Finding Speed Bumps: Web Server Performance Analysis and Anomaly Detection via Wide-Spectrum Microbenchmarking

Yaoping Ruan   *Michael S. Cohen*[†]   *Vivek S. Pai*
Department of Computer Science
[†] Department of Electrical Engineering
Princeton University

Abstract

We investigate the use of parameterized microbenchmarks to explore Web server performance over a wider range of workloads than is traditionally represented in "realistic" benchmarks. We find that these "wide-spectrum" tests not only give insights into the strengths and weaknesses of various server software, but also reveal long-standing performance problems that have either been undiscovered or improperly diagnosed. To enable our approach, we have developed "Flexiclient", a componentized testing system that allows the easy creation of parameterized workload generators to test HTTP-based servers.

Even with simple static file workloads, we uncover a number of performance problems on both popular and experimental systems, including the following: severe deterioration of FreeBSD's mincore system call performance with growing memory size on event-driven servers, process scheduler overhead crippling persistent connection benefits for Apache on Linux, and low multiprocessor performance gains on Linux. We additionally discover some other interesting behaviors that run counter to conventional wisdom: the most popular web server benchmark software generates surprisingly small working sets, and select-based servers may not suffer as much from idle persistent connections as previously assumed.

1 Introduction

Network servers continue to be a dominant form of information delivery for a large and growing worldwide population of Internet users. We believe that several factors will drive the demand for network-based information delivery: increases in user population, last-mile bandwidth improvements, and the growing quantity of online information. Web services are currently in their infancy, and their widespread adoption may drive a surge in automatically-generated web traffic, furthering the load on network servers.

Researchers within academia and industry have responded to this trend both by developing optimizations for servers and by developing mechanisms to test their performance. One of the challenges in this work is locating the performance bottlenecks in these increasingly complicated systems. Benchmark developers have focused on developing reality-derived macrobenchmarks that attempt to model many different aspects of behavior seen in "average" systems [5, 22]. While this approach may be useful for server sizing, it hides many performance details resulting from strengths and weaknesses in HTTP software, operating system implementation, and hardware.

One approach to exposing these important details is using narrow, focused microbenchmark techniques. These types of tests analyze performance under very specific and sometimes unrealistic conditions to test particular features of the server. While these tests are clearly useful for gaining insight into specific aspects of server performance, their widespread acceptance is hampered both by the view that these narrowly-focused tests may not reflect real-world performance, and also by the lack of consensus on exactly how these tests should be conducted. Without consensus, results from different sources are rarely comparable.

Our goal in this research is to explore a wide spectrum of server performance by combining elements of micro- and macrobenchmarks. We are particularly interested in characterizing overall performance on varying web server workloads, examining the strengths and weaknesses of different server/OS/hardware combinations, and providing a testbed to fairly evaluate different server software and optimizations. Using both production and experimental server software, we run a bat-

Name	Primary Axis	Seconday Axis	Purpose
Single File	file size	requests / connection	maximum performance
Hot/Cold	total # connections	% hot connections	persistent connection overhead
FileSet	data set size	locality parameter	in-core/disk performance
Degradation	data set size	locality parameter	disk-bound performance

Table 1: Summary of the workload generators

tery of tests on different OS/hardware combinations to provide a comparative analysis of performance. Within each set of tests, the individual test configurations vary only slightly from run to run, allowing more insight into what changes in workload are responsible for differences in performance. Additionally, replacing or modifying the server software can provide insight into the strengths and weaknesses of other servers. Our test framework, called Flexiclient, is designed to be highly configurable and automatic, so we expect that as other interesting scenarios arise, we can incorporate them into our framework.

The outline of the rest of this paper is as follows: We first discuss the general design of our wide-spectrum tests in Section 2, followed by results in Section 3. Our test framework, Flexiclient, is discussed in Section 4, and some avenues for future work are covered in Section 5. We discuss related work in Section 6.

2 Design

To meet our goal of testing a wide scenario of workloads, we design a series of *workload generators* that cover a broad range of the performance spectrum. Each workload generator is a program parameterized along two axes, allowing us to generate large numbers of test scenarios with similar characteristics. The individual workloads attempt to measure different aspects of the server's performance: upper bounds, large filesets, idle connection overhead, and disk behavior. While any particular data point may not provide great insight into server behavior, the trends in performance across multiple tests may be more valuable than a single macrobenchmark. The workload generators are summarized in Table 1 and are discussed individually later in this section.

The rationale for parameterized workload generators is the observation that fixing certain values, such as average transfer size or locality, is likely to omit interesting workloads that may be present in important real-world environments. For example, a site serving banner ads (e.g. DoubleClick) is likely to be handling a relatively small set of large transfers, whereas a community-building site (e.g. GeoCities) is likely to have a very large set of files accessed with low locality. The SpecWeb static file workload is modeled after a web hosting center, and uses a file popularity model following a Zipf distribution [25] with alpha value of 1.0. However, analysis of actual web access logs have shown alpha values ranging from 0.6 [7] to 1.5 [18], leaving possibly large portions of the performance space underrepresented in standard benchmarks.

Furthermore, we also believe that using a large number of narrowly-tailored tests is more likely to aid in performance debugging. Comparing results from tests with only a single changed variable provides a simple way of determining which aspect of a workload is causing performance problems. For example, if two tests using the same workload only differ in the data set size used, any performance difference is likely to result from the parts of the server/OS that are related to data set size. In contrast, when tests scale many factors with throughput, developers do not have a simple way of identifying which changed factor is the throughput limit.

Our approach to characterization via a suite of tests has parallels in the computer architecture community, where benchmarks such as SpecCPU [21] use a suite of 26 tests to determine the expected performance of a processor. These tests are split into two groups, where one is dominated by integer performance and the other is largely floating-point. The differences stem from the source of the tests – the programs in SpecCPU are drawn from real programs, while our workload generators are largely synthetic. However, early efforts at CPU benchmark often in-

class size	100 – 900	1 – 9 KB	10 – 90 KB	100 – 900 KB
class weight	35%	50%	14%	1%

file size	1	2	3	4	5	6	7	8	9
file weight	3.9%	5.9%	8.8%	17.7%	35.3%	11.8%	7.1%	5.0%	4.4%

Table 2: Class and File Weights – In our Fileset workload, we use the same request distribution characteristics as SpecWeb. Each class consists of 9 evenly-spaced files, and the file's probability is the class weight times the weight within the class.

volved synthetic workloads, and we expect that over time, our testing framework will accommodate workloads derived from a variety of web sites.

2.1 Single File Workload

In order to determine the maximum capacity of the system, this test has all clients continuously ask for the same file. These conditions are admittedly unrealistic, but we expect that after the first response, hardware and software caches are loaded and the file is always served in the fastest path possible in the server. Successive runs of the test use different files, and the set of files requested during the various tests ranges from 100 bytes to one megabyte. Non-persistent connections as well as persistent connections of various durations (number of requests served per connection) are tested. We use the term "groups" to refer to the number of requests per connection.

The resulting graph from this test quickly shows two aspects of a server's performance: its maximum performance at various file sizes, which provides a baseline for other tests, and the difference between its per-connection and per-request overheads. These differences are apparent as a "spread" of values at a given file size. A large ratio of spread size to bandwidth indicates that the per-connection cost is significantly higher than the per-request costs. Likewise, a narrow spread indicates that the per-request processing is the dominant component of processing cost. Particularly with small files and aggressive servers, per-request costs can be dominated by TCP setup/teardown overheads. In these situations, using persistent connections may significantly improve server capacity.

2.2 Hot/Cold Workload

This test measures the overhead of idle connections when using persistent connections. In wide-area network environments, the use of persistent connections can reduce the delays associated with TCP connection establishment [16], but idle connections can cause extra processing on the server and reduce its performance. Idle connections may arise from client browsers that have finished downloading all of the objects for one page and are waiting for the user to select the next link. Server capacity can increase due to the elimination of TCP overhead, but it can also degrade due to managing the idle connections. At the same time, server administrators must consider the benefit to users stemming from eliminating many of the TCP round-trip delays for connection establishment.

During the test, each client opens a number of connections to the system under test, but only a given percentage of the connections have HTTP requests in progress at any time. The connections which are sending requests are called "hot" connections, while connections which do not have any HTTP requests in progress are called "cold" connections. Over the course of the test, all connections are hot at some time to prevent the server software from identifying hot connections and short-circuiting the test. The test measures performance for different numbers of total connections as well as different percentages of hot connections.

2.3 Fileset Workload

To measure the impact of data set size and document popularity on server performance, we borrow the workload profile of the SpecWeb [22] tests. The dynamic object sizes in this test attempt to model the requests of real web sites in a web hosting environment. In this model, the data set size is controlled by specifying the number of "sets." Each set consists of four classes of files, and each

Data Set Size(MB)	$\alpha = 1.0$				$\alpha = 0.0$			
	50%	90%	95%	99%	50%	90%	95%	99%
512	0.79	9.48	16.69	49.02	2.49	15.15	25.53	56.47
1024	1.18	17.35	30.95	93.11	4.98	30.26	51.03	112.94
2048	1.75	31.83	57.51	176.25	9.93	60.33	101.79	225.33
4096	2.57	58.47	107.27	333.37	19.82	120.52	203.32	450.12

Table 3: Working Sets for Fileset workload – For each data set size, we calculate the working set size (in MB) for various memory hit rates. The sizes are for the Zipf parameters of one (default) and zero.

class contains nine files. The details about the file weights and class weights are given in Table 2.

An other mechanism used to create unequal popularity is to weight the various sets using the Zipf distribution. In this model, the n^{th} most popular object is given a weight of 1/n. Variations of the Zipf distribution introduce a factor, α, which adjusts the popularity by making the weight $1/n^{\alpha}$. The default Zipf distribution, which is used in SpecWeb, uses an α value of 1.0. Larger values for α concentrate the probability, while values near zero cause the set's probability to approach random.

We depart from the SpecWeb usage model by testing throughput at various data set sizes and by adjusting the α factor as well. The resulting graph shows the system capacity not only as the data set size changes, but also as document popularity changes. We hope that these changes allow sites to find a data point that is reflective of their document popularity distribution, rather than assuming a distribution pattern intended to reflect some "average" site. By testing a range of data set sizes, we expect to see system performance on in-memory workloads and disk-bound workloads.

2.4 Degradation Workload

To our surprise, we found that the Fileset workload, modeled after the SpecWeb workload, produced much higher locality than the data set sizes might imply. The result was that while the actual data set of this test was large, the working set of the files being actively accessed was fairly small. In the Fileset workload, the 36 files which comprise one set consume a total space of roughly 5 MB, yielding an average static file size of over 130 KB. However, with the unequal selection of classes and files within a class, the average dynamic object size is roughly 15 KB.

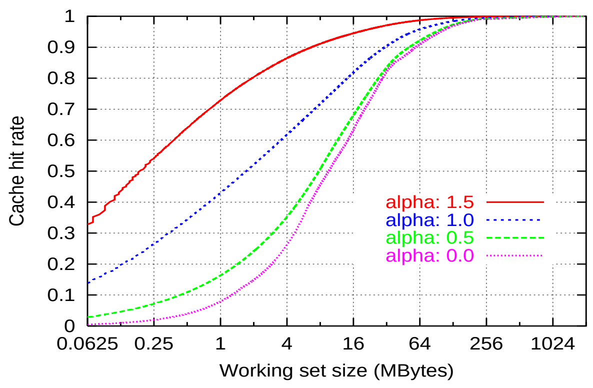

Figure 1: Memory size needed for various hit rates - 2GB data set size with Fileset workload

The calculated working set sizes for the Fileset workload are shown in Figure 1 and Table 3. Note that even the 99% in-memory hit rates requires a working set only about one-tenth the size of the entire data set. The high cost of disk access relative to memory access will cause noticeable performance degradation even at a 1% miss rate. However, other aspects of the system, such as cache locality, TLB performance, etc., may be under-tested in this scenario.

To better understand the effects of larger working set sizes, we devised a new workload that eliminates the size bias and uses a large number of fixed-size files to generate the working set. We considered using 15KB files to have an average transfer size that matches our previous workload. However, it would have required roughly 68 files to occupy 1 MB of space, or roughly 70,000 files per GB. To reduce this number slightly, we chose a fixed size of 50KB, similar to what an image serve might experience. We still preserve the ability to adjust the locality of the workload by adjusting the Zipf value α.

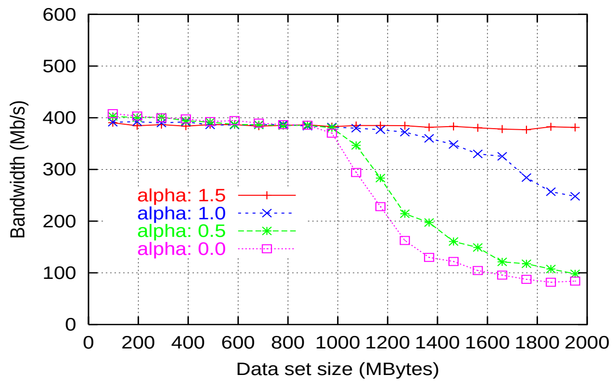

Figure 2: Flash/Linux Fileset workload

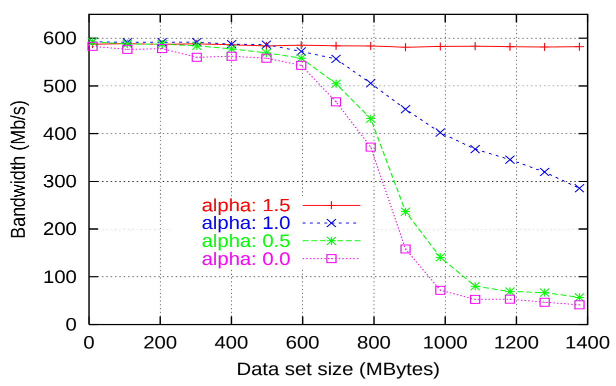

Figure 3: Flash/Linux Degradation workload

3 Experimental Results

We run our full suite of tests to evaluate OS/hardware combinations, and we also examine the performance of some other web server packages. Our original goal was not to search for any specific performance problems, but rather to conduct comparative performance analysis. However, in the process of testing, we observe some interesting performance anomalies, some results that run counter to conventional wisdom, and some evidence that addresses the complexity of comparing performance of different operating systems. We present our experimental setup and these comparative results in this section.

3.1 Experimental Setup

Our test server is a Pentium III Xeon running at 933 MHz. This machine uses an Intel SBT2 motherboard with 1 GB of physical memory and a Promise Ultra DMA 66 IDE controller for a 60GB 5400 RPM Maxtor Diamond Max Hard Drive. This motherboard is capable of dual processor support, but we run it as a single-processor system unless otherwise indicated. All clients and the server are connected to an Intel Express Gigabit Ethernet switch using Intel E1000 Gigabit Ethernet adapters. All machines are configured to use the default (1500 byte) MTU. The clients consist of six Pentium II machines running at 300 MHz, with 128 MB of memory per machine. The relatively slow clients do not present any performance bottlenecks in our tests, since our client software uses the same event-driven design common in high-performance servers.

We test various hardware/OS/server combinations and present some of the more surprising observations below. For uniprocessor testing, we use the FreeBSD 4.5 operating system and Redhat 7.1 with the Linux 2.4.17 kernel. For multiprocessor testing, we use the Linux 2.4.10 kernel with SMP support. We use the Flash [19] and Apache 1.3.20 [1] web servers to test both experimental event-driven servers as well as production multiple-process servers.

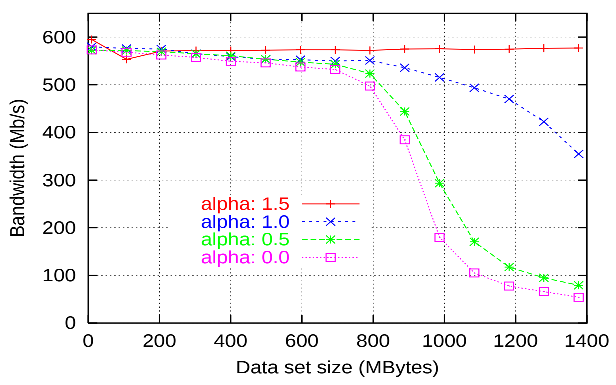

Figure 4: Flash/FreeBSD Degradation workload

3.2 Linux Fileset & Degradation workloads

In our initial testing of the workload generators, we observed some surprising results when running the SpecWeb-inspired Fileset workloads. These results led us to the creation of the Degradation workload. The results for Fileset workload on Linux with the Flash Web Server are shown in Figure 2.

The qualitative properties of the graph are not surprising – the performance tends to drop once the data set size exceeds the size of physical mem-

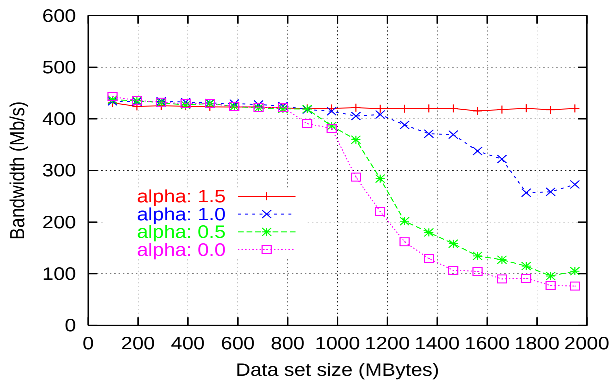

Figure 5: Flash/Linux Fileset workload, dual processor

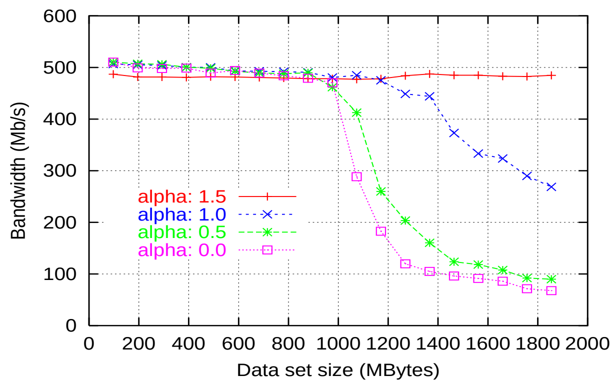

Figure 6: Flash/Linux Fileset workload, dual processor, two processes

ory, which is 1 GB for our server. The rate of the drop in performance, however, is surprising. The line that models the SpecWeb workload is the line with the α value 1. Even at a data set size 50% larger than the machine's physical memory, the performance drop is only on the order of 15%. Naively, we would expect that one-third of the requests would result in cache misses, causing a sharp drop in throughput as disk performance becomes the limiting factor in performance. However, as the cumulative distribution plots in Figure 1 shows, one property of the SpecWeb file frequency distribution is that surprisingly high hit rates can be achieved with small amounts of memory.

The Degradation workload was designed to place more stress on the filesystem, and the results in Figure 3 confirm its premise. While the larger average file size causes the in-memory bandwidths to be higher, the performance drops sharply when the data set size exceeds the system's physical memory. The effect is most noticeable when compared to the Fileset results in Figure 2. The Degradation workload results for Flash on FreeBSD are shown in Figure 4, with a marked performance improvement over Flash/Linux on this test. As we will see in later tests, FreeBSD's disk-based performance outperforms Linux on both web servers tested.

3.3 Linux single / dual processor

To measure opportunities for parallelism, we add a second processor to the system and rerun the Fileset workload tests on Linux. Surprisingly, we find that even on a workload that one assumes is embarrassingly parallel, the performance gains we see are relatively modest.

The results for the Flash Web Server are shown in Figures 5 and 6, run using both one and two instances of Flash on two processors. Since Flash uses an event-driven architecture, we have the capability of running one main Flash process on a two-processor system. In this scenario, we expect that performance gains may come from top-half/bottom-half parallelism in the kernel. With interrupt-intensive workloads like web servers, one processor can handle low-level interrupts while the other processor handles system calls.

Figures 7 and 8 show the performance of the Apache web server using one and two processors. We see performance gains in the range of 30-50% on the in-memory portions of the workload even though the multiple-process model of Apache should easily take advantage of a second processor. However, the Apache server performs a large number of system calls per request, and contention may cause the limited performance gain.

While the absolute performance gains using Flash are higher than that seen for Apache, the relative gain using Flash is lower. Neither server, however, sees linear scalability. Even top-half/bottom-half parallelism only yields marginal gains. We have not explored what causes the bottleneck in these tests. One possibility is that the two processors are inefficiently sharing interrupts. In this scenario, using multiple network cards with processor interrupt affinity is a com-

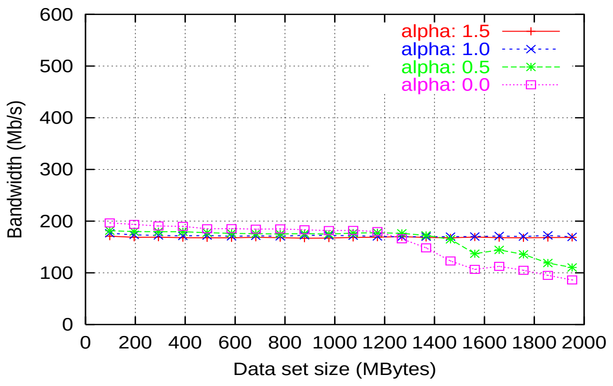

Figure 7: Apache/Linux Fileset workload

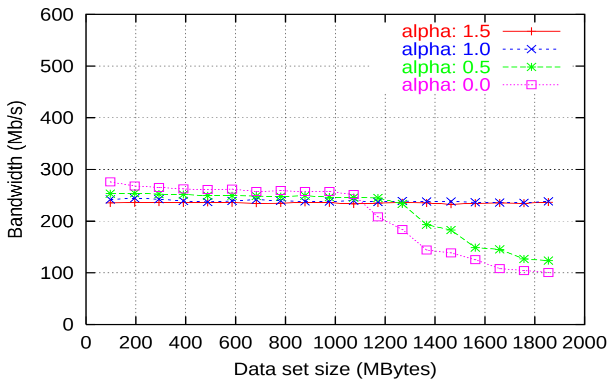

Figure 8: Apache/Linux Fileset workload, dual processor

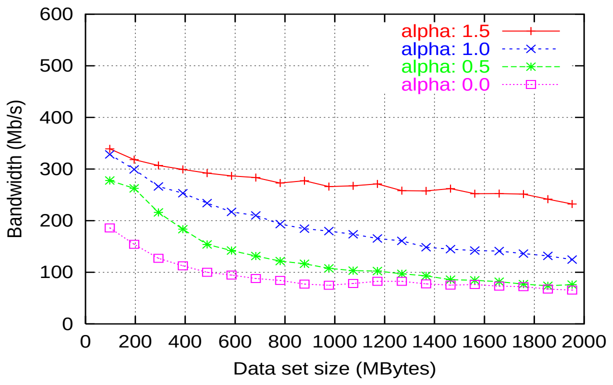

Figure 9: Flash/FreeBSD Fileset workload

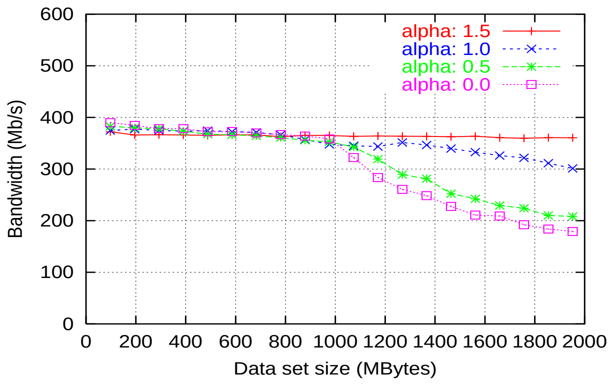

Figure 10: Flash/FreeBSD Fileset workload without mincore

mon technique for improving performance. However, if network access was the limiting factor, we would expect larger gains for Apache, which stresses the network less than Flash.

3.4 FreeBSD mincore performance

When we ran the Flash web server on FreeBSD, we were quite surprised with the initial results for the Fileset workload. These results are shown in Figure 9, and have some surprising traits compared with the results in Figure 2 for Linux: the absolute performance numbers are much lower than those for Linux on the same hardware, the server performance drops even when the entire workload should fit in physical memory, and the performance drop seems correlated with the working set size and locality. No sharp performance drop is visible once the data set exceeds physical memory size, but this effect is less of a concern than the poor in-memory performance.

We immediately suspected something related to the virtual memory (VM) system since the Flash web server aggressively uses memory-mapped files with caching, avoiding filesystem/disk access for in-memory workloads, and this problem does not occur with the results of Apache, which is shown in Figure 11. For in-memory workloads, once the cache is loaded, we expect no cache misses or memory map/unmap operations. In order to avoid page faults when sending files, the Flash web server heavily relies on the mincore system call to determine the memory-residence of cached memory mapped pages. This call is one of the few VM-related system calls to be expected on cached workloads, so we tested the server with this call elided.

The performance results of a mincore-less version of Flash on this test are shown in Figure 10, confirming our suspicion that the performance of the mincore system call on FreeBSD is respon-

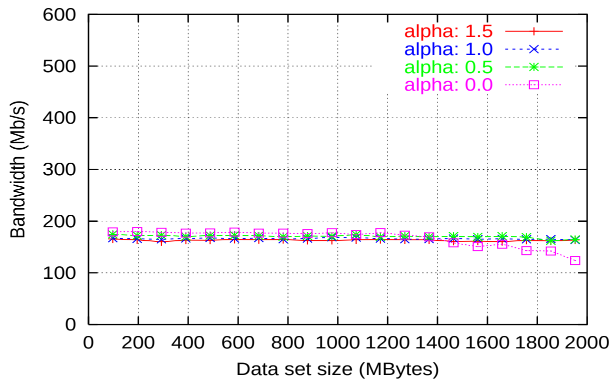

Figure 11: Apache/FreeBSD Fileset workload

sible for the poor performance we observed. To the best of our knowledge, this problem has never been documented, despite the growing popularity of memory-mapped event-driven servers. After corresponding with a FreeBSD maintainer, we received a patch that replaced a linked list VM map structure with a splay tree, and the performance of the mincore-enabled Flash returned to levels comparable to Figure 10. The acceptable performance of mincore on Linux is likely due to its relatively recent introduction, where tests with or without mincore enabled show little difference.

3.5 Linux vs FreeBSD

Comparing the Fileset results for Linux and FreeBSD (Figures 2 and 10) also reveals some other interesting performance comparisons. Note that the in-memory performance of the Linux tests yields a bandwidth that hovers near 400 Mb/s. In comparison, the same portion of the FreeBSD graph shows performance that is roughly 5% lower, in the range of 380 Mb/s. In this portion of the performance spectrum, Flash running on Linux shows a slight edge. However, the performance for out-of-memory workloads is very different for these two operating systems. As the data set size approaches 2 GB, the FreeBSD performance degrades to roughly 200 Mb/s. In contrast, the Linux results show much steeper degradation, on the order of 100 Mb/s. We have not explored the details of why this reversal occurs, other than noting that different aspects of the operating system are stressed in the two regimes. We do note, however, that these results demonstrate why narrow-spectrum benchmarks may not reveal many of the interesting facets of performance.

The results for the Apache server are shown in Figures 7 and 11, and a similar pattern can be observed. The in-memory Linux results are marginally higher than those for FreeBSD, but the out-of-memory portions of the test show greater degradation on Linux. When compared with Flash, Apache shows relatively smaller degradation when accessing disk, but this effect is due to its lower in-memory performance. The disk-bound results for Apache show lower absolute numbers than Flash.

3.6 Persistent Connection Behaviors

Our tests also give some insight into the behavior of persistent connections on different servers, including some results that run counter to conventional wisdom. The single-file tests in Flash and Apache are shown in Figures 12 and 13. These tests measure the bandwidth obtained by repeatedly requesting a file, and the various lines in the graph show the effects of using the same connection for varying numbers of requests. While these tests are unrealistic, they do show the maximum bandwidth achievable by the servers, both for non-persistent connections and for various numbers of requests on each persistent connections.

The "spread" of the lines in these figures shows the maximum capacity benefit of persistent connections at various file sizes. A large spread indicates potential capacity improvement by using persistent connections. However, even using persistent connections with a small-spread server may have latency benefits by avoiding most round-trip delays associated with TCP connection setup/teardown.

The more surprising results are shown in Figures 14 and 15, where the Hot/Cold workload is used to measure the impact of persistent connections. In both graphs, the performance initially improves as the total number of connections increases. We attribute this behavior to the delay between the server finishing one response and the client starting the next request. This effect stabilizes as the number of in-progress requests overcomes the latencies. The results for Flash then stabilize, whereas the Apache performance drops with increasing numbers of connections.

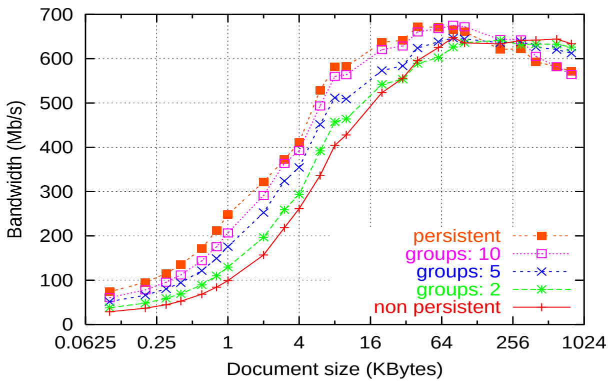

Figure 12: Flash/Linux single file workload

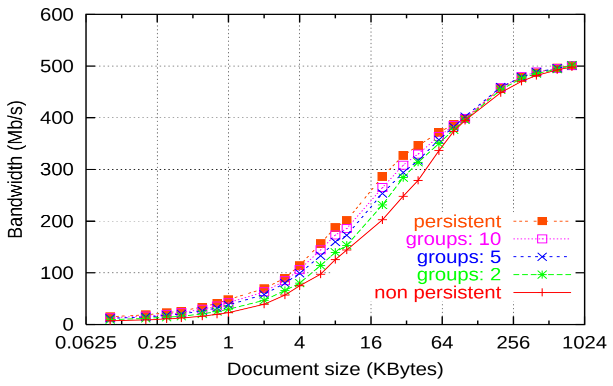

Figure 13: Apache/Linux single file workload

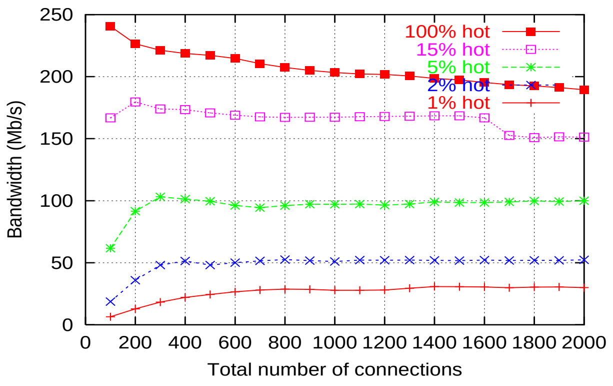

Figure 14: Flash/Linux Hot/Cold workload

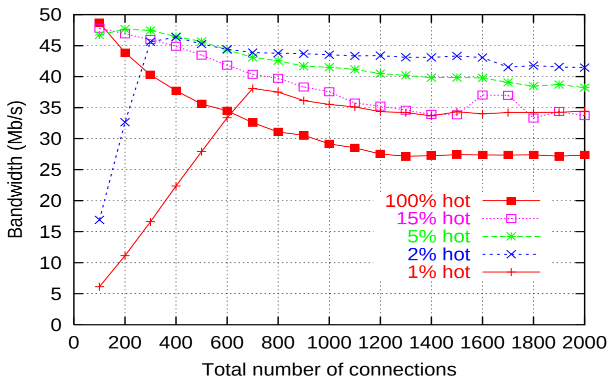

Figure 15: Apache/Linux Hot/Cold workload

The Flash results are qualitatively intuitive (more idle connections degrade performance) but quantitatively surprising – having 99% of the connections being idle causes less than a factor of 8 drop in performance, while 98% of the connections being idle causes only a factor of 4 drop in performance. By the time the percentage hot reaches 5%, less than half the capacity of the system is being wasted on idle connections. Furthermore, the performance drop is mostly stable with the total number of connections, indicating that busier servers may not suffer disproportionately from cold connections as long as the percentage of cold connections is comparable to less-busy servers.

3.7 Linux Scheduler Overhead

The surprising aspect of the Apache on Linux results is the degradation of performance with increasing numbers of connections, especially when all connections are busy. In fact, the 100% busy line performs worse than the 1% busy line and the degradation rate seems to be related to the number of active connections/processes. We can likely eliminate the filesystem and the VM system as possible causes, since this test uses only a single 1KB file.

What remains after eliminating these choices is the process management overhead in Linux, and to confirm our suspicion, we run the same test on FreeBSD. The results, shown in Figure 16, demonstrate that this problem seems to only occur with Apache on Linux, suggesting an OS issue rather than an application-level problem. Further investigation suggests the reason for Apache's degradation appears to be Linux's scheduler, which scans all ready processes at every context switch. This behavior is consistent with the observation that having 99% of the processes idle is yielding better performance than having all processes busy.

Our suspicions were confirmed by testing a new scheduler in development by the Linux kernel

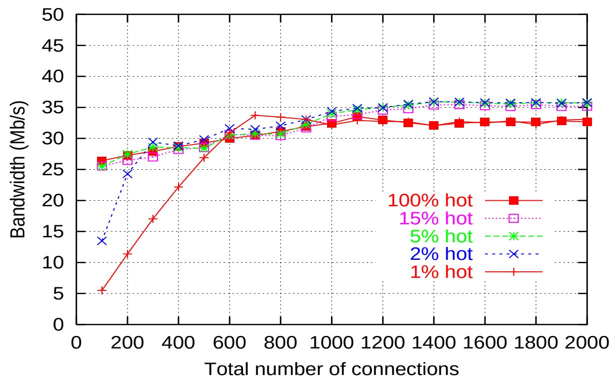

Figure 16: Apache/FreeBSD Hot/Cold workload

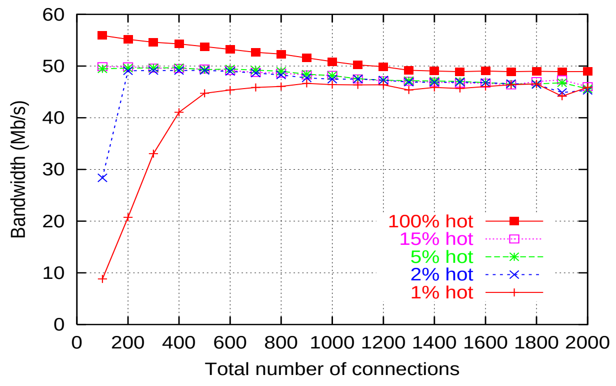

Figure 17: Apache/Linux Hot/Cold workload with O(1) scheduler

community. This scheduler, known as the O(1) scheduler, is targeted toward large SMP machines and does not degrade with the number of ready processes. The results from this scheduler are shown in Figure 17. It appears that other researchers may have been unknowingly affected by this problem. An earlier study on the performance of persistent connections [6] found few benefits using a combination of Linux and Apache.

4 Flexiclient

To facilitate the tests in the previous section, we developed a composable testing framework called Flexiclient, which consists of four components: the workload generators, the testing configurations, the client core, and the report generators. The testing process is automated by scripts that control the entire end-to-end process for our predefined suite of tests. However, the individual components can be run independently and used in other contexts. The main difference between Flexiclient and earlier benchmark tools is the intended scope of operation – our goal is to develop a simple framework that enables the development of new workloads and new tests, rather than a single monolithic testing tool.

One of the novel features of Flexiclient is the separation of workload generation from the mechanics of request/response management. The workload generators are separate programs that communicate with the "client core" via Unix pipes. As a result, tailoring benchmark workloads becomes relatively simple, requiring no knowledge of HTTP mechanics. Furthermore, workloads are composable, so that behaviors can be changed across a number of workloads without altering each test. For example, developers can easily alter web server workloads to customize cacheability behavior when testing caching proxy servers.

The client core in Flexiclient performs all of the HTTP request generation and response processing in an efficient manner, using the same kind of event-driven framework used by high-performance servers. The core can currently issue requests at predetermined rates or in an closed-loop infinite-demand model. If client machines are of the same caliber as the server under test, only a small number of client machines should be required to saturate the server. By reducing the hardware requirements of the test infrastructure, we expect that it will have a wider appeal than more resource-hungry tests.

The typical usage for this tool is the following: a configuration file specifying the server machine and the set of clients is created, and the testing scripts are started. A controller process launches Flexiclient processes on each of the client machines and starts the test. The script will run each of the workload generators in dozens of configurations, yielding hundreds of individual test results. These results are then collected by the report generators, and the final output is produced. This process is repeated for each server software package under test.

5 Future Work

For this paper, we intentionally focused only on the performance of static HTTP requests to demonstrate the power of our wide-spectrum microbenchmark approach. However, our framework is well-suited for other types of HTTP testing, and we will pursue these in the future. Two main avenues for future development of Flexiclient are incorporating a wider variety of workloads, and handling more complicated types of traffic.

With respect to other workloads, we currently do not address the issue of dynamic content or web services, and these may be interesting areas for the development of new workload generators. The issues that naturally arise would be what kinds of dynamic content to include and how to meaningfully test its performance. We expect to begin by incorporating some "standard" kinds of CGI applications, such as ad rotation, customization, and search. However, since the Flexiclient core is largely workload-agnostic, the mechanics of adding such generators should be relatively simple.

We have also used only a small subset of HTTP features, limiting ourselves current to just the GET method and some pipelining. However, we could easily begin using some of the features that are heavily used by real client but neglected in current HTTP testing, such as conditional features (e.g., if-modified-since) or byte-range support. A more ambitious approach would be attempting to use Flexiclient for other HTTP environments, such as proxy caches. While specialized tools such as Web-Polygraph [23] exist to test proxies under realistic traffic loads, Flexiclient's philosophy of wide-spectrum testing may locate performance bottlenecks in proxy servers in much the same way that we have shown with web servers.

6 Related Work

Several researchers have used both narrowly-targeted benchmarks and instrumentation to examine particular areas of operating system and web server performance. Yates et al. [24] examined operating system behavior under web workloads. lmbench [14] is a portable suite for OS performance analysis. And Seltzer et al. [20] proposed application-specific benchmarking. But we differ from them by the workloads and are looking for wider range performance characteristics.

Microbenchmarks are also used for system optimization. Banga et al. [4] have examined the performance drain caused by idle connections. Maltzahn et al. [12] have examined disk behavior and memory page fault behaviors for web proxy workloads. Some of these developments led to improvement of existing server software. For example, Hu et al. [9] optimized the behavior of Apache. Other servers have used entirely different software architectures to achieve performance – some in user-space [8, 19], and others inside the kernel [11, 10]. Some of these systems use special instrumentation for the measurement and the microbenchmarks used in them are heavily purpose oriented compare to those we discuss here.

We share some similarities with existing web server benchmarks though we have different approaches and intended goals. WebStone [15] has grown from relatively simple tools released by SGI to a family of benchmarks. Arlitt et al. [2] performed some of the earliest studies of traffic properties, while Manley et al. [13] examined how those properties were changing over time. These types of studies fed back into the testing arena, and WebStone's early popularity with hardware vendors has been largely supplanted by the committee-governed SpecWeb [22] benchmarks, which attempt to generate realistic traffic. Study of more accurate profile modeling of request characteristics led to the development of the SURGE macrobenchmark [5]. This focus on more accurate traffic generation can be incorporated into future workload generators in Flexiclient. These benchmarks try to evaluate system's overall performance while our work provides insights for performance debugging.

In the area of measurement tools, httperf [17] is a benchmark tool that extends some of the overload generation work from S-Client [3] while adding support for a variety of workloads. While we share the goals of flexibility as httperf, we focus on wide-spectrum testing as a component of test infrastructure development.

7 Conclusion

This paper presents a new technique for exploring server performance, wide-spectrum testing via parameterized microbenchmarks. We describe the framework, known as Flexiclient, that we have developed to automate the entire process. This approach allows us to characterize server performance over a wide range of workloads and to more easily understand the effects of workload variables on performance. We show that even with a seemingly simple set of workloads, we can uncover a range of performance problems in widely-studied servers and operating systems and often identify the exact source. We believe that this approach supplements the “realism”-oriented macrobenchmarks that are becoming more popular in server testing.

8 REFERENCES

[1] Apache Software Foundation. The Apache Web Server. http://www.apache.org/.

[2] M. F. Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. In *Proceedings of the ACM SIGMETRICS '96 Conference*, pages 126–137, Philadelphia, PA, Apr. 1996.

[3] G. Banga and P. Druschel. Measuring the capacity of a web server. In *USENIX Symposium on Internet Technologies and Systems*, 1997.

[4] G. Banga, J. C. Mogul, and P. Druschel. A scalable and explicit event delivery mechanism for UNIX. In *USENIX Annual Technical Conference*, pages 253–265, June 1999.

[5] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151–160, 1998.

[6] P. Barford and M. Crovella. A performance evaluation of hyper text transfer protocols. In *Proceedings of ACM SIGMETRICS '99*, May 1999.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, pages 126–134, 1999.

[8] J. C. Hu, S. Mungee, and D. C. Schmidt. Techniques for developing and measuring high-performance Web servers over ATM networks. In *Proceedings of the IEEE Infocom Conference*, San Francisco, CA, Mar. 1998.

[9] Y. Hu, A. Nanda, and Q. Yang. Measurement, analysis and performance improvement of the apache web server. In *Proceedings of the 18th IEEE International Performance, Computing and Communications Conference (IPCCC '99)*, February 1999.

[10] P. Joubert, R. King, R. Neves, M. Russinovich, and J. Tracey. High performance memory based web caches: Kernel and user space performance. In *USENIX Annual Technical Conference*, June 2001.

[11] M. F. Kaashoek, D. R. Engler, G. R. Ganger, and D. A. Wallach. Server Operating Systems. In *Proceedings of the 1996 ACM SIGOPS European Workshop*, pages 141–148, Connemara, Ireland, Sept. 1996.

[12] C. Maltzahn, K. J. Richardson, and D. Grunwald. Performance Issues of Enterprise Level Web Proxies. In *Proceedings of the ACM SIGMETRICS '97 Conference*, pages 13–23, Seattle, WA, June 1997.

[13] S. Manley and M. Seltzer. Web Facts and Fantasy. In *USENIX Symposium on Internet Technologies and Systems*, pages 125–134, Monterey, CA, Dec. 1997.

[14] L. W. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conference*, pages 279–294, 1996.

[15] Mindcraft. Webstone: The benchmark for web servers. http://www.mindcraft.com/webstone/.

[16] J. C. Mogul. The Case for Persistent-Connection HTTP. Technical Report 95/4, DEC Western Research Lab, Palo Alto CA, 1995.

[17] D. Mosberger and T. Jin. httperf: A tool for measuring web server performance. In *First Workshop on Internet Server Performance*, pages 59—67. ACM, June 1998.

[18] E. Nahum. Deconstructing SPECweb99. In *7th International Workshop on Web Content Caching and Distribution (WCW)*, Boulder, CO, August 2002.

[19] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *USENIX Annual Technical Conference*, pages 199–212, June 1999.

[20] M. I. Seltzer, D. Krinsky, K. A. Smith, and X. Zhang. The case for application-specific benchmarking. In *Workshop on Hot Topics in Operating Systems*, pages 102–, 1999.

[21] Standard Performance Evaluation Corporation. SPEC CPU Benchmark. http://www.spec.org/osg/cpu2000/.

[22] Standard Performance Evaluation Corporation. SPEC Web 96 & 99 Benchmarks. http://www.spec.org/osg/web96/ and http://www.spec.org/osg/web99/.

[23] Web Polygraph. http://www.web-polygraph.org/.

[24] D. Yates, V. Almeida, and J. Almeida. On the interaction between an operating system and Web server. Technical Report TR-97-012, Boston University, CS Dept., Boston MA, 1997.

[25] G. K. Zipf. *Relative Frequency as a Determinant of Phonetic Change*. Reprinted from the Harvard Studies in Classical Philiology, Volume XL, 1929.