Asynchronous non-blocking IO model approach to avoid the problem C10K in web servers for static content

Hristo Nenov [1] and Sevdalin Todorov[2]

Abstract – **The paper describes one of the main problems in processing of the web servers – C10K. A proposal for implementation of the server for static content is made and test benchmark and comparison results with chosen etalon are shown.**

Keywords – **static content, asynchronous non-blocking IO, event, event handler, pool, queue, thread, C10K.**

I. INTRODUCTION

The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Delivered pages are most frequently HTML documents which may include images, style sheets and scripts in addition to text content. A web server has defined load limits, because it can handle only a limited number of concurrent client connections (usually between 2 and 80,000, by default between 500 and 1,000) per IP address (and TCP port) and it can serve only a certain maximum number of requests per second depending on:

- its own settings;
- the HTTP request type;
- whether the content is static or dynamic;
- whether the content is cached;
- hardware and software limitations of the OS of the computer on which the web server runs.

When a web server is near to or over its limit, it becomes unresponsive.

II. *C10K* PROBLEM IN SERVER JOB

The name C10k is a numeronym for concurrently handling ten thousand connections. The C10K Problem refers to the inability of a server to scale beyond 10,000 connections or clients due to resource exhaustion. Servers that employ the thread-per-client model, for example, can be confounded when pooled threads spend too much time waiting for blocking operations-usually I/O. The native thread implementations on most OSes allocate about 1 MB of memory per thread for stack. As a result, blocking operations easily frustrate scalability by exhausting the server's memory with excessive allocations and by exhausting the server's CPU with excessive context-switching. For that reason the blocking operations should be avoided at all cost from servers that are likely to be challenged by the large number of customers at peak load.

The C10k problem is the problem of optimizing network sockets to handle a large number of clients at the same time. The problem of socket server optimization has been studied because a number of factors must be considered to allow a web server to support many clients. This can involve a combination of operating system constraints and web server software limitations. According to the scope of services to be made available and the capabilities of the operating system as well as hardware considerations such as multi-processing capabilities, a multi-threading model or a single threading model can be preferred. Concurrently with this aspect, which involves considerations regarding memory management (usually operating system related), strategies implied relate to the very diverse aspects of the I/O management.

III. DESIGN AND IMPLEMENTATION OF WEB SERVER FOR STATIC CONTENT (STATIX V.0.1.0)

For the implementation of the experimental server is chosen “Asynchronous non-blocking I/O” model. HTTP server listens for new connections from clients. During the accepting process of a new connection, the server reads the contents of the application submitted on the link and then makes its analysis according to the HTTP protocol. The resulting structure of the application is processed and according to it an appropriate response is generated. The response may be the require content or mistake. Finally the formatted response is sent to the appropriate client link and the connection is closed if it is not checked “keep-alive” options, otherwise the connection does not close, only resets the counters and buffers to it.

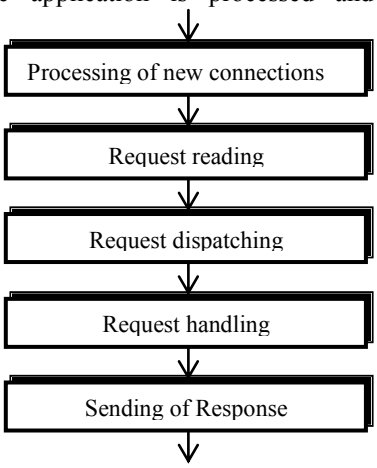

Fig.1 Generalized Work Scheme

Acceptation of new connections is processed only from one main thread. Once the connection is accepted a working

[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.comBulgaria.

thread is selected and read request in its event queue is registered. As addition, deletion, modification and retrieval of events are done through system calls (according to event-notification system) and those system calls are serialized i.e. they are thread safe and additional synchronization is not necessary. Each working thread has its own event queue.

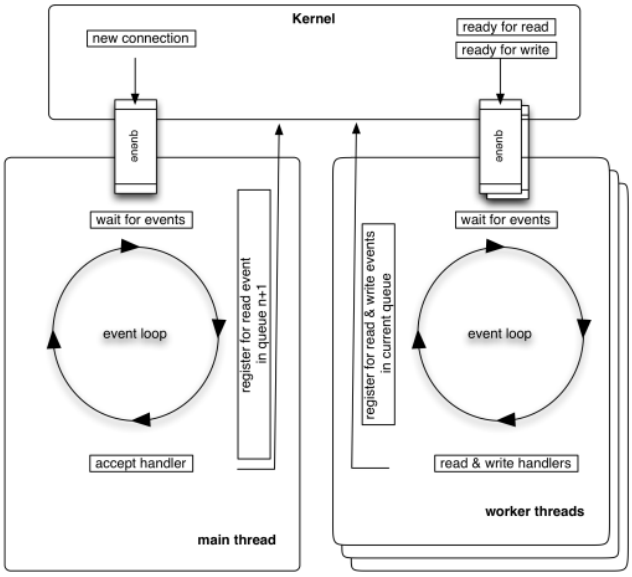

Fig.2 Generalized Work Architecture

A. Handling of new connections

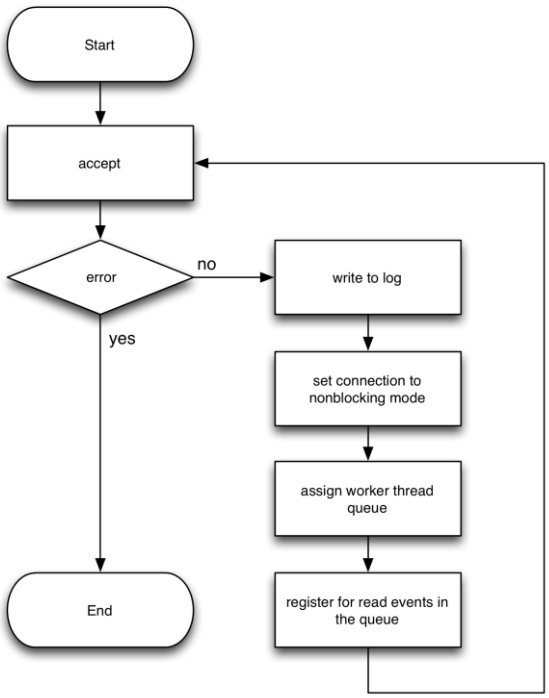

Fig.5 Processing of new connection

Upon the occurrence of new connection event, all pending connections are administering simultaneously as during the time in which the event is generated and the time in which it is processing new connections requests can be received. Once the connection is accepted, it is set appropriately, the event is recorded in the journal and event request for queue reading is registered in one of the working threads event queue. The thread chooses simple round principle, thus achieving balancing distribution of working load between threads.

B. Reading of request

Upon receipt of the event for reading the first thing is to be determined whether the connection is still active, if it is not then it closes itself. In the first event for reading to a certain connection is verified the capacity of working thread in which the processing is executed, and if it is reached then the connection closes itself, otherwise the application is initialized. The data reading from connection can be executed repeatedly until all data is read or an error occurs. The error may be due to too much data that cannot fit in the buffer request, a transmission error if the client closed the connection or if the result from the operation will be a process of blocking. By error of blocking process, a new request event for reading is registered and the processing ends. If the client closes the connection, the server processing ends. At large volume of data or other error in the response the respective code answer is included. Control is passed to the next stage - parsing and processing application.

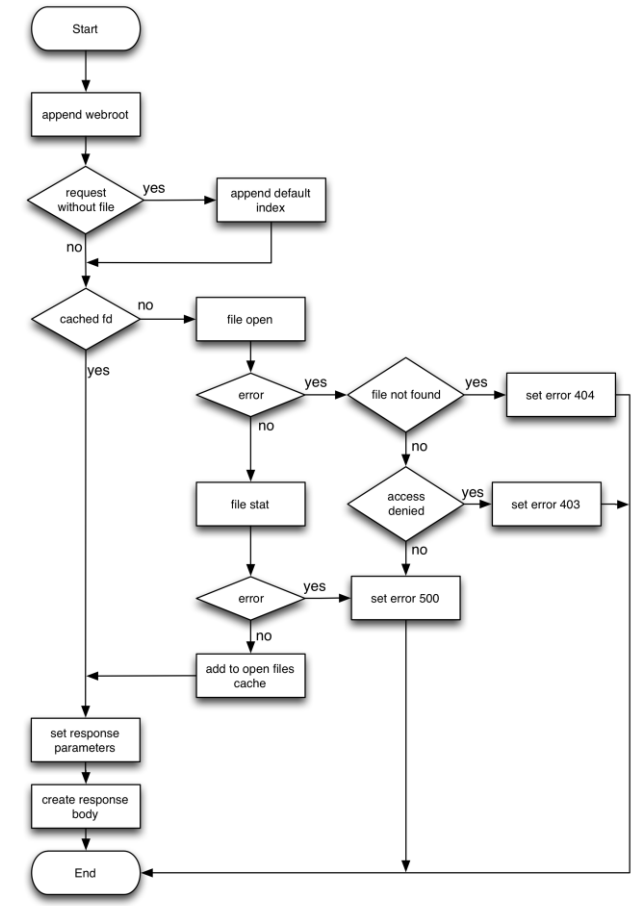

Fig.6 Request processing

C. Request handling

The Request handling consists in few steps:

- processing of the file request;
- determination of the appropriate header for different content type;
- generating of response.

In the time of file processing its absolute path on the file system server is generated. The check is if there is such a file and is it available for reading. For a high level of performance the cache of the open files is used. At successful open of the file, the respective status of the response and its size are set.

The content type of the response is determined by a static table based on the extension of the request. The construction of the response based on the formation of the text, depending on the status, type, size, and the subsequent closing of connection. If there is not an available opened file, the body of the response is formed by static table based on the status of the response. Once the request is processed, the response is send to the client.

IV. RESULTS

Tests were performed on a clean server especially designed for these tests. There are no unnecessary services installed and used otherwise. For comparatively analysis is used web server Nginx. Tool for the benchmark is Weighttp with Weighttp wrapper.

Technical part:

- HP ProLiant DL380p Gen8
- 2 x Intel(R) Xeon(R) CPU E5-2650 @ 2.00GHz
- cores: 16
- threads: 32
- RAM: 64GB DDR3 1600 MHz

Software environment:

- Ubuntu 14.04.1 LTS (64bit)
- Kernel: Linux ubuntu 3.13.0-35-generic
- Statix 0.1.0 (our project)
- Nginx 1.4.6 (etalon for benchmark)
- Weighttp 0.3
- Weighttp wrapper 5.10.7

TABLE I
STATIX V.0.1.0 RESULTS

Connections	Min	Average	Max	CPU % (usage)	CPU % (kernel)	Memory (MB)
1000	24464	24880	24967	51	264	40,83
2000	48378	49051	49317	41	258	41,11
3000	70766	70960	71334	61	236	48,45
4000	86460	90592	92163	52	248	53,57
5000	90668	102468	108517	52	245	62,62
6000	94759	114013	120446	47	270	62,78
7000	88030	115415	128845	50	251	68,53
8000	86079	127605	135004	34	277	68,63
9000	71328	107752	138751	57	262	68,73
10000	72418	114732	141538	54	265	68,75

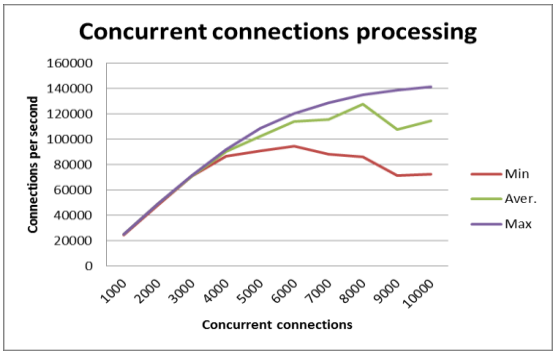

FIG.7 PROCESSING PERFORMANCE OF STATIX V0.1.0

TABLE II
NGINX 1.4.6 RESULTS

Connections	Min	Average	Max	CPU % (usage)	CPU % (kernel)	Memory (MB)
1000	23642	23786	23932	155	170	47,49
2000	22204	30646	36092	129	146	49,69
3000	28981	38987	66015	134	199	49,91
4000	27260	32057	40562	131	146	50,41
5000	13682	31937	51481	137	164	50,04
6000	14126	35815	65161	71	115	50,28
7000	13990	29367	32551	92	140	50,53
8000	13629	25508	32106	158	185	50,36
9000	13586	23564	31398	173	215	51,79
10000	13507	25480	31678	135	208	51,47

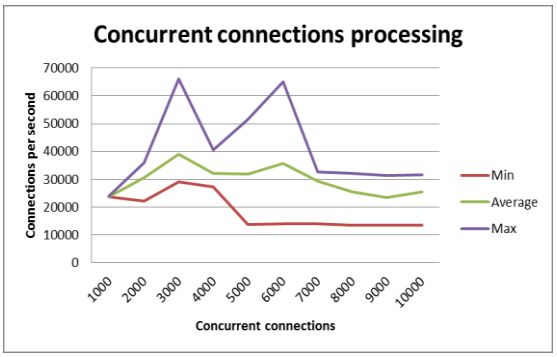

Fig.8 processing performance of Nginx 1.4.6

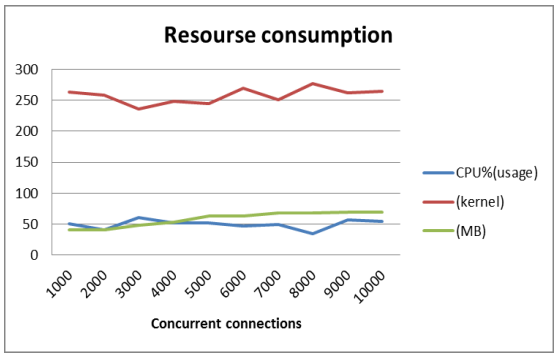

Fig.9 Resource consumption of Statix 0.1.0

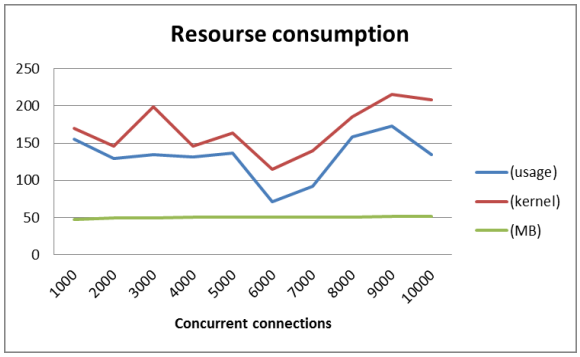

Fig.10 Resource consumption of Nginx 1.4.6

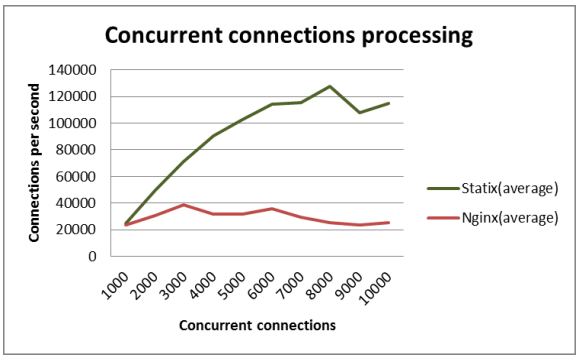

Fig.11 Comparison on request processing
Statix 0.1.0 - Nginx 1.4.6

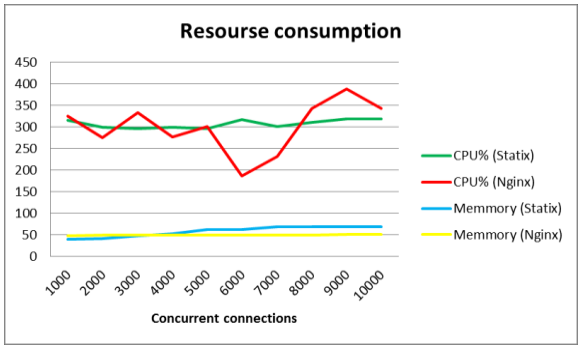

Fig.10 Comparison on resource usage
Statix 0.1.0 - Nginx 1.4.6

V. CONCLUSION

From the experiments made (shown in the preceding tables and graphs) the following can be concluded:

- at increasing of the number of client connections almost linearly increases the number of concurrent connections that are processed. At the target 10k connections, the server is doing extremely well. Moreover it reaches performance nearly five times better than the comparator etalon.
- the results are not at the expense of using a huge amount of resources – CPU, memory and etc.. Consumption of resources is in very good range. A good impression makes their stable behavior.

High productivity, efficient operation and predictability resource consumption are the results of the choice of scalable architecture for building Web server - “Asynchronous non-blocking I/O” model.

Experiments clearly show that this type of architecture is especially suitable for high load web servers.

REFERENCES

[1] S. Todorov,. Design and Implementation of web server for static content, diploma thesis 2014.

[2] D. Rubio,. Web application performance and scalability. <http://www.webforefront.com/performance/webservers_statictier.html> (15.09.2014)

[3] “Asynchronous I/O.” Wikimedia, 11.09.2014, <http://en.wikipedia.org/wiki/Asynchronous_I/O> (15.09.2014)

[4] T. Jones, “Boost application performance using asynchronous I/O. Emulex, 29.08.2006

[5] “Usage of operating systems for websites.” W3Techs, 15.09.2014, <http://w3techs.com/technologies/overview/operating_system/all> (15.09.2014)