A Multi-Threaded PIPELINED Web Server Architecture for SMP/SoC Machines[*]

Gyu Sang Choi, Jin-Ha Kim, Deniz Ersoz and Chita R. Das
Department of Computer Science and Engineering
Penn State University
University Park, PA 16802
{gchoi, jikim, ersoz, das}@cse.psu.edu

ABSTRACT

Design of high performance Web servers has become a recent research thrust to meet the increasing demand of network-based services. In this paper, we propose a new Web server architecture, called multi-threaded PIPELINED Web server, suitable for Symmetric Multi-Processor (SMP) or System-on-Chip (SoC) architectures. The proposed PIPELINED model consists of multiple thread pools, where each thread pool consists of five basic threads and two helper threads. The main advantages of the proposed model are global information sharing by the threads, minimal synchronization overhead due to less number of threads, and non-blocking I/O operations, possible with the helper threads.

We have conducted an in-depth performance analysis of the proposed server model along with four prior Web server models (Multi-Process (MP), Multi-Thread (MT), Single-Process Event-Driven (SPED) and Asynchronous Multi-Process Event-Driven (AMPED)) via simulation using six Web server workloads. The experiments are conducted to investigate the impact of various factors such as the memory size, disk speed and numbers of clients. The simulation results indicate that the proposed PIPELINED Web server architecture shows the best performance across all system and workload parameters compared to the MP, MT, SPED and AMPED models. Although the MT and AMPED models show competitive performance with less number of processors, the advantage of the PIPELINED model becomes obvious as the number of processors or clients in an SMP/SoC machine increases. The MP model shows the worst performance in most of the cases. The results indicate that the proposed server architecture can be used in future large-scale SMP/SoC machines to boost system performance.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Client/server; C.4 [**Performance Of Systems**]: Design studies; D.2.8 [**Operating Systems**]: Organization and Design

[*]This research was supported in part by NSF grants CCR-9900701, CCR-0098149, EIA-0202007 and CCR-0208734.

General Terms

Algorithms, Design, Performance and Measurement

Keywords

Multi-Process, Multi-Thread, Single Event-Driven Process, Asynchronous Multi-Process Event-Driven, Symmetric Multi-Processor, System-on-Chip

1. INTRODUCTION

Improving the performance of Web servers has become a critical issue to cope with the increasing use of network-based services. The critical nature of many online transactions and distributed service, provided through SOAP, mandates design of high performance Web servers since such servers are anticipated to be the bottleneck in hosting network-based services [20]. Three techniques have been proposed and adopted to improve the performance of a Web server[8]; (i) software scale-up, (ii) hardware scale-up and (iii) cluster-based scale-up.

Several software and hardware scale-up techniques have been proposed to enhance the performance of single node servers. Typically, a software based approach attempts to improve a Web server's cache hit ratio, and thus, minimize the disk access latency in satisfying user requests. It has been observed that by employing a larger data cache [12, 6, 8] and suitable cache replacement techniques [7], server throughput can be significantly improved. On the other hand, a hardware scale-up provides additional computing facility by adding more processor and memory to a single system. Finally, the cluster-based solution is aimed at providing a cost-effective solution by utilizing a cluster of homogeneous or heterogeneous nodes under a single domain name. Commercial servers like Google and e-Bay have used this technique quite effectively [3].

Although cluster-based Web servers seem a viable solution from performance, scalability and economic standpoints, this is certainly not the only choice, and any application specific design should exploit the novelty of the state-of-the-art architectural trend. The motivation of this paper relies on this context, and attempts to see how the server design can benefit from the architectural innovations. Towards this goal, we focus on two types of high performance architectures that can be used in designing Web servers; Symmetric Multi-Processors (SMPs) and System-on-Chip (SoC) architectures.

Recently, a Dual-Core CPU [15] is released by Intel and

AMD to target the high-performance server market. Four and Eight core SMPs are expected to be released soon. In addition, with the advent in deep sub-micron technology, SoC architectures have become a reality, and by the end of the decade, SoCs with billions of transistors are likely to dominate the high performance computing landscape [5, 10, 4]. With technology scaling down to 35nm, it would be possible to fabricate SoCs with up to 32/64 processors. Hence, we expect that many Web servers will be employed on a SoC system to provide high-performance throughputs.

To our knowledge, there is little research on designing SMP/SoC-based Web servers. PalChaudhuri et al. [14] conducted a performance comparison of Web server architectures for SMP systems. They proposed a coordinated AMPED (Co-AMPED) model with multiple AMPED processes. The experiments on a 4-way SMP machine revealed that multiple AMPED servers suffered from scalable performance. This study, in our opinion, provides a limited view since it does not conduct an in-depth analysis of existing Web server models to understand the limitations, so that an efficient server model can be used in the SMP environment. Furthermore, there is no study that examines the performance and scalability implications in the SoC domain.

To understand the performance implications of current server architectures, we first analyze the memory usage of four server architectures; Multi-Process (MP) [18], Multi-Thread (MT) [18], Single-Process Event-Driven (SPED) [21] and Asynchronous Multi-Process Event-Driven (AMPED) [13]. This is done through measuring the memory requirements of Flash (AMPED model) [13] and Apache [18] Web servers on a Sun solaris machine. This analysis is then extended to three other architectures (MP, MT and SPED). The data cache and cache overhead analysis of the four models in an SMP/SoC environment reveals that the MT server model is ideal in providing a large data cache per server to enhance throughput. However, the synchronization overhead of this model can be significant with a large number of threads. Thus, an MT model with small number of threads should provide high throughput in SMP/SoC-based servers.

Based on this rationale, we propose a new Web server architecture, called a multi-threaded PIPELINED Web server, for SMP or SoC systems in this paper. A PIPELINED Web server consists of multiple pipelined thread pools, each of which is composed of 5 basic threads and 2 helper threads. The main advantage of the proposed model is that a thread can share the global information (i.e. data cache, path translation structure, etc.) with other threads. Thus, like the MT model, it needs relatively small memory to maintain the global information. However, unlike the MT model, it can alleviate the synchronization overhead by limiting the total number of threads to $7 \times N$, where N is the number of processors. In addition, by utilizing separate helper threads, the main 5 threads do not block for I/O operations, and thus, it helps in boosting performance.

We have developed a detailed simulator to analyze the performance of four prior server models and the proposed model. We have conducted exhaustive performance analysis by varying several parameters such as the number of processors in the SMP/SoC system, memory size, and number of clients with six Web server traces; Penn State CSE [17], UC Berkeley [11], Penn State [16], Clarknet [2], WorldCup98 [1] and NASA [2] workloads.

The main conclusions of this paper are the following: First, our proposed PIPELINED Web server architecture shows the best performance across various environments and workloads compared to the MP, MT, SPED and AMPED models. The MP model is the worst performance due to available small data cache size per process. Second, the AMPED and SPED Web servers suffer from decreasing data cache size with more number of processors in an SMP/SoC machine due to little sharing of global information. Third, the MT model can provide competitive throughput as the PIPELINED model with smaller system configurations, and less number of clients. However, as the number of processors as well as the number of clients increases, the PIPELINED model becomes a clear winner. All these results indicate that the PIPELINED server model is a viable candidate for deploying server architectures in SMP/SoC machines.

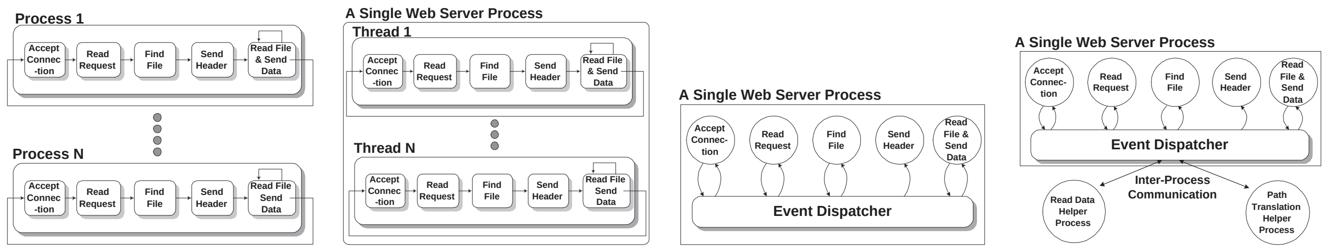
The rest of this paper is organized as follows: In Section 2, we provide a summary of all prior Web server architectures. Section 3 analyzes the memory requirements of Web server models. The multi-threaded PIPELINED Web server architecture is presented in Section 4. Section 5 narrates the simulator design and workload used for performance analysis. The performance results are analyzed in Section 6, followed by the concluding remarks in the last Section.

2. WEB SERVER ARCHITECTURES

Generally, an HTTP server consists of six request processing steps. The first step is the *accept client connection*, which accepts an incoming connection from a client based on the socket operations. Second, the *read request* operation reads and parses an HTTP request from the client's connection. Third, the *find file* operation checks whether the requested file exists in the file system, and the client has appropriate permissions. Fourth, the *send response header* step sends an HTTP response header to the client through a socket connection. Next, the *read file* operation reads the requested data from the file system or from the memory cache. Finally, the *send data* step transmits the requested content to the client. Especially, for larger files, the *read file* and *send data* steps are repeated (shown by the self loop in Figure 1) until all of the requested contents are transmitted [13].

Four HTTP server architectures have been proposed in the literature as shown in Figure 1. The Multi-Process (MP) model, as shown in Figure 1 (a), has a process pool and each process is assigned to execute the basic steps associated with servicing a request. Since multiple processes are employed, many HTTP requests can be served concurrently. However, the disadvantage of this model is the difficulty to share any global information (e.g.: shared cache information) among the processes, since each process has its own private address. An MP-based Web server needs more memory to maintain the same cache size per process compared to other server models. Thus, the overall performance of this model is expected to be lower than that of other models [12, 6].

The Multi-Thread (MT) model, in the other hand, consists of multiple kernel threads with a single shared address space. In Figure 1 (b), each thread takes care of a client's request and performs the request processing steps independently. The advantage of this model is that the threads can share any global information. Especially, the data cache is shared among all threads. However, not all Operating Systems (OSs) support kernel threads, and sharing the data cache information among many threads may lead to high



(a) Multi-Process (MP) Model

(b) Multi-Thread (MT) Model

(c) Single-Process Event-Driven (SPED) Model

(d) Asynchronous Multi-Process Event-Driven (AMPED) Model

Figure 1: Web Server Architectures

synchronization overhead. The widely used Apache Web server was originally designed as an MP model. Later, it is enhanced to support both MP and MT models, since the MT model tends to yield better performance than the MP model [18].

Next, Figure 1 (c) shows a Single-Process Event-Driven (SPED) Web server architecture that uses non-blocking I/O operations. SPED can avoid context-switching and synchronization overheads among threads or processes, because it is a single Web server process. This model is implemented by the Zeus Technology [21]. However, the non-blocking I/O operations in this model are actually blocked [13] when it performs disk-related operations due to the limitations of current OSs. This is the reason why SPED doesn't show better results than the MT model for disk-bound workload [13].

The last architecture is the Asynchronous Multi-Process Event-Driven (AMPED) model [13], which has been proposed to alleviate the weakness of the SPED model. Figure 1 (d) depicts the AMPED server architecture, which consists of one main Web server process and multiple helper processes to mainly handle I/O operations. As this model has multiple helper processes to serve disk-oriented requests, the main Web server process only serves cache-hit requests. If there is a cache miss, the main process forwards the request to a helper process, and then, the helper process fetches the data from the disk and sends it back to the main process by Inter-Process Communication (IPC). Especially, using the mmap operation in the AMPED model, additional data copying between a Web server and the helper processes can be removed.

All these server models were originally proposed for single CPU systems. The scalability of these server architectures has not been examined for SMP/SoC systems.

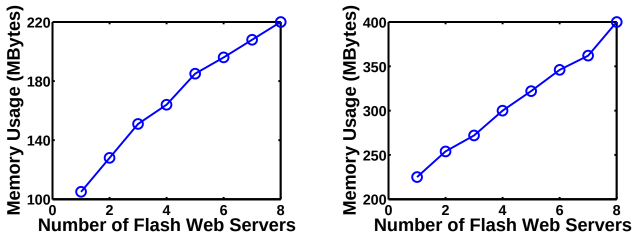
3. MEMORY ANALYSIS

In this section, we analyze the memory requirements for the four Web server models on an SMP or SoC machine.

We define three terms to conduct the memory analysis. First, we use *system_memory* as the maximum available main memory for a Web server, although normally system memory is referred as the main memory. For example, if the size of main memory is 100MBytes and an OS uses 10MBytes, then the *system_memory* is 90MBytes. Next, the available memory space for caching the Web contents is called the *data_cache*. Finally, we refer to all additional memory spaces as *cache_overhead* which is equal to *system_memory* - *data_cache*. This includes memory overhead for the Web server to maintain the *data_cache* (e.g. name, size, and path translation of cached contents).

3.1 Memory Usage in Flash Web servers

First, to show the scalability problem of the AMPED model, we measure the total *memory usage* of a Flash Web server by varying the number of servers in a single node. The total *memory usage* (i.e. *system_memory*) is the sum of the *cache_overhead* and *data_cache* in a Web server. The *cache_overhead* of a Flash Web server consists of several terms. First, the major component of the *cache_overhead* is the space required for maintaining the information of cached files, and it is 850Bytes per file. If we assume that the average Web file size is 15KBytes [13] and the *data_cache* is 100MBytes, the maximum number of the cached web files is approximately 6800. Then, the *cache_overhead* to maintain 100MBytes of *data_cache* is approximately 5MBytes. Second, since the maximum number of path translation entries in a Flash Web server is 6000 and the average size of a path translation entry is about 125Bytes, path translation consumes around 0.7MBytes. Third, two helper processes in a Flash Web server, *read* and *path translation helpers*, consume additional 3MBytes [13]. Thus, besides the *data_cache*, a Flash Web server needs an additional 8.7MBytes to maintain the 100MBytes *data_cache* in the main memory. While this *cache_overhead* seems small for a single AMPED server process, it increases linearly with the number of servers.



(a) 100MBytes Cache Size

(b) 200MBytes Cache Size

Figure 2: Memory Usage of the Flash Web servers (AMPED Model)

Figure 2 shows the *memory usage* as a function of the number of AMPED-based Flash Web servers in a single node. To examine the *cache_overhead*, we fix the *data_cache* size to 100MBytes and 200MBytes and increase the number of servers from 1 to 8 in a node. We measured the *memory usage* using the system monitoring tool in a single CPU Sun Solaris machine. In Figure 2 (a), for a single Web server, the *cache_overhead* is only around 5MBytes for 100MBytes of *data_cache*. However, when the number of Web servers is eight, the *cache_overhead* becomes 120MBytes, which is even larger than the *data_cache* size. Figure 2 (b) shows that when the number of the Web servers is eight, the *cache_overhead* becomes almost 200MBytes with 200MBytes of *data_cache*. This high overhead is attributed to the fact that processes cannot share the global information. The reason why the *cache_overhead* in Figure 2 (b) is larger than in Figure 2 (a) is that a Web server has more cached files in the *data_cache* due to a larger *data_cache* size.

This experiment clearly shows that the *cache_overhead* in Flash Web servers is a major bottleneck.

3.2 Memory Usage in Other Web Server Architectures

Based on the previous memory usage results of the Flash Web server, we analyze the *cache_overhead* in other Web server architectures.

In an MP model, each node has typically 16 to 32 processes, and each process should have its own *data_cache*. Thus, the *data_cache* size per process reduces accordingly. In addition, each server process needs space for the *cache_overhead* to maintain the *data_cache*.

Since threads in an MT model can share the global cache information, the memory requirement should not change significantly with an increase in the number of threads. To verify this, we increased the number of threads in an Apache Web server [18] from 25 to 50, and measured the memory usage. Each thread consumed about 25KBytes in the MT model configuration. In addition, we ran a simple multi-threaded barrier program and varied the number of threads from 10 to 1000. We measured the *memory usage* as we increased the number of threads. We observed that the additional memory requirement is only 10KBytes per thread.

However, the problem with the MT model is the synchronization overhead among threads. When the number of threads increases, the synchronization overhead becomes non-negligible. This overhead can be a major bottleneck in a large-scale SMP or SoC node. Since usually the number of threads per CPU is 32 or 64 and these types of machines can have 16 to 64 Processing Elements (PEs), the total number of threads, T, can be in hundreds or thousands. These threads compete to access the cached contents and path translation information, which needs to be synchronized.

Since the SPED Web server architecture is basically similar to the AMPED model, both the models have similar *cache_overhead*. The main difference between the two models is the helper processes in the AMPED model. Due to the memory requirements of the two helper processes, the AMPED model needs additional 3MBytes compared to the SPED model.

3.3 cache_overhead in Web Server Architectures

Now, we calculate the *cache_overhead* of prior four server models in an SMP or SoC system. Here, we assume that all Web servers have the same cache structure.

Assuming 16 or 32 server processes per CPU in an MP model, the total number of the processes, P, could be very high in an SMP or SoC system. In the SPED and AMPED models, one Web server process usually runs in one node, while Zeus [21] recommends to launch two server processes for better performance. In an SMP or SoC machine, we may launch one Web server process per CPU. In the MT-based model, there is only one Web server process, no matter how many threads are running.

Based on this, we compute the *cache_overhead* to maintain the cache information in each server architecture in Table 1, when *system_memory* size is known. Since we assume that the size of an average Web file is 15KBytes [9], the number of files that can be cached is $file\_entries = data\_cache\_size/15KBytes$. The memory requirement for keeping the information of these entries is $file\_entries \times 850Bytes$, since the average size to maintain each cache element is about 850Bytes in a Flash Web server. The total *cache_overhead* of an MP-based Web server is $file\_entries \times 850 \times N \times P$, where N is the number of processing elements, and P is the number of Web server processes per CPU. We ignore the memory space for the path translation, because this value is relatively small.

In an MT-based Web server, the total memory usage of the server model does not change significantly. The *cache_overhead* can be calculated as $file\_entries \times 850Bytes + T \times 25KBytes$, since one thread in the Apache server consumes 25KBytes. In the SPED-based Web server, the *cache_overhead* is $file\_entries \times 850Bytes \times N$, if we assume that we launch a single Web server per CPU. In an AMPED-based Web server, since read and name translation helpers consume about 3MBytes per server, the total *cache_overhead* becomes $file\_entries \times 850Bytes \times N + 3MBytes \times N$.

3.4 data_cache in Web Server Architectures

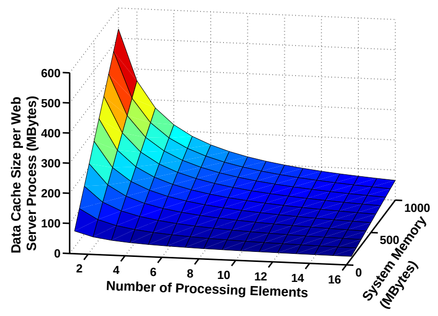
In this subsection, we calculate the available *data_cache* of four server architectures, where the available *data_cache* for the Web contents is $system\_memory - cache\_overhead$. Since the available *data_cache* size affects the performance of a Web server, we can predict a server's performance based on the calculated *data_cache* values. In this experiment, we examine the relationship between *data_cache* size, number of PEs (or CPUs) and *system_memory*.

Figure 3 shows the variation of available *data_cache* as a function of PEs and the *system_memory*. Figure 3 (a) depicts that the *data_cache* size in an MP model reduces drastically when the number of PEs increases. Because of this reduced *data_cache* space, the cache hit ratio of an MP-based server would be low, which in turn would affect the latency and throughput due to frequent disk accesses.

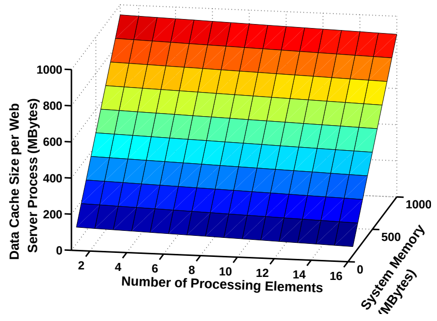
In Figure 3 (b), the *data_cache* for the MT-based Web server is almost constant because there is only one Web server process, and a thread can share cache information with other threads. However, as we will see later, the performance of the MT model might suffer from high synchronization overhead as the number of threads increases. Figures 3 (c) and (d) show that the *data_caches* of the SPED and AMPED servers reduce when the number of PEs increases. It is attributed to the non-sharing nature of the global information. An interesting observation is that the available *data_cache* size in the AMPED model shrinks more compared to the SPED model, because the helper processes in the AMPED model consume more additional memory.

Web Server Architecture	*cache_overhead* (Bytes)
Multi-Process Model	$file\_entries \times 850 \times N \times P$
Multi-Thread Model	$file\_entries \times 850 + T \times 25K$
Single-Process Event-Driven Model	$file\_entries \times 850 \times N$
Asynchronous Multi-Process Event-Driven Model	$file\_entries \times 850 \times N + 3M \times N$

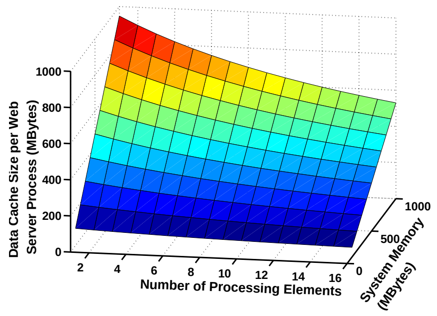
Table 1: *cache_overhead* **in Web Server Architectures, where** $file\_entries$ **is the number of cached files,** N **is the number of processing elements,** P **is the number of Web server processes per processing element, and** T **is the total number of threads**



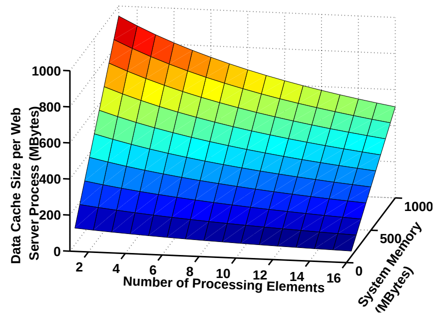
(a) Multi-Process Model



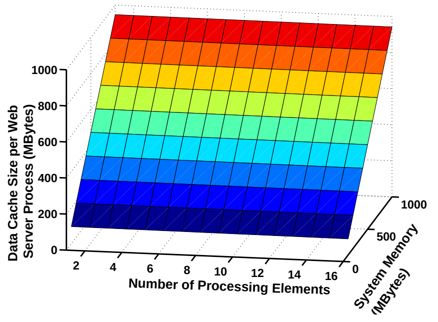
(b) Multi-Thread Model



(c) Single-Process Event-Driven (SPED) model



(d) Asynchronous Multi-Process Event-Driven (AMPED) Model



(e) Multi-Threaded PIPELINED model

Figure 3: *data_cache* **Size in Web Server Architectures**

4. A MULTI-THREADED PIPELINED WEB SERVER ARCHITECTURE

In this section, we propose a new Web server architecture, called *multi-threaded PIPELINED Web server*, which takes advantage of the MT model, but mitigates the synchronization overhead by limiting the number of threads. Figure 4 depicts the architecture of the multi-threaded PIPELINED HTTP server. In the proposed server, each basic operation of an HTTP request is mapped on to a thread. First, an *accept connection* thread takes care of a connection from a user, and forwards this request to a *read request* thread. The *read request* thread parses the request, and then, the *find file request* thread checks whether the request is in a path translation cache. If it is located in the path translation cache, the *find file request* thread gives it to a *send header* thread. Otherwise, it forwards the request to a *path translation helper* thread. Next, the *send header* thread sends the response header to the client. Finally, the *read file* and *send data* threads send the requested data to the client. If the file size is large, these threads repeat (shown by the loop in Figure 4) the process, until the file is sent. Due to its similarity to the PIPELINED model in computer architecture, we call this model a *PIPELINED* Web server.

We refer to these five basic processing threads and two helper threads as a *pipelined thread pool*. A PIPELINED Web server can have multiple pipelined thread pools. Two helper threads are dedicated to handle the I/O operations. One is the *path translation helper* thread, which converts a request's URL to a system path to the requested file. The other is a *read helper* thread, which reads the requested file from a disk, when a cache miss occurs. Whenever an I/O operation is required, the thread passes the request to one of the helper threads, and it handles the next request immediately. Whenever a helper thread completes an I/O operation, it sends back the response to the proper thread.

The most important characteristic of this model is that there is only one Web server process, even though there are multiple pipelined thread pools. While there will be one pipelined thread pool in a single CPU, a pipelined thread pool can be launched on each CPU in an SMP or SoC system and the threads can share the global information. Since the *data_cache* size in a single process model (i.e. the MT model) is larger than that in other models in a multi-CPU environment, the proposed model needs relatively small memory to maintain the global information compared to the SPED and AMPED models.

The main reason why the proposed model is suitable for an SMP/SoC environemnt is because of almost constant *data_cache* size (Figure 3 (e)). There might be multiple processes of the SPED or AMPED model in an SMP/SoC system. These multiple Web processes should have their own private cache, and thus, it significantly reduces the *data_cache* size as shown in Figures 3 (c) and (d).

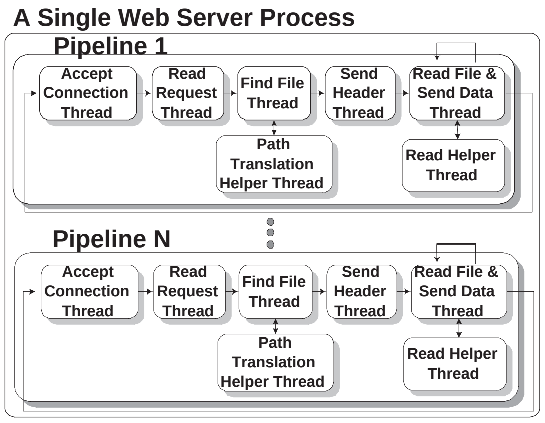


Figure 4: Architecture of the Multi-Threaded PIPELINED Web Server

Moreover, the PIPELINED model can alleviate the synchronization overhead, one of the drawbacks of the MT model. The total number of threads in the proposed model is $7 \times N$, while it will be 64 (or 32) $\times N$ in the MT model, where N is the number of processors. In addition, the threads in the MT model are likely to compete with each other to access the shared information (i.e. *data_cache* and path translation cache). However in a pipelined thread pool, since each thread plays a different role, the contention can be mitigated.

Unlike the MT model, we also adapt helper threads to prevent a thread being blocked due to an I/O operation. Without the helper threads, a pipelined thread pool cannot proceed before completing the blocking I/O operation. Thus, when a cache miss occurs, the *read file* thread forwards this request to a *read* helper thread. The helper thread sends back the static contents to the thread after reading it from a disk.

Next, we analyze the *cache_overhead* and *data_cache* sizes of a multi-threaded PIPELINED server. The *cache_overhead* $= file\_entries \times 850 Bytes + 7 \times N \times 25 KBytes$. Figure 3 (e) shows the *data_cache* size variation of this architecture when the number of PEs (N) and system memory are varied. The memory requirement of this model is marginally smaller than that of the MT model, since the number of threads in the proposed model is around 10 times less than that of the MT model and these two models have only a single Web server process. Thus, due to large *data_cache* size, we expect better performance in the proposed model than other models.

5. A SIMULATION TESTBED

In this section, we present a simulator for analyzing the five Web server models and summarize the six traces used in performance evaluation.

5.1 Simulator

We have developed a simulator platform, which can model the MP, MT, SPED, AMPED and PIPELINED server architectures. The simulator receives the number of PEs and the system memory size as input parameters. The client module reads a request from a trace file, and sends the request to the Web server. The main part of the simulator is the Web server module. Whenever a request arrives, the Web server module processes the HTTP basic operations. In our experiment, the client sends a next request as soon as the previous request is completed. Since disk latency is a critical factor in quantifying server performance, we modeled two hard drives from the Western Digital Technologies [19] and the hard drive specifications are depicted in Table 2. Throughput (number of completed requests per second) is the objective function analyzed in this paper.

	Hard Drives	
Model	Fast	Slow
Capacity (GBytes)	36.7	80
Average Latency (ms)	2.99	5.35
Rotational Speed (RPM)	10,000	5,400
Data Transfer Rate (MBits/s)	1200	594

Table 2: Performance Parameters of Two Hard Drives

We extracted performance-related parameters from a Flash Web server through measurements, and summarizes them in Table 3. *Average Entry Size* for the cache in Table 3 represents the memory required for maintaining the information of a cached file, and *Average Access Latency* for the cache is the required time to check whether the requested data is in the *data_cache*. *Average Entry Size* for path translation is memory required to map the requested file to an actual path on a disk, *Number of Entries* is the maximum number of entries in path translation cache, and *Average Access Latency* for the path translation cache is the time to look up the directory of the cached files and map from the requested file names to actual files on a disk. *Average Memory Usage* is memory consumption per helper process in the AMPED model [13] and *Average Memory Consumption* is memory required per thread in MT and PIPELINED models [18]. *Average Connection Overhead* is the overhead for a TCP connection between a client and a server, and *Average Transmit Latency* is the network latency to transmit data. Finally, *Context Switching Overhead* is the time to switch between two kernel threads in MT and PIPELINEd models.

5.2 Workload

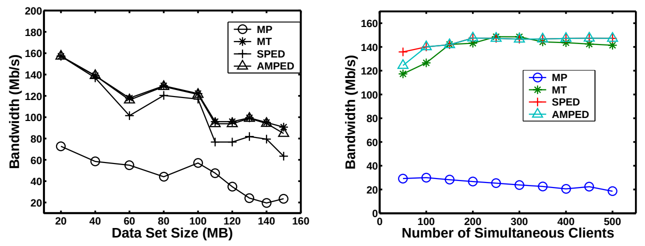
We use six trace-based workloads with different characteristics for performance analysis;, Penn State CSE [17], UC Berkeley[11], Penn State (PSU) [16], Clarknet [2], NASA [2] and WorldCup98 [1]. Table 4 shows the detail characteristics of the six traces including the number of files, average file size, number of requests and data set size. The main trace used in the experiments is Penn State CSE [17]. The average file size of Penn State CSE trace is 124.3KBytes, which is much larger than the average file size of other workloads.

Logs	Number of Files	Average File Size	Number of requests	Data Set Size
Penn State CSE	48079	124.3KBytes	1395148	5.7GBytes
UC Berkeley	511189	9.84KBytes	1383211	4.8GBytes
Penn State	139894	15.7KBytes	8853333	2.1GBytes
Clarknet	28864	14.2KBytes	2978121	320MBytes
NASA	9129	27.6KBytes	3147684	208MBytes
WorldCup98	13236	15.3KBytes	7000000	158MBytes

Table 4: Main Characteristics of the WWW Server Traces

Cache	Average Entry Size	850Bytes
	Average Access Latency	13.5μs
Path Translation Cache	Average Entry Size	125Bytes
	Number of Entries	6000
	Average Access Latency	6.16μs
Helper	Average Memory Usage	1.5MBytes
Network	Average Connection Overhead	129μs
	Average Transmit Latency per 512Bytes	24μs
Thread	Average Memory Consumption	25KBytes
	Context Switching Overhead	5μs

Table 3: Measured Parameters from a Flash Web server



(a) Data Set Size (MBytes) (b) Number of Clients

Figure 5: Simulator Validation

The reason is that it has many course material and multimedia files. The total data set size is about 5.7GBytes. We chose this as the main trace because it has a large data set to fit into an SMP or SoC system. The UC Berkeley trace [11] has the unique characteristics of very low spatial locality and the smallest average file size in all workloads. The characteristics of the other traces are listed in Table 4.

5.3 Simulator Validation

We have validated our simulator using a real trace, by comparing to the results of the Flash Web server, reported in [13]. While the authors in [13] used two workloads, they conducted most of results in their experiments by the trace of Rice University Computer Science. However, we decide to use the Clarknet [2] workload because we couldn't get this workload. Since both these workloads have different locality, the results are likely to be different. The Rice University CS trace [13] has a very high locality. It means that an MP model can have comparable throughput to those as MT, SPED and AMPED server models. In contrast, the Clarknet has less locality, compared to the CS trace in [13]. It implies that an MP model will show less throughput compared to other models, when the *data_cache* size is smaller than data set size.

We choose two experiments in [13], to verify our simulator. In the first validation experiment, the data set size is varied from 20MBytes to 150MBytes, and we set the number of clients to 64. These configurations are the same as the Flash Web server study [13]. In [13], the system memory size was 128MBytes, but the actual *data_cache* size might be around 100MBytes, since the bandwidth significantly dropped after the 100MBytes data set size and the Solaris OS itself consumes around 30MBytes. Hence, we set the *data_cache* size to 100Mbytes in this experiment. We use a slow disk model, since the results in [13] were measured in 1999. We measured bandwidth (Mb/s) as the performance metric, which was used in [13]. As Figure 10 in [13] showed that the bandwidth dropped after the data set size is 100MBytes, our result in Figure 5 (a) also shows the similar trend. The interesting point is that the MP bandwidth in our result is different from that in [13]. This is because the Clarknet workload has less spatial locality than the Rice University CS trace.

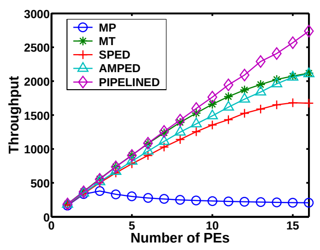
In the second validation experiment, we varied the number of clients, while the data set size is fixed at 90MBytes, and the other configurations are the same as the first validation. In Figure 5 (b), the results exhibit the similar trend, as reported in [13], while the bandwidth of the MP model drops slowly after 200 clients. As we expected, the MP model has less bandwidth/throughput than the results in [13] due to low spatial locality of the Clarknet trace.

6. PERFORMANCE ANALYSIS

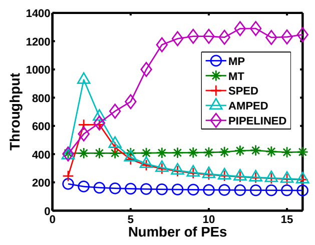
In this section, we present the performance comparison of the five Web server architectures under a variety of workload and system parameters. The performance analysis is done in detail through a series of experiments. In all experiments, we use a warm-up period of 5000 seconds to avoid the transient phase.

6.1 Impact of Workload

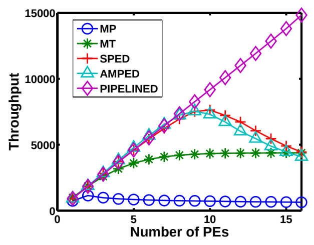
First, we show the performance results with various workloads. We use six traces, listed in Table 4. Since the Penn State CSE and UC Berkeley workloads have very large data set size, we use 4GBytes as the *system_memory* for the Penn State CSE [17] and 3.5GBytes for UC Berkeley workload. We set 1.5GBytes as *system_memory* for the Penn State trace [16], 220MBytes for the Clarknet trace, 110MBytes for the WorldCup98 trace, and 150MBytes for the NASA workload [2]. We set the *system_memory* to 70 $\sim$ 75% of the total data set size to compare each workload's characteristics, while the number of clients is fixed at 1000. We used a simple program to measure the locality of the traces, by fixing the



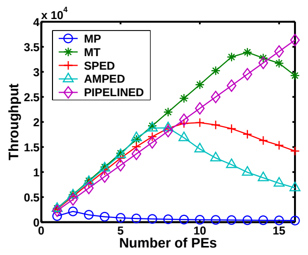
(a) Penn State CSE
4GBytes *system_memory*



(b) UC Berkeley
3.5GBytes *system_memory*



(c) Penn State
1.5GBytes *system_memory*



(d) WorldCup98
110MBytes *system_memory*

Figure 6: Performance Comparison among Web Server Architectures with Several Workloads

date set size and *data_cache* size. From our measurements, Table 5 presents the locality when the *data_cache* size is given. For example, 4GBytes *data_cache* in Penn State CSE trace has 99.6% locality, while 3.5GBytes *data_cache* in UC Berkeley trace shows only 88.3%.

Wrokload	*data_cache* Size (Bytes)	Locality (%)
Penn State CSE	4G	99.6
UCB	3.5G	88.3
Penn State	1.5G	99.47
Clarknet	220M	99.5
WorldCup98	110M	99.9
NASA	150M	99.82

Table 5: Spatial Locality of the Workloads

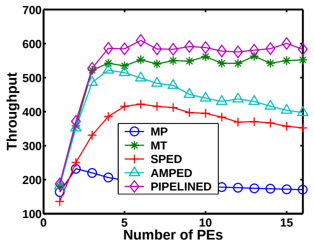
Figure 6 shows that the proposed PIPELINED Web server outperformed all prior Web servers except with the WorldCup98 trace, when we vary the number of CPUs/Processing Elements (PEs) on an SMP/SoC machine. With the WorldCup98 trace, the MT server shows competitive performance as the PIPELINED server. On the other hand, the MP model shows the worst throughput across all traces. Since the MP model requires more space for the *cache_overhead* and this overhead increases as the number of the PE increases, it suffers from high cache miss ratio, and the disk becomes the performance bottleneck.

In Figure 6 (a), the PIPELINED model shows up to 30% improvement in throughput compared to the MT, SPED and AMPED models. The performance difference between the PIPELINED model and the other models increases with the number of processors. This is because of the higher cache hit ratio of the PIPELINED Web server. The MT model has high synchronization overhead and this overhead increases as the number of threads increases. The AMPED and SPED models have more *cache_overhead* than the PIPELINED and MT models because they share the global information. The AMPED model needs 8 times more *cache_overhead* than the PIPELINED model for 16 processors. Thus, the available *data_cache* in the AMPED and SPED models decreases as the number of PEs increases. In addition, since the SPED model cannot handle the disk access efficiently, it yields poorer performance than the AMPED model.

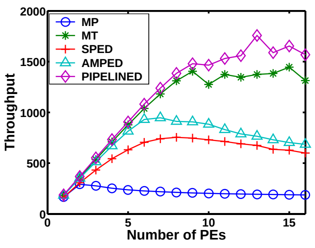
Figure 6 (b) shows the results of the UC Berkeley workload. As shown in Table 4, it has relatively low spatial locality compared to other traces. While the average file size is the smallest among the traces, its data set size is very large and the number of the requests is the highest. Thus, it needs a large *data_cache* to prevent frequent disk accesses, and the performance also depends on how a Web server can handle small contents efficiently. In Figure 6 (b), the PIPELINED model noticeably outperformed all other models. The reason that the AMPED and SPED models do not show good performance is that the smaller *data_cache* hurts performance more than the other trace files due to the low spatial locality ratio of UCB trace. However, with up to 3 PEs, the AMPED model shows the best performance because it can efficiently handle the disk-bound request [13], and can avoid the context switch overhead compared to the PIPELINED and MT models.

Although the MT model guarantees a large *data_cache*, each thread handles many requests. Since the size of the contents in the trace is small, it results in high synchronization overhead. In addition, during disk accesses, the threads in the MT model are blocked. Therefore, the MT model does not have the benefit of increasing concurrency like with other trace files. The PIPELINED Web server, unlike the other models, can handle these requests very efficiently due to helper threads, and thus, is a very good candidate when the workload shows very low spatial locality.

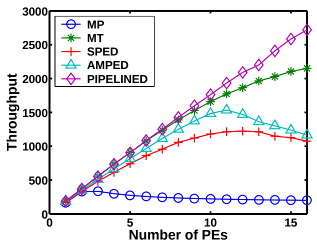
With the Penn State trace, the PIPELINED model is the best performer in Figure 6 (c) when number of PEs is greater than 8, while the SPED and AMPED models have comparable performance with the PIPELINED model with up to 8 PEs. As the number of PEs increases, total *cache_overhead* size in SPED and AMPED models increases but *data_cache* size per process in these models decreases moderately, because processes can't share global information. Thus, throughput in these models significantly drops beyond 9 PEs. The MT model shows lower throughput than the PIPELINED model. This is because the synchronization overhead among threads increases due to high cache hit ratio. We skip the result of the Clarknet trace due to its similarity with the Penn State trace. The NASA and WorldCup98 traces have the smallest data set size and *system_memory*. As the number of processors increases, the reduction of the *data_cache* size can affect the throughput more than other traces, due to the small *system_memory* size. The characteristics of the WorldCup98 trace are that it has not only the smallest data set among all workloads, but also has the highest spatial locality. Thus, the throughput in Figure 6 (d) is the highest among all workloads. Since the MT model does not suffer from high disk accesses due to the smallest data set size, it shows comparable performance to the PIPELINED model. We omit the result of NASA trace, since it shows similar trend as the WorldCup98 trace.



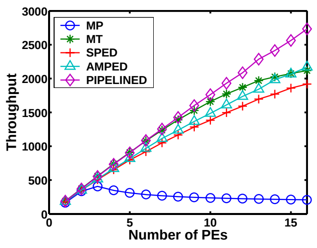
(a) 1GBytes *system_memory*



(b) 2GBytes*system_memory*



(c) 3GBytes *system_memory*



(d) 5GBytes*system_memory*

Figure 7: Performance Comparison of Web Server Architectures by Varying *system_memory* with the Penn State CSE trace

6.2 Impact of Data Cache Size

In this experiment, we examine the impact of *data_cache* by varying the *system_memory*, when the number of clients is fixed at 1000. The workload used for this experiment is the Penn State CSE. Figure 7 (a) depicts the throughput results of the five Web server architectures when the *system_memory* is 1GBytes, while the data set size of the trace is about 5.7GBytes. In this experiment, the Web servers incur large number of cache misses due to the small cache size. Thus, the throughputs of all Web servers are bounded by the disk speed. In other words, the throughput is not scalable as the number of PEs increases. The PIPELINED model shows the best throughput among these five models, while the performance of the MP model is the worst, as expected. The throughputs of the AMPED and SPED models increase until the number of PEs is 5, and decrease after that. An increase in the number of PEs reduces the *data_cache*, consequently causing more disk accesses. More PEs in the MT and PIPELINED models doesn't reduce the *data_cache* significantly, and thus, they yield stable throughputs.

The throughput results with 2 GBytes and 3 GBytes as *system_memory* size are plotted in Figures 7 (b) and 7 (c), respectively. The performance difference among the PIPELINED, AMPED and SPED models is more evident as the *system_memory* increases. Since the cache miss rate is decreased, the throughputs of all servers improved drastically. However, the disk is still the bottleneck in the SPED and AMPED model, when the number of PEs is greater than 12 in Figure 7 (c). Results with 4GBytes *system_memory* are depicted in Figure 6 (a). In Figure 7 (d), all servers except the MP model benefit from the large cache, because the disk operation is not the bottleneck any more. The PIPELINED model shows 32% throughput improvement compared to the MT model. The results of this section indicate that the cache size has a great impact on the throughput for all Web server architectures. We observed similar results with other workloads, and the proposed Web server had better throughput across all cases of *system_memory*.

6.3 Number of Clients

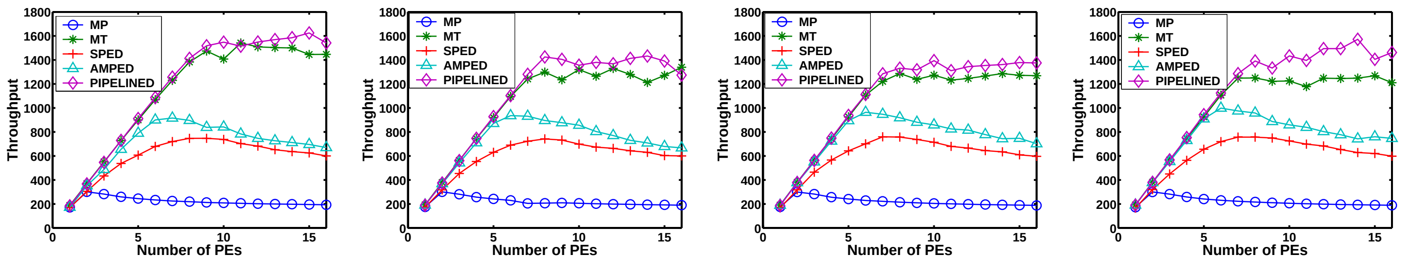
In this experiment, we vary the number of clients to examine the scalability issue in terms of the number of concurrent connections. We chose the Penn State CSE workload, and the *system_memory* size is fixed to 2GBytes. Other workload results are omitted due to space limitation. In Figure 8 (a), when the number of clients is 500, the PIPELINED and MT models are the best performers, while the PIPELINED model shows slightly better throughput than the MT model (up to 12%). Figure 8 (b) depicts the throughputs with 1500 clients, where the performance difference between the PIPELINED and MT models becomes more pronounced. The throughput difference gap is about 18% and 26%, in Figures 8 (c) and (d), respectively.

The results indicate that when the number of clients increases, the PIPELINED model outperforms the MT model. This is because the MT model needs many active threads to satisfy the requests. The synchronization overhead of the threads in the MT model with 2500 clients increased by 20% (measured as queuing time at the synchronization point) compared to with 500 clients. In addition, the memory consumption of these threads increases, and thus, available *data_cache* size of a server reduces. While the memory consumption of the MT model is only 25KBytes per thread, it can affect the system performance, when the number of threads becomes high. The MP model in all these cases exhibits the worst performance.

7. CONCLUSIONS

Design of high performance Web servers is essential for providing adequate support to the increasing demand of Internet-based services. Although several server models have been proposed towards this goal, very little attention has been paid in exploring the server design space with SMP and SoC architectures. In this paper, we have investigated the design of a multi-threaded PIPELINED architecture, suitable for SMP/SoC machines.

To understand the performance implications of current server models, we measured the memory overhead of a Flash Web server in a Sun Solaris machine, and based on this measured data analyzed the data cache and cache overhead of the prior MP, MT, SPED and AMPED models. The analysis revealed that a multi-threaded server model with small number of threads is ideal for providing high performance. The proposed multi-threaded PIPELINED Web server consists of multiple thread pools, where each thread pool is composed of 5 basic threads and 2 helper threads. The main advantage of the proposed model is that the threads can share the global information such as data cache and path translation structure. Thus, like the MT model, it needs relatively small memory to maintain the global information. However, unlike the MT model, it can alleviate the synchronization overhead by limiting the total number of threads to $7 \times N$, where N is the number of processors in an SMP/SoC machine. In addition, by utilizing separate helper threads, the main threads do not block for I/O operations, thereby helping in improving performance.



(a) 500 Clients (b) 1500 Clients (c) 2000 Clients (d) 2500 Clients

Figure 8: Performance Comparison of Web server models by Varying the Number of Clients with the Penn State CSE trace

A simulation-based performance analysis of the prior four server models and the proposed PIPELINED model with six Web server traces shows that the proposed server architecture can deliver the best performance across various system configurations and workloads. The MT and AMPED models are close competitors with the PIPELINED model for several system configurations, specifically when the number of processors is small. However, the proposed model outperformed the MT and AMPED designs as the number of clients or the number of processors increased. Furthermore, while the MT, SPED and AMPED models suffered due to low application locality and inadequate system memory, the proposed model exhibited good throughput across all experimental conditions. The MP model, as expected, is the worst performer.

These results indicate that the PIPELINED server architecture is a viable design option for SMP/SoC machines. We plan to implement the proposed model in an SMP machine. In addition, we will analyze the performance impact of dynamic Web contents on the proposed model.

8. REFERENCES

[1] M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup Web Site, February 1999. http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html.

[2] M. F. Arlitt and C. L. Williamson. Web Server Workload Characterization: The Search for Invariants. *ACM SIGMETRICS Performance Evaluation Review*, 24(1):126–137, 1996.

[3] L. A. Barroso, J. Dean, and U. Hlzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23(02):22–28, 2003.

[4] G. Bell and J. Gray. What's Next in High-Performance Computing? *Communications of the ACM*, 45(2):91–95, 2002.

[5] L. Benini and G. D. Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 35(1):70–78, 2002.

[6] A. Bestavros, R. L. Carter, M. E. Crovella, C. R. Cunha, A. Heddaya, and S. A. Mirdad. Application-level Document Caching in the Internet. In *Proceedings of the 2nd International Workshop on Services in Distributed and Networked Environments*, page 166, 1995.

[7] P. Cao and S. Irani. Cost-Aware WWW Proxy Caching Algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*.

[8] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys (CSUR)*, 34(2):263–311, 2002.

[9] E. V. Carrera, S. Rao, L. Iftode, and R. Bianchini. User-Level Communication in Cluster-Based Servers. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02)*, pages 248–259, 2002.

[10] D. Edenfeld, A. B. Kahng, M. Rodgers, and Y. Zorian. 2003 Technology Roadmap for Semiconductors. *IEEE Computer*, 37(1):47–56, 2004.

[11] S. D. Gribble. UC Berkeley Home IP HTTP Traces, July 1997. http://www.acm.org/sigcomm/ITA/.

[12] E. P. Markatos. Main Memory Caching of Web Documents. In *Proceedings of the fifth international World Wide Web conference on Computer networks and ISDN systems*, pages 893–905, 1996.

[13] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An Efficient and Portable Web Server. In *Proceedings of the USENIX 99 Annual Technical Conference*, June 1999.

[14] S. PalChaudhuri, R. Kumar, and A. K. Saha. A Web Server Architecture for Symmetric Multiprocessor System. Project Report for Comp520, Department of Computer Science, Rice University, December 2000.

[15] PC Magazine. Intel Serves up Double Chips, Sep. 2004. http://www.pcmag.co.uk/news/1158033.

[16] Pennsylvania State University. http://www.psu.edu, 2004.

[17] Pennsylvania State University Computer Science & Engineering. http://www.cse.psu.edu, 2004.

[18] The Apache Software Foundation. The Apache HTTP Server Project, 2003. http://httpd.apache.org.

[19] Western Digital Corporation. Enterprise Hard Drives, 2004. http://www.westerndigital.com/en/products/.

[20] T. Wilson. E-Biz Bucks Lost Under SSL Strain, May 1999. Available from `http://www.internetwk.com/lead/lead052099.htm`.

[21] Zeus Technology Limited. Zeus Web Server, 2003. Available from `http://www.zeus.com/`.