Performance Characterization of the Pentium III Using Web Workloads

http://www.cs.cmu.edu/~bianca/architecture/project.html

Shuchi Chawla
shuchi@cs.cmu.edu

Rose Hoberman
roseh@cs.cmu.edu

Bianca Schroeder
bianca@cs.cmu.edu

1 Introduction

Recent work shows that the characteristics of commercial applications can be significantly different from the numerical and scientific applications that are typically used in benchmarks (e.g. the SPEC suite). Commercial applications differ significantly from technical applications in branch behavior, percentage of operating system activity, I/O behavior, and the number of user processes. As a result, commercial applications cannot fully benefit from many of the recent architectural innovations and generally have reduced performance [13].

Recently there have been a significant number of studies focusing on architectural performance for database applications [4]. Additional studies of commercial workload performance have analyzed the performance of multimedia [16] and desktop applications [12] as well as operating systems [15]. However, no studies have been completed on one of the most prevalent non-scientific application: web servers.

This study measures the performance of a modern architecture when running a web server. To obtain the most realistic and thus the most useful performance statistics, we use real world workloads and a server configuration that is among the most common for commercial web servers today.

More specifically, we measure the performance of two web servers (Flash and Apache) on a Pentium III hardware platform running Linux. In order to do this, we generate requests based on a web trace and use hardware performance counters to measure the performance of the hardware in terms of CPI, resource stalls, cache performance and branch mispredictions.

In this study we find that the hardware performance when running a web server is still far from the performance achieved when running standard integer benchmarks. More surprisingly, even compared to database systems, which are known to have suboptimal hardware performance, web servers perform worse with respect to many hardware performance metrics. We also found that even though Flash is considered to be a high-performance web server, it doesn't compare favorably with Apache in terms of CPI and cache misses.

2 Experimental Setup

In this section, we describe our experimental setup. Section 2.1 explains the architectural layout in our experiments. Section 2.2 describes the web server software run by the server. Section 2.3 defines the trace-based workload that we use and the request generator at the clients. Section 2.4 specifies the metrics we use to evaluate performance, and Section 2.5 describes the software tools we use to gather statistics for these metrics.

2.1 The Hardware

Our experiment setup involves two machines connected by a 10/100 Ethernet switch.

Each machine has an Intel Pentium III 700 MHz processor, 256 MB RAM and an 8-way associative second level cache of 256 KB. The first level cache is split with 16 KB for data and 16 KB for instructions.

One of the machines is designated as the server and runs one of the two web servers described in Section 2.2. The other machine acts as web clients and generates requests as described in Sections 2.3. The switch and the network cards of the machines are forced into 10Mbps mode.

Both the client and the server machine run Linux 2.4.13.

2.2 The Web Server Software

Today's web servers can be divided into two classes based on how they handle multiple requests simultaneously. Web servers in the first class use multiple threads or processes to handle several requests at a time. The second class of web servers uses a single process event-driven (SPED) architecture, and is based on the "select()" call which checks for I/O operations that have completed.

To see how these web server design choices affect the architectural performance, we conducted experiments on both types of web servers. As a representative of the SPED architecture, we chose Flash [14], a high-performance research server. As a multi-threaded server we used Apache 1.3.14 [7], a commercial-grade server with freely-available source-code. Apache is the most widely used web-server on the web with a 57% market share as of November 2001 [1].

The Apache web server uses multiple processes to handle different requests. It spawns a new process on every request and this process handles all the tasks required for serving the request. The Flash web server uses a very different scheme. It uses only two processes: one main process handling requests, and a helper process handling disk accesses.

2.3 The Workload

The client machine generates requests based on a 1-day trace from the Soccer World Cup 1998 that we downloaded from the Internet Traffic Archive [8]. The 1-day trace contains 4.5 million HTTP requests, virtually all of which are *static*.

Some statistics about our trace workload that will be useful in the analysis of our measurements: The mean file size requested is 5K bytes. The min size file requested is a 41 byte file. The max size file requested is a 2.020644 MB file. The distribution of the file sizes requested fits a *heavy-tailed* Pareto distribution. The largest $< 3\%$ of the requests make up $> 50\%$ of the total load, exhibiting a strong heavy-tailed property. 50% of files have size less than 1K bytes. 90% of files have size less than 9.3K bytes.

In our experiments, we use `sclient` [5] for creating connections at the client machines. The original version of `sclient` makes requests for a certain file in periodic intervals. We modify `sclient` to read in traces and make the requests according to the arrival times and file names given in the trace.

In our experiments we run the web server at different system loads. Running the server at different loads is interesting because the number of requests that the server handles *simultaneously* varies with the load at the server.

To create a particular system load, say 0.8, we simply scale the interarrival times in the trace's request sequence until the average number of bytes requested per second is 8Mb/sec (recall that the server's network card is forced into 10Mb/sec). The scaling factor is easy to compute analytically. It is important to make some adjustments empirically however because simply looking at request sizes doesn't take into account header size and other small idiosyncrasies.

The system loads we used in the three experiments are 0.1, 0.4 and 0.8. Each experiment was run for 100 seconds and repeated multiple times for consistency and to obtain all the necessary measurements.

2.4 Hardware Performance Metrics

Our goal is to evaluate the hardware performance of web servers and compare it to that of benchmarks and other commercial applications.

Ideally, we would like to be able to determine exactly where the time goes. We want to break down the execution time into the following four categories: useful computation, memory stalls, resource stalls, and stalls due to branch mispredictions. In order to do this, we would have to know the exact time penalty associated with events like branch mispredictions or cache misses. However, these penalties are processor dependent, and deriving these numbers is beyond the scope of this project.

Although we are not able to compute an exact breakdown of the time, we can count certain hardware events which give us an idea of relative performance in terms of the categories listed above. We measure the events listed in Table 1 to obtain the following performance metrics:

- General Performance:
  - Average cycles per instruction (CPI).
  - Percent of issued instructions which were completed.
  - Percent of cycles which were stalled on any resource.
- Cache Performance
  - Percent of instruction cache accesses which are misses for L1 and L2 caches, respectively.
  - Percent of data cache accesses which are misses for L1 and L2 caches, respectively.
- Performance of Branch Prediction Hardware
  - Percent of instructions which were conditional branches.
  - Percent of conditional branches which were taken.
  - Percent of conditional branches which were mispredicted.

2.5 Counting Hardware Events

The Pentium III provides two counter registers for measuring hardware events. Each register can measure one of a large number of preset events. To count more than two events, multiple runs are necessary [6].

Instructions related events	Memory access related events
Total cycles	Level 1 instruction cache accesses
Cycles stalled on any resource	Level 1 instruction cache misses
Instructions issued	Level 1 data cache accesses
Instructions completed	Level 1 data cache misses
Branch instructions	Level 2 instruction cache accesses
Conditional branch instructions	Level 2 instruction cache misses
Conditional branch instructions taken	Level 2 total cache accesses
Conditional branch instructions mispredicted	Level 2 total cache misses

Table 1: Events Measured

To access these counters we used software supplied by the Performance API (PAPI) Project [2], which specifies a standard API for accessing hardware performance counters. We used the low level PAPI interface, which provides functions to initialize the counters, assign event codes, and read their values at any time.

To use the PAPI software, we first patched the kernel of the server machine. For both Apache and Flash, we then instrumented the source code, adding code to initialize, read, and output the counter values in a non-intrusive manner. We gathered numbers for each of the 16 events in Table 1 and for system loads of 0.1, 0.4, and 0.8. In addition, each experiment was repeated 6 times, to ensure that our numbers were consistent and reliable. In all cases the standard deviation of each event counted is *less than 3 percent*.

All figures in this report show results for a system load of 0.4, but the statistics for loads of 0.1 and 0.8 are virtually identical.

All results for both web servers are compared to standard benchmarks [10] and database performance numbers reported in [4] and [11].

3 Results

In this section we describe and evaluate the results of our experiments. In Section 3.1 we look at the general hardware performance in terms of instructions per cycle (CPI), and resource stalls. Section 3.2 describes the performance results regarding the branch prediction hardware. Section 3.3 evaluates the cache performance of our system.

3.1 General Performance

We begin by looking at resource-related stalls. The IA-32 Intel Developer's Manual [6] defines resource stalls to include register renaming buffer entries and memory buffer entries, but not stalls due to a full bus queue or too many cache misses, etc. According to this definition, the percent of cycles for which there was a resource related stall was $(15 \pm 0.5)\%$ in all experiments. Because the definition of resource stalls varies with each architecture and study, we were not able to compare these numbers directly with other applications.

Next we look at the average number of cycles per instruction (CPI). Since modern proces-

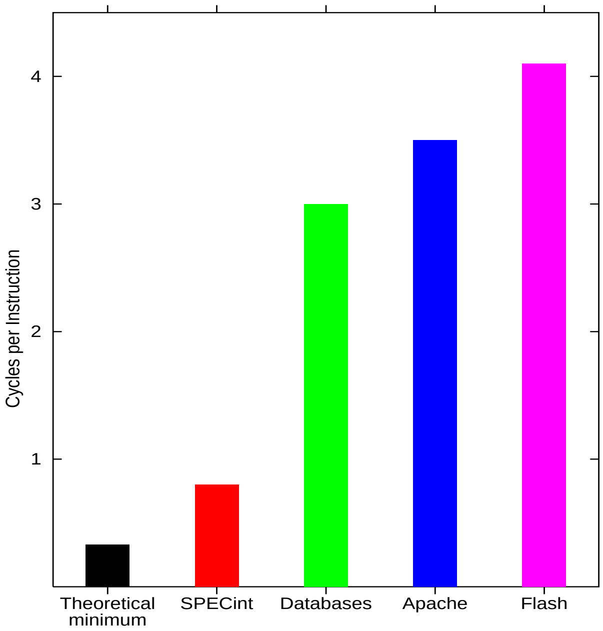

Figure 1: Cycles per instruction

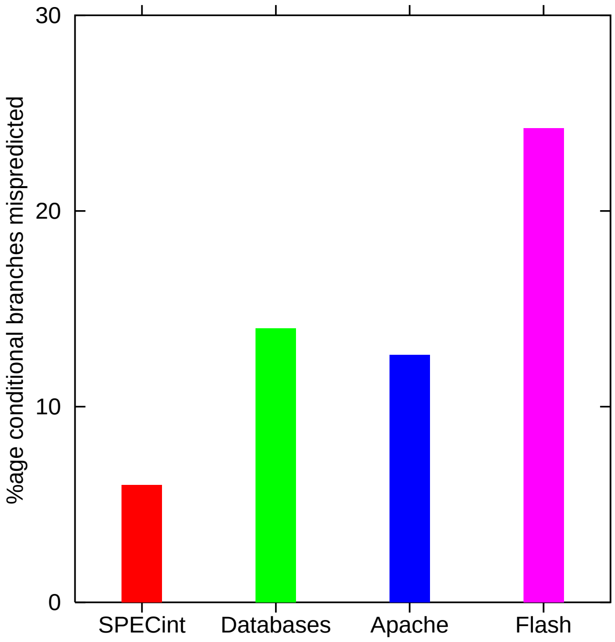

Figure 2: Branch mispredictions for Apache and Flash compared with other applications

sors are able to issue several instruction per cycle this number is ideally below 1. However, we find that the CPI of our web servers is far away from this ideal: Apache has a CPI of 3.5 and Flash has a CPI of 4.1. Figure 1 shows how this compares to the theoretical minimum, the SPECint benchmark, and another commercial application (an Online Transaction Processing System, a database system with a large number of users making many small transactions). While SPECint comes relatively close to the theoretical minimum, the CPI of the commercial applications are up to an order of magnitude higher than the optimum. In particular, observe that the CPI of the web servers is higher than that of any other application we consider!

The poor performance indicated by the high CPI is a result of sub-optimal utilization of many of the components of the processor. Performance results for some of these components are discussed in the following two sections.

3.2 Branch Mispredictions

The performance of the branch predictor is crucial for a processor's performance because branches limit the amount of instruction level parallelism that can be exploited. A large amount of recent research in branch prediction has led to modern architectures which are able to predict more than 90% of the branches in standard benchmarks correctly [17, 9].

Unfortunately, we find that the performance of the branch prediction hardware is not nearly

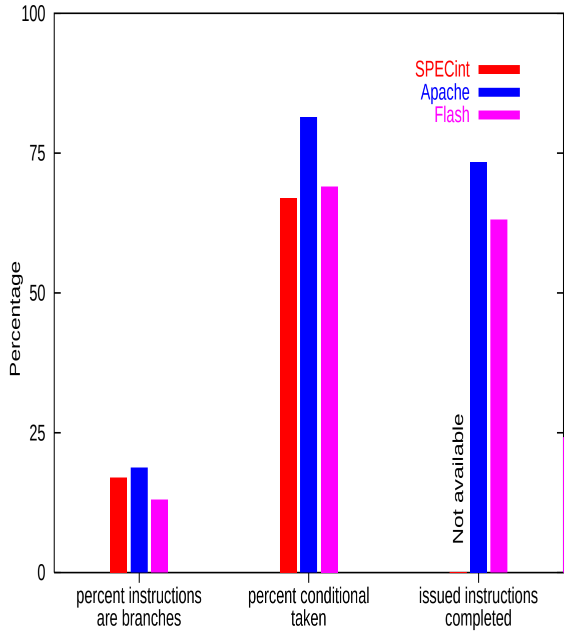

Figure 3: Branch behavior for Apache and Flash compared with SPECInt

as impressive for our web servers. As shown in Figure 2, 12% of the branches are mispredicted for Apache and nearly 25% for Flash.

In the remainder of this section we discuss the performance implications and possible underlying reasons for these high branch misprediction rates.

3.2.1 Performance Implications

The effect that mispredicted branches have on a system's performance depends on the percentage of instructions that are branches. Clearly, if there are hardly any branches in the application branch mispredictions will not affect the overall performance by much. We therefore looked at the percentage of instructions that are branches. Figure 3 shows that for both Flash and Apache the percentage of instructions that are branches is comparable to that of the SpecInt benchmark.

The fact that web servers, compared to benchmarks, have a much higher branch misprediction rate and an equally high percentage of branches might partially explain the high CPI that we observe in Section 3.1.

To validate this assumption we compared the total number of instructions that are issued to the total number of instructions that are retired. This gives us an idea of how many cycles were wasted because of instructions that were speculatively issued and later cancelled because of a branch misprediction. Figure 3 shows that for Apache only 74% of the instructions that are issued are also retired. For Flash the discrepancy between issued and retired instructions is even bigger: only 62% of all issued instructions complete.

This shows that a significant percentage of cycles is wasted due to branch mispredictions. These wasted cycles contribute to the high CPI.

3.2.2 Reasons for Poor Prediction

Having seen that the poor performance of the branch predictor might negatively affect the performance of web server software the question is what are the reason for the branch predictor's poor performance.

One reason might be similar to those given to explain similar observations for data base systems: branch predictors perform well on the high-iteration tight loops which are used in scientific applications, but are not optimized for the non-looping branches present in commercial applications.

Another reason could be that many branches might depend on network events that are hard to predict, in particular with the limited knowledge that branch predictors have about the application. However, this does still not explain the difference in performance of the branch prediction between Flash and Apache.

A reason for the difference in the performance of the branch predictor for the Flash and Apache server might be the branch pattern. A branch is hard to predict if it is randomly taken in close to 50% of the cases and not taken in the other cases. If you look at the percent of conditional branches that are taken we see that for Apache around 80% of the branches are taken, while for Flash only 69% of the branches are taken.

Another reason for the difference between Flash and Apache could be that Flash is a research web server that implements only the core functions of a web server. On the other hand Apache executes for every request some security related functions and functions that do error checking. Branch prediction for these function is most likely easy since the branch outcome is usually the same (assuming errors are rare).

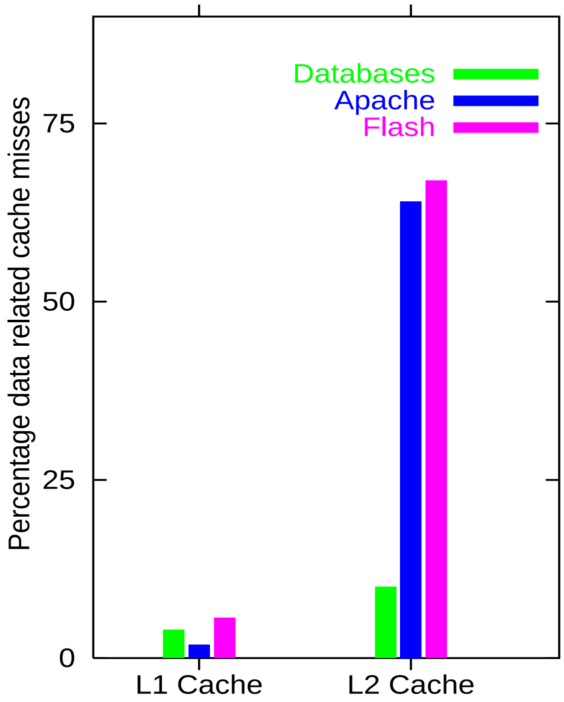

Figure 4: *Data-related* Cache Misses

3.3 Cache Performance

Memory system behavior is one of the most significant components of CPI. For database systems, for example, memory stalls can constitute up to 40% of the total execution time. When the percentage of cache misses is high, the processor eventually runs out of independent instructions to overlap with the memory accesses, and many cycles are spent stalled.

Figure 4 shows the percentage of data related misses in the L1 cache as well as in the L2 cache. Figure 5 shows the percentage of instruction related misses in the two caches.

3.3.1 Data-Related Cache Performance

The L1 data cache miss rates are consistently low for both servers (about 2% for Apache and 6% for Flash). These rates are similar to the miss rates observed in database systems, which tend to lie around 4%. In contrast, the data-related misses in the L2 cache are significantly higher than those observed for other applications: misses account for about 70% of all data-related L2 cache accesses.

One explanation for the low percentage of L1 data cache misses in web servers could be that the server code has internal data structures

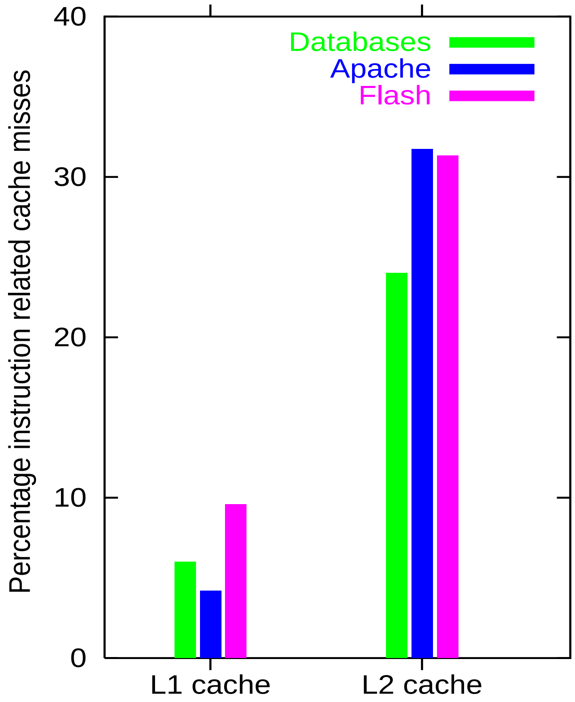

Figure 5: *Instruction-related* Cache Misses

which are small enough to fit in the L1 cache and accessed frequently enough to not be replaced by file data. For example, Flash maintains a table of pathname translations (mapping requested filenames to actual directories) and HTTP response headers (these remain the same when the same file is accessed).

Another explanation for the data related cache performance could lie in the nature of web workloads. As mentioned in Section 2.3 the large majority of requested files are very small (on the order of bytes). On the other hand there is a small percentage of extremely big files (on the order of several MBytes). For example, a page on a commercial web site typically consists of one larger file which contains the page's body and many small files which are icons. The icons (e.g. navigation tools or the company's logo) appear on every page requested from this site. These small icons therefore end up in the web server's L1 cache and are not evicted because they are accessed frequently. On the other hand, the body of a page is different for every page and the corresponding files are therefore accessed less frequently. Also, these files are larger and might not even fit into the L2 cache. Therefore the request for the page body is likely to miss in both caches.

This explains the low miss rate in the L1 cache and the high miss rate in the L2 cache.

3.3.2 Instruction-Related Cache Performance

Figure 5 shows that the L1 instruction cache miss rates lie between 4% and 10% for all applications shown, with Flash exhibiting the highest miss rate at 10%. The percentage of instruction-related accesses that miss in the L2 cache is consistently around 25%.

Although the instruction-related miss rates in the L2 cache are lower than the data-related miss rates, their performance impact might be larger. First, the absolute number of instruction accesses is *twice* the number of data accesses (numbers not shown). Second, because instruction misses cannot be overlapped the penalty incurred by an instruction miss is higher than that of a data miss. For the same reason, even the relatively low L1 instruction cache miss rates could have a significant impact on the performance.

These observations identify another possible contributor to the high CPI discussed in Section 3.1.

multaneously access different parts of the code.

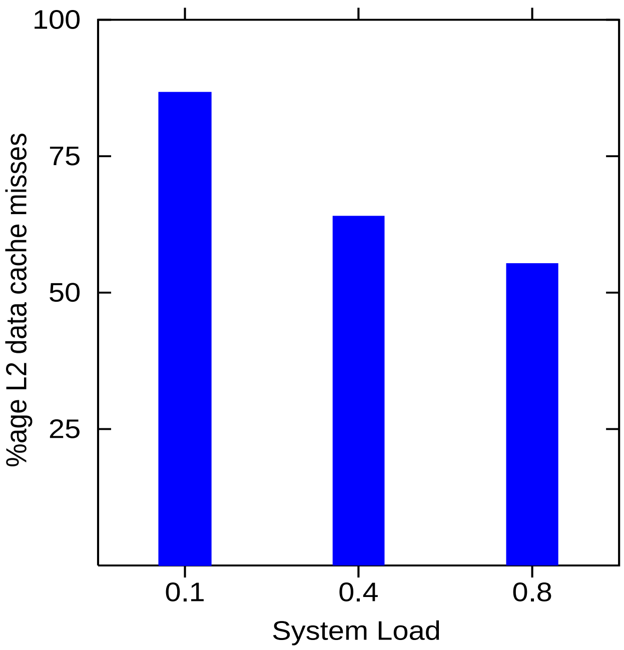

Figure 6: Effect of varying load on L2 data cache misses for Apache

3.3.3 Effect of System Load

Although we have shown results only for a system load of 0.4, we have also repeated all the experiments for system loads of 0.1 and 0.8. Interestingly, the results are quite similar for the three different system loads. The only exception is L2 cache misses for Apache (see Figure 6). Contrary to what we expected, we observe that cache misses decrease as we increase the load.

Another surprising result with respect to varying system loads is that increasing the load does not increase the percentage of instruction cache misses. We expected that the additional processes spawned by Apache would cause more cache misses because many processes might simultaneously access different parts of the code.

4 Conclusions and Future Work

In this study we find that the hardware performance when running a web server is still far from the performance achieved when running standard integer benchmarks. More surprisingly, even compared to database systems, which are known to have suboptimal hardware performance, web servers perform worse with respect to many hardware performance metrics. We also found that even though Flash is considered to be a high-performance web server, it doesn't compare favorably with Apache in terms of CPI and cache misses.

This work is limited to workloads consisting of static requests of the form "GET me a file". Most commercial web servers serve a combination of static and dynamic requests. It would be interesting to look at the effect of dynamic workloads like CGI scripts on the hardware performance. Unfortunately, generating realistic dynamic workloads is difficult since the characteristics of dynamic workloads are less well understood than that of static workloads. A good starting point might be experimenting with the SPECWeb99 workload [3].

Another limitation of our work is, compared to a commercial web server the system load we used in our experiments was relatively low. While the server in our experiments served data at a rate between 1 and 8 Mbps, real web servers run at a rate on the order of tens of Mbps. Given more time we would like to experiment with wider ranges of system loads.

Finally, we compiled both Flash and Apache using the standard gnu-compiler. It would be interesting to use Intel's C compiler that optimizes the executable for the Intel PIII and repeat the experiments with the optimized executables.

References

[1] Netcraft web server survey. Available at URL `http://www.netcraft.com/survey/`.

[2] Papi programmer's reference. Available at URL `http://icl.cs.utk.edu/projects/papi/`.

[3] SPECweb99 benchmark. Available at URL `http://www.specbench.org/osg/web99/`.

[4] A. Ailamaki, D.J. DeWitt, M.D. Hill, and D.A. Wood. DBMSs on a Modern Processor: Where Does Time Go? In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, Edinburgh, UK, September 1999.

[5] Gaurav Banga and Peter Druschel. Measuring the capacity of a web server under realistic loads. *World Wide Web*, 2(1-2):69–83, 1999.

[6] Intel Corporation. IA-32 Intel Architecture Developer's Manual, 2001.

[7] The Apache Group. Apache web server. `http://www.apache.org`.

[8] Internet Town Hall. The internet traffic archives. Available at `http://town.hall.org/A rchives/pub/ITA/`.

[9] T. Heil, Z. Smith, and J. Smith. Improving branch predictors by correlating on data values. In *Proceedings of 32nd Annual International Symposium on Microarchitecture*, pages 25–37, 1999.

[10] John L. Hennessy and David A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[11] Kimberly Keeton, David A. Patterson, Yong Qiang He, Roger C. Raphael, and Walter E. Baker. Performance characterization of a quad pentium pro SMP using OLTP workloads. In *International Symposium on Computer Architecture*, pages 15–26, 1998.

[12] Dennis C. Lee, Patrick Crowley, Jean-Loup Baer, Thomas E. Anderson, and Brian N. Bershad. Execution characteristics of desktop applications on windows NT. In *International Symposium on Computer Architecture*, pages 27–38, 1998.

[13] Ann Marie Maynard, Colette M. Donnelly, and Bret R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994.

[14] Vivek S. Pai, Peter Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of USENIX 1999*, June 1999.

[15] Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod, Emmett Witchel, and Anoop Gupta. The impact of architectural trends on operating system performance. In *Symposium on Operating Systems Principles*, pages 285–298, 1995.

[16] Deependra Talla and Lizy K. John. Execution Characteristics of Multimedia Applications on a Pentium II Processor. In *Proceedings of 19th IEEE International Performance, Computing, and Communications Conference (IPCCC)*, Phoenix, Arizona, Feb 2000.

[17] Cliff Young, Nicholas C. Gloy, and Michael D. Smith. A comparative analysis of schemes for correlated branch prediction. In *Proceedings of 32nd Annual International Symposium on Computer Architecture*, pages 276–286, 1995.