# Programming approaches for implementing web servers for static content

Hristo Nenov[1] and Sevdalin Todorov[2]

*Abstract* – **The paper describes the role of the web servers for static content, and what part of the overall concept of the internet they occupy. Different models and algorithms for implementation of servers are described, and their strengths and weaknesses are pointed.**

*Keywords* – **Servers for Static Content, Approach, Model, Thread, Process, I/O Operation.**

## I. INTRODUCTION

Popularity of Internet and widespread usage of Web requires timely improvement of the technologies and software used for Internet communications and servers. The Increasing number of users of WWW, as well as its penetration in more areas makes web servers critical component in this area. The enormous number of users whose requests must be processed increases the requirements for hardware and communication resources, which in turn increases operational costs. With the dynamic development of the network consumers are becoming more demanding of the server response time when processing their requests. Most of the resources that web servers serve are static – images, CSS, client-side JavaScript, audio, video etc.. There are few popular web servers with general purpose as well as many not so common. The main goals of most of them are flexibility, lots of functionality and cross-platform, thus the performance and system resources usage remain in the background.

## II. WEB SERVER IMPLEMENTATION APPROACHES

### *A. WEB PAGE CONTENT*

The information in the World Wide Web is contained in hypertext documents or information resources, known as web pages. Usually the web page format is HTML or XHTML and allows navigating to other web pages using hyperlinks. Web pages may include other resources like multimedia, CSS and JavaScript. Content that does need extra processing prior serving it to the user is referred to as static content, if the content needs processing it is referred to as dynamic content. Web servers use HTTP protocol to transmit the content. Often the Web server is used to provide information in other formats - most often these are images in the formats PNG, JPEG and GIF; XML documents, CSS and JavaScript files, etc.. so-called static content.

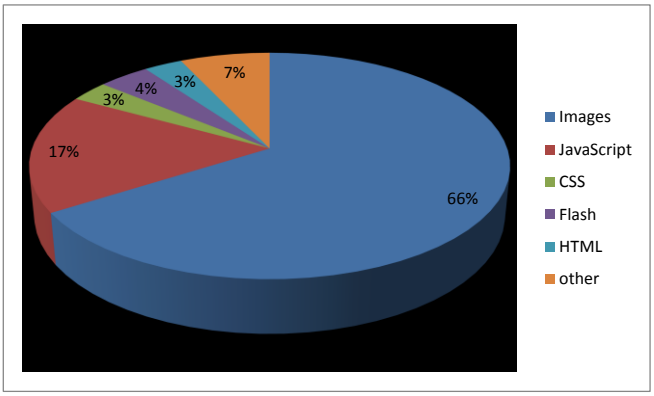


Fig.1 Average bytes per page for different type of context

Usually the Web server uses external programs to process the information before it is sent to the user using CGI scripts or application servers, however the processing module can be built-in into the server code. These scripts can be written in one of many programming languages, but the most commonly used languages are Java, PHP, Python, Ruby and Perl.

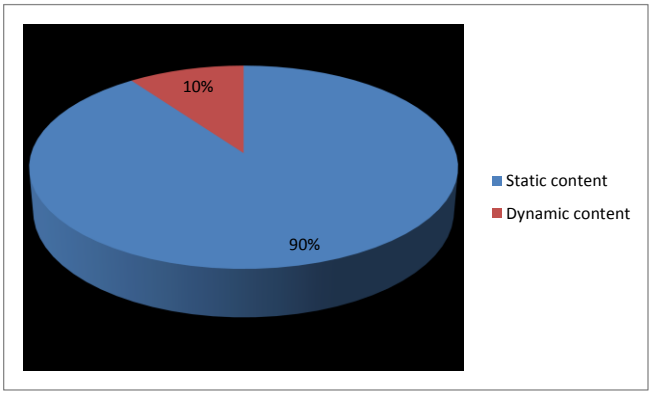


Fig.2 Average bytes per page for different type of content

### *B. LOADING SPEED*

In September 2009 - Akamai Technologies, Inc. published a key study conducted by Forrester Consulting, which examines eCommerce web site performance and its correlation with an online shopper's behavior. The most compelling results reveal that two seconds is the new threshold in terms of an average online shopper's expectation for a web page to load and 40 percent of shoppers will wait no more than three seconds before abandoning a retail or travel

[1]Hristo Nenov assist. prof. at Faculty of Automation and Computing at Technical University Varna, 1 Studentska str.Varna 9000, Bulgaria, E-mail: h.nenov@tu-varna.bg
[2]Sevdalin Todorov graduated bachelor engineer at Faculty of Automation and Computing at Technical University Varna, 1 Studentska str.Varna 9000, Bulgaria,
E-mail: dincho.todorov@gmail.com

site. Additional findings indicate that quick page loading is a key factor in a consumer's loyalty to an eCommerce site, especially for high spenders. 79 percent of online shoppers who experience a dissatisfying visit are less likely to buy from the same site again while 27 percent are less likely to buy from the same site's physical store, suggesting that the impact of a bad online experience will reach beyond the web and can result in lost store sales. In a similar study conducted in 2006, Akamai found customer expectations at four seconds or less. Considering both studies and the period of three years between them, it may be concluded that user expectations of web pages loading time were doubled. Given that, and the rapidly increasing mobile users and their significant requirements, because of the constraints of their environment (end of 2014) it can be concluded that expected loading time of web pages is less than one second. After internal research conducted in 2009 by Google, they also found that the loading times of web pages is of great matter, the greater is the delay - the less users use the service. On the one hand, from technical point of view many Internet publications argue that reducing the load time of pages dramatically reduces the load of the service equipment and therefore reduces operating costs by up to 50%. This is expected since theoretically faster request processing leads to faster resources freeing and available for use by other requests and programs, thereby increasing the ratio of efficiency of the equipment.

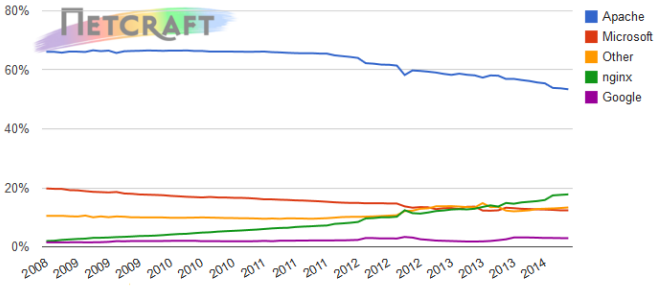


Fig.3 Web Server Survey

## C. APACHE SERVER APPROACH

Apache is among the most widely used web servers. According to NetCraft -- which provides Internet research services -- Apache web servers dispatch over 50% of the overall content -- static and dynamic -- on the web.

The Apache software foundation produces two types of web servers. Apache HTTP used for static content that can also be equipped with modules to serve dynamic content (e.g. PHP, Ruby), as well as Apache Tomcat which is a web-container (i.e. application server) used for serving dynamic content written in Java. The modules add additional functionality such as CGI, SSL, virtual hosts and processing of applications written in almost any programming language. Modules can be loaded dynamically without the need for recompilation of the web server.

The main modules through which Apache processes client requests are called modules for multi-processing. These modules are responsible for client requests processing and their distribution to the worker threads. These modules interact directly with the operating system by system calls.

*Multiprocess module*

This module creates a lot of child processes, but with one thread per each. Therefore, a process can handle only one request at a time. It is obvious that this architecture is not scalable. The module is mainly used for compatibility with older modules that are not thread safety.

*Multiprocess-multithreaded module*

This module creates a lot of child processes, and many threads for each of them. Therefore, each process can serve multiple requests. This is the best option of both modules because it is scalable, but resource consumption increases dramatically with increasing of concurrent requests.

*Advantages and disadvantages*

The advantages of Apache web server are its modular architecture and dynamic loading of its modules. Because Apache is developed and used so for many years, there are a lot build modules for it. Another advantage is its cross-platform characteristics.

The outdated and scale-dependent architecture that uses multiprocess-multithreaded processing of requests can be pointed as a major disadvantage. The support of large number of operating systems can also be specified as disadvantage because such a realization needs a lot additional source to achieve platform independence.

## D. NGINX SERVER APPROACH

Nginx is a web server, developed by Igor Sasoev. Initially used for sites with high traffic in his home country – Russia, Nginx has developed since then and is now one of the fourth most used web servers in the world according to Netcraft. Besides being able to handle static content, Nginx modules also support FastCGI - allowing processing and dynamically generated content. Nginx functionality is based on modules like Apache, however, unlike the Apache modules which can be built into the web server or loaded dynamically, Nginx modules can be added only at compile time. Nginx is designed with asynchronous event-driven architecture so you can use only one thread to processing many requests in concurrent. At high load, this architecture uses less and predictable amount of RAM- to process each request, compared to Apache model which uses "Multiprocess-multithreaded" oriented approach - depending on the selected module. Therefore Apache creates a new thread to handle new request - which requires additional memory. Nginx can handle new requests using existing threads, because of its architecture. Nginx also is cross-platform and supports many different operating systems.

*Advantages and disadvantages*

The biggest advantage of this web server is its architecture - asynchronous event-driven. A disadvantage is also the relatively high amount of source code needed to build: the

modular architecture, work with dynamic content and numerous of extra functionalities.

### *E.UNIX BASED OPERATING SYSTEMS*

UNIX based systems are dominant in the sector of web services. According W3Techs their share was 67.5% percent as of May 2014. Normally, the kernel and the application layers of these operating systems are open source, which contributes to their popularity, security and stability. Also, most of these systems are free, which is an essential factor in the choice of operating system, for such applications. UNIX operating systems are proven in the years of operation, superior performance and efficiency in the use of system resources. The most common of these is Linux, which is the reason that was chosen as a platform for research in this paper.

*Linux asynchronous input/output (I/O) model*

Linux asynchronous I/O is a relatively recent addition to the Linux kernel. It's a standard feature of the 2.6 kernel. The basic idea behind AIO is to allow a process to initiate a number of I/O operations without having to block or wait for any to complete. At some later time, or after being notified of I/O completion, the process can retrieve the results of the I/O.

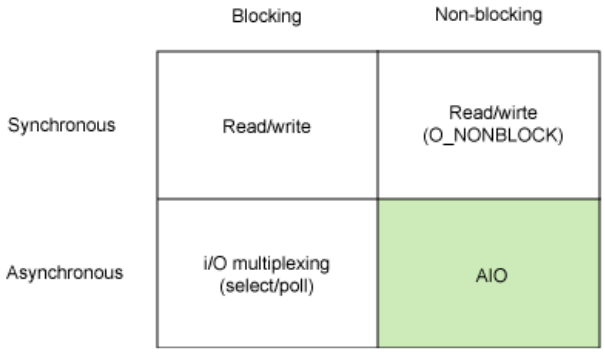


Fig.4 Simplified matrix of basic Linux I/O models

*Synchronous Blocking I/O*

One of the most common models is the synchronous blocking I/O model. In this model, the user-space application performs a system call that results in the application blocking. This means that the application blocks until the system call is complete (data transferred or error). The calling application is in a state where it consumes no CPU and simply awaits the response, so it is efficient from a processing perspective.

Figure 5 illustrates the traditional I/O blocking model, which is also the most common model used in applications today. Its behavior are well understood, and its usage is efficient for typical applications. When the read system call is invoked, the application blocks and the context switch to the kernel. The read is then initiated, and when the response returns (from the device from which you're reading), the data is moved to the user-space buffer. Then the application is unblocked (and the read call returns).

From the application's perspective, the read call spans a long duration. But, in fact, the application is actually blocked while the read is multiplexed with other work in the kernel.

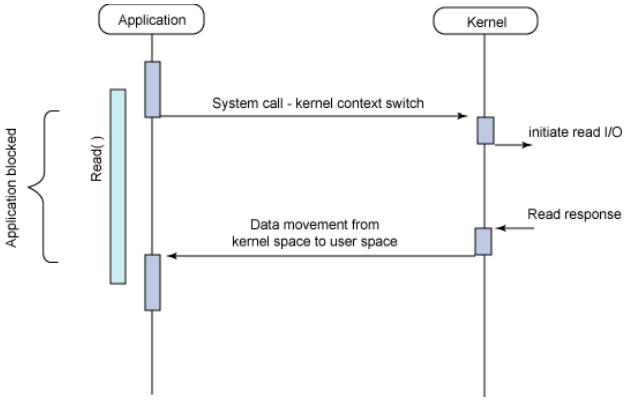


Fig.5 Typical flow of the synchronous blocking I/O model

*Synchronous non-blocking I/O*

A less efficient variant of synchronous blocking is synchronous non-blocking I/O. In this model, a device is opened as non-blocking. This means that instead of completing an I/O immediately, a read may return an error code indicating that the command could not be immediately satisfied (EAGAIN or EWOULDBLOCK), as shown in Figure 6.

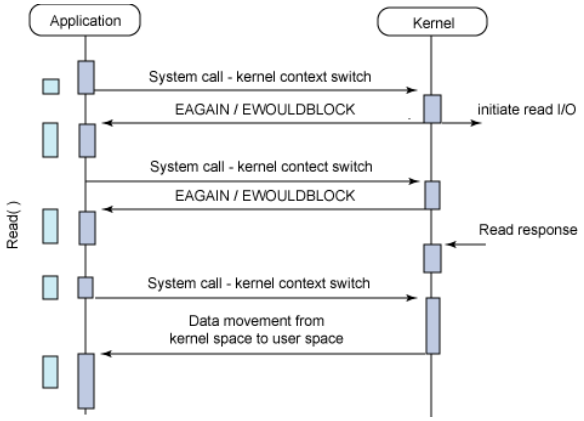


Fig.6 Typical flow of the synchronous non-blocking I/O model

The implication of non-blocking is that an I/O command may not be satisfied immediately, requiring the application to make numerous calls to await completion. This can be extremely inefficient because in many cases the application must busy-wait until the data is available or attempts to do other work while the command is performed in the kernel. As also shown in Figure 6, this method can introduce latency in the I/O because any gap between the data becoming available in the kernel and the user calling read to return it can reduce the overall data throughput.

*Asynchronous blocking I/O*

Another blocking paradigm is non-blocking I/O with blocking notifications. In this model, non-blocking I/O is configured, and then the blocking select system call is used to

determine when there's any activity for an I/O descriptor. What makes the selected call interesting is that it can be used to provide notification for not just one descriptor, but many. For each descriptor you can request notification of the descriptor's ability to write data, availability of read data, and also whether an error has occurred.

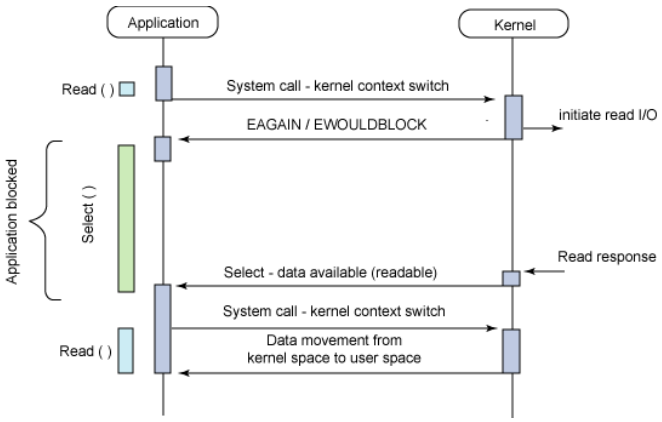


Fig7.Tipical flow of the asynchronous blocking I/O model (select)

The primary issue with the select call is that it's not very efficient. While it is a convenient model for asynchronous notification, its use for high-performance I/O is not advised.

*Asynchronous non-blocking I/O (AIO)*

Finally, the asynchronous non-blocking I/O model is one of the overlapping processing with I/O. The read request returns immediately, indicating that the read was successfully initiated. The application can then perform other processing while the background read operation completes. When the read response arrives, a signal or a thread-based callback can be generated to complete the I/O transaction.

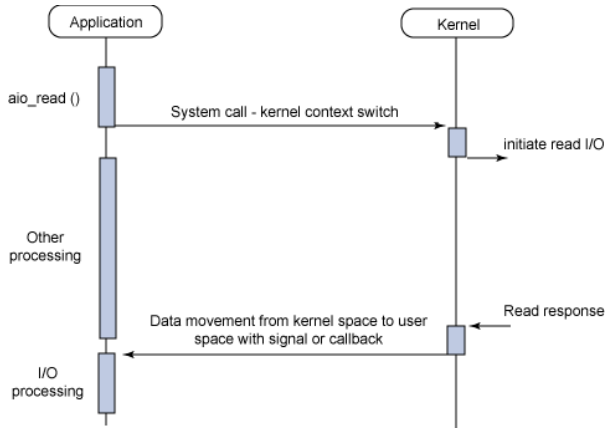


Fig.8 Typical flow of the asynchronous non-blocking I/O model

The ability to overlap computation and I/O processing in a single process for potentially multiple I/O requests exploits the gap between processing speed and I/O speed. While one or more slow I/O requests are pending, the CPU can perform other tasks or, more commonly, operate on already completed I/Os while other I/Os are initiated.

## III. Conclusion

In this article an overview of different programming approaches and principles for creating web server for static content is shown. From the presented facts can be made the following conclusions:

- the bottle neck in job of the web servers for static content is the performance of input output operations;
- choice of the proper architecture according to the specific role of the server significantly increases productivity and efficiency.

Future point in the study will be a real implementation of a web server by choosing some of the architectures described above.

## References

[1]. S. Todorov, “Design and Implementation of web server for static content”, diploma thesis 2014.
[2]. Л. Николов,. UNIX: Системно програмиране. - София: Сиела, 2009.
[3]. Л. Николов,. Операционни системи. - София: Сиела, 2012.
[4]. П. Наков, П. Добриков, Програмиране =
[5]. ++Алгоритми. - София: TopTeam Co, 2012.
[6]. Asynchronous I/O. Wikimedia, <http://en.wikipedia.org/wiki/Asynchronous_I/O> (15.09.2014)
[7]. eCommerce Web Site Performance Today. Akamai Technologies, Inc., <http://www.akamai.com/html/about/press/releases/2009/press_091409.html> (15.09.2014)
[8]. D. Rubio, “Web application performance and scalability.” <http://www.webforefront.com/performance/webservers_statictier.html> (15.09.2014)
[9]. J. Brutlag, “Speed Matters for Google Web Search.” Google, Inc., 22.06.2009, <http://googleresearch.blogspot.com/2009/06/speed-matters.html> (15.09.2014)
[10]. “Usage of operating systems for websites. W3Techs“ <http://w3techs.com/technologies/overview/operating_system/all> (15.09.2014)