site Designer	6	7	0	14
Government Agency	7	5	1	-7
Free Software	95	81	23	24

Table 3: The monthly growth patterns of each site and its traffic
The total growth of the site is reported as the average growth per month. Most sites exhibit exponential growth patterns in traffic and site size. The level of exponential growth does differ by site.
* This site no longer exists. The reported data trace its reduction to destruction.

In industry, engineers commonly joke that by licensing Java from Sun, a company's stock price immediately jumps by ten points. A similar assumption fills references to the Web - supposedly, anything involved with the World Wide Web grows exponentially. This assumption is true. Table 3 details the average monthly growth of a site's data and the corresponding traffic patterns. On the Web, since all growth can be classified as exponential, one needs to differentiate between the different exponents. Two of the business sites double their request traffic in less than two months.On the academic sites, traffic doubles every four to six months. Finally, at what seems a standstill compared to the other sites, the government site and Web developer's site traffic doubles every year. An analysis of the sites with decreasing traffic follows in Section 3.3.

One of the basic assumptions about the Web is that the average file size grows as quickly as traffic. More sites now use images, sounds, and movies. Inline images used for buttons, and eye-catching color have relegated the text only Web page to the status of historic relic. The data in Table 4 indicate that average file size has grown rapidly. Concurrently, the median file size has been decreasing. Rather than each file on the Web fueling the growth of the average file size, the sites actually show an increasing disparity in file sizes. The largest files have grown extremely quickly, and small files have become smaller and more common, to offset the enormous growth of the largest files. For every movie or large animated image on a site there exist hundreds of inline buttons, pictures, and thumbnail pictures

Site Name	Average File Size (KB)	Growth of Avg. File Size	Max File Size (KB)	Growth of Max File Size	Median File Size (KB)	Growth of Med File Size
Traditional Business	8.155	22%	101	36%	1.7	8%
Harvard EECS	33.5	-2%	12,000	14%	3.6	2%
Harvard FAS	13.25	0%	5300	25%	1.75	-2%
ISP	11.18	11%	723	34%	0.53	-7%
Rice ECE	24.8	7%	4500	10%	0.80	-7%
Adult Content	24.5	-1%	1625	-11%	0.4	-1%
Organization	8.4	5%	117	3%	3.2	4%
Web Site Designer	6.3	15%	77	68%	2.6	12%
Government Agency	6.3	-8%	15	-2%	0.21	-30%
Free Software	33	0%	6500	20%	0.087	-32%

Table 4: Site file average, maximum, and median file sizes for the most current logs, and their corresponding growth patterns over the history of the server logs.
Note that the median file size generally decreases, while the maximum file size increases so rapidly as to increase average file size. This indicates a growing dichotomy in file sizes on Web sites.

The Web has helped turn the Internet into a system of sight, sound, and text. As the Web has grown, the images have become a substantive portion of traffic. Unfortunately, images are not only worth a thousand words, one image file is larger than a file of a thousand words. This distinction matters little to students and to sites with limited growth, who either do not pay for bandwidth, or do not see their costs increasing rapidly. On the other end of the growth curve, however, the sites that experience the most rapid growth have been re-examining their image policy. Table 5 details the current popularity of image files. It also shows that different types of image files, JPEG and GIF, demonstrate different growth patterns on some sites. JPEG and GIF are both image formats, with the characteristic that the better compression of JPEG enables the file to present a larger picture with less data than a GIF compressed image. The two sites that displayed the greatest growth patterns have begun shifting away from the inefficient GIF format to the JPEG format of images. The enhanced compression enables the server to reduce its bandwidth consumption. The cost of the image has not been cut, however, just re-apportioned. Since the image has better compression, the client machine must process the file more to display the image. The reduced bandwidth consumption saves companies money with the ISP, which charges for bandwidth consumption, but in a time of powerful machines and limited bandwidth, JPEG files are a simple way of deferring load to the client machine.

Site Name	**%Requests for images, most recent month**	**6 month change in %requests for JPEG files**	**6 month change in %requests for GIF files**
Traditional Business	63	16	-14
Harvard EECS	49	3	-8
Harvard FAS	58	1	9
ISP	27	0	-2
Rice ECE	51	1	1
Adult Content	86	6	-10
Organization	57	9	0
Web Site Designer	72	0	4
Government Agency	45	0	0
Free Software	73	8	-9

Table 5: The percentage of traffic due to images, and the change in the relative percent of image requests for GIF and JPEG formats in the most recent six months of the server logs
The sites with the greatest growth have shifted from GIF to JPEG to take advantage of its greater compression of image data.

By caching documents, a client need not always download a file from the server. Instead, the browser or proxy cache can avoid sending a request altogether, or validate the consistency of the cache file with the original by sending a conditional GET request. For a conditional GET, the client sends a request of the form, “If-Modified-Since <Last Date of Guaranteed File Consistency> GET <URL>”. If the file has not been modified, the server sends a response of “304 - Not Modified.” Across all of the servers, in the most recent month of the log, the“304 - Not Modified” responses comprised between 10 and 25% of total responses to requests. Throughout each of the logs, in the past six months, the percentage of the “304 - Not Modified” responses was stable. Across the incredible variety of sites, with their concurrent differences in content and growth patterns, conditional GETs comprise nearly ten to twenty percent of all requests. A substantive number of connections to the server now return only a header, so it is worthwhile to find ways of improving performance for this operation. This trend forms a compelling argument in favor of persistent connections, analyzed further in section 3.6.

Site Name	**% Requests with a 304 Response**
Traditional Business	13
Harvard EECS	10
Harvard FAS	20
ISP	16
Rice ECE	8
Adult Content	9
Organization	26
Web Site Designer	13
Government Agency	13
Free Software	24

Table 6: The percentage of responses that return “Not Modified”
These responses indicate the success of current caching policies.

As the Web changes, users and developers desire more options with which to interact with the server. The original model of the Web enabled clients to “GET” files from a remote server. Recently, however, methods that allow clients to “POST,” or send, data to the server have gained attention. Of the sites, only one did not handle at least 97% of requests to GET files. Nearly one-third of all user traffic to the organization server POSTed data. The site charges for membership, so a user must send a name and password each time the site is accessed. Furthermore, the site is driven by a menu based system, in which the user selects an area of interest from a menu, and sends the results to the server. The functionality differs in no way from a standard tool bar, and most sites require a one-time validation rather than the constant demand for validation from this site. Therefore, this site should be considered an outlier. The only evident trend is that the Web is still a read-only medium, driven by users downloading files from the server.

While many of the growth patterns described in this section apply to all sites, not all Web sites grow in the same way, nor for the same reasons. The sites have different types of files, traffic, and growth patterns. In the next section, I present ways of distinguishing a variety of sites. Analysis of growth patterns, user behavior, and types of traffic enable one to characterize different sites.

3.3 Differentiating among sites

Taxonomy has presented challenges in all disciplines and times, from Aristotle trying to classify animals to television commentators trying to classify the color of Dennis Rodman's hair. Despite the difficulty, naming and categorization facilitate easy communication on complicated subjects. All Web sites do not have the same goal, the same type of traffic, or the same growth pattern. To begin to fill the void of taxonomy among Web sites, I detail a classification system for different types of growth patterns.

Site Name	Requests	Megabytes Transferred	Files on Site	Megabytes on Site
Traditional Business	321,747	3819	347	2.8
Harvard EECS	106,001	1322	5865	196
Harvard FAS	2,328,401	15,097	34, 348	455
ISP	8139	39	134	1.5
Rice ECE	85,763	854	4655	115
Adult Content	69,906	857	223	5.5
Organization	42,301	251	95	0.8
Web Site Designer	43,523	104	119	0.7
Government Agency	26,049	214	185	1.2
Free Software	15,982,085	76,315	4070	136

Table 7: The size of sites and the traffic handled in the most recent server log
The disparity in levels of traffic and site size illustrate the fundamental difference in Web sites.

Those involved with the Web tend to assume all interesting sites look like Microsoft and Netscape. Each site has set up more than ten servers to handle tens of millions of requests a day. These sites claim to be two of the most popular on the Web. However, as the most heavily loaded sites, they cannot also be the common case. While the sites presented in section 3.2 all show a great deal of growth, the actual load on each varies greatly. Table 7 shows the number of requests handled per month on each server. A site handling fifteen million requests for seventy-six GB of data a month must be thought of as quite different from a site processing forty-five thousand requests for 250 MB of data each month. Trying to compare the heaviest sites to average or lightly loaded sites is akin to comparing a Mack truck with a Yugo. Not all sites are Microsoft or Netscape. In fact, the data indicates that most sites have not yet reached one hundred thousand requests a month. The differences also surpass those of mere load. The difference among sites extends far beyond the number of requests per month. The varying sizes of sites show that all sites cannot be considered equal. However, size and level of user generated load cannot classify sites in a unique and definitive way.

To create a reasonable analysis of a site's growth pattern, one must examine the issues that define a site. For a Web site, the three primary issues are: site composition and growth, growth in traffic, and user access patterns. While the data for the first two factors can be found in Tables 3 and 6, no accurate portrayal of the way users explore sites has been presented. The distribution of requests per file and distribution of number of requests per user indicates whether users tend to visit many pages on a site, or only a few. These figures also help ascertain whether all users visit the same subset of pages, or tend to view different subsets of pages on the site. The sites demonstrate six different growth patterns.

First, the site that makes Internet software freely available has seen its traffic has double every month. The cause of growth is simple - other sites have been licensing the software and downloading files to run on the software. As the software's popularity has exploded with other businesses, a wide variety of clients became interested in the product, and the site's traffic doubles almost monthly. This method of growth is simple - make something that everybody wants. Figure 1 illustrates that very few of the pages account for the majority of requests - as many users access the path of pages to receive the free software tool. This site's traffic grows because many people download the most popular pages.

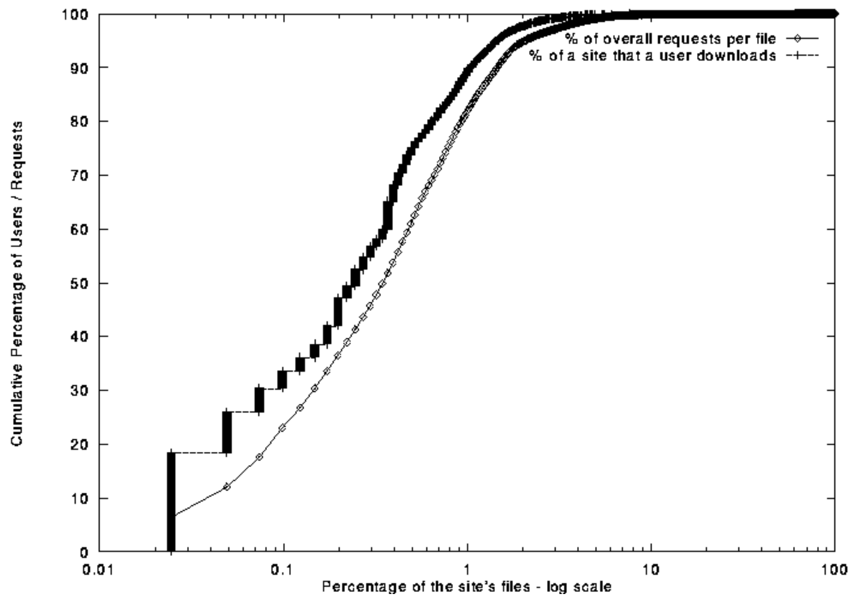


Figure 1. The cumulative distribution of percent of total requests for each file, and the percentage of total files accessed by the user for the Free Software Site. A small number of files account for a large percentage of requests, while most users access a very limited percentage of the files. This site grows because its most popular files are extremely popular

In fact, the site has been expanding, but the new files receive very little attention and constitute a trivial amount of the traffic.

The second most rapidly growing site has taken a different approach to generating growth. It continues to improve its site, adding more pages of information, and making it more visually appealing. Naturally, the number of files and bytes residing on the server have grown nearly as quickly as traffic. This method of growth adheres to the idea that a bigger and better site leads to increased traffic. This method is not surprising to find in a mainstream business using its Web site as an advertisement. The aggressive site maintenance and enhancement constitutes the fundamental difference between this site and the software site. To attract new users, the site frequently adds new sections of information to the site. Figure 2 shows the rate of growth of the site and its popularity. Between the 1st and 2nd, 4th and 5th, and 12th and 13th months, traffic undergoes a large increase, in response to changes on the site. These shifts, or "whomps", of traffic permanently change the level of traffic on the site. Between whomps, traffic remains extremely stable. The user activity on this site also reveals information about the site's growth. Figure 3 shows that most users examine only a limited set of pages, but some users explore large parts of the site. It also shows that many documents are accessed fairly frequently. In other words,

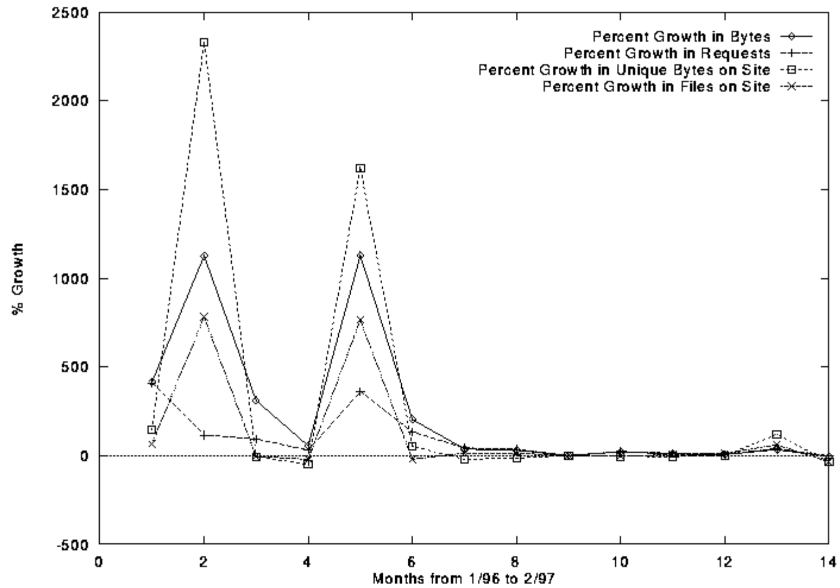
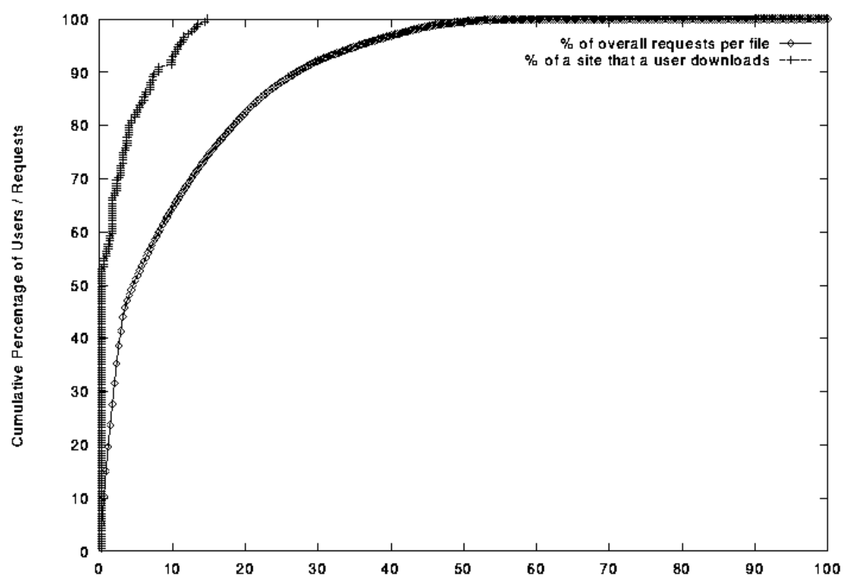


Figure 2 The growth of the traditional business site and its traffic, over time. The site grows during three periods, triggered by a massive enhancement of the site. I call this phenomenon of growth the "whomp factor."

Figure 3 The cumulative distribution of percent of total requests for each file, and the percentage of total files accessed by the user for the Traditional Business Site. Users tend to access a limited number of pages, but the pages tend to be different, so the distribution of requests by files shows a variety of popular files.



while most users download only a few pages, each examines a different set of pages. Thus, the site has achieved its goal. Its variety and general quality has attracted clients with a variety of interests.

The academic sites also show growth in their set of files and amount of data on the server. One of the most obvious difference between the academic sites and other sites is the magnitude of the site, compared to its traffic. The site grows as more students make Web pages. Not surprisingly, the wide diversity of student home pages on the Harvard FAS site have led to traffic growth virtually identical to the growth in number of files. As a student makes a new set of Web pages, or the administration adds to the commonly accessed university Web pages, clients with interest in the page access the site. Figure 4 shows that all users download less than 1% of the sites files. Nobody visits an academic site to look at the entire spectrum of pages -- but rather to look at a specific student's site or, more likely, the main University information. Figure 5 illustrates that he traffic experienced by files supports this claim, as it shows that beyond the extremely commonly accessed FAS front page, the remaining files are equally popular. The ratio of site size to load, and user access patterns distinguish academic sites from other sites.

The next pattern of growth can be traced to attracting clients who are extremely interested in the material on the site. The government site and non-aggressive business sites rarely change the scope of their sites, and yet the sites exhibit growth in traffic. Figure 6 shows that over time, users tend to download a greater percentage of the sites' files from the site. Figure 7 shows that over a series of months, the requests tend to distribute more evenly across the server's files. Essentially, as time passes, the users who visit the site do so to find information for which they examine the site more thoroughly. Essentially, traffic grows because the users most interested in the site's information begin to dominate the use of the site.

Two of the sites represent negative growth patterns. Figure 8 shows that the organizational site generated a negative whomp, or decrease, in traffic. The number of requests directed to the site decreased by an order of magnitude when the site began charging for access to the site. As with anything, once it costs money, fewer people want to use it. After this drastic change, the site tends to behave much as the government and non-aggressive business sites detailed above - only those who are most interested view the site, and tend to

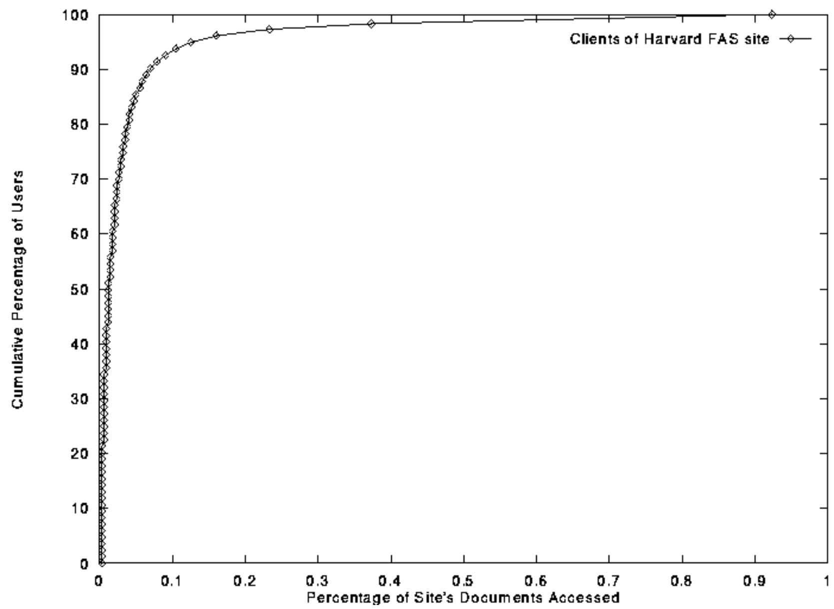
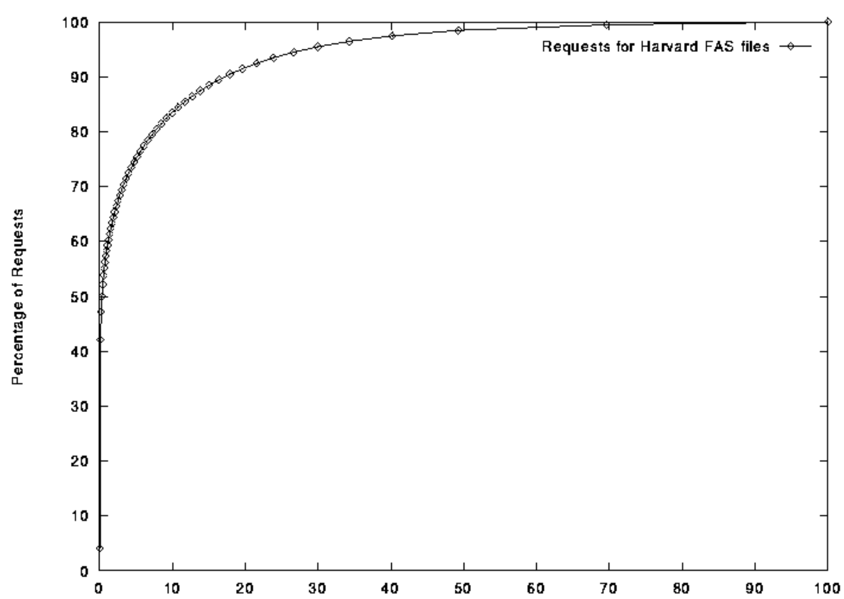


Figure 4 The cumulative distribution of the percentage of files at which a user looks for the Harvard FAS site. Users do not visit an academic site to browse through all student pages. The user is either attracted to University information, or a one student's Web pages.

Figure 5 The cumulative distribution of requests to the percentage of files on the Harvard As site. Beyond the extremely popular University pages, the requests spread evenly across the remainder of files. Each students' Web page has a limited audience because their sites cannot exceed 4MB.



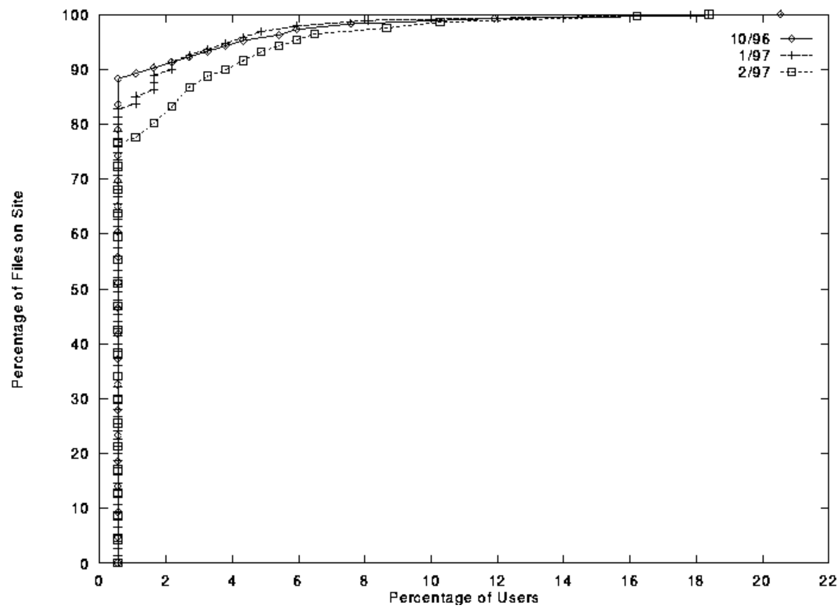
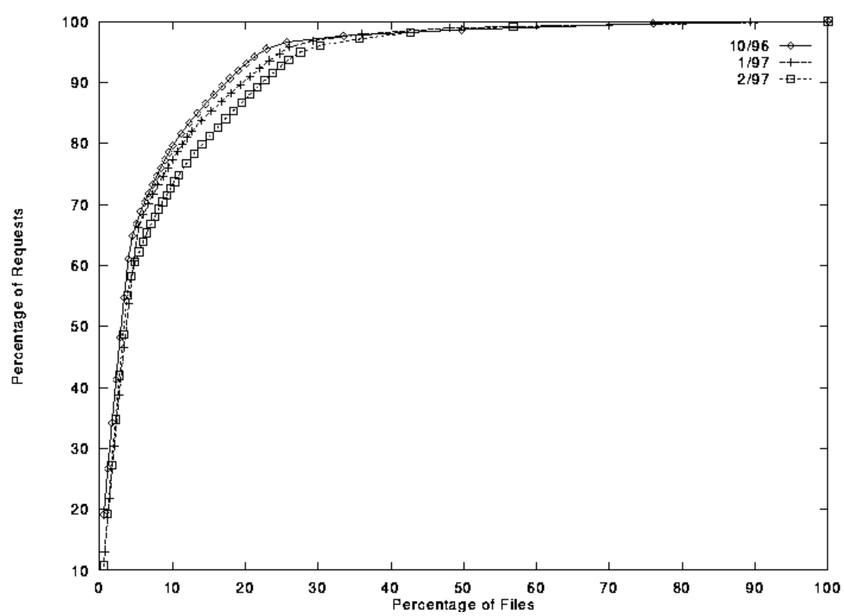


Figure 6 Cumulative distribution of the percentage of total files accessed per user for the Government Site, over time. The site tends to be explored more fully by users over time, to gain more information about the agency's activity.

Figure 7 Cumulative distribution of percent of total requests for each file for the Government Site over time. Due to the users interest in the content of the site, the pages approach more uniform popularity over time.



look at large portions of the site. The interesting traffic pattern to note, however, is the tremendous effect caused by charging for services and information on the Web.

The final pattern of growth can be seen in a site which has ceased to exist - the adult entertainment site. The business model of the site is like many on the Web - a limited subset of free material followed by a request that the user send money to get access to the remaining material. With the tremendous growth of the Web, and almost ubiquitous nature of such sites the best means of attracting users is through the search engines. Not surprisingly, the site experienced growth of almost a factor of 60, when it began to receive requests that were traced to the most commonly used search engines - Alta Vista, Yahoo, and Excite. The sites' popularity began to decrease, however, without the site or user access patterns changing drastically. The number of unique users began to drop. Figure 9 shows that the patterns of growth of user requests mirrors that of the negative growth of references from the Web's search engines. Those sites that depend on search engines for rapid growth can also suffer a rapid downfall, when the search engine does not return the site's URL as one of the best matches of the search.

These six growth patterns are easily distinguishable, and can help categorize different types of growth of Web sites. By examining the rate of change of a site, user behavior, and general growth, one can begin to classify different types of sites. Of course, these categorizations are very general, and not necessarily complete. Furthermore, within each large category, one can make distinctions based on the types and sizes of files on a site. However, such distinctions require far more subtle analysis. Most sites will have a combination of all types of files, in multiple sizes, so differentiation by such measures can be difficult to accomplish.

3.4 CGI

The perceived importance of CGI has motivated a great deal of server development. Both Netscape and Microsoft have changed the interface for CGI traffic, to improve performance. Microsoft's site includes two servers dedicated to processing their equivalent of CGI. Yet, with all of the clamor, most sites derive little functionality from CGI. Table 8 shows that most servers process very little CGI traffic. In fact, only three sites handle more

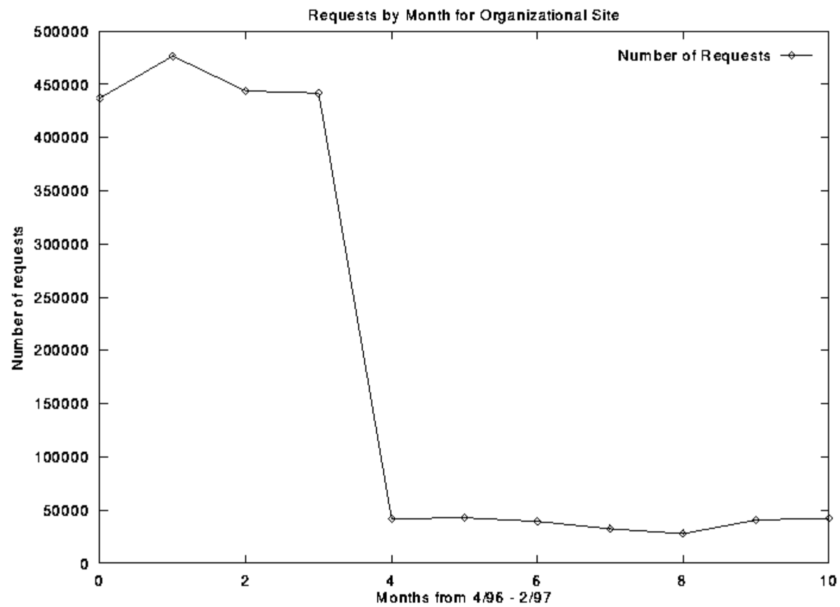
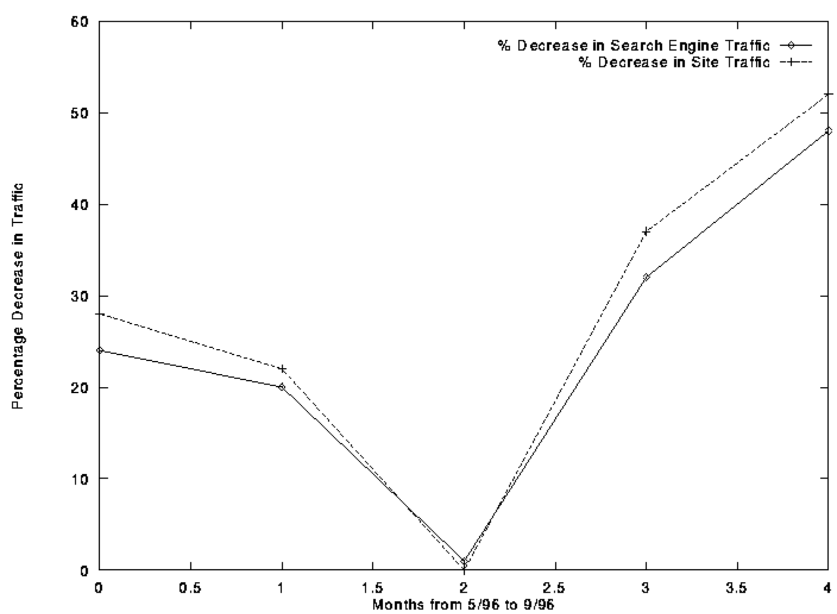


Figure 8 The Organization Site shows a dramatic instance of negative growth -- a negative whomp. When the Organization Site began to charge for access, the popularity of the site diminished greatly.

Figure 9 The Relation of References from Search Engines to Requests on the Adult-Content Site. As the number of references from the major search engines decrease, so does the site's popularity.



than 2% of requests as CGI. The types of programs, known as CGI scripts, that sites use is

Site Name	%Requests as CGI	%Bytes Transferred from CGI
Traditional Business	1	0.4
Harvard EECS	8	15
Harvard FAS	1.4	1.6
ISP	0	0
Rice ECE	0	0
Adult Content	2	0
Organization	34	62
Web Site Designer	1	0
Government Agency	0	0
Free Software	10	5

Table 8: The % of requests due to CGI
The amount of CGI traffic on most servers is trivial.

just as trivial as the traffic generated by these scripts.

Most of the sites handle a trivial amount of CGI traffic, and the set of functionality reflects this fact. The counter is a universally used CGI script, and access to the counter constitutes all of CGI traffic on most of the sites. Some sites also use CGI to provide redirects when a page has moved. Both of these scripts generate very little data, explaining why, in most cases the server handles a negligible percent of total bytes due to CGI. The free software, adult content, and organizational site all have used CGI scripts to allow users to login to the site. While the software site does not charge for its software, it does require users to register. The traditional business and the FAS site include a search engine. The only site with a great variety in CGI scripts is the Harvard EECS site. Unlike FAS, where the students cannot write CGI scripts, EECS allows many users to have access to the CGI directory. Thus, a wide variety of scripts reside on this site, many of which generate a large number of bytes to be sent to a user. The organizational site's CGI traffic can be traced to its unusual decision to force all requests through a CGI driven interface. Finally, the level of CGI traffic has not exhibited any growth on the sites. CGI has found its niche - with limited functionality and mostly trivial use.

3.5 Persistent Connections

Persistent connections modify the HTTP protocol to reflect the changes in the Web since its inception. Web pages have changed from a single text files written in HTML to major layout projects, with limited amounts of text in conjunction with multiple inline images, ranging from buttons to animations. As a user downloads the main page, a typical browser will begin making requests for the inline images - one connection per request. As a result, many connections are generated in the retrieval of a single page. By downloading multiple files with one persistent connection, the resource demands decrease. Clients generate fewer connections, leading to less traffic to setup and destroy connections, and fewer connections on the server. The general traffic patterns reveal that “Not Modified” response headers can comprise as much as 25% of total server responses. Setting up a separate connection for each of these responses can conceivably require more time and data packets between machines for connection overhead, than is actually generated by the server’s response. As is the case with any new technology, however, there are questions as to how best to optimize the use of these connections. Servers need to set a limit on how long a connection can be idle before closing it. When connections begin arriving so quickly as to overwhelm the number of connections currently open, the server may choose to close connections before time out, a dynamic time-out. Therefore, to identify the most important issues facing persistent connections, I simulated the effect of persistent connections on a server, using the logs of data.

To demonstrate the effects of different time out lengths for handling persistent connections, I have run simulations on many different types of server access logs. My method of simulation builds on Mogul’s design. The simulation allows for specification of the time out interval, the maximum number of connections a user is allowed to open, the maximum number of requests the server can leave open, and the algorithm for implementing the dynamic time-out.

Each request from the log sends the connection state machine through a set of tasks. First, the server reads the request from the log, and closes all connections that timed out before the timestamp of the request. Next, the simulator handles the user’s request The client, defined by machine IP address, either generates a new connection to handle the request, or re-uses an old one. If the request comes within 2 seconds of the most recent request, the client attempts to generate a new connection. The choice of 2 seconds can be changed in the simulator, but was chosen as a reasonable amount of time for a server to send a request to a remote client, and the client to send another request. A client will most likely generate requests that rapidly only for multiple file in a page. In this case, the standard browser behavior is to open multiple connections. If there are no open connections, a new connection is also created. However, if the request arrives more than 2 seconds later than the most recent request, and a connection still exists, the client re-uses the open connection. If, when the client tries to open a new connection, the server does not have any open connections available, the dynamic time-out closes a connection. The algorithms used to close connections before they time out are: least recently used connection, most recently used connection, and random connection closure. The realism of this simulation is not complete. First, it assumes that each IP address can only create one session for downloading documents from a server. Second, the logs are biased toward browsers that make four concurrent connections, the standard Netscape browser behavior. Such a bias could create side effects when analyzing only one or two open persistent connections per client. Finally, the simulation assumes no latency in handling requests. As discussed in

Name of site	Date simulated	Time period	Number requests
Traditional Business	2/28/97	24 hours	11,549
Harvard FAS	2/28/96	4 hours	16,741
Free software	2/28/97	1 hour	26,574

Table 9: The sites for which persistent connection handling was simulated, and the chosen traffic segment simulated

section 7, latency varies by users, so a general analysis of persistent connections cannot depend on one definition of latency. Therefore, these results will tend to be pessimistic about the effectiveness of persistent connections because connections will be closed before future requests can be generated. On a real server, this would not be the case. The estimations on the number of concurrently open connections will tend to be small, for the same reason.

Since the salient issues of persistent connections have not been identified, much less debated, I ran the simulations with multiple configurations for different types of Web sites.

The table details the logs used, the dates and time periods which were run, and the number of requests processed. The time periods for which simulations are run differ across tests because the levels of traffic vary so greatly. For each site, four sets of each time period were selected. The data presented is indicative of that seen in the other tests. The data collected reflects the number of requests handled per open connection, the maximum number of open connections, and the number of connections closed before the time out. For each of the sites, there were three comparisons being made: effect of different time outs, effect of different numbers of maximum connections, effect of different policies on closing connections early.

The first focus of the analysis was to simply discover the benefits of different lengths of time outs. Thus, the simulator was run with no limit on the number of open server connections, so the only way in which a connection would close was by timing out. Furthermore, at each step, the browsers made only one persistent connection - enabling the highest degree of socket reuse. Figure 10 shows that the improvement in performance due to longer time outs drops rapidly after 15 seconds. The primary benefit of persistent connections come in downloading a single page. A longer time-out produces little benefit. On these sites, the users who download only one or two pages tend to outnumber users who explore the site more fully. This type of behavior also leads to a linear increase in the number of open connections. In fact, during the simulation of the site with the heaviest traffic, a three minute time-out led to over 300 open connections.

A more realistic analysis of persistent connections requires setting a limit on the number of open connections. Mogul focused his analysis on 10, 64, 128, 1024 open connections. The data on the maximum number of open connections shows that the server never opens as many as 1024 connections. In fact, only Harvard's FAS site and the free software site ever have as many as 64 concurrently open connections. From here, the analysis focuses on the free software site. Its heavier traffic enables a better analysis of the stresses which could significantly affect persistent connections. Not surprisingly, the data shows that, regardless of the connection limit and level of traffic, closing the least recently used connection leads to the best performance. More surprisingly, as can be seen in Figure 11, whenever the time-out length leads to more active connections than permissible, an increased time-out interval provides no improvement. The dynamic time-out effectively

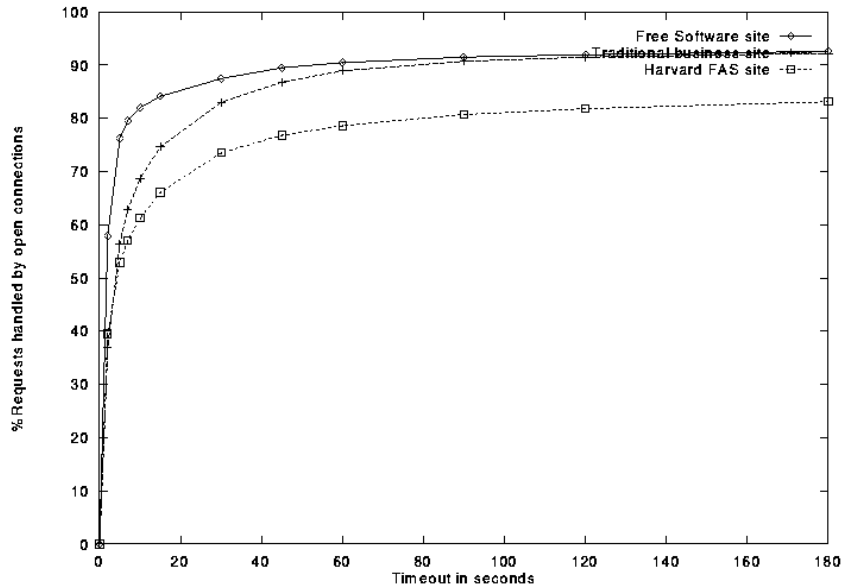
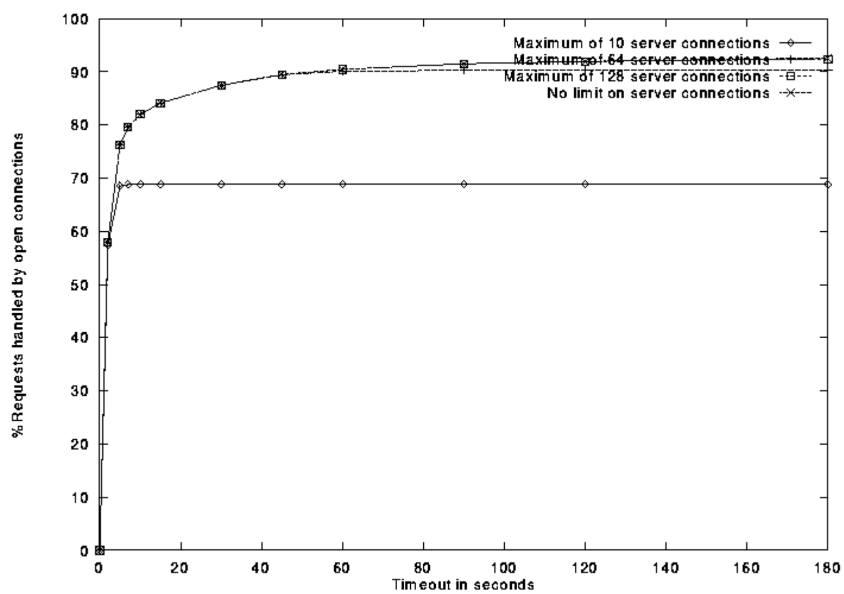


Figure 10 The percentage of total connections handled by open persistent connections, with no maximum connection limit. The persistent connections handle a large percentage of the requests on the server, but extending the time-out beyond 15 seconds yields little benefit.

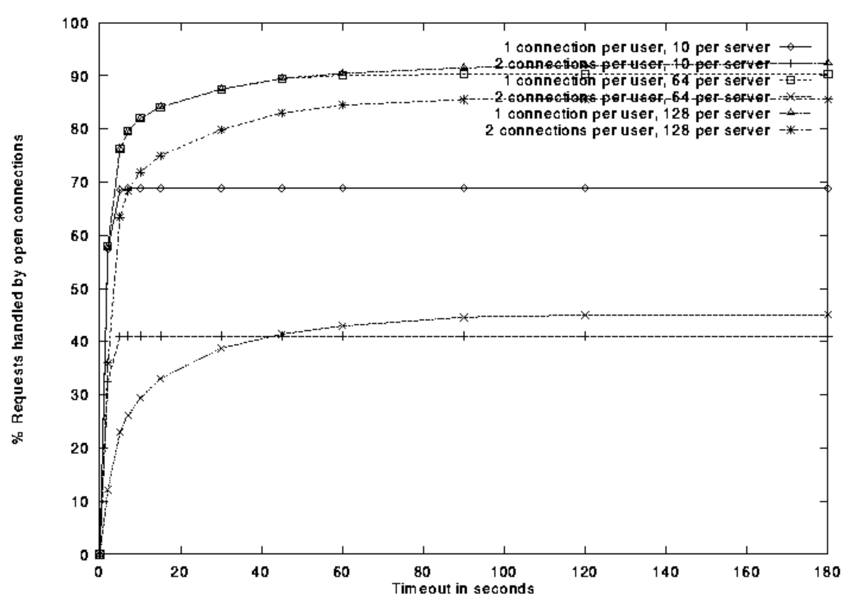
Figure 11 The percentage of total connections handled by open persistent connections on the free software site, with a variety of maximum connection limits. The connection limit can force connections to close prematurely and hurt the performance of the persistent connections. Still, 15 seconds provides most of the benefits.



implements a shorter static time-out because the connections were closed so frequently. The complexity of the dynamic time-out is unnecessary. Furthermore, the data for the heavily loaded software site shows a general decline in the performance of persistent connections from 64 to 10 server connections. The lesson to be learned is that one should permit as many open connections as the server would make for the chosen time-out, or performance will be the same as if the time-out were reduced.

Allowing one persistent connection per client provides the best possible performance gain. However the Web protocol standard, HTTP/1.1 [7], permits clients can set up two persistent connections to a server. The trends detailed above are only presented in the context of one connection per client, but one would expect similar patterns for two connections, with mild performance degradation. Figure 12 reveals a disturbing drop in benefits derived from persistent connections. Due to a side effect of open connection limits, the actual level of degradation of performance exceeded expectation. The side effects of allowing users to open two connections are very harmful. Since the site is so heavily loaded, allowing two connections per user doubles the number of open connections on the server. Therefore, at 10 and 64 connections, the server starts closing connections much more quickly than in the original model. As discussed before, this behavior has very

Figure 12 The performance of two persistent connects per user compared to one persistent connection per user. Allowing the user to create two persistent connections leads to the premature closing of many connections, and a degradation of performance



adverse effects on performance. Only by comparing the performance of the server allowing one connection per user, with 64 total connections, to the performance of the server when allowing one two connections per user, with 128 total connections, does one see the expected limited degradation in performance. Of course, such a decision does not come with costs. ISPs charge clients for extra connections, the implications of requiring the server to retain twice as many connections has serious implications for the cost structure of service provision. By allowing two connections per client, the server does not show the same benefit as in the ideal case, and it must also maintain more open connections, to prevent even more drastic performance degradation.

3.6 Caching

Web caching has become an increasingly important issue. As Gwertzman and Bestavros discussed, a server that can distribute copies of its data to external proxy caches, can reduce its load, as well as bandwidth consumption on the Internet. The storage needs of such a cache also play a significant role in the future of caching policies. The question underlying web caching is: “How much data need be cached to satisfy some percentage of requests from an ideally maintained cache?”

The computer community has typically looked for a skewing ratio of 90% to 10%. In the issue of caching, a good skew ratio would show 90% of requests downloading 10% of the unique bytes on the site. Figures 13 to 16 show the four different sites’ change in skew over time. These patterns correspond to the first four types of growth observed in section 3.2. The software site demonstrates the increased skewing due to its enormous growth in popularity for a few of its files. The traditional business' site demonstrates significant degradation in skew. The significant increase in the size of the site, in conjunction with its users showing interest in a wide variety of the site's features lead to its rapidly diminishing skew. The Harvard FAS site demonstrates consistent skewing over time. While the site increases in size and number of documents, users tend to access the new pages with the same patterns as the old. The final change in skew comes from the government site. Its skew decreases over time, though less drastically than that of the traditional business site. Since the users tend to access large portions of the site, one document does not comprise a significantly greater percentage of requests than the others. As users tend to explore more

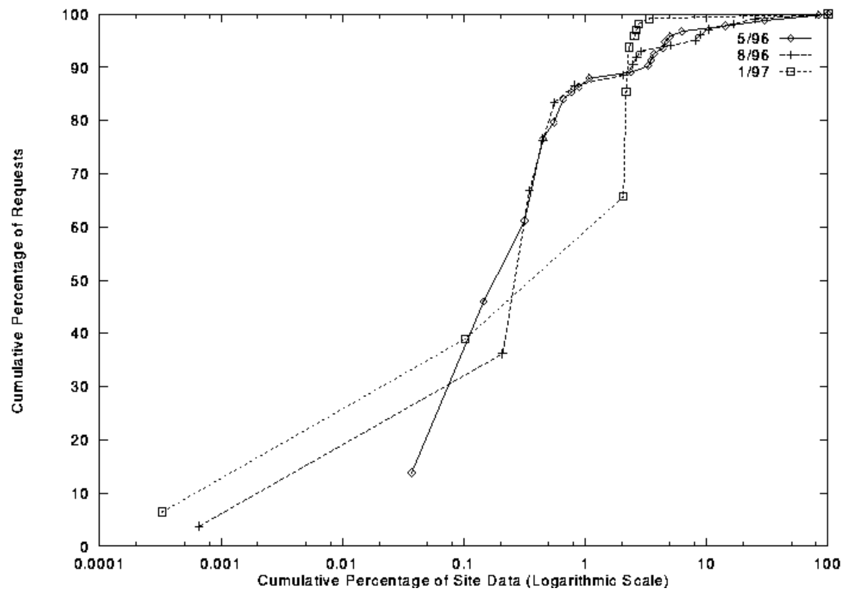
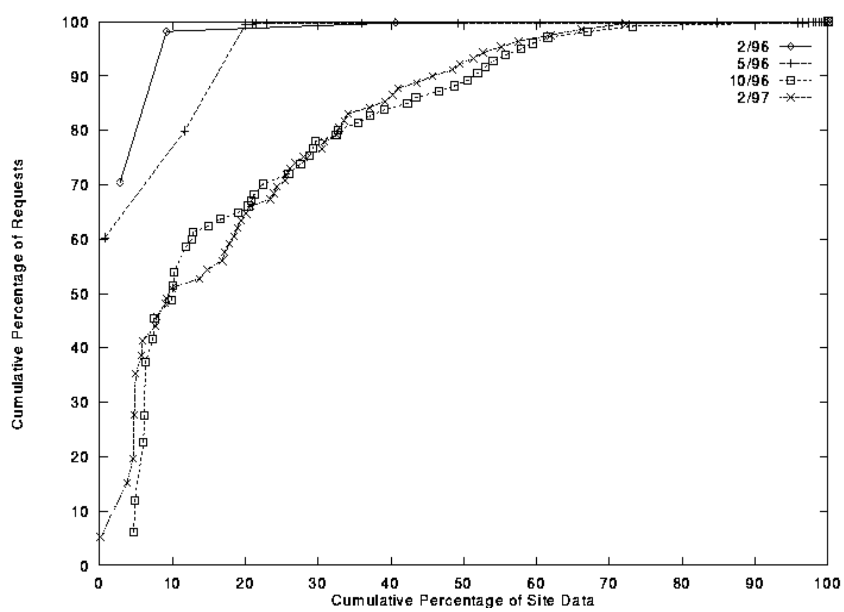


Figure 13 The Pattern of Skew on the Software Site over Time. Since the software site continues to get users who access a few pages and leave the site, its skew continues to increase.

Figure 14 The Pattern of Skew on the Traditional Business Site over Time. The traditional business site has different segments of popular pages. Therefore, as the site expands, more new users lead to diminished skew.



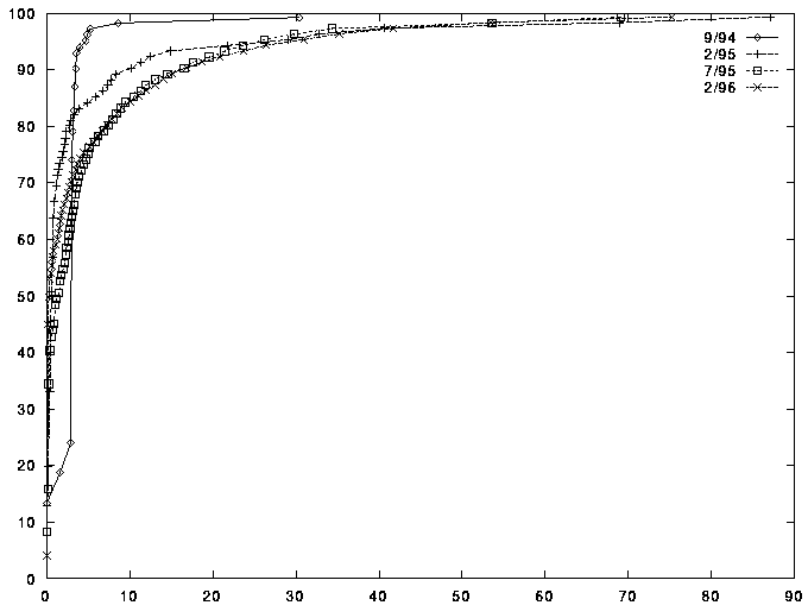
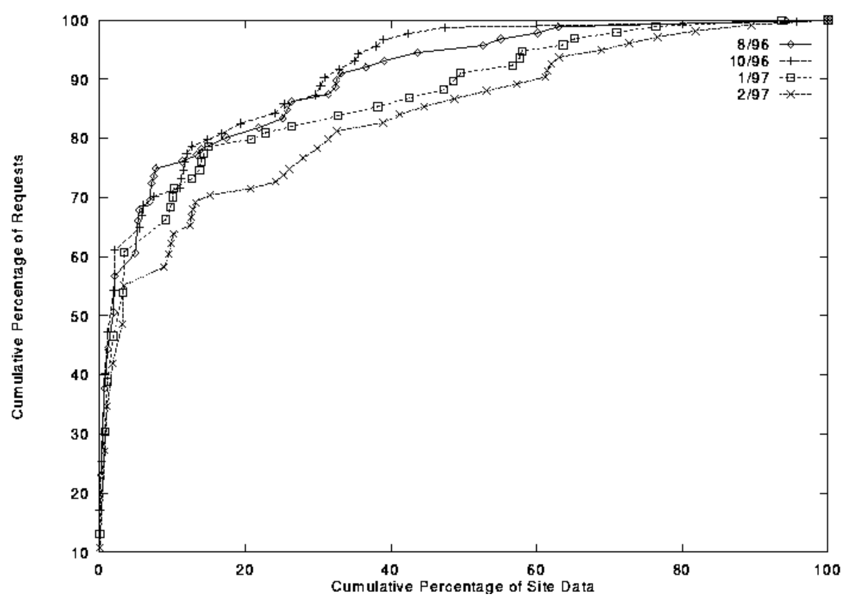


Figure 15 The Pattern of Skew on the Harvard FAS Site over Time. The pattern of skew does not change significantly because new users behave like old users, and the central page continues to be a locus of exploration.

Figure 16 The Pattern of Skew on the Government Site over Time. While the site does not change, users access more of the server. Therefore, skew decreases, though not as markedly as the traditional business site.



and more of a largely unchanging site, the skew will continue to diminish. Skew patterns such as these are discouraging. Only the site with a defined path of popular pages shows increasing skew. Even more disturbing, however, is the fact that only one of the sites examined shows a skewing pattern within the 90% - 10% parameter. The most distressing trend in Web site caching, however, was discussed in section 3.2. The number of unique bytes residing on servers is generally increasing by at least 10% a month on most sites. The degradation of most sites skewing ratios, in conjunction with being well below the 90%-10% skewing ratio, and the exponential growth of Web sites, leads to disastrous conclusion about the needs for caching. Table 10 lists the number of bytes necessary to maintain a cache to satisfy 90% of requests, and the growth of such needs, in the most recent 6 months of the logs. In some of the cases, the physical requirements seem small, but when

Site Name	Size of Cache to handle 90% of Requests (MB)	6th month % growth of cache	Projected size of cache in 6 months (MB)
Traditional Business	1.28	6	1.36
Harvard EECS	39.2	410	200
Harvard FAS	74.2	147	183
ISP	1.05	293	4.13
Rice ECE	13.34	-40	8
Adult Content	0.88	39	1.22
Organization	0.1	-80	0.02
Web Site Designer	0.063	0	0.063
Government Agency	0.713	13	0.81
Free Software	89	14,300	12,816

Table 10: Necessary sizes of caches to handle 90% of a site's requests from cache, the growth in the size of such a cache over the past 6 months, as well as a projection of the necessary growth of the cache for 6 months in the future.
The necessary growth in cache size to satisfy traditional cache requirements cannot be supported.

ISPs host multiple sites on a machine, the cache requirements begin to add up. Furthermore, as the requirements for caching grow exponentially, it won't take much time to overwhelm available hardware resources.

The traditional approach to caching will be ineffective. The tremendous growth in resources necessary to maintain at cache hit rate of 90% will require too much hardware and maintenance for the goal to be feasible. As an alternative, I propose a theory of caching that looks beyond typical views of caching, to integrate knowledge about caching and the Web. Traditionally, the concerns of caching have focused on scaling hardware capabilities to satisfy a specific cache hit ratio. The Web's growth prevents an effective allocation of hardware resources to meet new levels of demand. Furthermore, the inherent differences in traffic patterns among sites, and even within one site's history, make any choice of a cache-hit ratio completely arbitrary. My theory builds upon the notion of a Web site as being driven by a small subset of the site's files. In general, users access sites through the front page. Even when entering by means of a link to within the site, users find a way to the index page. As hit rates increase, these pages will still tend to receive a greater number of requests because these pages are used as a starting point, especially when users bookmark the front page to check for new developments. Furthermore, a site tends to have pages that are extremely popular for a limited amount of time, and these pages will lead to a large number of requests. Finally, many sites tend to reuse images such as including the same buttons or table of contents image bar on each page. The effects of these tendencies lead to a small subset of pages receiving a fairly constant percentage of requests over time. Therefore, by caching the limited subset of the site's “key” pages, one can maintain hit rates over a 6 month period. Within six months, a user can be expected to add more memory to the cache, to improve performance over the remainder of the year.

To test the efficiency of my hypothesis, I examined the performance of a static-sized cache over a six-month period. At the first month, I set aside cache space for 10% of the bytes on the site. While each site has a different size cache compared to the others, the size of each site’s cache remains constant over 6 months. For each month, I record the percentage of requests handled from the given cache size. The choice of 10% of system bytes as the basis of the cache was guided by the desire to include all of the key files, without demanding excessive amounts of hardware. Naturally, caching 10% of the bytes does not provide the optimal performance for each site, but it is unrealistic to expect a system administrator to be able to identify optimal cache size. Furthermore, the choice of 10% of system bytes provides very good cache performance, so the figure is not unrealistic. The data depicted in Figure 17 shows the effectiveness of the modified caching scheme. While the hit rate does vary with the type of site, the percentage of total requests handled by the cached data remains largely constant over the six month period. With the sites more than

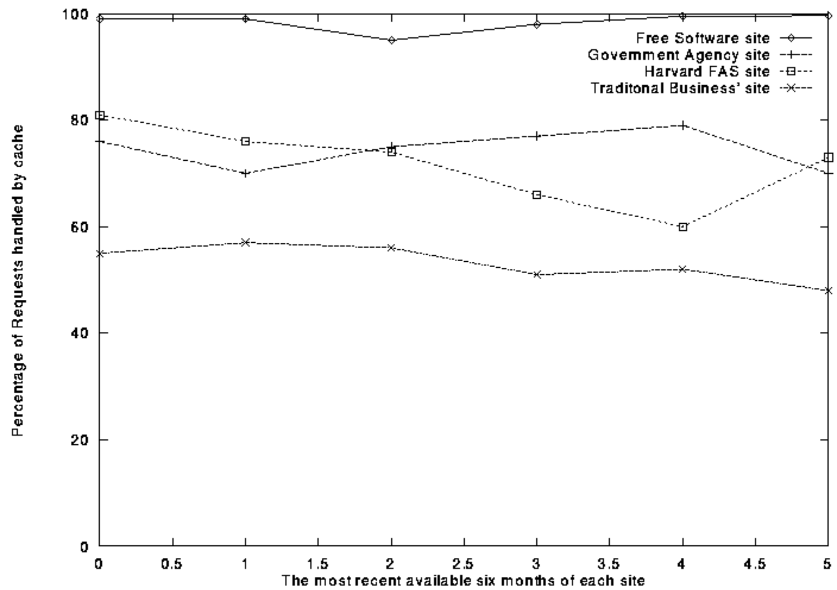


Figure 17 The Performance of a simple stable cache over time. The cache maintains its hit rate over time, despite the exponential growth of sites and traffic.

doubling in size over the six month period, and the number of requests showing even greater growth, the scheme is remarkably effective. Only the Harvard FAS site shows significant degradation in cache performance. An academic site will benefit least from this scheme because the site is not centrally controlled. Many users do not want to enter through the Harvard University main page, but would rather view a student's page. The large number of internal users will be especially prone to generate traffic in this manner. Most students set their browsers to access their home page at start-up. Moreover, one does not gain the benefit of a webmaster constantly reusing images, since each student constructs his or her own page. Still, hypothetical cache performance did not diminish unacceptably rapidly, and it did improve in the last month, as some pages' popularity increasingly dominated traffic. This theory of caching creates a new perspective on a complicated issue. Researchers have been working with increasingly convoluted caching schemes, with unacceptable hardware demands in an attempt to reach an artificially chosen cache hit ratio. The elegant simplicity of accepting the behavior of a Web site and creating a simple, and small cache, provides reasonable cache performance at little cost.

3.7 Latency

The latency in downloading World Wide Web pages has plagued and frustrated everybody who uses the Web. People often cry in vexation, “The server is so slow!” or “The server must be overloaded!” Many users refuse to wait tens of seconds to download one Web page. Since people who create Web sites want to ensure that clients use the sites, the issue of latency is paramount. Once again, by using the ISP’s modified logs, I can extract information about the latency in a server’s handling of requests.

Server latency has been attributed to four factors: number of non-deleted connections, number of concurrently active requests, CGI requests, and the size of the file requested. Unfortunately, the issue of number of open connections only applies to setting up the connection, a figure not included in server access logs. However, one can address the issue with an indirect approach. The theory that number of connections affects latency asserts that searching through the operating system control block hash tables consumes a great deal of server time. This latency, then, would affect all other requests, since the server is forced to use its CPU cycles to setup requests, rather than service them. A more salient limitation in this analysis arises from the ISP's use of virtual hosting. The final limitation

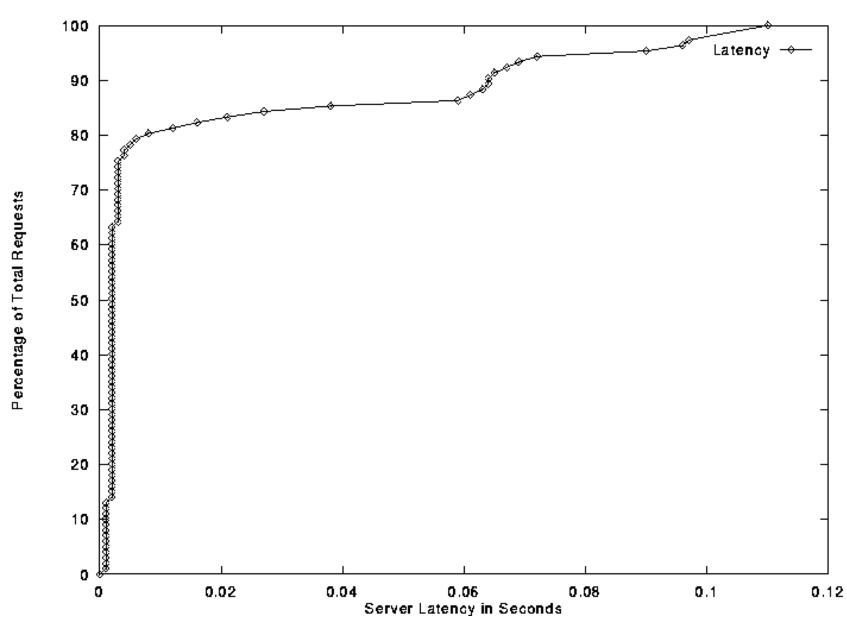
Site name	Requests in 15 minute test period
Government Agency	48
Traditional Business	233
Free Software	6722

Table 11: Description of the sites and loads used for latency analysis
The analysis will focus on the software site, which handles over 500,000 requests a day.

in the analysis comes from the way in which the server logs latencies. The latency calculation begins at the time the server begins to handle the request, which omits the time to setup the connection and send the request. The server then logs the latency as soon as it has written the last 8 KB of data. This leads to the omission of the final latency to transmit the data from the server to the user. Despite these limitations, however, the data still indicates that the server cannot be the source of latency in Web requests.

In analyzing server latency, I chose three types of logs, based on their different orders of magnitude of load. Table 11 provides a description of the sites and traffic used. For each log, I focused on four individual 15 minute segments of traffic from Feb. 28, 1997. The requests fell into two distinct categories. On the ISP's servers, any response is divided into 8 KB segments. Until the last segment of data, the server writes 8KB of data into the operating system's network buffers, and waits until the user has received all of the data. Then, it moves to the next 8KB segment of data. For the final segment of data, the Web server writes the data to the network buffer, and then returns, assuming the operating system will handle sending the data to the client. The time spent handling the final buffer can be defined as the latency of the actual Web server because it eliminates the effect of the operating system, the Internet, and the user from the latency. Therefore, for all requests that require sending fewer than 8 KB in total, the latency recorded in the log shows the latency of the server in handling complete requests. By examining only these latencies, the server software's role in latency can be shown as minimal. Figure 18 shows that the server handles 50% of requests in 1 ms, and all requests in under 1 second for each of the sites. This data is only available for responses under 8 KB, since the latency of larger requests reflects the influence of the Internet, the user, and other factors. However, since the server handles each 8KB segment of response separately, and the sites' logs included transfer

Figure 18 Server latency for requests under 8KB for the Software Site. The server is not the cause of latency in this case, as all requests are handled in approximately one-tenth seconds.



requests ranging from 200 bytes for a header to an 8000 byte file, the data is indicative of server performance. While it would be ideal to be able to separate the latency due to the server and non-server factors on all requests, we can only do so for requests for files smaller than 8KB. Still, the available data convincingly show that the server does not cause latency, even on a site handling over 500,000 requests a day.

Analysis of the latencies for files greater than 8KB reveals that the server's operating system and hardware cannot be responsible for the latency in handling Web requests. The latency figure for files above 8KB includes the other factors determining the actual latency that a user experiences. Since the server cannot send a new segment of data until the original segment has reached the client, the latency includes the role of the operating system, Internet, and local network of the client for all but the last block of the file. Figure 19 shows that the server response can take as long as 10 minutes. These latencies reflect the time most people actually wait for a Web page. Comparing such a disparate set of latencies will not yield interesting results because the effect of different file sizes will distort latency. All other factors held constant, a response of 64KB will have longer latency than one of 32KB. Therefore, to analyze latency, the most interesting metric is that of byte-latency, which is the average time necessary for a client to receive a byte of data from the server. Since the latency for the last segment of data has been effectively pared down to milliseconds, the size of transfers are reduced to the closest factor of 8 KB to determine byte-latency.

An analysis of byte-latency shows that the servers, regardless of load, face the same levels of latency. Figure 20 shows the general distribution of byte-latency per request on the two most heavily loaded sites. The much more heavily loaded server performs comparably with the traditional business' site. On average, the software site handled 20 open connections per second, with a maximum of 48 active requests in a one second period of time. The data show that servers have no trouble handling what amounts to 7 requests per second with the same latency as a server that handles one request every three seconds. Each of the servers, though, displays a small percentage of requests that suffer from more than an order of magnitude less throughput. Overall, the range of throughput per request amounts to almost 4 orders of magnitude. The causes of these variations cannot be traced to the commonly proposed theories of server latency.

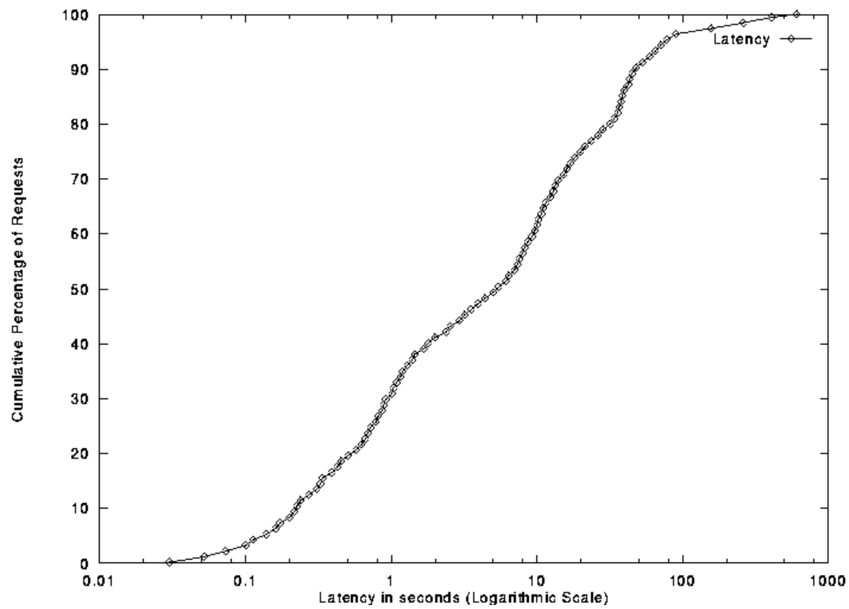
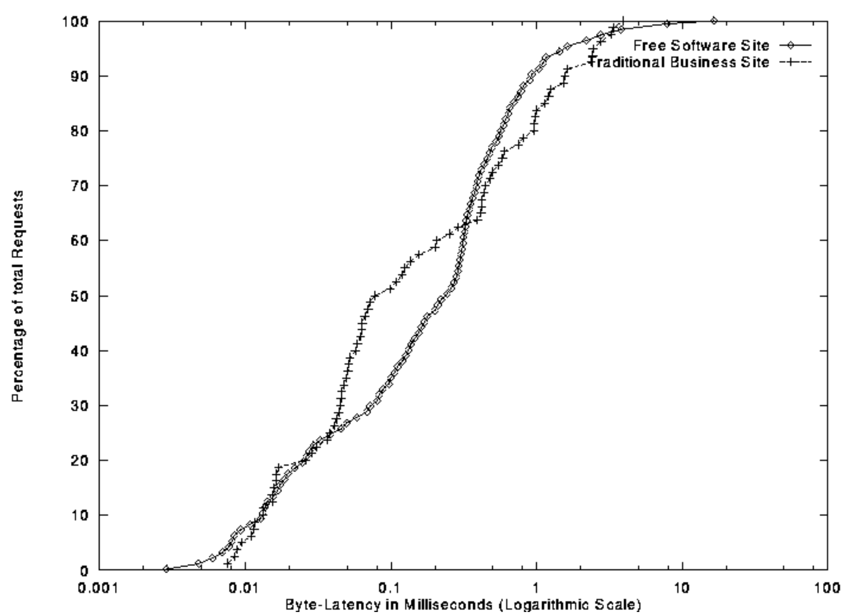


Figure 19 Cumulative Distribution of Latency for all Requests greater than 8KB on the Software Site. By including the latency due to the OS, the Internet, and user the latencies begin to resemble those that users really see.

Figure 20 Cumulative Distribution of Byte-Latency on Software Site and Traditional Company Site. Despite handling over 15 times as many requests as the Traditional Business site, the Software Site shows similar byte-latency as the less loaded site.



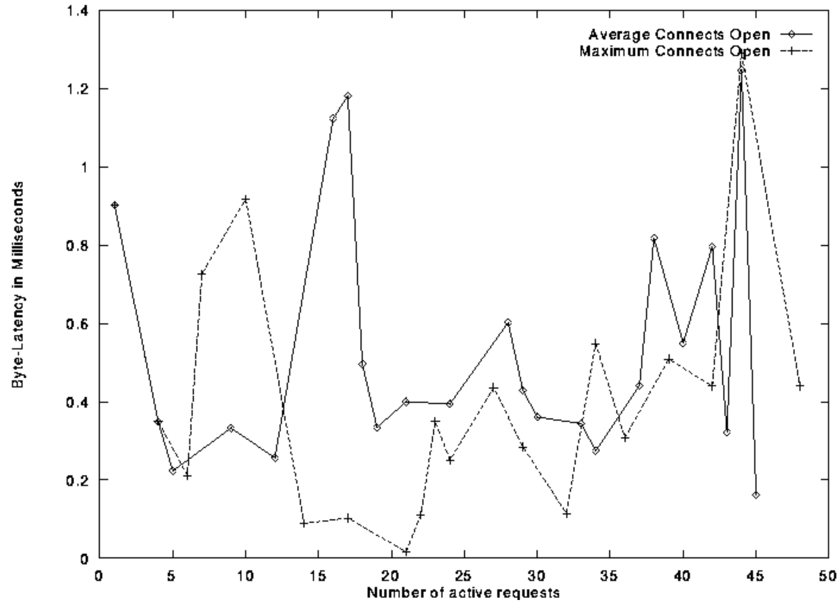
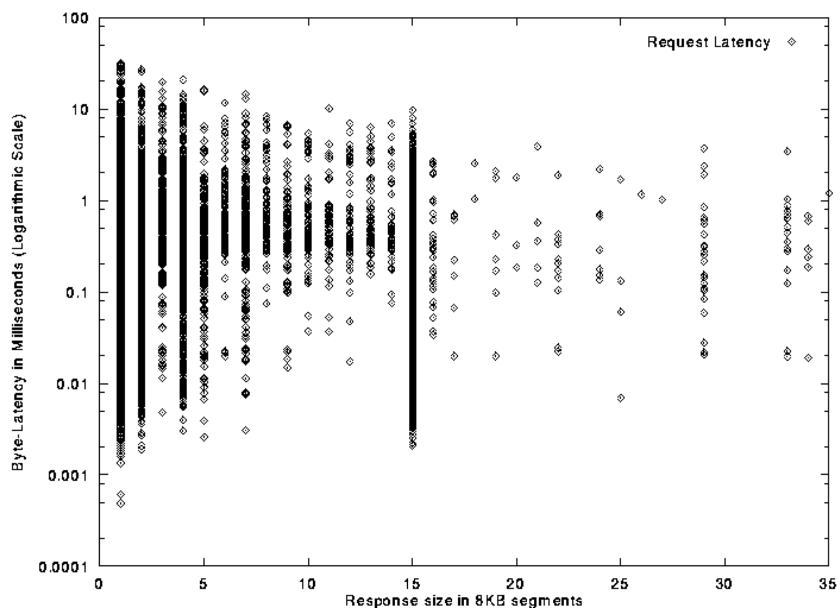


Figure 21 Byte-Latency as a function of average and maximum number of connections active while handling request on the Software site. Latency and number of active connections have no obvious dependencies.

Figure 22 Scatterplot of Byte-Latency as a function of the size of the response for all requests on the Software Site. Again, the variance is too large for there to be a simple dependence between file size and the rate of receiving data.



Regardless of the number of connections on either server, the load has no impact on the latency of handling a request. Even during spikes, the maximum number of open connections and server load on the two more lightly loaded sites does not approach the average load on the free software site. Since the average behavior of the free software site meets or exceeds that of the second, it is difficult to believe that load could cause latency as high as 40 seconds on the other server. In the following measurement, load is equated with number of active requests, since requests account for server load. Some may claim that number of dying connections also affects latency. However, over the time of the log, assuming a 2 minute time out, the number of closing connections remained at approximately 2000, with a variance of less than 10% in any second.

In analyzing the effect of load and open connections on server latency, there are two possible methods of measurement. One can measure latency as a function of the average number of requests active during the time the request was handled, or the maximum number of requests active at some point when the request was handled. To determine if there could be any correlation between the latencies of requests and server load, both of these possibilities were analyzed. There was no correlation between server load and latency. The data in Figure 21 shows that the behavior varies widely. Furthermore, by examining the set of requests that lead to a spike in latency, the latency can be traced to a small subset of requests handled with long latencies. The load has no correlation to latency.

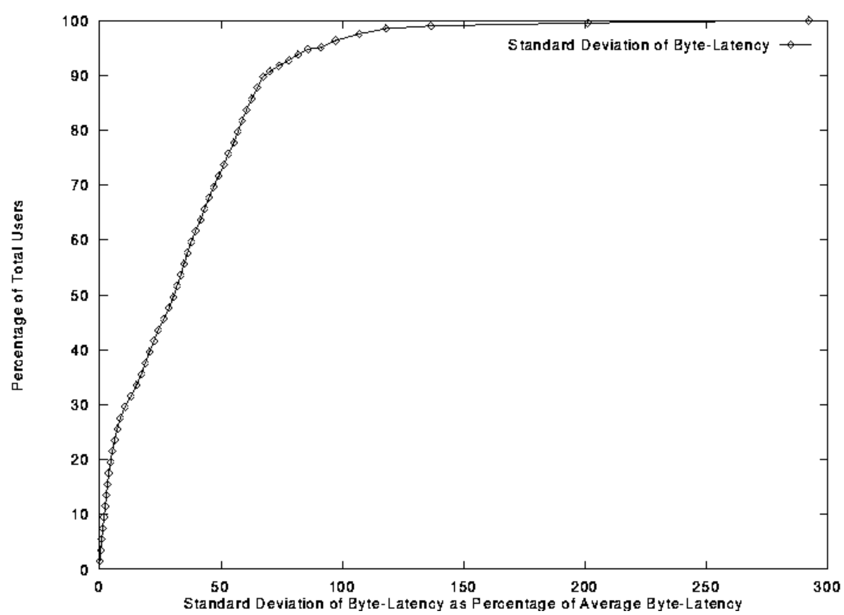
Another common myth declares that CGI traffic should cause a significantly greater latency in handling requests than traffic for static files. Unfortunately, for both sites, no CGI scripts generated a response greater than 8KB, so CGI was not the source of the latency. Furthermore, for response of smaller than 8KB, CGI was handled as efficiently as static files. However, this does not preclude it from affecting latency on other servers.

The file size also has no impact on the rates at which bytes are transferred from the server to the user. Figure 22 shows a scatterplot of all requests. Latency per byte has no correlation to the size of the file being transferred. Therefore, the logical notion that size of file affect latency holds -- if transfer rates remain constant, a larger response will cause greater latency.

The latencies of over 10 seconds cannot be explained by looking at the server. The server software has no difficulty in handling most requests in 1 ms. The data from the server logs shows that load, CGI, and file size do not cause the type of latencies seen in the logs. The data indicates that the causes of latency are traceable to the user and Internet. I have no data on whether clients are accessing the server by means of a modem or Ethernet. The state of the Internet and the routing path of request cannot be known. Still, since the logs include the IP address of the client machine, I was able to collect data on client behavior. The results corroborate my theory that latency cannot be attributed to the server.

While the various causes of client and Internet latency cannot be examined from server logs, the server data indicates that the cause of latency resides outside of its domain. An analysis of byte-latency per user reveals that, even for users who access the site over a period of five, ten, or fifteen minutes, the byte-latency remains constant. To quantify the level of constancy in performance, for all users who make more than one request, the byte-latency per request enables a calculation of average byte-latency over all requests, as well as the standard deviation of byte-latency. Figure 23 shows that the standard deviation of byte-latency per user never exceeds 290% of the overall average byte latency. On a log, in which users show variation of four orders of magnitude, the constancy of user behavior indicates that latency can be attributed to a non-server factor. Some of the possible options for client latency include: Internet backbone latency, local network latency, modem

Figure 23 Cumulative Distribution of Standard Deviation of Byte-Latency per User as a percentage of Byte-Latency. User byte-latency remains constant, indicating a dependence between the user and byte-latency



latency, or the latency of using a heavy loaded machine from which to generate requests. The causes of latency cannot be examined from server logs, but the logs do show that user latency across requests, shows remarkable similarity.

The results on latency indicate that servers may not be the bottleneck in the common case. While the load on these sites may not seem indicative of the types of loads experienced by Microsoft, Netscape, CNN Online, and ESPNet, recall that very few sites handle such extreme levels of traffic. Furthermore, servers which handle the load for these sites may be handling multiple sites per server. Therefore, while the load of one site may seem small, there could potentially be an order of magnitude more traffic per physical server. Regardless, the data show that a server handling over half a million hits a day can run Apache 1.1.3, and handle 90% of requests in under 0.1 s. The sites and data are both real and representative. The ISP running the sites is among the leaders in its field. The types of sites are diverse in traffic patterns and content. While there are a limited number of requests with large latencies, the data lead to no link between the server and those spikes of latency. Servers cannot be considered the primary source of latency on the World Wide Web.

Chapter 4

Design of a Web Server Benchmark

A comprehensive understanding of Web servers requires far more analysis and data than can be derived from logs. For example, new developments such as persistent connections need to be actively tested to understand the interaction of many persistent connections on a server. Such data could not be derived from logs, especially when persistent connections do not dominate web traffic. Furthermore, while current analysis shows that Web servers do not cause latency, this may not always be the case. As growth continues, and researchers reduce the latency due to other conditions, the server will become more important. Finally, some sites may actually need to test servers to ensure that the desired performance can be achieved. When money and business is on the line, companies leave very little to chance. Only a benchmark enables testing of a server under active, yet controlled conditions. Therefore, I propose a new self-configuring, scaling Web benchmark, that is based on Web server logs.

4.1 Philosophy of Benchmarking

The design of the new benchmark, hBench:Web, focused on improving past work in two areas: realism and statistics. For a Web server benchmark to be useful, it must generate traffic that resembles the load of real Web site. Without realism, the benchmark becomes as useful as an analysis of the performance of automobiles at either 120 miles per hour or 5 miles per hour. Neither of these metrics present interesting information to drivers. Moreover, if taken seriously, the automobile community would make optimizations on cars that do not affect real drivers. This analogy holds true for Web servers. Second, the benchmark must report meaningful statistics. To return to the car analogy, reporting the incorrect numbers can make a benchmark useless. If a testing group announced to the public that the new Ford Taurus used 0.01 gallons of gasoline to start the engine, and another 0.001 gallons to run the air conditioner for ten minutes, the announcement would be met with indifference. A benchmark must be able to sum up interesting results into a simple metric - such as the number of miles a car can drive per gallon of gas. Still, for producers, the more specific measurements must be made available, so that the causes of performance degradation can be found and improved.

The current sets of Web benchmarks makes little or insufficient attempts at creating a realistic, reproducible test. WebStone measures throughput with a five files on the server, with the average size of the files far much larger than we see in reality. They then report bandwidth figures for users who constantly download one file at a time, as rapidly as possible. This benchmark ignores the fact that Web sites have different distributions of files and that users rarely behave like WebStone clients. SPECWeb suffers from a similar problem. First, it bases its traffic on a limited set of sites that cannot be considered indicative. This weakness can be traced to SPECWeb not making any attempt to classify different types of sites as I do in Chapter 3. All sites are not the same, and most bear no resemblance to the traffic at the Microsoft and Netscape sites. Second, SPECWeb's simulation generates a target number of connections per second, by shifting behavior patterns further away from those of the original log. This same problem of dynamically mutating how users behave also plagues DBench. By changing reality, with no understanding of the side effects, the results from the benchmark become less indicative and useful.

Once a benchmark can generate realistic approximations of Web traffic, the statistics it collects become significant. Correct statistics, if focusing on unimportant issues, contribute little to the understanding of Web server performance. While throughput and connections per second provide interesting measures, hBench:Web focuses on the user perception of request handling latency. The logic behind the choice is simple. First, as argued by Endo et al[6], users care primarily about latency. From the logs analyzed in Chapter 3, 80% of users who experience latency of greater than 15 seconds for a request, immediately leave the site. Web users show a great deal of sensitivity to latency. Those who develop Web sites, do so to attract clients. As a result, if their clients cannot use the site effectively, the people who run the site lose money. Thus, both from a client and webmaster perspective, the importance of latency exceeds that of all other measurements.

4.2 Modeling Traffic from Server Logs

The analysis of Web server logs shows that Web sites and their traffic vary greatly. As a result, a realistic benchmark must be able to model a wide diversity of traffic patterns. Furthermore, as the sites change rapidly, a benchmark that creates static models of traffic patterns will soon become outdated. Therefore, a benchmark should be able to generate new models of traffic and constantly update those models. Finally, the benchmark must create simple and redistributable models of traffic. Since one of the values of a benchmark comes from being able to compare different configurations of hardware and software with the same test, the benchmark must be simple. To achieve these goals, I have chosen to implement a benchmark that pre-processes a real Web server log to generate a configuration that, when run through the benchmark, enables the benchmark to model the traffic described by the log.

Under the complexity of the technology and rapid changes on the Web, lies a simple model. The Web has been built on clients sending queries to servers based on the server's set of files. Servers then send information in response to the user. Clients include proxy servers, search engine robots, and individuals running on a LAN or modem. Regardless of the category of client, or the identity of individuals who are clients, each follow the same model - sending requests or posting data to the server. Meanwhile, each Web site also follows a similar model. Each site has some number of static files, as well as CGI scripts. These are the files and scripts that are accessed by users, at various intervals. As this model is simple, accurate, and constant, it is the basis of the new benchmark.

The models of traffic generated from Web server logs can be described by three simple parameters. First, the configuration script extracts the site's page set from the log. In constructing the set of files that constitute each page, the configuration script groups together files that one user accesses with a latency of fewer than 2 seconds. Since a user's browser will request the files that constitute a Web page as rapidly as possible, the only latency between requests for files on the same page is that experienced by users in receiving files. The 2 second limit was selected by doubling the latency by which over 80% of requests were handled in the logs analyzed in Chapter 3.7. Next the configuration script extracts user session profiles. A user is defined by an IP address, so that a proxy server or multiple clients on one time-sharing machine are considered one user. The session is defined as the list of Web pages, accessed, as well as the idle time between page accesses. If the user is ever idle for more than 5 minutes, the profile ends. The data indicate that if a user is idle for more than 5 minutes, he or she is likely to be idle for at least a day. In other words, if a user is idle for more than five minutes, that particular surfing session has been completed. The user profile captures all of the important information about a user: the requests made to the server and the idle time between each set of requests. From here, the benchmark can simulate actual clients accessing real Web sites.

The number of user profiles started per minute controls the load in hBench:Web. The number of users sending requests to a server in a given time period has a direct impact on the load experienced by a Web server. This element also conforms to standard Web behavior. Users arrive at a site, download information, stay idle, and then download more information. Eventually, that user leaves the site. While this user is active, however, other users send requests to the server. To effectively select the user interarrival time, or the time between the arrival of one user and the next, the configuration script divides the log into n/m sublogs where n is the number of seconds of traffic represented by the logs, and m is the time for which the benchmark will run. For each of these sublogs, the configuration process determines the average interarrival time. Then, the user can select one of these user interarrival times, by specifying the percentile time period for which it wants to generate traffic. Thus, if the user wants to simulate one hour's worth of traffic similar to the heaviest hour that the site experienced, he or she would choose the 100th percentile hour. Then, the benchmark generates users as rapidly as they arrived in the heaviest hour recorded in the log. The three factors of the model characterize a Web site by the most basic underlying factors of Web traffic -- users accessing a site.

4.3 Statistical Basis for Model

The model, while simple and intuitive, needs mathematical justification. At any given point in time, the traffic on a site can be described by the users and the files they access. This information is captured by the log. However, rather than selecting parts of the log to re-run, one would prefer to be able to create an abstraction of a full month of traffic, and be able to select different levels of load, without having to search the log for a segment that generates the desired load. The structure of the model forces the set of files and users to remain constant, while the rate at which users access the site changes. Therefore, the model can only be considered valid if each subset of traffic in a log exhibits similarity to the other traffic patterns, and the log taken as a whole.

In determining the similarity of different segments of the traffic, one needs to determine which aspects of traffic are most important to track. In this case, the focus should be on the type of traffic, since the amount of traffic is controlled by a non-static aspect of the benchmark. Five factors, other than load, characterize a site's traffic pattern. The similarity in the page set can be seen in the distribution of file size and file type by request. The similarity in user set can be determined by the distribution of number of requests per user. The similarity in general traffic patterns can be seen in the percentage of requests per method, as well as percentage of requests per server response("200-OK", "304-Not Modified", "404-Not Found", etc.). From these factors, one can determine whether the traffic on a site follows set patterns, regardless of the number of users accessing the site.

A chi-square test of homogeneity quantifies the degree of similarity between each of the factors of traffic similarity. The chi-square test of homogeneity can compute the goodness-of-fit between two multinomial distributions. A set of discrete buckets forms the multinomial distribution for this test. Two of the factors(method type, response type) already fit into buckets. Most file types are rarely accessed, so file types were divided into buckets of: HTML, CGI, images, and other. The file sizes were bucketized by being of size less than: 500 bytes, 1KB, 5KB, 10KB, 50KB, 100KB, 500 KB, and any size larger. The granularity of bucket sizes decreases as file size increases. This reflects the pattern of many small files and a few rapidly growing large files in the server logs. The number of requests by user are broken into buckets of size less than: 5 requests, 10 requests, every ten requests up until 100, and greater than 100 requests. The chi-square test then compares relative frequency of elements in buckets to create a metric for the similarity of distributions.

The chi-square test convincingly shows that a site's traffic patterns over a period of a week show little variance. For each site, the chi-square test shows over 99% similarity between ten sub logs and the overall log. Each sublog consists of 0.5% of the site's total requests. To extract the logs, a random point in the log was selected, and an appropriate number of consecutive requests were removed. Therefore, each sublog represents a traffic pattern of approximately an hour in each case. The traffic was then bucketized, and the chi-square test conducted. In each case, on each log, the chi-square test indicated that the similarity between sites was greater than 99%. Essentially, the test declares that the model of each sublog is the same because the variance seen in the different sublogs can be attributed to random variation over 99% of the time. This data indicates that the model of the Web site does remain valid for any given time period to be examined.

To further prove the model's accuracy, there must be a relationship between idle time between users and server load. The analysis of the constancy of factors such as number of requests per user, and the distribution of file requests by size indicates that the only way to adjust load would be to add users. As a verification, for each month of the logs, the log was broken into hour-long sublogs. In each month, the hour with the greatest number of requests always ran the greatest number of users in that hour. The same did not hold for bytes transferred. In some of the lightly loaded hours (the lowest 50% in the log), a few large requests led to more traffic that many small requests. However, the pattern held in all cases with significant traffic. The statistical analysis shows that the model of the monthly log corresponds closely to the smaller increments of traffic, and that the number of users does determine the level of traffic.

4.4 Reducing the Model

On a large site, the model can become extremely large. The set of pages can reach the thousands and consume hundreds of megabytes of space. For a benchmark, the act of reconstructing an entire fileset can be daunting. Furthermore, the space devoted to retaining user logs can become nearly as large as the original log - which can be as many as eighty megabytes large. Such a model can be unfeasible to retain, share, or even run as a benchmark. Therefore, the model can be reduced by means of simple heuristics.

The user can select from two methods of reduction of the file set. The goal of the first level of reduction is to combine all pages which come from a templated model. The reduction compresses two pages in which 75% of files are the same, and the remaining 25% of files are half or twice as large as its counterparts. This causes little change in the actual file set, except for possible side effects in changing the percentage of types of files, such as HTML, CGI, or image files. The next level of reduction leads to a great departure from the original model. Here, all pages which have 25% of files in common, and the remaining 75% are within a factor of two in size.

Two levels of reduction can compress the set of users. In reducing the user set, combining users merely involves incrementing a weighting factor, such that, when users are combined, the weighting factor becomes the sum of the original weights. When combining pages, the chosen user profile does not change its idle times between page accesses. In creating the userset, users with identical access patterns, those users who access the same pages in the same order are always combined. The first reduction tool combines all users who access the same pages, but in different order. The second means of reduction combines all users who request similar pages. Pages are defined as similar when they have between 50% and 200% of the same number of files on a Web page, and the sum size of these files on these pages are also between 50% and 200% of one another. Naturally, the second method of reduction significantly alters the original model.

4.5 Changing Traffic Patterns

While a realistic benchmark helps provide good analyses of current server performance, researchers, companies, and webmasters often want to test the effects of a new development. The simple self-configuring model of the benchmark allows the user to only set the load generated, and even then the load cannot exceed that of the original server load. Therefore, the basic hBench:Web can do little to help analyze the effects of changes in traffic patterns. To augment standard hBench:Web, a user can also specify the ratio of traffic for different types of requests. While users want the power to change the pattern of traf-

Category	Traffic Type to Specify
Method of Accessing Server	GET, POST, HEAD, DELETE, TRACE
Server Response	200 (OK), 302 (Redirect), 304 (Not Modified), 403 (Not Permitted), 404 (Not Available)
File Type	HTML, CGI, JPEG, GIF, Other
File Size	Less than 500 bytes, 1KB, 5KB, 10KB, 50KB, 100Kb, Larger

Table 12: Available means of altering traffic patterns
Note that each of these types of traffic are changed not at a file level, but by selecting user profiles with the desired characteristics

fic, the resulting model should still be based on real traffic patterns. Just as before, realism gives credence to the results, and helps identify the issues involved in the change of traffic.

By changing the ratio of user profiles, hBench:Web succeeds in providing a realistic model of altered traffic. Table 12 lists the possible means of modifying traffic. All of the available methods of altering traffic patterns involve changing the type or size of responses to the user. Ideally, one would be able to determine exactly the necessary user behavior to generate the chosen traffic patterns. In an attempt to apply such an approach, the changes to the model are all achieved by manipulating the set of user profiles. Each user profile models what an actual user does. The profiles, when combined, form the entirety of a site's traffic. Therefore, to change the traffic model, hBench:Web finds all user profiles with a set of requests that will shift the overall structure of the traffic toward the goal, and creates copies. The model includes only profiles of actual user access patterns, so only the ratio of user profiles changes. In other words, the benchmark models the increased traffic pattern as being caused by more clients who behave in a given way.

While this method retains much of the original model, there are flaws in the chosen method. First, if the desired level of traffic is not demonstrated by any user, the model cannot be modified to meet user specifications. In other words, if a researcher wanted to increase a site's CGI traffic from 2% to 50%, and no user profile includes over 50% of requests for CGI, the request will fail. Furthermore, if a limited number of user profiles does fit the requirement for the alteration to succeed, creating the altered model could lead to a negative side effect. The reproduced user profiles make requests other than those that are specifically desired. The small number of profiles, and their other requests, could dominate the model of the log, and shift non-desired types of behavior. In general, these types of shifts improve the reality of modification - a change in one type of behavior likely induces other changes. However, as the set of user profiles used to change the site's traffic decreases, the realism of the new model must be questioned. Despite the flaws, this method of changing traffic patterns fits within the framework of the model. It still retains the notion of users accessing files, and does not presume to generate an "ideal" user, or modify one user's behavior. Instead, it uses actual client profiles to provide a template for modeling the chosen traffic pattern. This method preserves the variety of side effects that come with changing a site's traffic and retains the other user profiles to preserve the variety of traffic on a site.

4.6 Scaling the Model to Predict Future Traffic Patterns

As discussed in Chapter 3, the one word that best characterizes the Web is "growth." A benchmark that cannot model growth suffers the weakness of being a rear-view mirror in a speeding car. If the benchmark can only model traffic that has occurred, then it suffers from the problem of being outdated. Furthermore, since many people like to benchmark servers and software on a extremely heavy load, the benchmark needs to be able to scale its traffic.

The scaling on hBench:Web uses historical server logs to project the future traffic pattern for a site. The benchmark requires data from at least three months of a site's traffic, and generates a model of the site's traffic for as many as six months into the future. The model of the benchmark remains unchanged - the future model must also be based on fileset, userset, and user arrival times. Of course, any model would struggle to determine changes in a site's set of files, and the behavior of the users. After all, each of these factors are controlled by individuals, who are far too complex to model. On the other hand, user arrival time can be based on a set of millions of Web users. One person's choices become insignificant in the face of such a large set of users. Therefore, scaling focuses on predicting the rate at which new users will access the site.

Site Name	Correlation in Dist.of File sizes	Correlation in File Type	Correlation in Dist. of Files per User	Correlation in Server Responses	Correlation in Client Methods
Trad. Business	99%	99%	95%	99%	99%
Harvard EECS	95%	97.5%	94%	96%	97.5%
Harvard FAS	98%	99%	99%	96%	99%
ISP	99%	99%	99%	99%	99%
Rice ECE	85%	99%	97%	99%	95%
Adult Content	**96%**	**5%**	**18%**	**96%**	**90%**
Organization	**5%**	**10%**	**21%**	**87%**	**5%**
Web Site Designer	99%	99%	96%	99%	99%
Gov't Agency	99%	99%	93%	99%	99%
Free Software	97%	99%	99%	97%	98%

Table 13: The result of a chi-square test for similarity of six months of sites
For virtually every site, the model of traffic scarcely changes over the six month period. Only the organization and adult-content sites, which completely restructured, show degradation in traffic patterns

To even attempt a model of future traffic the benchmark must do more than not change the model incorrectly. By generating more users, can one actually claim to model the future? As discussed in chapter 3, user access patterns change on sites, and the sites themselves change exponentially. Despite such changes, the general strength of the model stands. Since the original model has been built upon two of the fundamental aspects of the Web, the model remains accurate, even in the future. While sites change, and user behavior changes, fundamental traffic patterns do not undergo major transformations. To test the behavior of the model over time, a chi-square, goodness of fit test is conducted on the most recent 6 months of each server log. The test mirrors that conducted on the original model, except that rather than determining the similarity of many sets of traffic from one month, it determines the similarity of each month over one-half year. The results, shown in table 13, indicate that the fundamental features of fileset and user behavior remain constant. In fact, only the organizational site has changed significantly. As discussed in chapter 3, the webmaster changed the entire model of the site - charging for access and driving the site through a CGI interface. Still, even sites such as the traditional business, which “whomps” the site, do not change the nature of the site - just the magnitude. One potential weakness in the model is that it does not separate GIF and JPEG files. This enhances the similarity in the type of files requested for the software and traditional business' sites, which shifted to the new image type. However, since the analysis in chapter 3 showed that the average size of the file remained constant, the difference in behavior between the files does not make such a decision unjustified. The similarity in the downloaded file sizes further confirms this assessment.

Since the model of a Web site's traffic remains valid over a six month period, the model can scale. Since the manner in which users accessing files does not fundamentally change, the basic features of the model do not vary, the file set and user profiles will still model real traffic patterns. The user profiles may not be for users in the current month and the file set may be outdated, but the traffic patterns they represent are still valid. Therefore, to model the load on the site, one needs only to modify the arrival time of users. To generate the future arrival times, the benchmark extrapolates on the changes in past arrival times of users, with respect to the growth on the site. In determining the projected change of user arrival time, the configuration script computes the percentage change of that factor from one month's model to the next. The simplest solution would then be to average the growth patterns and simulate linear growth, but with the exponential growth and change on the Web, such a linear change would be incorrect. Instead, the weight given to past growth rates should decrease exponentially by the rate of growth of traffic on the site.

The strength of the original model enables the benchmark to scale to predict future traffic. Over time, the type of traffic on a site undergoes very little change. While the site grows, the type of content varies little. Still, even such a model can fail if external events significantly alter the site, as happened with the organizational site. Such a change emphasizes the assets of this model. The model has been built upon the most static elements of a Web site, to minimize the susceptibility to the rapid changes of the Web.

4.7 Run Time Environment

The most realistic model of Web traffic does not guarantee a good benchmark. The actual run time of the benchmark must use the strengths of the model to create realistic traffic and record accurate and meaningful statistics. The general structure of the benchmark builds upon the WebStone master-client relationship. Unlike WebStone, and the benchmarks that followed, hBench:Web adds a third element - the statistics daemon. At each point when a new user should be generated, the master randomly selects a user profile, and spawns a client process to carry out that user profile. The way in which a client requests from a server can be easily configured to match a range of browser behaviors. At the conclusion of handling the profile, the client sends its data to the statistics daemon. A script post processes the data to generate useful statistics.

The standard method of handling statistics causes a litany of problems found on other benchmarks. All current Web server benchmarks collect statistics at a client level because the client has access to the information about the requests it has made and the responses it received. On the standard benchmarks, however, after one request has been handled, the client then updates computes a few simple figures, such as the average latency per file, size of file and header, and the standard deviations of all of the above. After the client has finished generating requests, the client returns its data to the master. This approach causes a number of problems. First, when the master process should be generating new clients, it must instead read statistical information from clients. Second, with the clients processing statistics as they run, the client process cannot to generate traffic as efficiently. Additionally, the machine that should be using its cycles for generating requests wastes CPU time on statistical calculation. The generation of clients is further limited because the client process must connect to the master process as it is being spawned. The master must then spawn the client on a remote machine to harness the power of multiple machines. Unfortunately, users often do not have system set up to handle such behavior. Even if they can, the master still remains a bottleneck - clients can only be generated as rapidly as the master can spawn them. If another client master process were run on another machine, the statistics from the two master processes would be difficult to integrate. Even when only running one master process, the standard model limits the utility of reporting statistics, since the clients compute their data as they run, and throw away useful information, the benchmark can only calculate and report a limited amount of data. Not only does the client not provide an in depth analysis of its own behavior, but the interaction of clients cannot be derived after the data has already been converted and returned.

The problem of gathering statistics far exceeds the question of gathering data. The way in which most benchmarks mishandle statistics inhibits their ability to scale traffic, and prevents any capture of the interaction of users. The separate statistics daemon, and post-processing script in hBench:Web solves these problems. Clients only connect to the daemon when they have completed the user profile. The statistics daemon can run on any machine because the port and machine on which the daemon runs has been set up before the benchmark runs. Immediately, the daemon solves the difficulty of connecting to a master process at start-up and transferring data to the master when it should be spawning clients. Furthermore, multiple master processes can be run on different machines. The amount of power in the benchmark increases enormously. Second, the daemon simply receives information about each request from the client. The client makes no calculations, and throws away no data. After each user profile, the client sends information about the overall session, each page retrieval, and each file request. Reported numbers include: header size, number of packets sent, number of packets received, connection latency, header latency, body latency, page latency(the time for all files to download), number of connections per page (for persistent connection analysis), and most important, the time at which each session, page retrieval, and file request began. This data enables the post processing script to make a thorough analysis of the issues that affect one users request. Furthermore, the inclusion of a timestamp allows the post-processing script to create a picture of the interaction between different user profiles. Various data that can be extracted include: file and page latency by the number of connections, each file request's latency broken into the time spent making the connection, sending the request, receiving the header, and receiving the entirety of the body. Receiving such large amounts of data could potentially make the statistics daemon a bottleneck. The inclusion of the timestamp, though, means that multiple daemons can be run, with the clients connecting to different daemons. In the end, the daemon merely puts the data into a specially formatted file. The data from multiple daemon sessions can easily be concatenated so that the post-processing script treats the data as if it were generated by one daemon.

The client process also enables a variety of options for requests. Originally, WebStone supported only GET requests for static files. Currently, WebStone includes limited support for CGI, but it does not take into account any of the types of CGI traffic. Instead, its only CGI script returns a random set of bytes to the client program. SPECWeb has made no attempt to model CGI. DBench, as it depends on running on the original site and precisely replaying segments of a Web log, models exactly the CGI scripts on a server, conditional

GET requests, failed requests, and limited Authorization. Unfortunately, as the benchmark runs only on the site on which it is installed, the benchmark's model cannot be considered appropriate for hBench:Web. To make the benchmark easily distributed and run, as well as enabling the benchmark to scale with future traffic, hBench:Web takes a very modular approach to the generation of traffic. First, unlike the other benchmarks, hBench:Web enables the user to specify the number and type of concurrent connections per client. The standard behavior has been set at four concurrent non-persistent connections, to match the default of the Netscape browser. Furthermore, since requests are not created until right before being sent to the server, as with a real browser, any type of traffic can be easily generated. Currently, the benchmark supports GET and POST, but other methods can be easily inserted. The ability to add headers also marks the benchmark. Currently the benchmark only modifies the conditional GET "If-Modified-Since" header, but could easily add authorization data, bytes range requests, or other additions. Finally, the type of traffic handled by the benchmark includes static files, a CGI counter, a CGI script which accepts POST data, a CGI script which generates redirects, a CGI search engine, and a default CGI script like that included in WebStone. The benchmark distribution will include the standard scripts to handle the CGI traffic Once again, more CGI can be easily added by an interested user.

4.8 Conclusion

Quality benchmarks give the computer community tools with which to conduct quality research and analysis. The enormous weaknesses of each available Web server benchmark contributes little to our understanding of the Web. Whether the benchmark is based on data that is not freely available, data that is non-representative and outdated, or nothing at all, the current set of benchmarks do not facilitate reproducible, resdistributable, and modifiable testing of servers. Furthermore, the benchmarks' focus on only one number to the exclusion of all others ignores the need to determine causes of bottlenecks. Finally, the inability to test a variety of client behavior patterns compromises the effectiveness of each benchmark.

In response to the dire need of the Web community, hBench:Web provides a simple, yet effective set of tools to measure Web server performance. The benchmark can generate a model of Web traffic based on the three factors of site design, user behavior, and frequency of users. The benchmark can abstract the factors to reduce the model to a small set that can be easily redistributed, without revealing information about a site or its users. Furthermore, the model can be adjusted to realistically model dramatic or subtle shifts in traffic patterns. Finally, the benchmark can scale models of sites to predict traffic patterns as much as a year into the future. The benchmark models client behavior in every way from the headers a user's browser sends, to the implementation of persistent connections. The benchmark enables the user to test four standard CGI scripts, to determine the effect of a real CGI load. The benchmark collects a variety of statistics, and can easily generate load from multiple machines. Finally, a tool can produce a variety of information about the relation of latency to factors in an individual request, as well as the interaction of one request with the entire set of other requests. This benchmark gives those who run it control, realism, and accurate and interesting data.

Chapter 5

Experimental Proof of the Accuracy of the Benchmarking Model

The simplest and most elegant theory means little if, in practice, the theory fails. Therefore, any model must be tested to verify its strengths and weaknesses. For example, creating clients to behave like real browsers and creating a statistics daemon both provide significant enhancements to any benchmark's realism. Of course, these developments would be just as useful to WebStone or SPECWeb as hBench:Web. The self-configuring model, reduction, selection of traffic patterns, and scaling provide the true innovation of hBench:Web. Therefore, in this chapter, I analyze the approximation of the modeled traffic to real traffic patterns.

In Section 1, I describe the experimental setup of the tests. In Section 2, I discuss the effectiveness of the simple model in creating traffic similar to the original. Section 3 presents a comparison of the traffic generated from the various reduction methods. Section 4 details the effectiveness of generating the specific type of traffic desired by the user. Section 5 uses the historical set of logs to compare the scaled traffic patterns to that which actually occurred. Section 6 concludes.

5.1 Experimental Setup

For each aspect of the benchmark, I present an analysis of the traffic generated by the benchmark and compare it to the original log. The analysis presented comes from tests based on the software site because its load makes it the most interesting to researchers. The Harvard FAS site confirmed the results presented, but as its traffic does not match that of the software site, its results have been omitted. For the software site, the length of the benchmark was 15 minutes, with 5 minutes time to start users. In this way, complex user interactions and burstiness characterize the entire benchmark, as on a real server. Each of the tests was run three times, with different random seeds by which to choose user profiles. To test the benchmark's ability to scale load, each test was run to generate different levels of load. The projected goals were: the 50th, 60th, 70th, 80th, 90th, and 100th percentile heaviest loaded 15 minute segments from the original month-long log. The data, unless mentioned, varied little. Each of the reported numbers is the most pessimistic available.

The tests were run on available, non-optimized components. An x86 PC with a 120 MHz Pentium processor and PCI bus running BSD/OS 2.1 with Apache 1.1.3, served client requests. Apache was modified to log latency times, in the manner of the ISP machines. The machine used an Intel Endeavor Motherboard (i430FX chipset). The memory consisted of the on-chip cache(8KB i + 8 KB d), a 512KB pipeline burst off-chip cache and 32MB of 60ns EDO RAM. The machine held the relevant files on the web site on a 1033MB, 5400 RPM Fujitsu M2694ES disk, with a 10ms average seek time and a transfer rate of 5MB/second accessed through a BusLogic BT946C PCI SCSI controller. The Ethernet adapter was a 10Mbps Western Digital Elite 16 with 16KB of 100ns RAM.

The machine that generated the requests was a SparcStation 20 with two 75 MHz SuperSparc processors running Solaris 2.4. The memory subsystem included 128MB RAM. The network subsystem was the built-in 10MB Lance Ethernet. The tests were run on the same local subnetwork, but not a private network.

5.2 Analysis of the Basic Model

The traffic from running the benchmark with the original model closely resembled the original log. The configuration script created the model based on one day's traffic because the analysis in chapter 4 showed that the subsets of traffic in a month's log mirror those of the full log. The model included a fileset of 1530 files, grouped into 700 pages, accessed by 3222 user profiles. Many pages included the same file, leading to an average of four files per page. The standard chi-square test used in chapter 4 also is used to confirm the similarity between the model and the actual log traffic. Since each user runs from the same machine, one cannot distinguish between different user patterns, so the user set analysis is omitted. Added, however, is an analysis of the similarity in traffic growth. Since the benchmark allows the user to specify different levels of traffic, by means of the user arrival time, the benchmark should be able to create traffic changes similar to those in the log. The test compares the load actually generated during the time period that provides the arrival time, with the benchmark figures. A fifteen minute test generated enough traffic to create an approximation of the original traffic. The only area in which the traffic differs form the original model is in the file size. The generated traffic makes 5% more requests for files under 500 bytes, and 5% fewer for files of size 5 KB than the original log. This behavior also leads to fewer bytes being downloaded, overall. That difference, however, is offset by the greater average size of the 1KB and 5KB in the generated traffic. This traffic shift is not found in each run of the benchmark. In another case, the average traffic erred in the other direction. The difference can be attributed to the noise expected from simulating a month's worth of traffic in fifteen minutes. The accuracy of the results, combined with the convenience of running the benchmark for only fifteen minutes, indicates that the modeling has been successful.

5.3 Analysis of the Reduced Models

Each of the four methods of reduction trades reality of traffic for simplicity in the model. The page set reduction reduced the site by as much as 400%. The user set reduced the set of user profiles by 250%. Of course, the file set reduction leads to a greater variance in the distribution of requests by file size, and the user set reduction leads to a noticeable degradation in the generation of load. Table 14 shows the factors of reduction, as well as the fit to the original traffic.The factors of method and response from the server have been omit-

Reduction Method	Reduction Factor	Correlation in Bytes per Load	Correlation in Requests per Load	Correlation in File Type	Correlation in Dist. of File Size
File Reduction 1	25%	99%	90%	93%	80%
File Reduction 2	400%	70%	10%	90%	60%
User Reduction 1	12%	97%	83%	95%	55%
User Reduction 2	250%	43%	8%	94%	30%

Table 14: The reduction and goodness-of-fit to the original log per reduction method
The user reductions cause a greater deviation from the original model. The models tend to deteriorate very quickly, as reduction increases.

ted. The similarity of these features, throughout all tests was above 99%. None of the traffic patterns could affect these numbers.

The chart demonstrates that as reduction intensifies and information is discarded, the benchmark becomes less realistic. For example, the first method of reducing the page set only condenses 25% of the pages, and the traffic shows little degradation from the original model. The variance in behavior becomes more apparent when the file set has been reduced by a factor of four. Not surprisingly, the variance is largest in the areas where the information is disregarded in performing the reduction. The number of bytes of traffic sent by the server in the benchmark approaches that of the original log. The reduction technique focuses on retaining a similar number of bytes on each Web page. The number of requests increase, due to pages with different numbers of files being combined. The distribution of file sizes is heavily weighted toward small files, which causes the disparity from the original log. Essentially, both of these changes can be attributed to the reduction technique - pages with few large files are combined with similar size pages of many small files. Thus, the total number of bytes remains close to the original model. The distribution of size and requests, as a consequence, vary more greatly. One factor that seems surprising is the lack of degradation of the type of file being accessed. The reduction techniques completely ignore the file type when condensing files. The close approximation of original traffic comes from the basic model of the Web page - each page is made of an HTML front page and multiple images. Therefore, even by condensing pages, one cannot significantly alter the model. The methods and responses show no change because the reduction techniques do not affect these factors.

The reduction of the user profiles have a more significant effect on the traffic generated by the benchmark than those of the fileset. Again, the results demonstrate the effects of the reduction technique. Affecting user profiles can affect the load generated by the benchmark, and also the type of traffic. Users often download many files, so changing a user profile also introduces a significant effect on the page set. Once again, the more aggressive the reduction technique, the greater disparity from the original model. Most of the changes can be attributed to the greater variance in combining different user profiles that generate different numbers and types of requests. The data even shows some degradation in the types of methods used by the benchmark and the server responses to the requests. Combining users affects not only the number of requests and the files requested, but also the methods used. Combining users can also lead to a fluctuation in the number of conditional GETs(304), requests resulting in redirects(302), and requests for non-existent files(404). A change in each type of these responses resulted in the concurrent variance in the benchmark behavior.

The analysis of the reduction techniques demonstrates the benefits, as well as the pitfalls of abstracting traffic patterns. First, any method of abstraction leads to a loss of information, and diminishes the reality of the model. The more aggressive the abstraction, the greater the loss of information. Changing user profiles leads to a greater variation in behavior because the user affects the entirety of the model, while the page set's influence is limited. Finally, the degradation due to even simple reduction techniques very rapidly compromise the model. Therefore, benchmarks such as dBench and SPECWeb which very significantly change user behavior at run time and do not generate a wide variety of users will very likely lose most of the information which formed the basis of the model. The models of Web server benchmark cannot simply be manipulated without consequence.

5.4 Analysis of Generating Specific Types of Traffic

The analysis in the previous two sections indicates that any attempt at generating realistic traffic would be compromised by making the significant changes to the user set that this technique implies. Fortunately, the usefulness of this feature comes not from its approximation of reality, but its configurability. The measure of effectiveness of this feature comes from the benchmark generating traffic proportional to the users' request.

This aspect of the benchmark did not perform extremely well. One of the great difficulties in specifying traffic by making a user profile more likely to be run, is that the randomness of choosing the user profile can negate any attempts to specify traffic. When there are 3000 user profiles, each with different probability of being chosen, increasing the likelihood of a subset of users' choice may not affect the model enough to generate the proper results. Table 15 shows some of the different types of tests conducted, and the results. Each time that a significant increase of an uncommon traffic pattern was attempted, the benchmark did not reach the goal. This limited number of user profiles were not selected often enough to affect the traffic as desired. However, the increase in common traffic enables the script to increase many profiles' likelihood of being run, so that traffic patterns actually changed. The pattern did not continue when trying to reduce factors of traffic because profiles can easily be deleted, for both a common and uncommon type of traffic. For dramatically increasing specific types of traffic, the method fails. Otherwise, the tool performs very well. Unfortunately, as this feature has been constructed with the idea that researchers might want to significantly increase a new type of traffic, the results are disappointing.

Traffic Specification	Current Traffic Level	Goal Traffic Level	Adjusted Traffic Level
Increase level of “Not modified” Responses - 304	20%	60%	41%
Increase level of “Not modified” Responses - 304	20%	25%	25%
Increase level of “OK” Responses -200	77%	85%	86%

Table 15: Sample data from tests run on setting traffic parameters
The sample of tests shows that the benchmark cannot dramatically increase traffic,
but it can decrease the likelihood of traffic, and make small changes in a sites' traffic pattern.

Traffic Specification	Current Traffic Level	Goal Traffic Level	Adjusted Traffic Level
Decrease level of CGI traffic	10%	3%	2%
Decrease level of GET requests	97%	40%	40%

Table 15: Sample data from tests run on setting traffic parameters
The sample of tests shows that the benchmark cannot dramatically increase traffic, but it can decrease the likelihood of traffic, and make small changes in a sites' traffic pattern.

5.5 Analysis of Scaling

The model of scaling generated traffic remarkably similar to that of the original log. The testing only scaled the behavior of the original model of the software site, and not those of the reduced models. The test analyzes scaling traffic for the one, three, and six months into the future. Since I have access to the access logs for the past 10 months, I was able to use three months of logs to generate the proper configurations, and then test the result against actual logs from the future. Two issues were analyzed in the scaling: the effectiveness of the selection of user arrival times and the approximation of the traffic to reality.

The method of selecting user arrival time achieved moderate success. Again, assuming the base model of user profiles and page sets to be constant, I compared the user arrival times generated by the scaling script to those generated by the standard configuration script, for the months to be analyzed. Table 16 shows the figures generated by the configuration script, and those generated by the scaling script. The user arrival time deviates somewhat in the third month, and cannot completely recover by the sixth month. The site undergoes something of a user spike in the third month, that could not be predicted. Still, the values seem to be similar enough to justify confidence in the method of scaling user arrival times.

Months into the future	Real User InterArrival Time	Projected User InterArrival Time
One	4.91 s	4.87 s
Three	2.03 s	2.41 s
Six	1.65 s	1.84 s

Table 16: Times between new user session generated by the benchmark versus the real figure
While the growth in Month three throws the simulation off, the scale time begins to approach the real figure by the sixth month.

The traffic generated by the benchmark does not approach the reality of the original model, but achieves great success. The results of the standard chi-square tests, listed in table 17 show that the benchmark's performance deteriorates over time, but it never falls below 80% similarity to the original model in any category of traffic. The results also indicate that the selection of user arrival time makes the greatest impact on the accuracy of scaling. The reality of the benchmark does not change from the third to the sixth month. Therefore, the benefit of more closely approximating user arrival time offsets the deterioration of the underlying model. Furthermore, the degradation in benchmark's prediction of traffic works almost uniformly across all of the factors of measurement of similarity. The underlying model does not change in any specific way, further indicating that the page and user sets of a site remain extremely constant.

Number of months into the future	**Correlation in Bytes per Load**	**Correlation in Requests per Load**	**Correlation in File Type**	**Correlation in Dist. of File Size**
Month 1	94%	96%	98%	98%
Month 3	86%	87%	97%	98%
Month 6	85%	84%	90%	93%

Table 17: The scaled model's goodness-of-fit with the actual traffic patterns
The model is remarkable effective, creating very accurate simulations of future traffic.

The results from the scaling performance of the benchmark are extremely encouraging. The results provide practical justification of the statistical analysis of site traffic. Furthermore, approximating future user arrival times can be done because the space of Web users is so large, that the role of an individual user can be minimized. The selection of future user arrival times can be made a more refined process, but the simple proof of concept shows great potential for the idea.

5.6 Results

This section provides both proof of concept of the ability of the benchmark to generate accurate statistics and to offer final confirmation of the server's ability to handle load. This section presents the statistics gathered from running the benchmark with the original load, on the heaviest load on the software site. While the test did not exploit the persistent connections, it did utilize the benchmark's counter, POST script, redirect functions, conditional GET requests, and requests for files that do not exist on the server. In this way, the

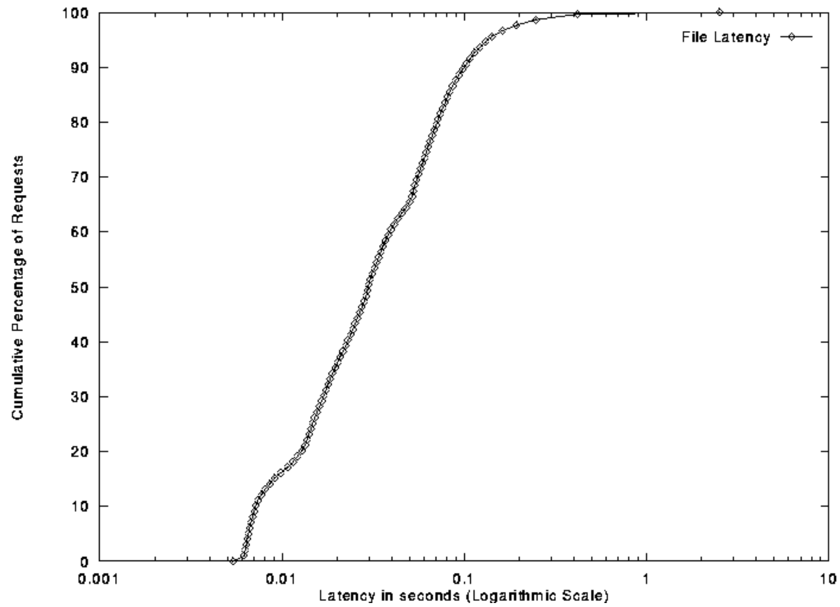


Figure 24 Cumulative Distribution of Latency Per File on Non-Reduced, Non-Scaled model of heaviest 15 minute period of Software Site. By effectively removing the user variance, and the Internet, the benchmark shows that the server does not cause the large latencies that plague the Web.

run-time environment was used in ways that other benchmarks do not allow. The verification of their implementation can be seen in the traffic analyses, by the accuracy in levels of these types of traffic.

The statistics from running the benchmark show that, even under heavy load, clients do not experience significant latency. Figure 24 shows that no requests incurred a latency of over 2.5 seconds. The requests ranged from simple conditional GETs to accessing a 133 KB file, and the latency never exceeded 2.5 seconds. As conjectured in Section 3.7, when removing the Internet and slow user connections, the latencies become trivial. Additionally, the graph shows that the items omitted from the server statistics in Chapter 3, such as connect time, constitute a trivial amount of total transfer time. Furthermore, the load placed on the server reached a maximum of 60 connections in one second. Not only can a simple 120 MHz Pentium handle 60 connections per second, but the latency is virtually negligible. First, this statistic offers conclusive proof of the conclusion from chapter 3 - the server cannot be the primary source of latency on the Web. Both real Web logs and a proven realistic benchmark indicate that the server can easily handle loads of over 800,000 hits a day.

The unique structure of hBench:Web enables the benchmark to generate such large loads. For example, in the test run the 60 connections generated by hBench:Web exceeds the maximum number generated by WebStone's most intensive configuration on the running on the same hardware platform. WebStone, running from 40 to 120 users could generate only 45 connections per second, marking a 33% improvement in hBench:Web's ability to simulate very intense traffic patterns. On the other hand, the total amount of requests did not approach that of WebStone in a 15 minute run. WebStone generated nearly 36,000 requests, while hBench:Web sent only 8000. Not only can hBench:Web generate more traffic per second than WebStone, but it does so in a bursty model - which models true Web behavior.

The unique structure of hBench:Web also enables the collection of such statistics that have facilitated the above analysis. Marking requests with timestamps enables the post-processing script to determine the number of connections per second, as well as the number of outstanding requests at any given time. The recording of each request also enables an analysis of the cause of latency, which can be masked by merely returning averages. The general organization and statistical focus of the benchmark enhance the modeling, by extracting all possible information, and enabling the user to extract the statistic of choice.

5.7 Conclusion

The need for a quality, realistic benchmark motivated the creation of hBench:Web. By building upon a sound, simple mathematical model hBench:Web, models traffic, reduces the model, allows the user to specify traffic patterns, and scales into the future. The implementation of the benchmark was largely successful in modeling realistic traffic. While the model did not fulfill the goal of enabling fine-grained control of traffic patterns, the results of hBench:Web exceeded expectations. Furthermore, an analysis of the effects of changing user and page sets showed that the models of real sites are extremely sensitive to manipulation. This result casts further doubt on the methodology and results generated by other benchmarks using access logs as the baseline of their benchmark. The improved run-time design simulates a variety of traffic that other benchmarks ignore, and collects a wider variety of statistics. The redesign also allows the benchmark to generate more load that other benchmarks, while doing it in a bursty manner, as is common on the Web. The benchmark, therefore, achieves its stated goal of generating realistic traffic and collecting meaningful and accurate statistics.

Chapter 6

Conclusion

This thesis has shown that the server cannot generally be the source of latency when using the Web. Log analysis indicates that servers can handle the traffic of the Web, and benchmarking confirms this conclusion. While some sites may require a more robust server environment than that described in this thesis, those sites are the exception, not the rule. Therefore, to truly enable the Web to scale, research and corporate efforts must be refocused. By continuing to enhance the performance of servers, users will see no reduction in latency. Instead, work must be done to enhance the performance of moving the data from the site's point in the network to a client's screen.

The log analysis indicates that the traffic on sites continues to grow exponentially. To be able to satisfy 90% of requests, any cache must grow exponentially each month. New sites continue to join the Web, and old sites continue to expand their presence. Different sites grow in different ways, but the reality is: the exponential growth continues. The number of sites increases. The number of users increases. The number of bytes on a server exponentially increases. The number of bytes transmitted by a server exponentially increases. The crisis of expansion has begun to affect everybody. On a personal level, we have each been frustrated by the latency on the Web, and often we blame the server. This is wrong. To avoid a crisis in Web expansion, it is imperative to refocus the attempts at improving Web performance. First, scientific analysis must replace conjecture as a means of guiding research development. The development of a quality set of benchmarks such as hBench:Web for servers, can help gauge the effect of new developments. It is important to have quality benchmarks, so that new "solutions" do not end up causing more trouble than the original problem! Furthermore, if the benchmarks have the same fundamentally strong basis as hBench:Web, no longer will the community be limited by tests that become outdated before they are released. The computer community does not have the same freedom for trial and error in developing solutions to Web scaling problems, as it had when improving systems, hardware, and other applications. In those cases, the growth was in the capabilities of machines. Now, the exponential growth being talked about in the computer community is about demand. If the public's demand for Web services goes unsatisfied, the Web will collapse.

6.1 Future Work

This work must be expanded in two directions. First, general analyses of the Web must continue. Sites need to be forthcoming with data, or they will suffer a collapse of their livelihood. Without the logs from the ISP, I could not have conducted quality research. More businesses need to realize what this isolated ISP already has - research can only help make their lives better. When beginning this project, I did not expect to find that servers do not cause latency on the Web. It seemed very unlikely. Such unsubstantiated theory dominates the Web community, and these fallacies must be addressed. Only by conducting an impartial analysis of real statistics, can one distinguish assumption from fact. Then, once the real causes of latency can be determined, it becomes imperative to address those issues. Two issues I have discussed in this paper are persistent connections and caching. These issues are far from resolved. Both can help decrease the amount of traffic on the Web, but the real data is needed to conduct a proper analysis of these potential developments.

Second, hBench:Web must be fully exploited. The benchmark provides an excellent tool for measuring the effect of persistent connections and other developments on server performance. When focusing on enhancing performance on the Web, it is imperative that side effects of developments be known. A realistic self-configuring benchmark, that can scale into the future provides a tool that can provide such information. Still, the scaling performance of the benchmark should be improved to generate more realistic data. Most important, however, configurations for the benchmark should be generated and distributed. These configurations reveal no information about a site, but allow the benchmark to reproduce traffic from a variety of sites. Information, our greatest asset in conducting research, should not be the scarcest resource on the World Wide Web.

6.2 Availability

The benchmark, hBench:Web, and a set of configuration are available at:

http://www.eecs.harvard.edu/vino/web

6.3 Appreciation

This thesis, while written by me, has been the work of many. Thanks to Margo Seltzer for helping with ideas, encouragement, contacts, machines, and advice in writing the thesis. Thanks to the unnamed ISP, and my contact there. Without your tremendous assistance with logs, information, and industrial insight, this thesis never would have happened. Special thanks to Michael Courage, who never hesitated to talk about web benchmarking, and who worked on the statistics daemon and the request structure. Yasuhiro Endo contributed his enormous knowledge of measurement, while Nick Gloy made the late nights in Aiken interesting. Chris Small taught me my first lessons about research, and I hope I've applied them well. Keith Smith helped with all aspects of manipulating systems. Aaron Brown allowed me to dominate his machine for long stretches of time, and was an excellent source of commiseration.

Special thanks and love go to my parents who helped me become the person I am today. My heartfelt appreciation goes to David Manley, who kept the entire project in perspective, and served as my idol for 21 years. Thanks to my little brother Michael Manley, who has been the best little brother anyone ever could have hoped to have. Of course, as far as I can tell, he did not actually affect the project. Thanks also to my roommate who has heard the battle over the Web more times than he'll ever want to remember.

Finally, and most importantly, I thank my fiancee, Lara Izlan. She has supported me unfailingly along this road, read the work more often than she wanted, and has been the light that will guide me through the rest of my life.

Bibliography

[1] T. Berners-Lee. Hypertext Transfer Protocol. *Internet Engineering Task Force Working Draft*, November 1993.

[2] A. Bestavros. Demand-based Data Dissemination in Distributed Multimedia Systems. *Proceedings of IASTED/ISMM/ACM International Conference on Distributed Multimedia Systems and Applications*. August 1995.

[3] J. Chen, M. Day, A.Wharton. Benchmarking the Next Generation of Internet Servers. Whitepaper available at http://domino.lotus.com. March 1997.

[4] M. Crovella and A. Bestavros. Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes. *IEEE/ACM Transactions on Networking*, 1997.

[5] N. Edwards, O. Rees. Performance of HTTP and CGI.
Available at http://www.ansa.co.uk/ANSA/ISF/1506/APM1506.html

[6]Y. Endo, Z. Wang, J. Chen, M. Seltzer. Using Latency to Evaluate Interactive System Performance. *Proceedings of the 1996 Symposium on Operating System Design and Implementation (OSDI).*

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1. *Internet Engineering Task Force Working Draft*, August 1996.

[8] J. Gwertzman. Autonomous Replication in Wide-Area Internetworks. Technical Report 34-94. April 1995.

[9] J. Mogul. The Case for Persistent Connection HTTP. SIGCOMM '95 Conference on Communications Architectures and Protocols.

[10] J. Mogul, Network Behavior of a Busy Web Server and its Clients. DEC WRL RR 95.5

[11] H Nielsen, J. Gettys, A Baird-Smith, E. Prud'hommeaux, H Lie, C. Lilley. Network Performance Effects of HTTP/1.1, CSS1, and PNG. W3 Consortium Note available at http://www.w3.org/pub/WWW/Protocols/HTTP/Performance/Pipeline.html

[12] J. Pitkow, M. Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns.
Available at : http://www.vuw.ac.nz/~mimi/www/www-caching/caching.html

[13] SPECweb96 Benchmark Information can be found at
http://open.specbench.org/osg/web96/

[14] SPECweb97 Benchmark Information can be found at
http://open.specbench.org/osg/web97/

[15] G. Trent, M. Sate. WebSTONE: The First Generation in HTTP Server Benchmarking. White paper available at http://www.sgi.com/Products/WebFORCE/WebStone/paper.html

[16] D. Wessels. Intelligent Caching for World-Wide Web Objects.
Available at http://morse.colorado.edu/~wessels/Proxy/Thesis/