Performance Evaluation of Distributed Web Server Architectures under E-Commerce Workloads[1]

Xubin He, and Qing Yang[2], *Senior Member,IEEE*.

Dept. of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI 02881
E-mail: {hexb, qyang}@ele.uri.edu

[1] This research is supported by National Science Foundation under grant No. MIP-9714370.
[2] Corresponding author

Abstract

One of the central and key components to make the E-Commerce a success is high performance and highly reliable web server architectures. Extensive research has been sparked aiming at improving web server performance. Fundamental to the goal of improving web server performance is a solid understanding of behavior and performance of web servers. However, very little research is found on evaluating web server performance based on realistic workload representing E-Commerce applications which are usually exemplified by a large amount of CGI (Common Gateway Interface), ASP (Active Server Page), or Servlet (Java Server-side interface) calls. This paper presents a performance study under the workload with a mixture of static web page requests, CGI requests, Servlet requests, and database queries. System throughputs and user response times are measured for five different server architectures consisting of PCs that run both a web server program and a database. We observed that performance behaviors of the web server architectures considered under this mixed workload are quite different from that under static page workload and sometimes counter-intuitive. Our performance results suggest that there is a large room for potential performance improvement for web servers.

Index Terms—Distributed web server, performance analysis, E-Commerce workload, CGI, servlet

I. Introduction

WITH the phenomenal growth of popularity of the Internet, performance of web servers has become critical to the success of many corporations and organizations. Much research has been sparked to improve web server performance such as through caching [11,12], OS [5,6,24,25], and others [3,9,18,23]. Fundamental to the goal of improving web server performance is a solid understanding of behavior of web servers. A number of performance studies on web server performance have been recently reported in the literature [1,2,4,7-10,13,16,19,21,22]. Most of these studies characterize server performance using workloads that either consist of mainly static web page accesses or many static web page accesses blended with a small percentage of CGI (Common Gateway Interface) scripts that perform very simple computation functions. Historically, these studies can be considered realistic because the Web was originally evolved from academia and research institutions. The Internet sites of this sort including some ISP (Internet Service Providers) are usually content-oriented meaning that they provide information to the public in the form of HTML files. Users can obtain information from these sites through static web pages including text, images, audio, as well as video information.

In the last couple of years, the most noticeable development of Internet applications is commercial applications or E-Commerce. E-Commerce has grown explosively as existing corporations make extensive use of the Internet and thousands of Internet start-up companies emerge. The workload characteristics of web servers in E-Commerce are dramatically different from academic institutions and content-oriented ISPs. Static web pages that are stored as files are no longer the dominant web accesses in this type of application environments. We have studied over 150 commercial web sites to investigate how these web sites work. Observation of these web sites reveals that CGI, ASP (Active Serve Page), or Servlet (the server side Applet of Java

language) dominant most web accesses except for the very first one or two pages when a user comes to a site. Among the sites that we investigated, about 70% of them start CGI, ASP, or Servlet calls after a user surfed the first or second homepage of a site. A typical example of such a web site is online shopping center. When a user comes to this kind of site, the first page to be seen contains a keyword search field and product category selections. Entering a keyword or clicking on a category link by the user results in a form being posted to a program that queries databases. All subsequent interactions between the user and the web server such as shopping cart and purchase transactions are performed by the database application programs at the web server machine through web interfaces. Other types of commercial web sites such as search engines, whole sales, and technical support and services exhibit quite similar behavior as we observed online.

In order to design high performance and reliable web servers for E-Commerce applications, it is critical to understand how web servers work under this application environment. We need to gain insight into the server behavior and the interaction between web server and database servers. In particular, we need to quantitatively measure and evaluate the performance and resource utilization of such web servers.

Despite the importance of measuring and understanding the behavior of such web servers, there is no published research assessing the impact of database, web server, and interaction between database server and web server on the overall performance of web server architectures. In this paper, we present an experimental study on web server performance under E-Commerce workload environment. The server architectures being studied include a single machine server such as a Pentium PC, and distributed servers consisting of multiple machines interconnected through an Ethernet. In case of single machine servers, both the web server program and

database reside on the same machine. In case of distributed servers, several different system configurations are considered depending on where web server program and database reside and how the proxy works. The web server programs studied here include Apache web server which is the most popular web server on the Internet, Java Web server which has some new features, and IIS which is also fairly popular. The database server that we use in our measurement is MSQL 2.0[14]. We measured the performance of 5 different architecture configurations using 6 types of synthetic workloads varying the percentage of CGI and Servlet calls.

Our measurement results show that workloads with mixed static pages, CGI and Servlet requests exhibit significantly different server behavior from the traditional static web page accesses. Some of our observations from the measurements are counter intuitive and contradictory to common believes. For example, when CGI calls are added to the workload, the average system throughput is higher than that of static page only workload because of balanced resource utilizations at the web server. We also observed that with similar workload, servlet usually shows better throughput/response time performance than CGI. Our measurement results also suggest that there is a large room for potential performance improvement of web servers. Minimizing kernel overhead of OS, efficient caching mechanism for static pages, and caching database connections are a few example areas where architects can work on to improve web server performance.

II. Experimental Setup

Our experiments are based on the system that consists of one or several duplicated web servers (WS), zero or several duplicated Database Servers (DBS), a web server selector (WS selector). The WS selector is in charge of accepting HTTP requests and scheduling one of the servers to

respond to requests. A web server and a database server can share one physical machine implying a single machine acts as a web server and a database server simultaneously. The simplest case is only one web server and one database server both on the same machine.

Item	CPU	Memory	Disk Drive	OS	Web server	Database
PC Server 1	PII 450	64M	ATA-4/ 5400RPM	Windows NT server 4.0	IIS 2.0 Java Web server 1.1.3	
PC Server 2	PII 300	64M	ATA-4/ 7200RPM	Windows NT server 4.0	IIS 2.0 Java Web Server 1.1.3	
PC server 3 (DBS 1)	PII 400	64M	ATA-4/ 7200RPM	Linux 2.0.32	Apache 1.3.6 Java Web Server 1.1.3	MSQL 2.0
Proxy (WS selector)	PII 400	64M	ATA-4/ 5400RPM	Windows NT server 4.0		
DBS 2	Pentium 166	64M	SCSI-2 /5400RPM	Linux 2.0.32		MSQL 2.0
Clients	PII 450	128M	ATA-4/ 5400RPM	Windows NT 4.0		

Table 1: Hardware and software configurations

We performed our tests on a 100Mbps LAN network which is isolated from other networks. Our testbed includes 3 PCs as servers, a database server, a proxy as WS selector and 16 identical client machines. All these machines are interconnected by a 100 Base-T link. Table 1 lists the hardware and software configurations of these machines, where PC server3 may act as a web server and/or a database server.

We selected 5 typical architecture configurations as shown in Figure 1.

a) A web server and a database server reside on the same machine PC Server3 (Figure 1a).
b) A web server and a database server run on different machines (Figure 1b). PC server1 acts as the web server and PC server3 acts as database server.
c) 2 web servers and 1 database server: PC server1 and server2 act as web servers and PC server3 acts as database server (Figure 1c).
d) 2 web servers and 2 database servers: PC server1 and server2 act as web servers, while PC server3 and DBS 2 act as database servers (Figure 1d).

e) 3 web servers and 2 database servers: PC server1, server2 and server3 act as web servers, while PC server3 and DBS2 act as database servers (Figure 1e).

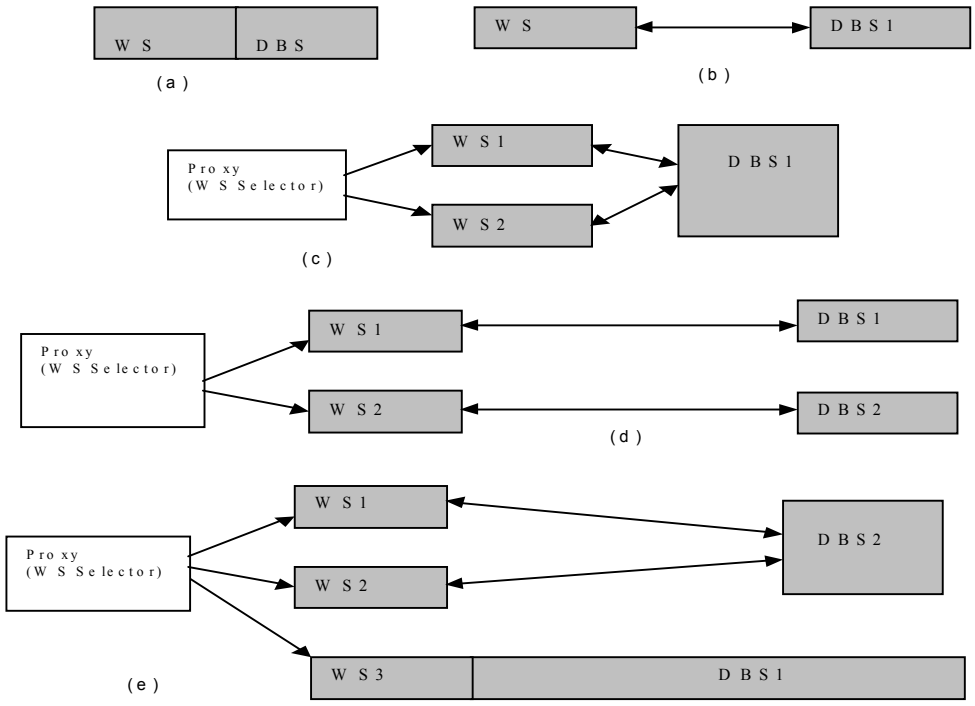


Figure 1.Experiment configurations

We use WebBench^{TM} 3.0[26] which was developed by Ziff-Davis Inc. as the benchmark program to perform our tests. WebBench, which is used and recommended by PC Magazine, measures the performance of Web server. It consists of three main parts: a client program, a controller program and a workload tree. To run WebBench, we need one or several web servers connected through a PC running the controller program and one or more PCs each running the client program. WebBench provides both static standard test suites and dynamic standard test suites. The static test suites access only HTML, GIF, and a few sample executable files. They do not run any programs on the server. The dynamic test suites use applications that actually run on the server.

The workload tree that is provided by WebBench contains the test files the WebBench clients access when we execute a test suite. The tree uses multiple directories and different directory depths. It contains 6,010 static pages such as HTML, GIF, and executable test files that take up approximately 60 MB of space.

We expanded above workload in two ways: firstly,we added more static pages to make the data set size up to 160MB; secondly, we added 300 simple CGI programs and 300 simple servlet prgrams which have simple functions such as counter, snooping, redirection and so on. Besides, we added other 400 servlet programs which access to databases using JDBC calls. We have two databases in our database server, namely BOOKS and EMPLOYEE. There are 15 and 18 tables in databases BOOKS and EMPLOYEE respectively, with different number of records ranging from 100 to 20,000.

Client processes running on client PCs continuously make HTTP requests to the server with zero think time. The traffic intensity seen by the server is therefore represented by the number of active clients making HTTP requests. The type of requests generated by each client is determined by the following 6 different types of workloads:

1) Static: All requests are static pages with 28% pages having size less than 1KB, 47% pages having size between 1K and 10K, 17% pages having size between 10K and 20K, 7% pages having size between 20K and 40K, and 1% pages having size larger than 40K.
2) Light CGI: 20% requests are CGI and 80% are static pages.
3) Heavy CGI: 90% requests are CGI and other 10% are static pages.
4) Heavy servlet: 90% requests are servlet and other 10% are static pages.
5) Heavy database access: 90% requests are requests with database access and other 10% are static pages.

6) Mixed workload: 7% requests are CGI, 8% are simple servlets, 30% are database accesses and others are static pages with different sizes.

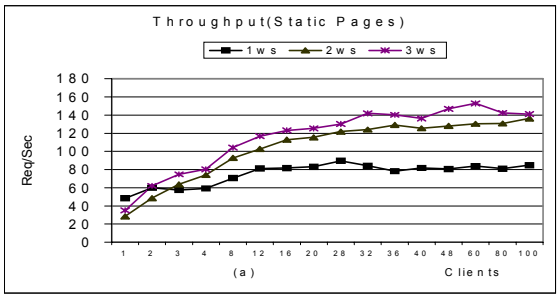
The web server software that we used for workloads 1), 2) and 3) is Apache under Linux, and IIS under Windows NT. For workloads 4), 5), 6), Java Web server[15] is used. The clients in our tests refer to logic client processes which are running evenly on 16 interconnected machines.

For each workload, we have 16 test mixes with increasing number of clients, and each mix lasts 100 seconds. After one workload was tested, we restarted all the machines and then begin another workload test.

III. Measurement Results

We started our experiments with measurement of static web page accesses. In this scenario, 100% of user requests are for static web pages that are stored in web server as files. Figure 2 shows the measurement results of three different architecture configurations in terms of throughput (Figure 2(a)) and user response time (Figure 2(b)). It can be seen from this figure that the throughput increases as the workload increases. The server with a single PC gets saturated when the number of clients exceeds 30 with throughput being about 80 requests/second. The user response time also gets excessively long after the number of clients is more than 30. The servers with 3 PCs get saturated at throughput being about 150 requests/second, approximately doubled the performance of the single PC server. The response times of these two servers are also significantly lower than that of the single PC server.

For static web pages, much of the CPU time is spent on I/O interrupt and file copying inside RAM [13]. Therefore, we can conclude that static pages place high demand on RAM and disk I/O since static web page accesses are mainly file access operations.



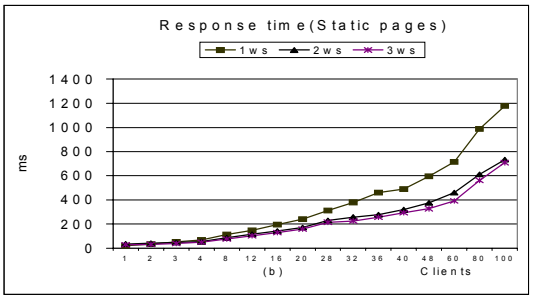
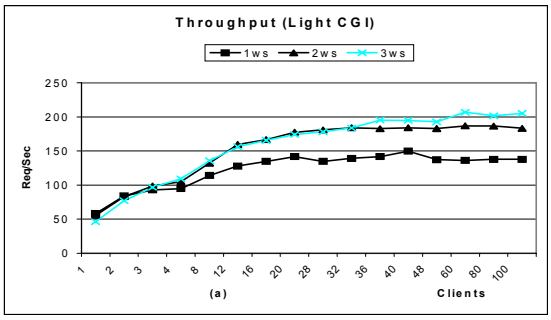


Figure 2 Measurements under static pages



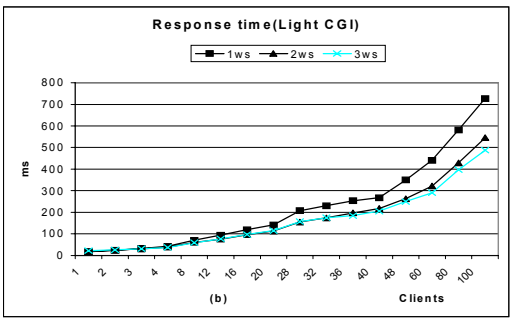
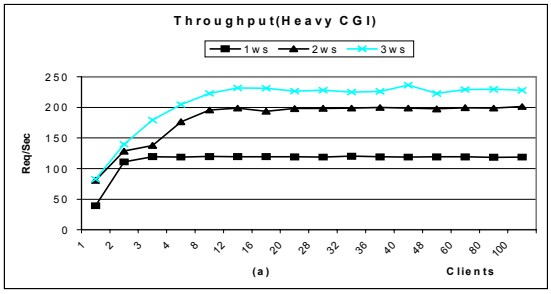


Figure 3 Measurements under light CGI



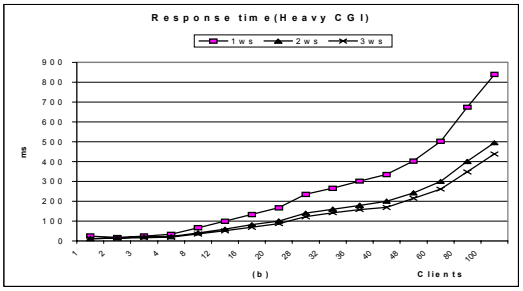
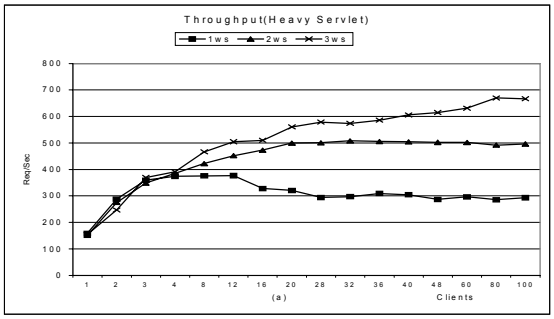


Figure 4 Measurements under heavy CGI



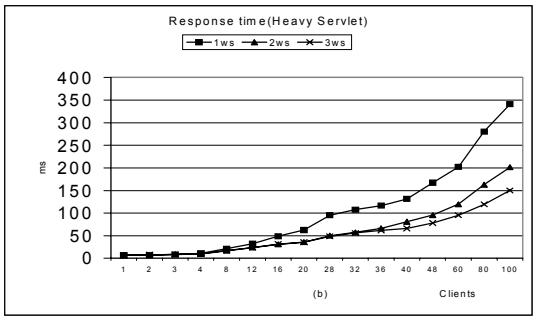


Figure 5 Measurements under heavy servlet

Legends in figure 2,3,4 and 5:

1ws: PC server1 as the web server

2ws: PC server1 and server2 as the web servers

3ws: PC server1, server2 and server3 as the web servers

Our next experiment is to add a small percentage of CGI requests into the static web page workload. The CGI requests added are mainly short and simple functions such as counter, redirection, snoop, and dynamic page generations. Figure 3 shows the measurement results of this experiment. In this figure, the workload consists of 80% of static web page requests and 20% of simple CGI calls. Contrary to common belief, adding CGI programs into the workload increases the system throughput rather than decreasing it. The maximum system throughput increases from about 150 request/second (Figure 2a) to about 200 request/second. Such performance increase can be mainly attributed to the fact that the server resources are better utilized in case of this mixed workload than in case of only static web pages. Although CGI calls add additional work to the server, the additional works are mainly performed by CPU not I/O systems. As a result, system resources at the server work in parallel with fairly good balance in handling web requests. The better performance is also shown in terms of user response time as shown in Figure 3b where the response times are flatter and lower than Figure 2b.

The results shown in Figure 3 and some subsequent figures unveiled a very important fact about performance evaluation of web servers. That is, realistic workload used in performance evaluation is extremely important. Intuition or using workload that is nonrealistic to do performance evaluation may give misleading results and conclusions that may send architecture researchers into wrong directions. The results shown above are a simple example indicating the importance of selecting correct workloads.

In order to observe further how CGI affects web server performance, we increase the proportion of CGI requests in the workload from 20% to 90%. As a result, we have a heavy CGI workload. The measurements for this workload are shown in Figure 4. Because a large portion of web

requests is CGI calls, the server load becomes CPU-bound not I/O-bound. It is clearly shown in Figure 4 that the more CPUs we have in the server, the better throughput we can obtain.

Replacing all the CGI calls in the workload by Servlet calls, we have heavy Servlet workload. The performance results for this case are shown in Figure 5. It is interesting to observe from this figure as compared to Figure 4 that the system throughput jumped up significantly. For the 2-PC server case, the maximum throughput jumps from 200 requests/second to a little less than 600 requests/second, almost a factor of 3! To understand why there is such a considerable performance difference, let us take a close look at how CGI and Servlet work.

To make a CGI request, a user usually fills out a form or clicks on a hyper link with predefined form data. When the form is submitted, a GET or POST command is sent to the web server to start a CGI program with the form data as input parameters. This program runs as a new process. In Unix systems, a server runs a CGI program by calling *fork* and *exec* to form a new process. Therefore, each CGI call results in a new process and the kernel will have to manage and maintain these processes. Communication between processes and between CGI processes and main server process are done through reading and writing files including standard input and output. Windows NT systems work in a similar way in handling CGI programs. Servlet, on the other hand, is a server API in Java language that is functionally similar to CGI. The name comes from Applet representing server side Applet. The most distinguished feature of Servlet as compared to CGI is that each call to the Servlet from a user results in a new thread to be started rather than a new process. All threads resulting from Servlet from users run in a single process. As a result, significant amounts of process management and communication overhead are not present in Servlet environment. Therefore, the web server can handle Servlet requests much faster than CGI requests. It is also noticed that the throughput drops after the saturation point as

shown in Figure 5a. This performance drop is caused by I/O activities after the number of threads is very large and the RAM cannot cache all the threads. This phenomenon is not present in heavy CGI case because I/O activities start from low workload giving rise to smoother curves.

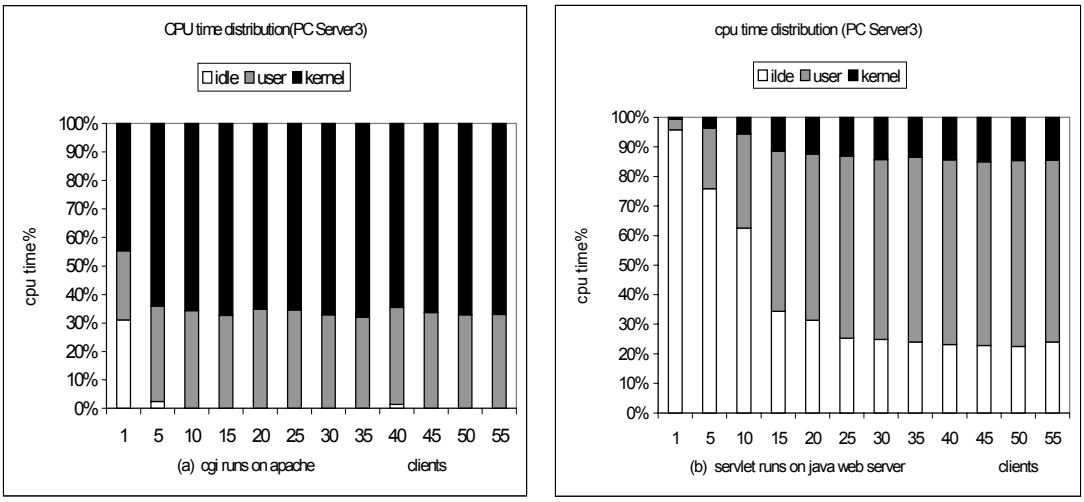


Figure 6. CPU utilization

To further prove our analysis above, we have done additional experiments to measure the CPU utilization for heavy CGI and Servlet workloads as shown in Figures 6(a) and 6(b). The CPU time is broken down into user time, kernel time, and idle time, respectively. Figure 6(a) shows the CPU utilization under heavy CGI requests that are called through Apache web server. It shows that over 60% of CPU time is spent on kernels for heavy CGI workload, implying large multi-process overheads. For heavy Servlet workload, the situation is quite different as shown in Figure 6(b). In this case, the proportion of time spent for kernel is significantly reduced to approximately 10% to 15% as shown in the figure. At high workload, most of CPU time was spent for user code and therefore high system throughput.

The above observation is very important because it suggests a large space for potential performance improvements on web server designs. Although fast CGI proposed by Open Market

Inc. improves CGI performance by means of caching and minimizing process management overheads, there is still a plenty of room for further performance improvement.

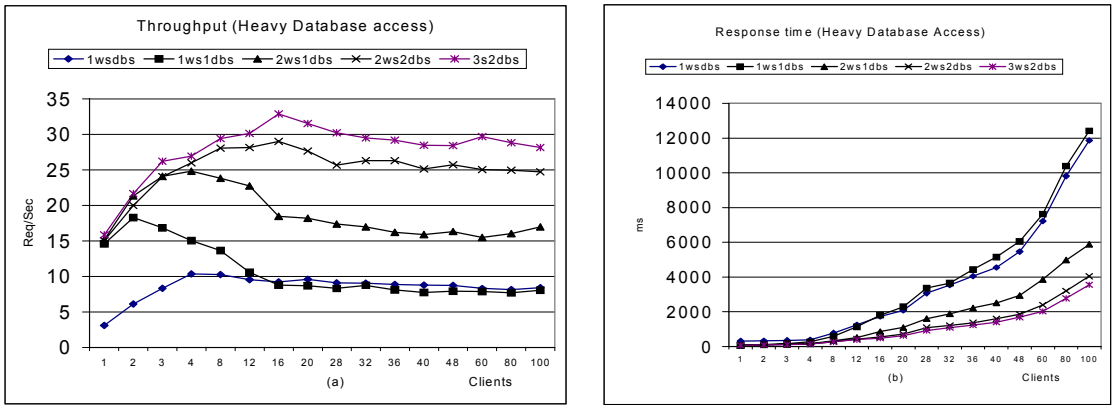


Figure 7 Measurements under heavy database accesses, where:
1wsdbs: PC server3 as the web server and database server simultaneously
1ws1dbs: PC server1 as the web server and DBS1 as the database server
2ws1dbs: PC server1 and server2 as web servers and DBS1 as the database server
2ws2dbs: PC server1 and server2 as web servers, while DBS1 and DBS2 as database servers
3ws2dbs: PC server1, server2 and server3 as web servers, while DBS1 and DBS2 as database servers

As mentioned in the introduction, most E-Commerce web servers see significant amount of database accesses among users' web requests. To evaluate how these database accesses affect web server performance, we inserted a large amount of JDBC (Java DataBase Connectivity) calls in Servlet programs. The JDBC calls are mainly simple database queries. The system throughput and average user response times are plotted in Figure 7. In case of single PC web server with a database server, the throughput drops down quickly as workload increases. There are several reasons for this phenomenon. First of all, heavy database accesses place high demand on disk I/O systems. When the number of database queries or database connections gets large, the RAM in the PCs is not large enough to cache all the database connections. Increasing number of database queries beyond this point will creates an adverse effects increasing the processing time for each database query. As a result, we observed reduced throughput. Secondly, heavy Servlet

calls at the web server also require a lot of CPU time as evidenced in our previous discussions. Adding an additional PC as a web server while keeping the same database server results in increased maximum throughput as shown in Figure 7a. However, the throughput still drops after the saturation point. In case of 2ws2dbs and 3ws2dbs architectures, the performance drop is no longer as steep because of one more database server.

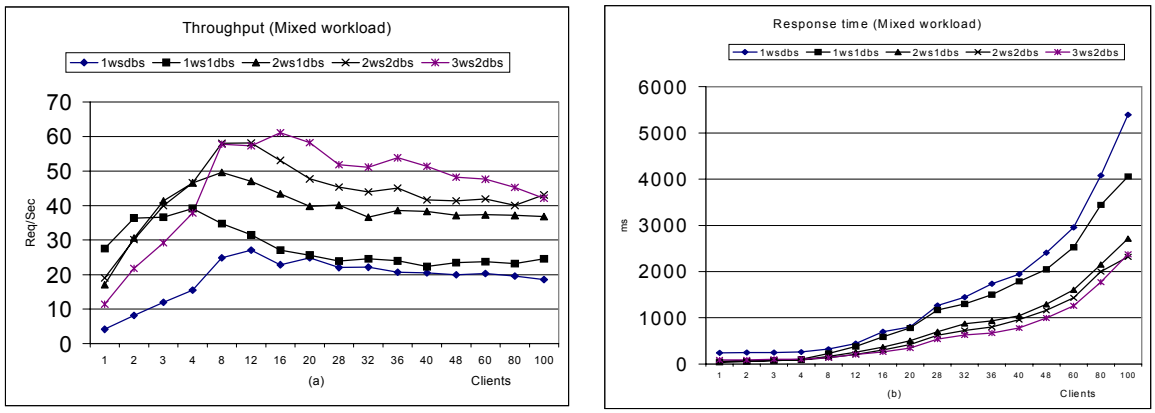


Figure 8 Measurements under Mixed workload, where:
1wsdbs: PC server3 as the web server and database server simultaneously
1ws1dbs: PC server1 as the web server and DBS1 as the database server
2ws1dbs: PC server1 and server2 as web servers and DBS1 as the database server
2ws2dbs: PC server1 and server2 as web servers, while DBS1 and DBS2 as database servers
3ws2dbs: PC server1, server2 and server3 as web servers, while DBS1 and DBS2 as database servers

Figure 8(a) and (b) show the measurements results for a more realistic workload with a mixture of static pages, CGI requests, and Servlet with database queries. Static pages, CGI requests, Servlet requests and database requests account for 55%, 7%, 8% and 30%, respectively. With this type of workload, static page requests demand high disk I/O activities, CGI and servlet requests require high CPU utilization, and database queries in Servlet programs place high demands on both CPU and disk I/Os. As a result, throughputs for this workload keep changing as request rate changes for all architecture configurations considered. They all increase at low workload and decrease after reaching saturation points though saturation points are different depending on architectures. These results suggest that it is very important to monitor and tune

web servers while at operation to avoid bad service. A threshold should be setup to stop accepting additional requests once the server is approaching saturation point.

To this point, all our experiments are based on static scheduling at the proxy, i.e. web requests are statically assigned to different web servers as they come based on a predefined percentage. In order to assign web requests dynamically to different web servers depending on their real time workload, we have developed a simple adaptive proxy server. The adaptive proxy distributes web requests to servers dynamically using the current load information of the servers. Figure 9 shows the performance comparison between static proxy and adaptive proxy. We observed about 10% to 15% performance improvement due to adaptive proxy at high workload.

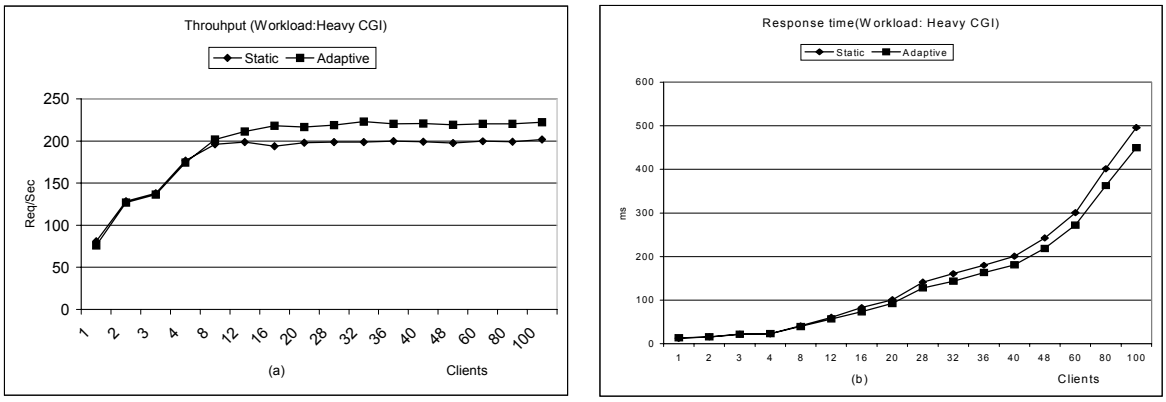


Figure 9 The influence of requests scheduling algorithm of proxy

IV. Related Work

Evaluating and measuring the performance of web servers has been the subject of much recent research. Banga and Druschel [4] proposed a method to generate heavy concurrent traffic that has a temporal behavior similar to that of real Web traffic. They also measured the overload behavior of a web server and throughput under bursty condition. Arlitt and Williamson [2] characterized several aspects of web server workloads such as reference locality, request file type

and size distribution. Almeida and Yates [1] provided a tool called WebMonitor to measure activity and resource consumption both within the kernel and in HTTP processes running in user space. They [25] also measured the performance of a PC acting as a standalone web server running Apache on top of Linux. Their experiments varied the workload at the server along two dimensions: the number of clients accessing the server and the size of the workload tree. Hu, Nanda and Yang[13] quantitatively identified the performance bottlenecks of Apache web server and proposed 7 techniques to improve its performance. Courage and Manley [10] created a self-configuring benchmark that bases WWW load on real server logs and analyzed general WWW server performance and showed that CGI application affected the latency of not only CGI, but other responses as well. Barford and Crovella [7] created a realistic Web workload generation tool SURGE and generated representative web workloads for server performance evaluation. Colajanni, Yu and Dias [9] discussed in detail the load balance and task assignment policies at DNS for distributed web servers. Pai[22] presents an approach called locality-aware requests distribution (LARD) [22] tries to distribute web requests to back-end nodes according to request contents in order to maximize cache hit ratio at servers.

In summary, all web server performance evaluations that we know of evaluate web servers using workloads that consist of static pages and small percentage of CGI. To the best of our knowledge, no research attempts to accurately measure the effects of heavy CGI, servlet, database accesses on the performance of web servers.

V. Conclusions

In this paper, we have studied the performance of web servers under the workloads that represent typical E-Commerce applications. We have measured the system throughputs and user response

times of 5 different types of server architecture configurations. For workload consisting of static web page only, disk I/O plays an important role at the server since all web requests are mainly file accesses. For this kind of workload, it is clearly beneficial to have a large RAM to cache as many web pages as possible. If CGI or Servlet calls are added to the workload, the server behavior changed significantly. Contrary to common believes, most web servers considered here can obtain higher throughputs under the mixed workload than under the static page workload. In this situation, efficiently handling multi-process or multi-thread becomes essential to system performance. Our measurement results suggest that there are rooms for potential performance improvement on server architectures. When database activity increases, server performance drops down significantly compared to other workload because of heavy demand on both CPU and disk I/Os. The average system throughput increases with workload at low load range and decreases at high load, which suggests that it is necessary to continuously monitor web servers and to set up threshold for accepting web requests.

Acknowledgments

Our thanks to ZhenQiang Shan for providing the databases and writing the servlet programs. Further thanks to Tycho Nightingale, Jonathan Short and Tim Toolan for providing client PCs and WWW log files.

References

[1] M. Almeida, V. Almeida, and D. J. Yates, "Measuring the Behavior of a World-Wide Web Server," *Seventh IFIP Conference on High Performance Networking (HPN),* White Plains, NY, Apr. 1997, pp. 57-72.

[2] M.F Arlitt, C.L. Williamson, "Web Server Workload Chracterization: The Search for Invariants," *In Proceeding of the ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems,* 1996, pp. 126-137.

[3] G. Banga, F. Douglis, M. Rabinovich, "Optimistic Deltas for WWW Latency Reduction," *In Proceedings of 1997 Usenix Annual Technical Conference,* Anaheim, CA, Jan. 1997, pp. 289-303.

[4] G. Banga, P. Druschel, "Measuring the Capacity of a Web Server," *Proceedings of the USENIX Symposium on Internet Technologies and Systems,* Monterey, Dec. 1997.

[5] G. Banga, J.C. Mogul, "Scalable kernel performance for Internet servers under realistic loads," *In Proceedings of 1998 Usenix Annual Technical Conference,* New Orleans, LA, June 1998.

[6] G. Banga, P. Druschel, J. C. Mogul, "Better Operating system features for faster network servers," *in the* Proceedings of the Workshop on Internet Server Performance*,* Madison, WI, June 1998.

[7] P. Barford, M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," *In Proceeding of the ACM SIGMETRICS'98 Conference,* Madison, WI, 1998, pp. 151-160.

[8] T. Bray, "Measuring the Web," *In Fifth International World Wide Web Conference,* Paris, France, May 1996.

[9] M. Colajanni, P.S. Yu, D.M. Dias, "Analysis of task assignment policies in scalable distributed web-server systems," *IEEE Transactions on Parallel and Distributed Systems,* Vol.9, No. 6, June 1998, pp. 585-600.

[10] M. Courage, S. Manley, "An Evaluation of CGI Traffic and its Effect on WWW Latency," Http://www.eecs.harvard.edu/vino/web, May 10, 1999.

[11] S. Glassman, "A caching relay for the World Wide Web," *In WWW'94 Conference Proceedings,* 1994.

[12] V. Holmedahl, B. Smith, and T. Yang, "Cooperative caching of dynamic content on a distributed web server," *in Proc. of 7th IEEE International Symposium on High Performance Distributed Computing (HPDC-7)*, Chicago, IL, July, 1998, pp.243-250.

[13] Y. Hu, A. Nanda, Q. Yang, "Measurement, analysis and performance improvement of the Apache Web Server," *in the 18th IEEE International Performance, Computing and Communications Conference (IPCCC'99),* Phoenix/Scottsdale, AZ, Feb. 1999.

[14] Hughes Technologies Pty Ltd., "Mini SQL 2.0 Reference," URL: http://www.hughes.com.au/library/msql/manual_20/, Oct. 12, 2000.

[15] JavaSoft Co., " Java Web server," URL: http://www.sun.com/software/jwebserver/, Oct. 3, 2000.

[16] S. Manley, M. Seltzer, "Web Facts and Fantasy," *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems*, Monterey, CA, Dec. 1997.

[17] S. Manley, M. Seltzer, M. Courage, "A Self-Scaling and Self-Configuring Benchmark for Web Servers," *Proceedings of the ACM SIGMETRICS'98 Conference,* Madison, WI, 1998, pp. 270-272.

[18] Microsoft Co., "CIFS: An Internet File System Protocol," http://msdn.microsoft.com/workshop/networking/cifs/, Oct. 5, 2000.

[19] J.C. Mogul, *Operating Systems Support for Busy Internet Servers,* WRL Technical Note TN-49, Http://www.research.digital.com/wrl/home.html, May 12, 2000.

[20] J.C Mogul, "The case for persistent-connection HTTP," *In SIGCOMM Symposium on Communications Architectures and Protocols,* Cambridge, MA, Aug. 1995, pp.299-313.

[21] E. M. Nahum, T. Barzilai, D. Kandlur, *Performance Issues in WWW servers,* Tech. Report, IBM Corp., Feb. 1999.

[22] V. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. M. Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers." *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VIII)*, Oct. 1998, pp. 205-216.

[23] D. C Schmidt, J.C. Hu, "Developing Flexible and High-performance Web Servers with Frameworks and Patterns," http://www.cs.wustl.edu/~jxh/research/research.html#jaws, Oct. 6, 2000.

[24] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, C. Yoshikawa, "WebOS: Operating System Services for Wide Area Applications," *Seventh Symposium on High Performance Distributed Computing,* July 1998.

[25] D.J Yates, V. Almeida, J.M. Almeida, *On the Interaction Between an OS and Web server,* Boston University Computer Science Dept., Boston Univ., MA, Tech. Report CS 97-012, July 1997.

[26] Ziff-Davis Inc., "WebBench 3.0." URL: http://www.zdnet.com/zdbop/webbench/, Aug. 10 2000.