Measurement and Analysis of Web Server Response Time

DongJun Shin, Kern Koh
djshin@oslab.snu.ac.kr, kernkoh@june.snu.ac.kr
School of Computer Science and Engineering, Seoul National University

Youjip Won
yjwon@email.hanyang.ac.kr
Division of Elec. And Comp. Engineering, Hanyang University

Abstract - Many researches have been focused on measuring the behavior of web server and improving performance. In this paper, we present another approach to measure the web server performance using the timing information gathered at client. We find the performance bottleneck of web server by varying the workloads and analyzing the effect of changes in various phases of response time.
From our observation, the response time of web server increases exponentially as the number of clients in workloads increases. This is mostly due to the limitation of server resources and the interaction overhead between OS and web server process.

I. Introduction

Web server measurement is necessary in finding the bottleneck and improving performance. Depending on the situation and characteristics of measured metrics, it is done either at server or at client.
Measurement at server is usually performed on per-connection-basis or sampling. While the former provide much detailed information about current status of server, it would incur too much overhead. The latter provide less detailed information with smaller overhead. Measurement at client introduces no overhead on the server being watched, but it is difficult to have a deep insight into the current status of server.
In previous works [1, 3, 4, 5], they used the client for generating specified workload and for obtaining overall statistical information – such as response time and throughput. We expect that we can get much more useful information from client if we carefully examine the interaction between client and server. Although this does not provide as precise information as we could obtain by server-side measurement, it is still useful in observing current status of the server and finding possible performance bottleneck.
In this paper, we used a workload generator based on Surge and measure the response time of Apache web server [6] on Linux in LAN environment. We first categorize the request time into three phases and analyze the effect of changes in the workload on each phase. With the assumption that the round-trip delay is almost negligible in LAN, the time taken at each phase represents the current load status of various part of server.

II. Detailed Analysis of Client and Server Interaction

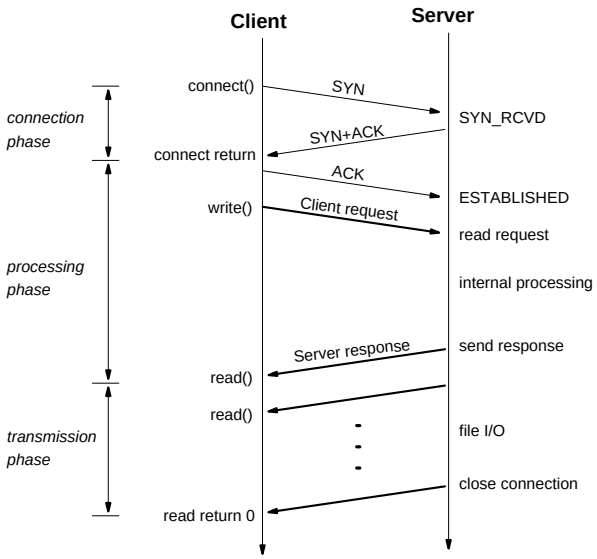

Figure 1: Packet exchange and duration of each phase in HTTP/1.0 transaction

Response time means the time between when the client initiates connection request and finishes receiving the last byte of response. We divided the

This work was supported by Brain Korea 21 Project

total duration of the response time into three phases: connection phase, processing phase and transmission phase. Figure 1 depicts the packet exchange and the duration of each phase for HTTP/1.0 transaction.

Connection phase begins when a client initiates connection request, and ends when the connection is established. Processing phase begins with sending HTTP request, and ends just before receiving HTTP response. Transfer phase begins with receiving the first byte of HTTP response, and ends when the connection is closed.

The time spent in each phase can be calculated as follows.

- Connection time = 1 RTT (Round Trip Time) + latency for handling incoming connection request at server TCP
- Processing time = 1 RTT (the time to send HTTP request is overlapped with client ACK) + time spent in accept queue + latency for internal processing at web server
- Transmission time = latency for network I/O to send requested file + latency for file I/O

The RTT of our experimental environment (Fast Ethernet) is as small as 0.2 milliseconds, which is almost negligible. Thus, we can assume that the duration of each phase represents the current load level of various parts of server components – the connection time corresponds to the connection processing time at server TCP, the processing time to the internal processing time at web server, and the transmission time to the I/O processing time at server.

III. Measuring Environment

A. Workload Generator

Our workload generator uses dataset generated by Surge [7]. The original workload generator simulates clients using threads. But this approach uses lots of system resources and needs several machines to run the test with sufficient number of clients. Instead, we borrowed the event-driven engine from s-client [2] for workload generation and modified it to fit our need. This implementation has two benefits over the original implementation. 1) It needs lesser machines to run the test. 2) We can easily control and observe the activity of entire clients. With this method, we can increase the total number of clients up to the point that system permits using only one client machine. The workload generator only simulates HTTP/1.0 client.

For each request, the following information is logged.

- req_time: the timestamp when request is made
- conn_time: the time spent in connection phase
- proc_time: the time spend in processing phase
- trans_time: the time spent in transmission phase
- size: size of HTTP response body

Number of unique objects	7000
Total data set size	158.1Mbyte
Maximum object size	5.76Mbyte
Average size of requested objects	14.7Kbyte

Table 1: Statistical summary of data set

The statistical summary for the data set used in our experiment is shown in Table 1 ([7] describes more detail information about the Surge dataset model).

The files serviced by web server are cached in the page cache because Apache uses mmaped I/O for file read operation. The maximum page cache size in our experiment is calculated as 96Mbytes, which is 75% of the total main memory (in Linux, the default setting for page cache size is between 15% and 75% of the total memory). We set the size of dataset to be larger than the maximum page cache size in order to measure the impact of disk I/O overhead. To avoid the effect of cold-start miss, we pre-loaded the page cache before the measurement begins.

To change the workload, we varied the number of clients from 100 to 1100, after that point we can't test any more with only one client machine. For every workload, we ran the benchmark for 20 minutes as mentioned in [7]. When the pre-computed dataset was not sufficient for the test duration, the workload generator repeats the requests from the beginning of the dataset.

B. Hardware and Software

	Client	Server
CPU	P-II 266Mhz	P-III 450Mhz
Main Memory	64Mbyte	128Mbyte
NIC	Ethernet 10/100Mbps	Ethernet 10/100Mbps
OS	Linux 2.2.13	Linux 2.2.13
Web Server		Apache 1.3.9

Table 2: Machine configuration

We used one client machine and one server machine for the experiment. The two machines are connected via 100Mbps Ethernet switch. The configuration of each machine is shown in Table 2.

We tune the Apache web server and the Linux kernel to experiment with larger number of clients than are available in default configuration. We increase the hard limit of Apache processes to 1024 and increase the maximum number of file descriptor per process to 2048. This enables us to experiment with as many as 2048 simultaneous connections with one client machine.

IV. Measurement Result

A. Connection time analysis

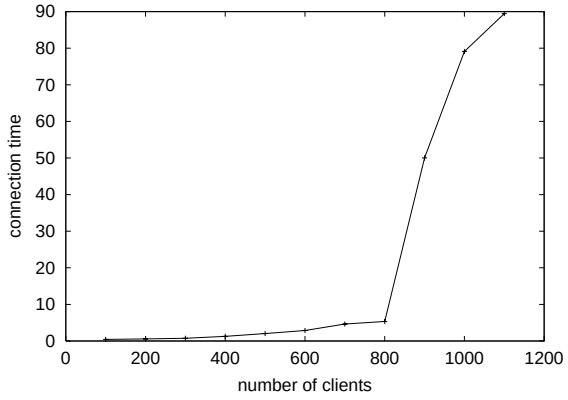

Figure 2: Connection time vs number of clients

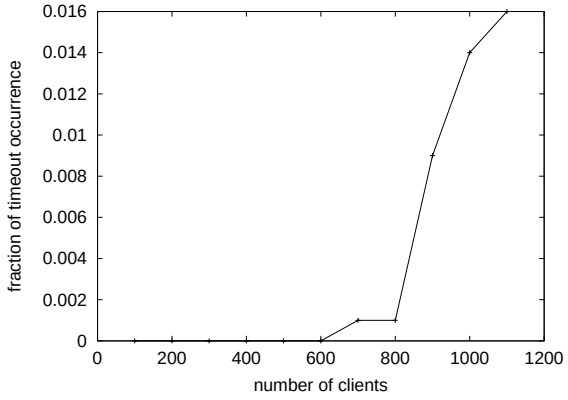

Figure 3: Fraction of timeout occurrences

Figure 2 illustrates the average connection time versus the number of clients. The hyper-linear increase for more than 800 clients is due to the timeout during three-way handshake in connection establishment (Figure 3).

If there are large number of simultaneous connection requests that exceeds the capacity of the SYN-RCVD queue, some connection requests will timeout and the clients re-send the connection requests following an exponential back-off rate. In Linux, if the client does not receive SYN-ACK packet for the initial connection request, the timeout intervals for subsequent re-transmissions of connection requests (SYN) are 3 sec, 9 sec, 21 sec, 45sec, 93 sec, and so on. The connection where timeout occurred will take hundreds or thousands of times longer delay than normal one and will greatly increase the average connection time. In real situation, client may abort the connection if he/she waits for long time (timeout occurs) without seeing the "Host Connected" message. It is important to lower the occurrence of timeouts to shorten the delay under acceptable level.

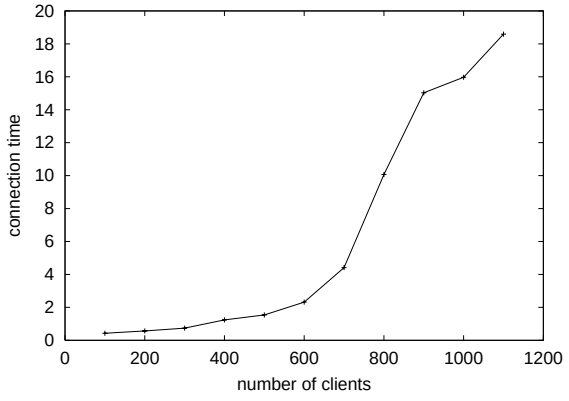

Figure 4: Connection time vs number of clients (SYN_RCVD queue = 1024)

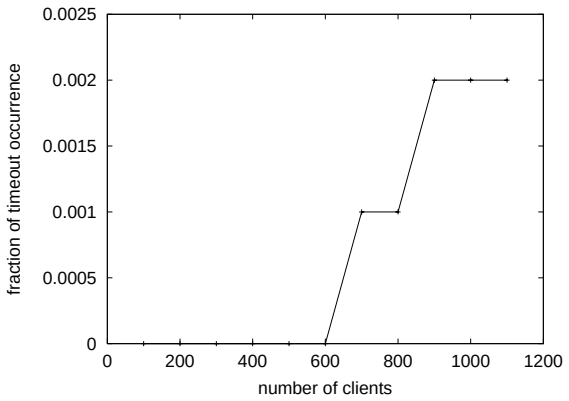

Figure 5: Fraction of timeout occurrences during connection phase (SYN_RCVD queue = 1024)

The default size of SYN-RCVD queue on Linux is 128. We increased it to 1024 and repeated the test to measure the effect of increasing the size of queue (Figure 4, 5). By just increasing the size, we got noticeable decrease in the number of timeouts and

the connection time. But the connection time still increase at hyper-linear rate due to the inefficient implementation (linear search of SYN_RCVD queue) of Linux TCP stack.

B. Processing time analysis

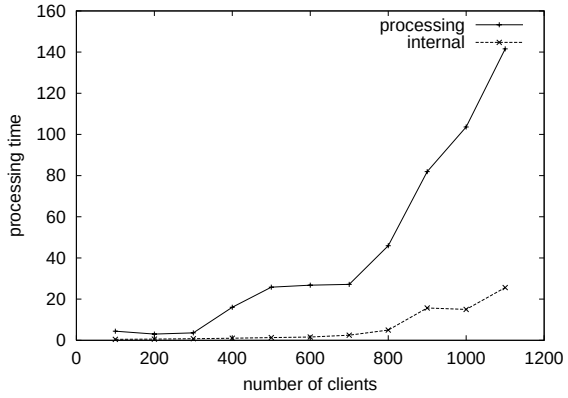

Figure 6: Processing time vs number of clients

Figure 6 illustrates the average processing time versus the number of client. As mentioned earlier, processing time includes both the time taken in kernel and user space. We can expect that the time taken in user space will increase at almost linear rate, because the throughput of web server linearly increased. To make this certain, we instrumented the Apache server to report the time it takes in user space via response header (labeled as “internal” in Figure 6). As expected, the user-level processing time increased almost linearly.

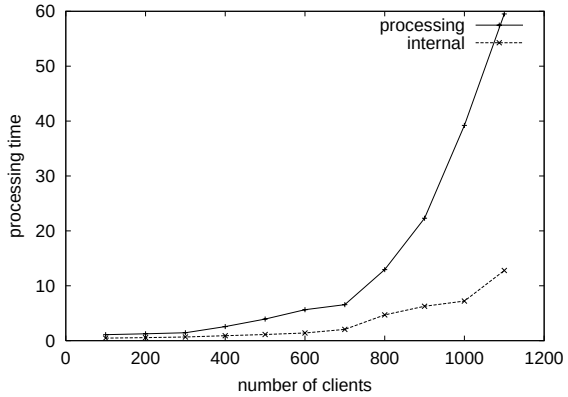

Figure 7: Processing time vs number of clients (accept queue = 1024)

The difference between “processing” line and “internal” lines - we will call this “accept” time afterwards- increases at hyper-linear rate as the number of clients increases. To see that if the limit of accept queue size (default is 128) result in the increase in accept time, we increase the size of accept queue from 128 to 1024 (Figure 7). The processing time with adjustment is less than 50% of normal one, but the accept time sill increase at hyper-linear rate.

An inefficient search of accept queue could explain the hyper-linear increase of accept time. If it really is, the processing time should be almost same as connection time assuming that the queue is not the limiting factor. But the processing time is as large as 300% of connection time with adjustment.

The main reason is that the user-level process doesn’t keep up with the rate at which TCP stack pumps the incoming requests. This is because: 1) As the Apache uses multiple processes to handle incoming requests, the context switch overhead - as many as 30000 times per second with 1100 clients (measured using *vmstat*) - became larger as the number of Apache processes increases. 2) TCP stack, which belongs to the kernel bottom-half, has higher priority than user-level process. The state transition in TCP stack initiated by Ethernet interrupts can’t be instantly delivered to user-level processes if the CPU is busy handling the bottom half routines. Efficient event-delivery mechanism and low context-switch overhead for server process is necessary to reduce the processing time, which takes major part of response time.

C. Transmission time analysis

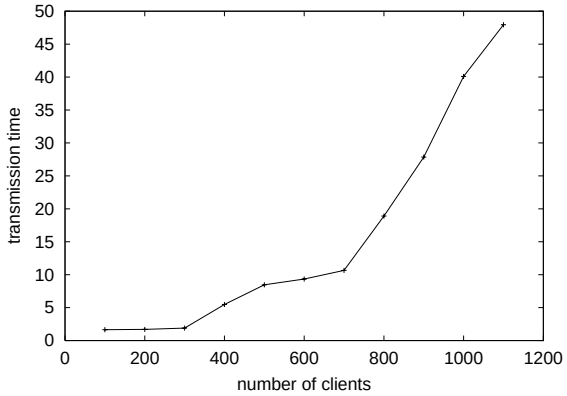

Figure 8: Transmission time vs number of clients

Figure 8 illustrates the transmission time versus the number of clients. The transmission time is largely affected by the size of requested object and whether it is cached in memory or fetched from disk. As the total size of dataset exceeds the size of main memory, some large and cold objects will result in page cache

miss and they must be fetched from disk, which will make the transmission time larger.

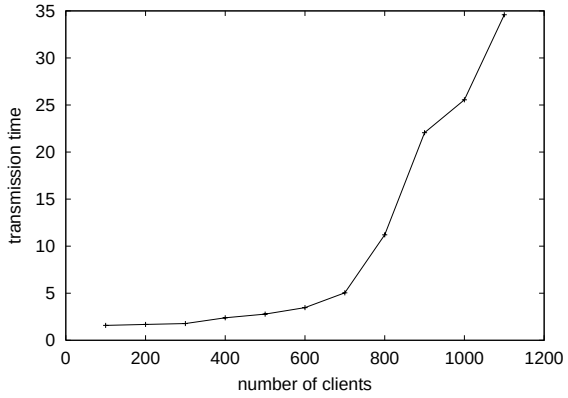

Figure 9: Transmission time vs number of clients (main memory = 256Mbyte)

To measure the effect of page cache misses, we increased the size of main memory to 256Mbyte and repeated the test (Figure 9). Transmission time takes 30% less with sufficient main memory to cover all the requested files. But the hyper-linear growth of transmission time implies that network resource will be a limiting factor in almost no time.

V. Conclusion

In this paper, we measured and analyzed the web server response time. By categorizing the response time into three phases, we could measure the effect of changes in workloads to the behavior of server components. We tried to get much useful information from the response time alone and used server-side instrumentation and monitoring as a method to verify our assumption.

There are two factors, which limit the scalability of web server performance: The limitation of server resources and the interaction overhead between OS and web server. Some simple remedies, such as kernel tunings and adding additional memories, can bring noticeable improvements in response time. But we carefully conclude that the interaction overhead will be the most important part to make web servers scalable.

The main contribution of this paper is in the approach to use client-side measurement data as a method to find the bottleneck of web server and in the quantification of web server tuning techniques. Although the experiment was restricted to controlled workloads in LAN environment, the results should be valuable in designing more scalable web server system.

ACKNOWLEDGEMENT

The authors would like to thank Paul Barford for providing the source code of the Surge.

REFERENCES

[1] Jussara M. Almeida, Virgilio Almeida, and David J. Yates. Measuring the Behavior of a World-Wide Web Server. In *Proceedings of the Seventh Conference on High Performance Networking*, White Plains, New York , April 1997.

[2] Gaurav Banga and Peter Druschel. Measuring the Capacity of a Web Server. In *USENIX Symposium on Internet Technology and Systems*, Monterey, California, December 1997

[3] James Hu, Sumedh Mungee, and Douglas C. Schmidt. Techniques for Developing and Measuring High-Performance Web Servers over High Speed Networks. In *Proceedings of the IEEE Infocom '98 Conference*, San Francisco, California, April 1998.

[4] Yiming Hu, Ashwini Nanda, and Qing Yang. Measurement, Analysis and Performance Improvement of Apache Web Server. In *IEEE International Performance, Computing, and Communications Conference*, Phoenix, Arizona, February 1999.

[5] Erich Nahum, Tsipora Barzilai, and Dilip Kandlur. Performance Issues in WWW Servers. In *Proceedings of the ACM SIGMETRICS '99 Conference*, Atlanta, Georgia , May 1999.

[6] The Apache Team, Apache HTTP server project, http://www.apache.org

[7] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation . In *Proceedings of the ACM SIGMETRICS '98 Conference*, Madison, Wisconsin, June 1998.