VNU Journal of Science: Comp. Science & Com. Eng., Vol. 31, No. 3 (2017) 28–34

Comparative Performance Evaluation of Web Servers

Van Nam Nguyen*

Department of Networking and Computer Communications,
Faculty of Information Technology,
VNU University of Engineering and Technology,
E3 Building, 144 Xuan Thuy Street, Cau Giay, Hanoi, Vietnam

Abstract

This paper surveys the fundamental software designs of high performance web servers that are widely used today for communication networks. Given the fast growing demands of the Internet and wide-ranging intranets, it is essential to fully and deeply understand which factors that mainly affect the web servers and how these systems can be designed to satisfy a maximum number of requests per second from multiple users. The paper presents empirical analysis of several popular web servers including Apache, NodeJS and NginX to precisely figure out the trade-off between different software designs to tackle the performance bottleneck and resource consuming problems.

Keywords: Web server performance, performance, bottleneck, load balancing

1. Introduction

Most websites work under the support of web servers that usually include both hardware and software. The hardware resource of a web server consists of CPU, RAM, disks and Internet communication line. The software architecture of a web server provides a web server service to conduct how a user access to the data files stored in the web server through HTTP protocol. Popular web server services are Apache, NodeJS and NginX. Much more than that, web servers also support application services such as PHP, Python, Java, database management services like MySQL, PostgreSQL, security services such as iptables, ufw and many other necessary services.

As the number of Internet user rapidly increases recently, the web servers need to serve millions requests per second from multiple users. This requires web servers to be upgraded in both hardware platform and software architecture. Hardware upgrade is costly. Meanwhile, software architecture of a web server faces two main challenges: performance bottleneck and resource consuming. In the former case, either the request handler of a web server service does not catch up the request arrivals, or the clients do not received as fast as possible the responses from web servers. If buffering and caching are used, these may consume large memory. If multiple request handlers are used, this may requires more CPU for managing handlers.

Recently, three preferable architectures of web server service are processed-based, thread-based and event-based. The first two architectures rely on concurrent threads and/or callback processes. Usually, the I/O such as socket or memory is blocked during request handling by a thread/process. Therefore, the resource consuming, in this case, is linearly increased according to the increasing number of threads/processes. Moreover, a context switching can be used to manage the processes and this may result in more CPU time waste. The third is, reversely, a nonlinear architecture which is

* Corresponding author. Email: van.nam@vnu.edu.vn

based on event handling. Events may be emitted from sockets as, for instance, a new connection is established, from the I/O notifications, or even inside the event handlers. This design is much more flexible than the others with non-blocking I/O and asynchronous request handling. However, the event-based implementation is clearly more complicated than the process-based and thread-based. This paper aims to precisely point out the trade-off of the above server architectures using empirical analysis from three widely used web server service Apache, NodeJS and NginX.

The rest of the paper is organized as follows:the second section introduces three fundamental server architectures. The third section presents the popular web server implementations including Apache, NodeJS and NginX. In the fourth section, we figure out the performance evaluation and analysis. The fifth section presents additional discussions. The six section summarizes the related works. Finally, the last section concludes our works.

2. Fundamental Server Service Designs

Currently, concurrency strategies used in web servers relies on process-based, thread-based and event-based architectures [1, 2].

2.1. Process-based Architecture

In process-based architecture (Fig.**??**), a main server process creates copies of itself to child process (preforking). The child processes handle the requests independently. This strategy ensure the stability of the system in such a case that one destroyed process does not affect the other ones. However, each child process occupies a separate memory space. The memory usage, in this case, is linearly proportional to the number of child processes. Moreover, the operating system needs to spend additional CPU time for switching between the child processes (context switching). The inter-process communication is also difficult due to the separation of child processes.

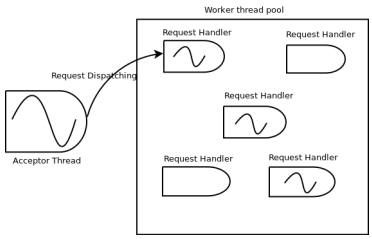

Fig. 1: Thread-based Architecture

2.2. Thread-based Architecture

In thread-based architecture (Fig.1), a dispatcher thread queues new connections of a socket. Request handler threads execute the connections received from the dispatcher queue. Since the thread pool contains the maximum number of threads, certain handlers may be idle and other's busy in case of not enough connections available in the queue. The threads can share memory, I/O operations and socket cause they are part of the same process. Thus, there is no context switching in this case. However, bad interactions on the same data and/or variables between threads may cause the whole system down.

2.3. Event-based Architecture

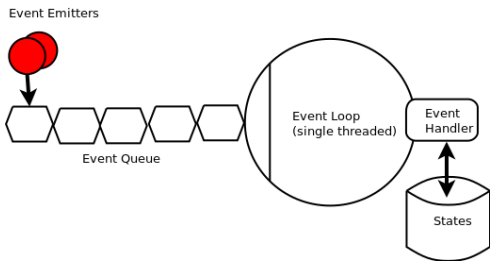

Fig. 2: Event-based Architecture

In event-driven architecture, a single process single threaded event loop executes multiple connections using non-blocking I/O mechanisms. Events are generated in different sources including the socket, the I/O operations or inside the event handler as there is a significant change in data or states. The event handler does not poll the event sources. Instead, events are pushed to the event loop as notifications. The event-based architecture can dis-burden the performance

bottleneck in case of high connection load thanks to its asynchronous communication. However, it is unable to carry out the non-blocking I/O operations in single process single threaded event handler.

3. High Performance Web Server Design

Today, most of high performance web server designs implement hybrid architectures. Apache provides flexible process-based and thread-based architecture. NodeJS successfully integrates event-based and thread-based. Meanwhile NginX is a powerful multi-process and event-driven architecture. In 2017, Apache and NginX account for more than 84% of the web server market share. NodeJS ranks at the seventh in this list but this is raising thanks to increasing real time applications.

3.1. Apache Web Server

Apache [3] is a free and open source Unix-based web server developed by Apache Software Foundation. In 2017, Apache accounts for more than 48% of the web server market share. Apache is lightweight, fully featured and more powerful than other Unix-based web servers. Apache's design is thread-based where a main process (Multi-Processing Modules-MPM) is called at start-up and preforks child processes/threads (module) to concurrently handle requests. Apache can act as multi-threaded, multi-process model or both and this can be specified by MPM. httpd is the core module in Apache which implements HTTP request/response processing.

3.2. NodeJS

NodeJS [4] is a single-threaded server-side JavaScript environment developed by Ryan Dahl in 2009. NodeJS can act as a web server service thanks to its built-in library for HTTP communication. With scalable event-based architecture, NodeJS can achieve high concurrency level. Although, NodeJS also uses multiple threads for non blocking I/O operations. Lots of main modules in NodeJS are written in Javascript and executed by Google V8 Javascript Engine. Thus, the whole codepath of NodeJS is asynchronous and non-blocking. This is why NodeJS is applied in most of real time applications.

3.3. NginX

NginX [5] is a free, open source and high performance web server developed by Igor Sysoev and then by NginX Inc. in 2004. In 2017, NginX ranks at the second (36%) of web server popularity after Apache. NginX uses also multi-process to efficiently profit available hardware resources. A master process is called at the startup along with cache loader to load disk-based cache in to memory and cache manager to control memory usage. This design aims to reduce context switching affects in multi-processes architecture. Depending on the workload, the master process balances load to a number of worker processes for simultaneous request handling. All worker processes implement single threaded event loop with asynchronous communication and non-blocking I/O operations.

4. Performance Evaluation

We evaluate the performance of three web server services under two aspects: performance bottleneck and resource consuming. The first problem arises due to the high concurrency level of requests. The second is caused by high arrival rate.

4.1. Environnement

The experimentation is implemented in four servers: one for testing and the three others for web server services. The testing server consists of 12 2.6 GHz CPU cores and 4GB RAM. Each web server contains 4 CPU cores and 4GB RAM. The servers are connected through a Ethernet network with 118 GB/s downstream and 90GB/s upstream.

4.2. Workload

We aim to generate high traffic load in term of concurrency and intensity using Apache Bench and Tsung. While Apache Bench can generate up to 20,000 concurrent requests. We can also get 40,000 requests/s with Tsung. In our experimentation, Apache Bench is used to benchmark the concurrency level of the web servers. We gradually create increasing number of concurrent connections with Apache Bench and measure the throughput, response time and error ratio of the web servers. In the other side, Tsung generates six millions requests to evaluate the resource consummation of the servers. We also observe the throughput, response time and bandwidth of the servers.

4.3. Tuning

The web servers can be configured for better performance in different load environments. Apache can act as prefork, worker or event model. In the prefork model, the MPM prefork a configurable number of process to handle requests in parallel. Since the processes are independent, the model is appropriate to web sites with unrelated requests. The worker model is a hybrid thread-bassed and process-based architecture. In this configuration, a process can create many threads for request handling. This configuration is applicable for high concurrent requests. The third model provides a mechanism to manage the processes and threads for performance optimization using event notifications.

NginX can work either with one process and a single threaded event loop or with many processes. However, NodeJS runs only with an event loop. For fair comparison, in this paper, we evaluate Apache prefork, NodeJS and NginX with one process.

4.4. Experimental results

We conduct two separate experiments, one with Apache Bench and the other with Tsung.

4.4.1. Concurrency level

In the first experiment, we carry out 20 continuous tests for each servers. The number of concurrent requests increases from 200 to 4000, step 200 for Apache, from 400 to 8000, step 400 for NodeJS and from 1000 to 20000, step 1000 for NginX. We choose a timeout of 15s for the response time of the requests.

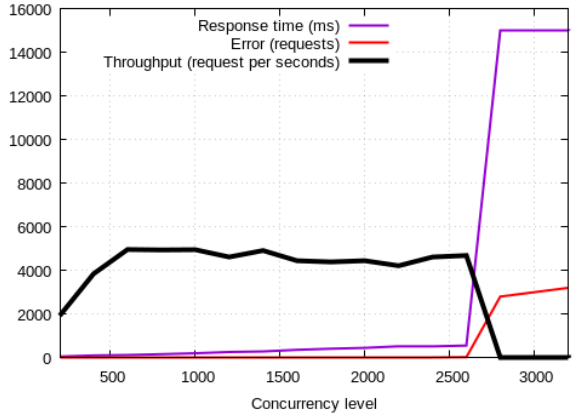

Fig. 3: Apache Concurrency Level

The Fig. 3 shows the evolution of throughput, response time and number of timeout requests for Apache with different concurrency level. We can see that at 2800 concurrent requests, the Apache server does not respond. At that time, the response time and the error ratio burst suddenly. The throughput is recorded at nearly 4100 requests per second.

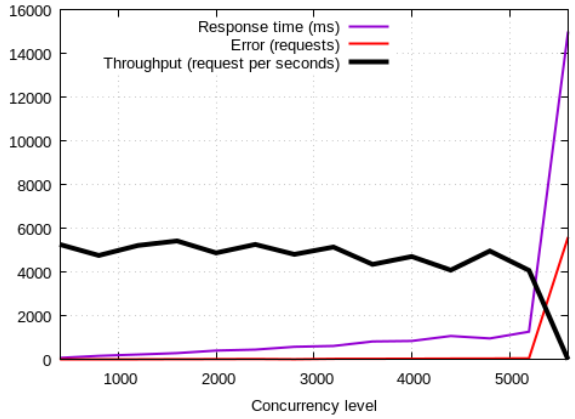

Fig. 4: NodeJS Concurrency Level

NodeJS, in this case, explodes better

performance than Apache. Effectively, at the concurrency level of higher than 5600 requests, NodeJS responds unstably but does not shutdown immediately. Its average throughput also reaches nearly 4300 requests per second (Fig. 4).

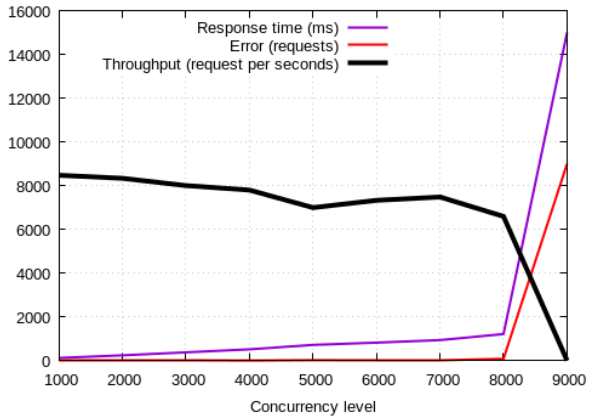

Fig. 5: NginX Concurrency Level

NginX achieves the best performance among the three tested servers. NginX can adapt stably to up to 8000 simultaneous requests. The server can unreliably serves up to the Apache concurrency level limit (20,000 requests). The average throughput over the experimental time is also above 7000 requests per second.

The above experimental results confirm that the event-based architecture used in NodeJS and NginX contributes for their higher concurrency levels. Effectively, in this architecture, the concurrent requests are asynchronously handled by only a single-threaded event-loop. Meanwhile, Apache prefork which is process-based is more vulnerable to the performance bottleneck. This is because as the number of concurrent requests gradually increases, more and more processes are forked in Apache. Each process is associated certain memory space and will spend certain CPU time while processing requests. Given a limited memory and CPU, the Apache server will not have enough memory space for handling and will go down immediately at a low level of concurrency.

4.4.2. Resource Usage Efficiency

In this experiment, we use Tsung to generate 6,000,000 requests to evaluate the resource usage efficiency of the servers. We also compare the evolution of the throughput, latency and the bandwidth of the web servers.

In Fig. 6, we show the evolution of average response time of Apache and NginX over the testing time along the total number of generated requests. We can see that average response time of NginX (about 0.44ms) is lower than a half of that in Apache server (approximately 1ms). This is even 100 times smaller than that of NodeJS (nearly 40ms). This implies that the request processing in NginX consumes less CPU time than in Apache and in NodeJS.

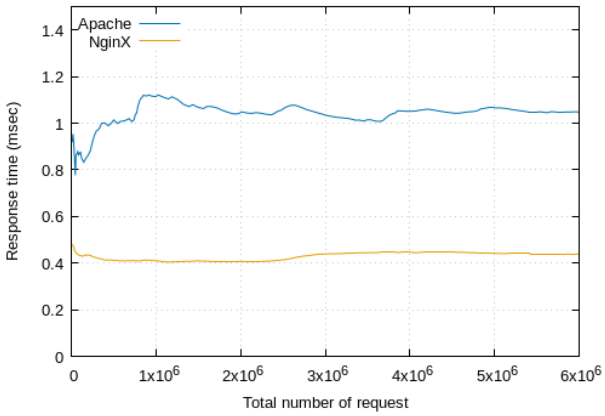

Fig. 6: Latency comparison

Moreover, over the time, NginX (Fig. 7) has always much higher throughput (about 4400 requests per second) than that of Apache (nearly 3800 requests per second) and of NodeJS (about 160 requests per second). This means that NginX has the most efficient memory management mechanism. In fact, NginX has cache loader which can allocate enough memory space to cache incoming requests and outgoing responses. Moreover, the cache manager in NginX is able to release the allocated memory in time as the requests have been processed or the responses have been transmitted to the clients.

Apache is based on MPM for memory management. MPM can fork an enough number of processes for request handling and kills the

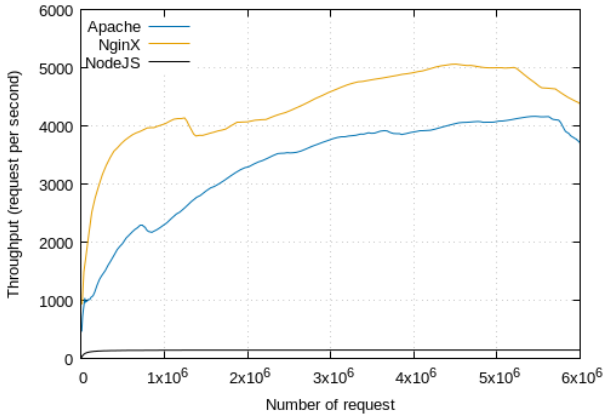

Fig. 7: Throughput comparison

unused process immediately. Note that each process is assign a memory space as forked. This mechanism, clearly, contributes to memory saving for Apache. Compared to NodeJS without any memory management mechanism, Apache uses the memory more efficiently.

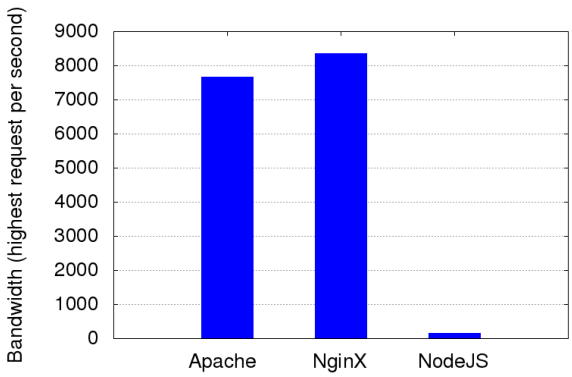

Fig. 8: Bandwidth comparison

NodeJS does not waste CPU time for context switching as in Apache prefork thanks to its event-based architecture. However, in such a high load, the context switching time is too small compared to the request processing time. While Apache can caches the requests and balances the request load through certain processes for request handling in parallel. NodeJS relies only on the event-handler which consumes as much as possible the CPU time for request processing. In other words, the event-handler in this case is overloaded. NginX has also an event-handler as in NodeJS. However, its caching mechanism can balance the load over the time before entering to the event-loop. Fig.8 shows the bandwidth comparison which highlights once more the resource usage efficiency of three web servers.

5. Discussion

The two above expriments reveal many different performance aspects of the fundamental server architecture as well as of the three hybrid designs Apache, NodeJS and NginX. Firstly, we find that event-based architecture can serve more concurrent requests than the process-based or thread-based one. However, in the long running at the low level of concurrency, the process-based and thread-based can process requests much faster than the event-based. Secondly, the hybrid design of process-based and event-based (as in NginX) can produce better performance than those between process-based and thread-based (as in Apache) or between event-based and thread-based (in NodeJS). This is because the event-based and process-based can complement the trade-off to each other. Thirdly, we find that a load balancer is necessary for mult-process and multi-thread architectures. Meanwhile, a scheduler makes the event-driven architecture to work better. NginX has both a load balancer and a scheduler while there is also a load balancing mechanism in Apache. Finally, a good memory management strategy contributes considerably to the web servers' performance (NginX over Apache and NodeJS). Note that the memory management can only be implemented using events.

6. Related Work

In literature, many works have been presented to evaluate the performance of web server designs and implementations. In [1], the authors compare the performance between event-driven, thread-based and pipeline-based architectures. They also find that the event-based and the pipeline-based can achieve up to 18% higher throughput than the thread-based. However, they do not concern

about the concurrency level of the architectures. In [2], the author finds that NginX, Apache, Cherokee are the best web servers for dynamic web pages. Meanwhile, Lighttpd and Cherokee are the best for static contents. However, the paper does not aware of the testing with high volume of load.

7. Conclusions

In this paper, we survey the fundamental architectures and popular implementations of web servers. We classify the architectures of the web server service in to three categories process-based, thread-based and event-based. Then we analyze that popular web server implementations does not relies on only one architecture. Apache is hybrid design of process-based and thread-based. NodeJS combines event-based and thread-based. NginX is a mix of process-based and event-based.

We also evaluate the performance of web server implementations in term of performance bottle neck and resource usage efficiency. Two experiments have been conducted using Apache Bench and Tsung with very high load. The experimental results show that NginX can achieve up to 42% and up to 200% higher level of concurrency than NodeJS and Apache prefork, respectively. The results also reveal that NginX can get up to 16% higher throughput than Apache in the long run with high volume of load. The response time of NginX also two times smaller than that of Apache.

We conclude that the hybrid design of process-based and event-based can explode better performance than the others. Besides, the performance of the web servers is considerably improved with a good memory mechanism.

Acknowledgments

We thank to Dr. Nguyen Dai Tho, University of Engineering and Technologies, VNU Hanoi for his useful reviewing.

References

[1] D. Pariag, T. Brecht, A. Harji, P. Buhr, A. Shukla, D. R. Cheriton, Comparing the performance of web server architectures, SIGOPS Oper. Syst. Rev. 41 (3) (2007) 231–243. doi:10.1145/1272998.1273021.
URL http://doi.acm.org/10.1145/1272998.1273021

[2] A. Hidalgo Barea, Analysis and evaluation of high performance web servers.

[3] R. Scoular, R. R. Scoular, Apache: The Definitive Guide, Third Edition, 3rd Edition, O'Reilly Media, Inc., 2002.

[4] A. Low, J. Siu, I. Ho, G. Liu, Introduction to node.js, in: Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, CASCON '14, IBM Corp., Riverton, NJ, USA, 2014, pp. 283–284.
URL http://dl.acm.org/citation.cfm?id=2735522.2735554

[5] W. Reese, Nginx: The high-performance web server and reverse proxy, Linux J. 2008 (173).
URL http://dl.acm.org/citation.cfm?id=1412202.1412204