On the Interaction Between an Operating System and Web Server

David J. Yates[1]
(djy@cs.bu.edu)

Computer Science Department
Boston University
Boston, MA 02215 – USA

Virgílio Almeida[2]
(virgilio@dcc.ufmg.br)

Jussara M. Almeida
(jussara@dcc.ufmg.br)

Depto. de Ciência da Computaçao
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais 31270-010 – Brazil

Technical Report CS 97-012
July 16, 1997

Abstract

This paper examines how and why web server performance changes as the workload at the server varies. We measure the performance of a PC acting as a standalone web server, running Apache on top of Linux. We use two important tools to understand what aspects of software architecture and implementation determine performance at the server. The first is a tool that we developed, called WebMonitor, which measures activity and resource consumption, both in the operating system and in the web server. The second is the kernel profiling facility distributed as part of Linux. We vary the workload at the server along two important dimensions: the number of clients concurrently accessing the server, and the size of the documents stored on the server. Our results quantify and show how more clients and larger files stress the web server and operating system in different and surprising ways. Our results also show the importance of fixed costs (i.e., opening and closing TCP connections, and updating the server log) in determining web server performance.

1   Introduction

Web server performance is an important issue for content publishers and web users alike. They have the common goal of delivering information to all interested end-users with good response time. Several factors determine whether or not the server is the bottleneck in this process. Some of these factors are machine dependent (e.g., CPU speed and memory size), whereas others are not.

This paper examines how and why web server performance changes as the workload at the server varies. We measure the performance of a PC acting as a standalone web server, running Apache on top of Linux. We use two important tools to understand what aspects of software architecture and implementation determine performance at the server. The first is a tool that we developed, called WebMonitor, which measures activity and resource consumption, both in the operating system and in the web server. The second is the kernel profiling facility distributed as part of Linux. We vary the workload at the server along two important dimensions: the number of clients concurrently accessing the server, and the size of the documents stored on the server. In all experiments we use a set of documents that has a heavy-tailed size distribution. This captures the fact that servers must simultaneously handle many requests for small files (e.g., text and image files) and a few requests for large files (e.g., audio and video files).

Our results show that our server becomes saturated as soon as 5 clients are accessing it at full speed (with zero think time), but sustains its aggregate throughput surprisingly well for up to 60 clients. Our results also show that the nature of the workload at the server stresses different parts of the operating system in different and surprising ways. For example, increasing average document size (by up to a factor of 16) actually increases server performance. Specifically, the throughput in bytes per second increases with document size, even though the connection throughput (requests per second) decreases. This is because the increase in file size doesn't significantly increase file system activity, but does increase the amount of useful work the network protocol stack can perform, as a side-effect of requiring longer TCP connections to transfer larger documents. These results also show the importance of fixed costs (i.e., opening and closing TCP connections, and updating the server log) in determining web server performance.

The paper is organized as follows. Section 2 presents an overview of the WebMonitor architecture and important aspects of its implementation. In section 3, we describe the experimental environment that was instrumented and measured, and the workload used to drive our experiments. Section 4 discusses and analyzes the results obtained by WebMonitor and the Linux profiling facility. Finally, we summarize the main contributions of our work in section 5.

2   WebMonitor Architecture

WebMonitor is a tool for measuring and understanding server behavior. It can be viewed as a combination of two main modules that operate at different levels of the system and collect performance data using different techniques. The Kernel Module (KM) runs independently of the web server and collects information about the operating system as a whole. The code for the Server Module (SM) is actually compiled and linked with the server code, and therefore runs as part of the server.

KM periodically collects resource usage data from both system-wide and web server standpoints. The information collected includes processor utilization, disk activity, paging activity and interrupt rates. This module also collects statistics about network activity, such as the number and state of TCP connections to the HTTP port at the server. Such statistics are useful for understanding the "lifetime" of connections in the server. KM also obtains per-process performance data including CPU and memory utilization. The Linux kernel provides a "virtual file system", the /proc filesystem [12], that keeps, in *main memory*, performance data regarding kernel activity. This information can be read by user programs through several read system calls. However, we found that obtaining performance statistics through /proc was very expensive, especially during the operation of our web server when it was busy. Thus, we decided to implement KM using four new system calls that summarize and return specific information about kernel activity [1].

Instead of being based on periodic sampling, like KM, the SM collects information about server performance based on a trace of events that occur during the handling of each HTTP request. The data collected includes bytes transmitted, connections established and disk operations. Another important piece of information is the response time of the server for processing a request. The time measured by the SM begins with the establishment of a connection and ends when the server (i.e., HTTP process) is ready to handle the next request. It is broken into three components, which are measured in processor time and elapsed time. *Parsing time* is the interval that begins just after the establishment of the connection and ends when the header of the request has been parsed and is ready to be processed. *Processing time* is the time spent actually processing the request. It does not include the server logging time. It accounts for the time spent reading the URL (Uniform Resource Locator) and the time needed to move the file from memory or disk to the network. *Logging time* is the time spent performing standard HTTP logging. After logging, a server process is ready to handle a new request.

Document size at web servers follow heavy-tailed distributions, that have very large variance [2]. Thus, average results for the whole population of requests would have no statistical meaning. However, keeping detailed information about each individual request is unfeasible in terms of overhead. As a compromise to keep overhead as low as possible without impairing the accuracy and significance of the measurements, SM categorizes requests into a small number of classes, defined by ranges of file sizes. The overhead introduced by WebMonitor is low. We compared the performance of our web server with and without the monitor (KM and SM modules). We found the difference in performance between these two cases to be less than 4% [1].

3   Measurement Methodology

This section describes the environment where we carried out the web server experiments. Our hardware platform was an Intel Pentium 75 MHz system, with 16 Megabytes of main memory, a 0.5 Gigabyte disk and a standard 10 Megabit/second Ethernet card. The operating system used is Linux version 2.0.0, which is distributed under the terms of GNU General Public License [12]. The server software is Apache, version 1.1.1, a public domain HTTP server [10]. Our Apache server was configured to run in standalone mode, since this is far more efficient than running it from the inetd system process.

To generate a representative WWW workload, we used WebStone [11] (version 2.0), which is an industry-standard benchmark for generating HTTP requests. WebStone is a configurable client-server benchmark, that uses workload parameters and client processes to generate web requests. It makes a number of HTTP GET requests for specific pages on a web server and measures the server performance, from a client standpoint. Each client process successively requests files from the server, as fast as the server can answer the requests and collects performance statistics. After all clients finish running, the master process generates an overall server performance report based on the statistics collected by each client. The main performance measures collected by WebStone are latency and throughput, measured in connections per second and also in bytes transferred per second. Table 1 gives parameters that define the baseline HTTP workload (referred to as workload A) used in our experiments. To vary document sizes, we multiply and divide the documents in workload A by 2 and 4, to give workloads A/4, A/2, A*2, and A*4. Table 2 gives the mean document size for each of the workloads. Truncation and rounding are responsible by the fact that averages do not increase exactly by a factor of 2 in this table.

		File size (KBytes)		Access probability	
Item	**Number of files**	**Total**	**Average**	**Total**	**Average**
HTML	24	180	7.5	0.192	0.008
Images	29	385	13.28	0.754	0.026
Sound	20	3580	179	0.05	0.0025
Video	4	9216	2304	0.004	0.001

Table 1: Characteristics of workload A.

Workload	**A/4**	**A/2**	**A**	**A*2**	**A*4**
Average File Size (KBytes)	7.00	14.82	29.31	58.73	117.46

Table 2: Average file size of the workloads.

4   Results

Our results evaluate web server performance for different workloads, and provide insight into what aspects of the operating system are most important in determining web server performance. We vary the web server workload along two important dimensions: the number of clients accessing the server, and the size of the documents stored on the server. In one set of experiments, we fix the set of documents accessed, and vary the number of clients concurrently accessing the server. In another set of experiments, we fix the number of clients accessing the server and vary the size of the documents stored on the server. The performance metrics we use in our evaluation are throughput and latency.

4.1   Throughput Behavior

Figure 1 shows both connection throughput and byte throughput for workload A, where the number of clients is varied between 1 and 60. As can be seen from both curves, our server is saturated when 5 clients are making requests at full speed (with zero think time). However, throughput is sustained surprisingly well over an order of magnitude increase in the number of clients. For example, consider the Mbit/s curve in Figure 1. With 5 clients the
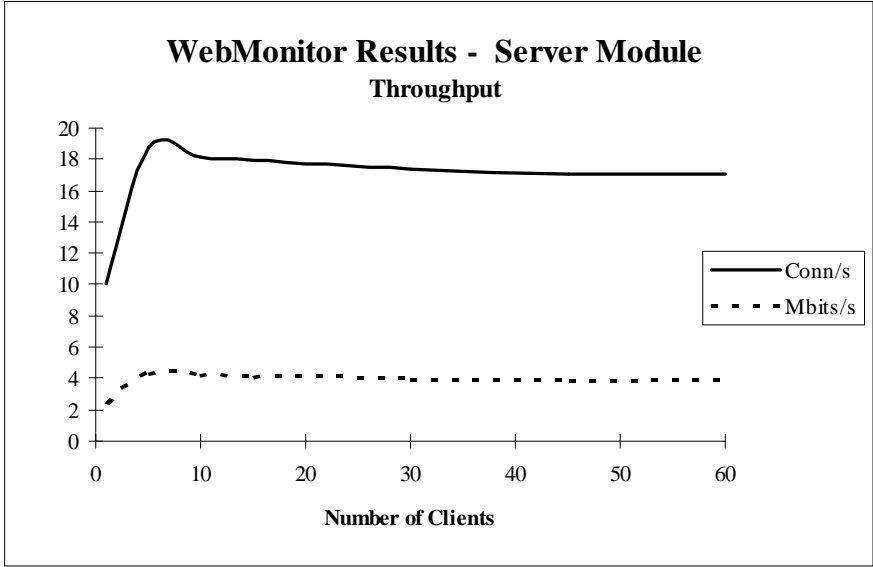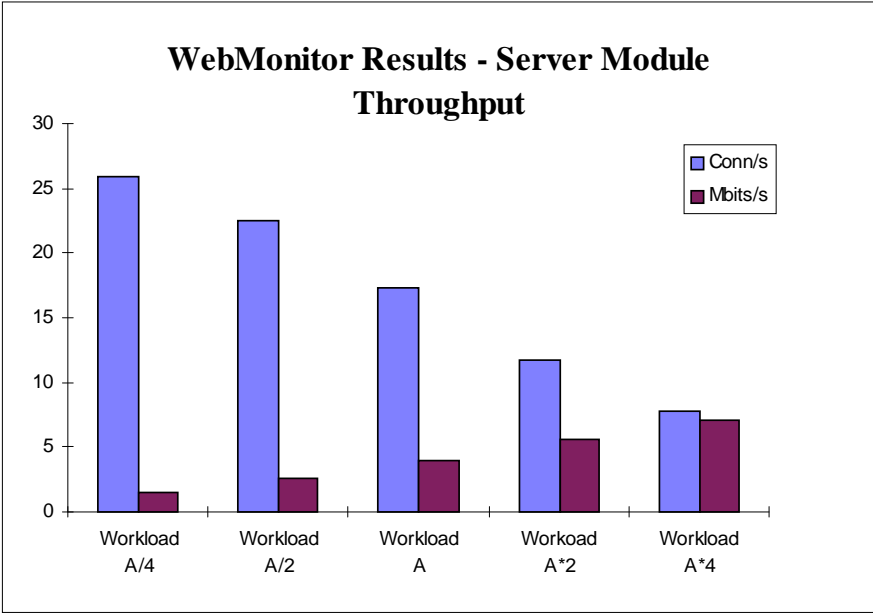
Figure 1: Throughput versus number of clients.

Figure 2: Throughput versus file size.

aggregate throughput is 4.23 Mbit/s whereas with 60 clients the aggregate throughput is 3.81 Mbit/s. This represents only a 10% drop in terms of byte (or bit) throughput. Since connection throughput is close to the byte throughput divided by the mean file size (about 29 KB for workload A), the decrease in connection throughput from 5 to 60 clients mimics the decrease in byte throughput.

Figure 2 shows the connection throughput and byte throughput for 30 clients, where the document sizes increase by a factor of 16 from workload A/4 to A*4. As expected, the connection throughput decreases as the document (or file) sizes increase. However, the byte throughput actually increases as the document sizes increase. This is because fixed per-connection costs are reduced for larger files, relative to the duration of the connection. We were somewhat surprised by this result since we expected more cache misses and paging in the memory system when transferring larger documents, and thus a decrease in byte throughput. If we just consider the footprint of the documents relative to the amount of physical memory in our server, workload A/4 represents about 21% of memory whereas workload A*4 covers memory more than 3 times over. These results suggest the importance of fixed costs associated with servicing each request from a client. In fact, the connection throughput results in Figure 2 also hint at this. If fixed costs were not important, we would expect connection throughput to drop by roughly a factor of 16 going from workload A/4 to A*4. However, we observe that this drop is only a factor of about 3.3. Our latency results in the next section confirm the hypothesis that fixed costs are important in determining web server performance, and show that the cost of updating the server request log is significant. We now turn our attention to understanding why byte throughput increases as document size increases.

To determine why byte throughput increases as document sizes increase, we reran our experiments for 30 Web-Stone clients with Linux kernel profiling enabled. We chose to profile the kernel rather than our application (i.e., the HTTP processes) since about 90% of the time is spent in the kernel on a web server [1], and the Server Module in WebMonitor already provides a coarse profile of where time is spent in the web server code. From the profiling data, we broke down how time was being spent in the kernel into six categories:

1. Low-level networking: Ethernet interrupt and device driver routines

2. High-level networking: TCP, IP, and socket routines

3. Low-level file system: IDE disk interrupt and device driver routines

4. High-level file system: read, write, and buffer cache routines

5. Timer management: timer routines used by TCP and device drivers

6. Other: routines for system calls, interrupt handling, scheduling, and memory management

Figure 3 shows where time is spent in the kernel for different document sizes. Note that the timer management routines consume a large fraction of processing in the kernel, especially when documents are smaller. For example, for workload A/4 over 50% of the time spent in the kernel is spent managing timers. We discovered that the vast majority of this timer management is due to maintaining timers for keeping TCP connections open at the server. This is required by TCP to guard against old data being received by a new connection. The Linux TCP implementation keeps these connections open for 60 seconds after the web server code has closed them, and during this time they linger in TCP's so-called TIME_WAIT state. These results extend those in [7, 8, 9], which point out the poor interaction between TCP and HTTP. Our results suggest that other performance optimizations may also be appropriate. For example, improving the implementation of timer management in Linux may yield a significant performance improvement. It is also interesting to note that as less time is spent managing timers, the time is mostly consumed by the low-level networking as it is reallocated. In contrast, the fraction of time spent performing high-level networking, high-level file system operations, and other activities doesn't change that much as file sizes increase. However, the number of low-level file system operations does become non-negligible as the file sizes increase, as one would expect.

We wanted to understand what inside TCP was changing as the time spent managing timers in the kernel dropped from about 51% for workload A/4 to about 11% for workload A*4. To do this we used the Kernel Module within WebMonitor to determine the number of connections in each of the different TCP states (including TIME_WAIT).
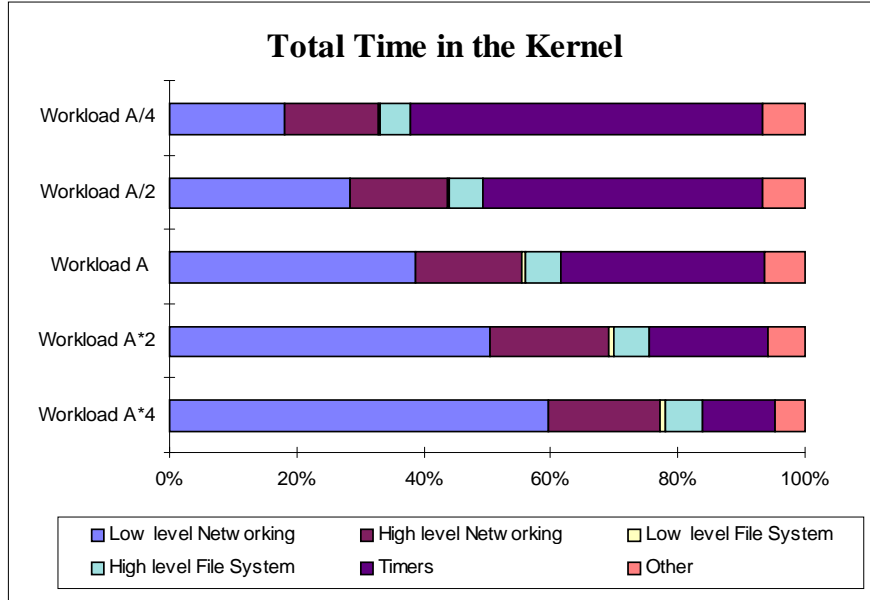
Figure 3: Kernel profiles for different file sizes.

Table 3 shows these results for the three TCP states where a significant number of connections were found. Note that there is a strong correlation between number of connections in the TIME_WAIT state and the percentage of kernel time devoted to timer management.

Workload	A/4	A/2	A	A*2	A*4
ESTABLISHED	29.59	29.25	29.25	28.20	26.51
TIME_WAIT	1399.85	1216.54	937.35	626.74	398.02
CLOSED	52.10	45.44	34.83	24.09	15.92

Table 3: TCP connection states for different file sizes.

The main conclusion we can draw from Figure 3 and Table 3 is that timer management for the TCP TIME_WAIT state is an important fixed cost associated with handling each HTTP request. As this fixed cost is reduced relative to the duration of the connection, the fraction of time spent managing timers in the kernel is reduced. This time is reallocated to performing more useful work on behalf of the clients communicating with the server. Thus, we observe that byte throughput actually increases as document sizes increase.

We also profiled the kernel for experiments where we varied the number of clients accessing the document collection in workload A (as in Figure 1). We found that the manner in which time was spent in the kernel did not change that much. Specifically, the kernel profiles for between 5 and 60 clients were very similar to the workload A profile shown in Figure 3. Thus, varying the file size changes the way the operating system is stressed, however, changing the number of clients (at least from 5 to 60) does not.

4.2   Latency Behavior

Figure 4 shows variation in response time and latency by number of clients. Transmission time is the reason for the difference between these metrics. As we increase the number of clients, the transmission time (which includes the time to get access to the network) also increases. As a consequence, clients spend more time at the client-side
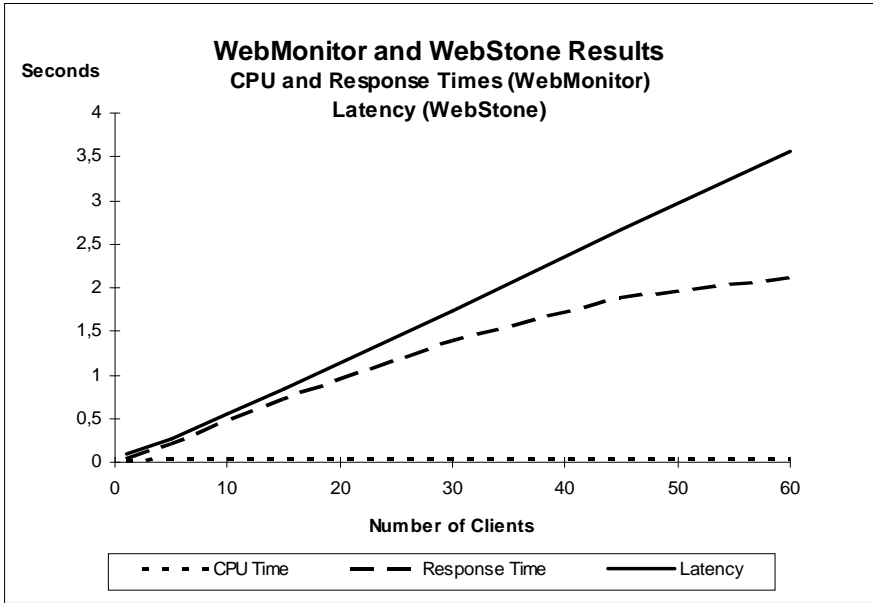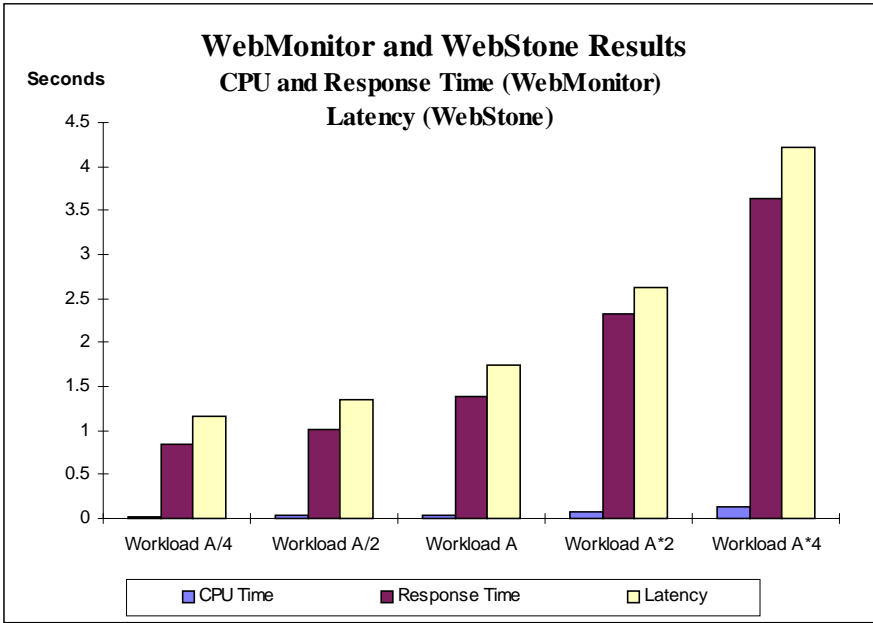
Figure 4: Time measurements versus number of clients.

Figure 5: Time measurements versus file size.

sending packets. The number of server processes (running on behalf of the client requests) does not increase linearly with the number of clients. For 1, 5, 10, 15, 20, 30, 45, and 60 clients, the number of running processes are 1, 6.47, 11.54, 15.09, 16.53, 24.90, 31.72, and 35.21, respectively. Figure 4 can be better understood in light of Little's Law, which states that $N = R \times X$, where N is the average number of clients in the system, R is the average time spent in the system and X is the throughput. As can be seen in Figure 1, throughput is sustained as the number of clients is varied between 1 and 60. We can roughly consider that throughput (X) is constant from 10 to 60. As a result, we then have R directly proportional to the number of clients, which explains the straight line for latency in Figure 4. Likewise, we can understand why response time (measured at the server-side) increases sub-linearly. This is because the number of running processes increases more slowly than the number of clients.

In Figure 5, we show latency measurements for 30 clients, where the size of the workload varies by a factor of 16, from A/4 to A*4. We used these measurements to estimate the fixed cost per connection for a zero-byte request, which is around 25 milliseconds. This includes logging time plus parsing time, and the time on the network to send the response header.

We wanted to further analyze the breakdown of response time at the server (see Figure 5) for our experiments varying document sizes for 30 clients. Again, our goal was to separate the fixed costs associated with each HTTP request and assess their importance in determining web server performance. To achieve this goal, we used the timing facility within the Server Module of WebMonitor to separate fixed costs (parsing and logging HTTP requests) from variable costs (i.e., processing requests).
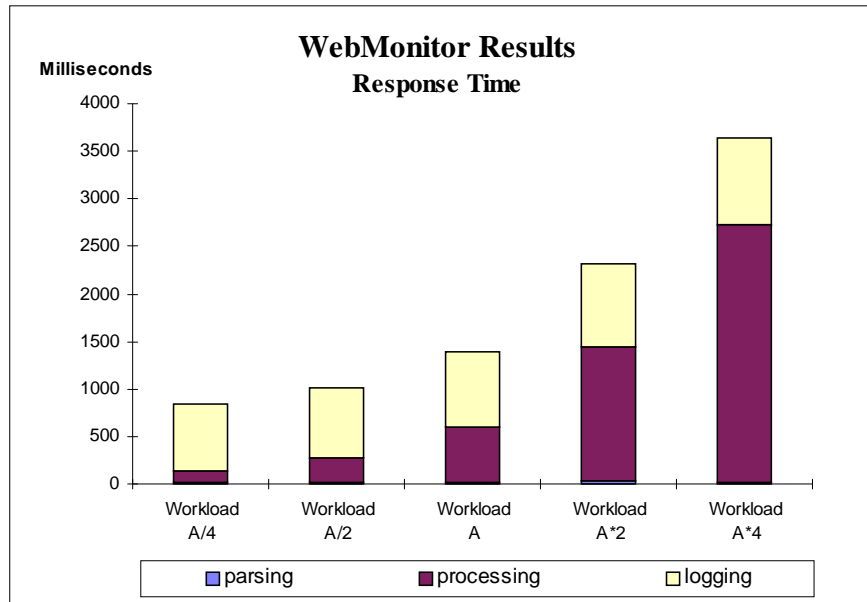

Figure 6: Breakdown of response time at the server.

Figure 6 shows the server response time (from Figure 5), broken into its three main components. What is most surprising about these results is the cost of updating the server log after each HTTP request. This logging cost dominates all other costs (including actually processing the request) until the mean file size is larger than 30 KB. The logging time is slightly less when document sizes are smaller, however, it still averages about 23 milliseconds per connection for workload A/4. In examining the Linux code that is executed when the server log is updated, there is a critical section in which a clean buffer cache entry containing the inode for log file is acquired by the HTTP process updating the log. The HTTP process then locks the inode, writes it, and then releases the lock. Although the

disk is not updated for every write, we believe that this critical section for acquiring and manipulating the inode for the log file is a bottleneck in our web server.

5 Conclusions

This paper examines how and why web server performance changes as the workload at the server varies. We have measured the performance of a PC acting as a standalone web server, running Apache on top of Linux. We varied the workload at the server along two important dimensions: the number of clients accessing the server, and the size of the documents stored on the server.

Our results demonstrate that more clients and larger files stress the web server and operating system in different and surprising ways. For example, we have seen that our web server is robust in terms of sustaining throughput as the number of clients increases. However, we have also seen that increasing average document size decreases connection throughput while increasing byte throughput. This increase in byte throughput is because increases in file size don't significantly increase file system activity, but do increase the amount of useful work the network protocol stack can perform, as a side-effect of requiring longer TCP connections to transfer larger documents. Our results also show the importance of fixed costs in determining web server performance. We have seen that the cost of managing timers for TCP connections in the TIME_WAIT state and updating the server request log are both significant fixed costs incurred for each HTTP request.

References

[1] Jussara M. Almeida, Virgílio Almeida, and David J. Yates. Measuring the behavior of a World-Wide Web server. In *Seventh Conference on High Performance Networking (HPN)*, pages 57–72, White Plains, NY, April 1997. IFIP.

[2] Mark Crovella and Azer Bestavros. Self-similarity in the world-wide web traffic: evidence and possible causes. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Philadelphia, PA, May 1996. ACM.

[3] Yasuhiro Endo, Zheng Wang, J. Bradley Chen, and Margo Seltzer. Using latency to evaluate interactive system performance. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, Seattle, WA, October 1996. USENIX.

[4] Nicholas Gloy, Cliff Young, J. Bradley Chen, and Michael D. Smith. An analysis of dynamic branch prediction schemes on system workloads. In *Proc. of 23rd International Symposium on Computer Architecture*. ACM/IEEE, May 1996.

[5] K. Lai and M. Baker. A performance comparison of UNIX operating systems on the Pentium. In *Proceedings of the 1996 USENIX Conference*, San Diego, CA, January 1996. USENIX.

[6] Carlos Maltzahn, Kathy J. Richardson, and Dirk Grunwald. Performance issues of enterprise level web proxies. In *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, Seattle, WA, June 1997. ACM.

[7] Jeffery C. Mogul. Network behavior of a busy Web server and its clients. Research Report 95/5, DEC Western Research Laboratory, October 1995.

[8] Jeffrey C. Mogul. The case for persistent-connection HTTP. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 299–313, Cambridge, MA, August 1995. ACM.

[9] Venkata N. Padmanabhan and Jeffrey C. Mogul. Improving HTTP latency. In *Proceedings of Second WWW Conference '94: Mosaic and the Web*, pages 995–1005, Chicago, IL, October 1994.

[10] D. Robinson and the Apache Group. *APACHE - An HTTP Server, Reference Manual*, 1995. URL: http://www.apache.org.

[11] G. Trent and M. Sake. *WebStone: The First Generation in HTTP Server Benchmarking*, February 1995. URL: http://www.sgi.com/Products/WebFORCE/WebStone/paper.html.

[12] M. Welsh. *The Linux Bible*. Yggdrasil Computing Incorporated, 2nd edition, 1994.