International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 5– No.3, February 2013 – www.ijais.org

Development and Evaluation of an Experimental Java-based Web Server

Syed Mutahar Aaqib
Department of Computer Science & IT
University of Jammu
Jammu, India

Lalitsen Sharma, PhD.
Department of Computer Science & IT
University of Jammu
Jammu, India

ABSTRACT
This paper compares the architecture of multi-threaded and event-driven web servers and highlights their advantages and disadvantages. Objective of this paper is to present a model of a novel web server architecture based on the best properties of multithreaded and event-driven architectures. Based on this architecture, an experimental java-based hybrid web server is implemented. This paper then evaluates and compares its performance with Apache and µserver for both static and dynamic workloads. The results of the experiments revealed that our experimental web server exhibits better performance than Apache and µserver for static workloads. For dynamic workloads, this web server shows substantial performance improvements as compared to Apache and slightly better performance than µserver web server.

General Terms
Performance Evaluation, Benchmarking, Web Server Development

Keywords
Web Performance Analysis, Web Server Architectures. Java Web Servers.

1. INTRODUCTION
The increasing dependence on the internet services emphasizes importance of stable underlying web server architecture [1, 2]. Basic goal in the design of efficient web server architectures is to provide capability of processing long lasting client sessions, handle large number of concurrent connections and sustain higher throughput. The most popular contemporary web servers are based on multithreaded and event-driven architecture. A multithreaded web server uses multiple threads to deal with blocking network I/O. In this architecture, if a thread is blocked in the queue, other threads can continue their execution. While handling a request, a web server either uses blocking or non-blocking socket operations. In case blocking socket operations, calling thread is blocked till all the data associated with the request is transmitted by the kernel. While in case of non-blocking operations, calling thread is allowed to execute while parts of data transmitted by the kernel are buffered. Former architecture is called multithreaded while as latter is referred as event-driven architecture. The multithreaded architectural model leads to a very simple and natural way of programming. Also multithreaded web servers are easier to build as they involve well defined and easily comprehensible logic and the numbers of possible errors while developing a multi threaded web server are also very less [3]. Context switching in multithreaded web servers incurs heavy overhead as the number of simultaneous executing threads may increase proportionally with the number of incoming requests [3]. Event-driven web servers like Flash [4] on the other hand resolve all the issues related with the multi threaded web servers but they are difficult to develop, debug and involves complicated logic and flow control [3]. Apache [5], IIS [6] and Tomcat [7] are based on the multithreaded architecture whereas Flash [4], Zeus [8], µserver [9] and SEDA [10] are examples of event-driven architectures. A number of studies [11, 12, 13, 14, and 15] have compared the features of these two web server architectures based on their mutually exclusive features. But till date no substantial work has been carried out to develop a server that takes advantage of best characteristics of these two architectures. This paper thus briefly describes working of multithreaded and event-driven web server architectures, highlights their advantages and disadvantages and presents the implementation of a new experimental web server based on the hybrid model proposed. Finally, the performance of this new experimental java based web server is evaluated and compared with Apache and µserver.

2. WEB SERVER ARCHITECTURES

2.1 Multi-Threaded Architecture
Multithreaded architecture is a natural way of programming a server and is the most common approach for implementing web servers, e.g. Apache [5], IIS [6]. In this architecture, one thread is in charge of handling incoming HTTP requests. After a connection is established, a worker thread is selected from a pool of threads and this thread is responsible for handling all the further client requests on this connection. The thread pool from which worker threads are allocated is either implemented in a static or dynamic manner. Static thread pool has the advantage that it has no overhead of costly spawning of new worker threads but in situations involving less number of connections, Static fixed pool of thread results in under-utilization of thread pool. Dynamically created worker threads spawns new threads based on the given workload and thus it helps scaling a server which results in better performance. Also context switching is used to interleave simultaneous execution of worker threads. However, there are a number of disadvantages to this architecture. A multi threaded web server normally handles hundreds or even thousands of concurrent connections. This results in large overhead for context switching and undue overhead due to the creation of a large memory stacks. The thread-library used by a web server also plays a significant role in the overall performance of a web server. Thread-model implemented by a web server can either use 1:1 model, where each user-level thread has its corresponding kernel thread

running inside the kernel or M: N threading model where multiple user threads are multiplexed over few kernel threads. Thread library that comes with Linux is called NPTL threads library [16] which uses 1:1 threading model. In this model, the kernel threads are allowed to wait on blocking system calls without hampering the execution of other user-level threads. The server in this case can thus overlap the execution with I/O and this model can thus be extended over multiple CPU's. The only disadvantage of 1:1 threading model is that it are not able to scale well when there are thousands of connections involved [17]. In order achieve higher scalability for thousands of threads, M: N threading model is normally used. In this model multiple user threads are multiplexed over few kernel threads and blocking calls are avoided by using wrapper functions over such system calls that make equivalent non -blocking system calls. An example of a web server based on such architecture is Knot [17].

2.2 Event-Driven Web Server Architecture

In an event driven web server architecture, one main thread is responsible for accepting new client connections and registering socket channel in a channel selector. This architecture has the advantage of eliminating the overhead of blocking I/O operations used by multithreaded servers, thus reducing the idle times incurred by worker threads. The worker threads are called only when data is available on the socket. Once the request is processed, the worker threads are freed from the channel selector and are assigned to new client requests. Using this architecture, large numbers of active clients can be connected to the server without the threads getting blocked. As there is no limit on number of active client connections, an admission control policy to limit the number of incoming connections is required. This architecture has been implemented as Flash [4], µserver web server [9] and Zeus [8].

3. MODEL OF EXPERIMENTAL HYBRID WEB SERVER

In this section, a hybrid architectural model of a web server is presented. This model takes advantage of features of both multithreaded and event-driven architecture. In the model presented, as shown in Figure 1, the incoming client connections are passed on to a module called *mod_thread_selector* in an event-driven fashion. Once a request is received, *mod_thread_selector* module chooses one worker thread from the pool of worker threads and assigns it the task to process this request. This request is then processed by the assigned worker thread in a multithreaded fashion. In this way, this hybrid model receives requests in an event-driven manner and processes them in a multi-threaded fashion.

3.1 Hybrid Implementation

To validate this hybrid architectural model, a new java-based experimental web server is implemented. In this implementation, a module called *mod_thread_selector* is responsible for the task of handling incoming client connections. Once a request is received, this module chooses one worker thread from a pool of worker threads already spawned and assigns it the task of processing each request. Each worker thread then processes the assigned request in a multi-threaded fashion till its completion. Once a request is completed and response sent, the *mod_thread_selector* module is invoked again to process clients sending new requests. This implementation divides the web server logic into various modules. For workload characterization, this web server processes static as well as dynamic workloads. For dynamic workloads, support for CGI files is implemented. As the objective of this work is to validate the hybrid architecture proposed, the support for dynamic file extensions like *.ASP* and *.PHP* is not included in this implementation. This web server implementation contains a configuration class file called *HTTP. class* with comprises of all the constants like URL of various directories, *HTTP* Port number in use, *SERVER_LOCATION* and various response *STATUS* flags. A variable *CONN_TMT* is used to provide the connection timeout period that enforces a connection timeout period. The only disadvantage of this implementation is the absence of a proper admission control module required to solve the problem that occurs when the number of simultaneous connections increases many folds as a result of which there is a considerable decrease in the overall performance.

4. EXPERIMENTAL METHODOLOGY

The test-bed setup for the experiments is depicted in Fig. 2. It consists of two client machines connected to a server machine via a switched Giga bit network. The client machines are running Scientific Linux CERN 5 (2.6.18). Each machine has a single 2.0 GHz Intel Pentium I3 processor with 1 GB of RAM and uses "RAM-disk" of 128 MB size for collecting measurement statistics [18]. The server machine in our test environment is Intel Pentium I5 machine, with 4 GB of RAM, running Scientific Linux CERN 5 (2.6.18).

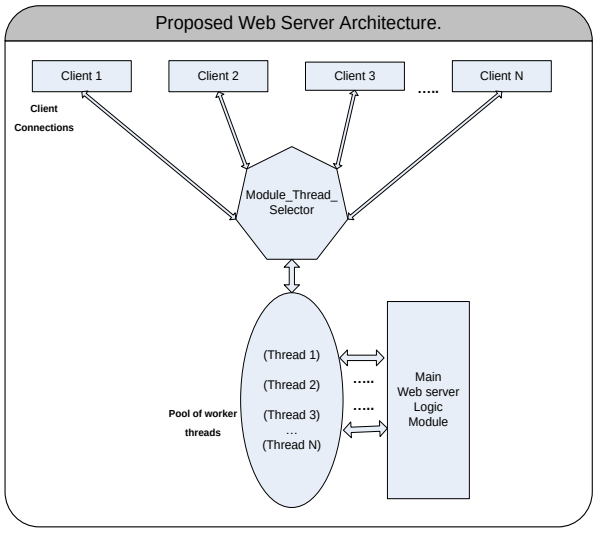

Fig 1: Proposed model of a hybrid experimental web-server.

4.1 Performance Tuning

The number of available file descriptors is increased from 1024 to 32,678 and the limit of the local port range is also increased. TCP *TIME_WAIT* recycling is enabled to free up sockets in a *TIME_WAIT* state more quickly [18], thus allowing clients to generate and sustain high request rate. Also, all the non-essential processes and services on the server as well as client machines are disabled. Also all the web servers are restarted before and after each experiment.

4.2 Client workload generator

The *httperf* [19] is an open source benchmark developed at Hewlett-Packard Research Labs. The *httperf* benchmark is a flexible HTTP client that requests a file from a web server multiple times and for number of parallel threads and then prints out detailed statistics. Its source code is modified in order to print the server response rate information more frequently. Thus the output of the *httperf* provides information about TCP connection rate; HTTP request rate and HTTP reply rates after every one second during the experiments.

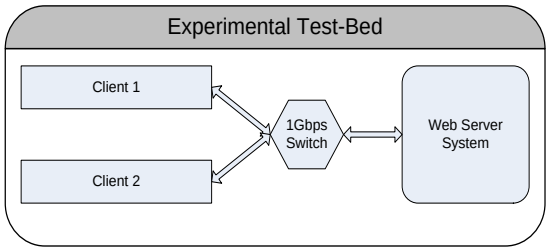

Fig 2: Experimental test-bed.

4.3 Web servers evaluated

The experiments carried out in this paper compare the performance of our experimental hybrid java based web server with multithreaded *Apache* 2.2 and event-driven *µserver*. These web servers were configured and tuned to optimize their performance. Arlitt [20] and Grottke et al. [21] in their work suggested that the two configuration parameters for Apache web server, *MaxRequestsPerChild* and *MaxClients* should be set to 0 and 250 respectively. Based on this insight, in the main experiments the Apache web server is tuned by setting *MaxClients* and *MaxRequestsPerChild* to these values. In addition to that, a kernel-based *TUX* web server has been used in the preliminary experiments to validate the request generation capabilities of the test-bed.

4.4 Validation of the Test Environment

In order to validate the request generation capabilities of the test-bed, *TUX* web server [22] is used. The purpose of this validation is to show that clients can generate and sustain high amount of requests rates during the experiments, possibly large enough to saturate the web server, and thus are not a bottleneck. These tests are performed for a 1KB static file. Figure 3 shows the result for this experiment. This figure has three sets of data plotted which includes average number of TCP connections, average rate of HTTP requests and the average number of HTTP responses. The results depict that clients are able to generate and sustain workload of 10,000 requests per second for a static 1kb file. Also the server platform and the network could support up to 10,000 responses per second for a static 1 KB file. Thus, the achieved response rates lower than these in the main experiments would indicate a bottleneck related to the particular server software technology being tested.

5. EXPERIMENTAL RESULTS

In this section, a comparison is made among our experimental hybrid web server, multi-threaded Apache and event-driven µserver for both static and dynamic workloads. The metrics chosen include TCP connection rate, HTTP request rate, HTTP reply rate, HTTP reply time. Experiments are designed for a static plain workload and dynamic workloads comprising of scripts written in Perl CGI. The workloads chosen in this set of experiments consist of plain static workload files. The results, as shown in Figure 4 depict that all the three servers' exhibit same performance up to the target request rate of 1500 reqs/sec after which there is a sudden decrease in the performance of multithreaded apache web server.

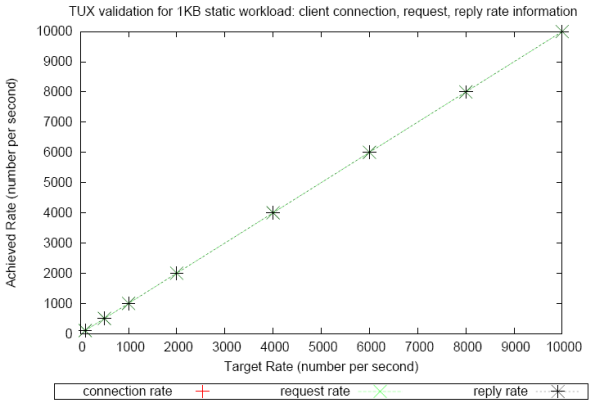

Fig 3: Validation experiment for test-bed involving TUX web-server

5.1 Results of experiments involving static workloads

After attaining saturation level Apache web server exhibited lower performance than the other two web servers. The performance of µserver and the experimental hybrid web server is approximately equal. The recorded saturation point of µserver, experimental hybrid, and Apache web server is 1711 reqs/sec, 1723 reqs/sec and 1619 reqs/sec respectively which are attained at the target request rate of 2000 reqs/sec.

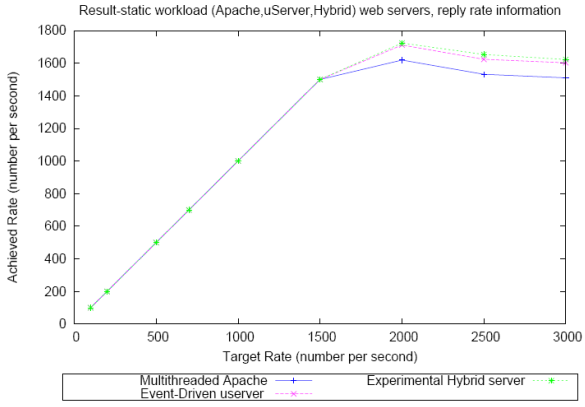

Fig 4: Result of the experiments involving static workloads using various web servers

5.2 Results of experiments involving dynamic workloads

This section presents results of the experiments carried out using Apache, µserver and our experimental hybrid web server involving dynamic workloads. The results, as shown in Figure 5 depict that the performance difference between these three servers increases at the target request rate of 500 reqs/sec. In these experiment, Apache web server exhibited lower performance than other two web servers and attained saturation level at the target request rate of 2000 reqs/sec. The performance of our experimental hybrid web server is better than multithreaded Apache and slightly better than event-

driven µserver. As depicted in Figure 5, µserver and experimental hybrid web server exhibits steady and a stable performance compared to multithreaded Apache with experimental hybrid server leading in terms of achieved response rate. The performance of the experimental hybrid web server slightly exceeds to that of µserver. At the target request rate of 2500 and 3000 reqs/sec, it is found that neither µserver nor the experimental hybrid web server saturates although there is some variations in the resulting achieved response rate.

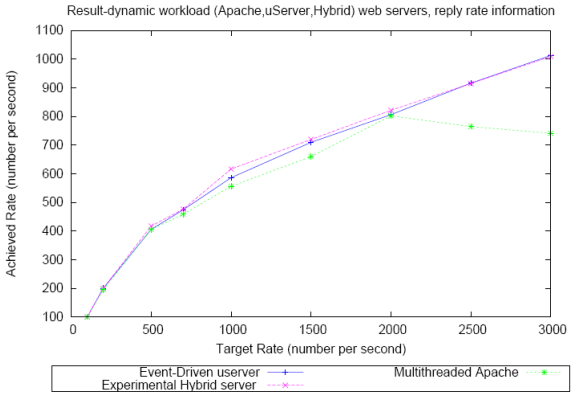

Fig 5: Result of the experiments involving dynamic workloads using various web servers.

6. CONCLUSION AND FUTURE WORK

This paper describes the working of multithreaded and event-driven web server architectures, highlights their advantages and disadvantages. Objective of this paper is to present a model of hybrid web server architecture. Based on this architecture an experimental java-based web server is developed. This paper then evaluates the performance of this web server with multi-threaded Apache and event-driven µserver. The workloads chosen include both static and dynamic files. The result of the experiments revealed that the experimental hybrid web server exhibits better performance than multi-threaded Apache and event-driven µserver for static workloads. For dynamic workloads, experimental hybrid web server showed substantial performance improvements as compared to multithreaded Apache and exhibited slightly better performance as that of the event-driven µserver web server. The goal of this evaluation is not to establish the experimental web server as better than other web servers, but only to validate the model proposed. The architecture thus presented can thus be used as reference model. Also, a proper admission control module can be included in this model. For further performance evaluation studies, web server based on the model proposed can be tested for scalability studies with respect to their horizontal and vertical characteristics.

7. ACKNOWLEDGMENTS

Authors are thankful to the Head, Department of computer Science & IT, and University of Jammu for his kind support.

8. REFERENCES

[1] S.M. Aaqib, L. Sharma, **"**Analysis of Compute Vs Retrieve Intensive Web Applications and Its Impact on the Performance of a Web Server", International. Journal Advanced Networking and Applications Vol. 03 Issue 04, pp. 1233- 1239, 2012.

[2] L. A. Adamic, B. A. Huberman, "Zipf's law and the Internet", Glottometrics,Vol. 3 pp. 143-150. 2002.

[3] Beltran, V., Carrera, D., Torres, J. and Ayguadé, E. (2004) 'Evaluating the scalability of java event-driven web servers', International Conference on Parallel Processing (ICPP'04) , pp.134–142. 2004

[4] V. S. Pai, P. Druschel, and W. Zwaenepoel. "Flash: An efficient and portable Web server", In Proceedings of the USENIX Annual Technical Conference, 1999.

[5] Apache software foundation. The Apache web server. URL :http://httpd.apache.org.

[6] Microsoft IIS web server, URL:- http://www.iis.net/

[7] Tomcat Apache, URL:- http://tomcat.apache.org/

[8] Zeus server. URL: - http://www.zeus.com/products/zws.

[9] The µserver home page. HP Labs, 2005. URL:- http://www.hpl.hp.com /research/linux/userver/ 2005.

[10] M. Welsh, D. Culler, and E. Brewer. "SEDA: An architecture for well-conditioned, scalable Internet services", In Proceedings of the eighteenth ACM symposium on Operating systems principles, pp 230–243, New York, NY, USA, 2001.

[11] Adya, J. Howell, M. Theimer, W. J. Bolosky, and J. R. Douceur. "Cooperative task management without manual stack management", In the Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference, pp.289–302,Berkeley, CA, USA, 2002.

[12] F. Dabek, N. Zeldovich, F. Kaashoek, D. Mazi`eres, and R. Morris. "Event-driven programming for robust software", In Proceedings of the 10th ACM SIGOPS European Workshop, pp 186–189, New York, NY, USA, 2002.

[13] H.C. Lauer and R. M. Needham. "On the duality of operating systems structures", In the Proceedings of the 2nd International Symposium on Operating Systems, IRIA, October 1978.

[14] J. K. Ousterhout. "Why threads are a bad idea (for most purposes)", Presentation given at the 1996 USENIX Annual Technical Conference 1996.

[15] R. Behren., J. Condit, and E. Brewer. "Why events are a bad idea for high- concurrency servers", In 9th Workshop on Hot Topics in Operating Systems (HotOS IX), 2003.

[16] U. Drepper and I. Molnar. The native POSIX threads library for Linux, URL:http://people.redhat.com /drepper/nptl-design.pdf.

[17] R. Behren, J. Condit, F. Zhou, G. C. Necula, and E.Brewer, "Capriccio:Scalable threads for Internet services", In Proceedings of Nineteenth ACM Symposium on Operating Systems Principles, pp. 268–281, New York, USA, 2003.

[18] S.M. Aaqib and L. Sharma, "Analysis of Delivery of Web Contents for Kernel-mode and User-mode Web Servers", International Journal of Computer Applications Vol 12, Issue 9 , pp 37–42, Published by Foundation of Computer Science. January 2011.

[19] Mosberger D, Jin T."httperf: A Tool for Measuring Web server performance." The First Workshop on Internet Server Performance, pp.59-67 Madison, WI, 1998.

[20] Grottke M., Li L., Vaidyanathan K., and Trivedi K.S."Analysis of software aging in a web server". IEEE Transactions on Reliability, Volume 55, Issue 3, pp.411-420, 2006.

[21] Arlitt M., Williamson C., "Understanding Web server configuration issues". Software: Practice and Experience. Volume 34, Issue 2, pp. 163–186, 2004.

[22] Lever C., Eriksen M., and Molloy S. "An analysis of the TUX web server". Technical report, pp.00-8 University of Michigan, 2000.