Scalability of a Web Server

How does vertical scalability improve the performance of a server

Ancuta-Petronela Barzu, Mihai Carabas, Nicolae Tapus
University POLITEHNICA of Bucharest
Computer Science Faculty
Bucharest, Romania
ancuta.barzu@stud.acs.upb.ro, mihai.carabas@cs.pub.ro, nicolae.tapus@cs.pub.ro

***Abstract*—This paper presents how the performance of a server is influenced by the applying a vertical scalability. The paper studies the results obtained in measuring the response time of the server and the processing time of the server when dealing with a large number of requests by modifying the configuration of the machine, increasing the number of cores the machine has and increasing the RAM capacity. This represents a test to see how many requests a server can process and complete when dealing with large amount of data in a short period of time.**

Keywords—server; scalability; vertical scalability; performance; improvements; client; response time; processing time; CPU; cores; RAM; capacity; machine; monitor; performance improvement; virtual machine; component;

I. INTRODUCTION

With the expansion of the Internet and the rapid evolution of technologies, a current problem when it comes to servers is their capacity to scale for a large amount of data. Since nowadays Internet connectivity does not represent an issue, the response time of the server depends on the server itself, how it handles its data by processing it.

Nowadays a server needs to be capable of processing a large number of requests in a short period of time without introducing latency. It is known that a machine is capable of performing a number of tasks as its hardware allows it to. It is known that a machine with a better configuration can process more tasks than a machine with a weaker configuration, for example a machine with 2 cores is theoretically capable of processing the doubled number of tasks as a machine with 1 core.

This paper presents the problem of the scalability of a server when dealing with a large amount of data and the technologies used for this server. This paper also presents an experiment which was conducted in order to determine how vertical scalability affects the performance of the server. The results of this experiment are also described as well as the analysis of the obtained results.

The main objective of this paper is to obtain a better response time from the server, and an improvement in performance by conducting the vertical scalability experiment.

The first chapter presents a short introduction for this paper, describing the problem at hand. The second chapter presents similar experiments conducted when dealing with the scalability of a server. The second chapter also present the previous work that has been done for this paper. The third chapter presents the changes that have been brought to the server's implementation as opposed to previous work. The forth chapter describes the experimental environment in which this experiment was executed. The fifth chapter presents how the server reacted to the experiment and the results obtained while conducting it. The sixth chapter presents the conclusions of this paper. The seventh chapter presents a list of ideas that could improve the server's performance in the future.

II. RELATED WORK

Scalability refers to a term that defines the capacity of a system, a network or a process to handle a large amount of work. When talking about the scalability of a system, a set of requirements must be specified in order to be considered important. We can consider that a system is scalable if, by adding hardware to the system, it gains an improvement in its performance.

Vertical scaling refers to the process of enhancing the system by adding resources to a single node, or removing resources from a single node. In most cases, vertical scaling refers to the addition of CPUs or memory capacity to a single machine.

Article [1] describes a detailed analysis of an experiment by measuring the vertical scalability of a Web Server and by performing an analysis using a performance analysis framework to determine the server's behavior to the vertical scalability. Their analysis concluded that by increasing the number or processors of the server, the performance of the server will also increase.

In order to conduct this experiment, the authors used a Tomcat Servlet Container v5.0.19 [10] for the application server. To test the system, they have developed the authors used an Auction Site Benchmark, namely RUBiS [4] to monitor the server's activity and its behavior to various requests. In order to generate requests for the server, the authors used Httperf [5].

This configuration was deployed on a 4-way Intel XEON 1.4 GHz with 2 GB RAM running a 2.6.2 Linux kernel and connected to the client machine through a 1 Gbps Ethernet interface. They also included a separate machine to run a MySQL v4.0.18 [13] database which was directly connected to the server through a 100 Mbps Fast Ethernet crossed-link. For the server, the authors have used Sun JVM 1.4.2 setting the maximum Java heap size to 512 MB.

The results of these experiments are described in the following table.

TABLE I. NUMER OF CLIENTS THAT SATURATE THE SERVER AND MAXIMUM ACHIEVED THROUGHPUT BEFORE SATURATION

Number of processors	Number of clients	Throughput (replies/s)
1	250	90
2	500	172
4	950	279

Through this experiment, they have concluded that increasing the number ok processors, the server is able to handle more clients before saturation.

TABLE II. AVERAGE SERVER THROUGHPUT WHEN SATURATED

Number of processors	Throughput (replies/s)
1	25
2	50
4	90

The results obtained from Table II reveal that the server can obtain better throughput when increasing the number of processors even if the server has reached a saturated state.

When analyzing the cause of the server's saturation, the authors concluded that the processors represents a bottleneck for Tomcat performance in a secure environment. Thus, they concluded that increasing the number of processors has a positive effect on the server's performance.

Article [2] presents an analysis a large Web-based shopping system is affected by a workload generated in five days. Their analysis was conducted in order to measure the scalability. Through their analysis, they have discovered that horizontal scalability isn't always an adequate method to improve the server's performance when trying to support increased workloads. Another conclusion would be that personalization and robots may have a significant impact for the system's scalability. They have run this test for two periods of time, a period being March 2000 and a second period being July 2000.

While conducting their analysis on the two periods of time, the authors concluded that for the March period the traffic describes a typical time-of-day pattern, while for the July period it is revealed that the day of the week can affect the server's workload. The authors also concluded that, in certain periods of the year, the traffic volume has a significant increase. These periods are often before important holidays such as Valentine's Day, Christmas, Easter, Mother's Day or Father's Day.

Article [3] performs an experiment to observe the evolution of horizontal and vertical scalability of a Cluster-based application server. The authors have evaluated the scalability of a server and relevant performance metrics when improving the server horizontally, by adding more machines to the system, and vertically, by improving the already existing machines in the system.

Their experiment started with a minimal cluster, having 2 servers each with one core, and then they doubled and quadrupled the number or servers in the system as well as the number of processors for each server. Their results are described in the following tables.

TABLE III. RESULTS OF MEASURING THE CLUSTER'S PERFORMANCE

Cores/ Node	Nodes	
	2	*4*
1	123.55 (150 EB)	238.04 (280 EB)
2	236311 (280 EB)	278.83 (320 EB)
4	284.80 (320 EB)	284.80 (320 EB)

TABLE IV. SCALABILITY METRICS AND EXPECTED RESULTS

Cores/ Node	Nodes	
	2	*4*
1	1.00 (1)	1.93 (2)
2	1.91 (2)	2.26 (4)
4	2.31 (4)	2.31 (8)

From the above tables the authors concluded that, even though the server presents an improvement in performance, the actual results are somewhat far from the expected results, especially concerning horizontal scalability.

Their results also concluded that horizontal scalability and vertical scalability, when referring to small cluster-based application server, are practically the same when it comes to the performance of the server.

A. Previous Work

In previous work, described in the following paper [6], a basic server was tested in order to observe how it reacts to a large number of requests. The results of this experiment were that, for the existing configuration of the server, it did not scale for a large number of requests.

The server used the MEAN [14] stack in order to develop the server. The technologies belonging to the MEAN stack are as follows:

- Node.js [15] in order to develop the server-side application;

- Express.js [16] is the web application framework designed for Node.js to build web applications;
- MongoDB [17] is the database used for this application;
- AngularJS [18] is the JavaScript [19] framework used to develop the front-end web application.

The main objective of the server used for the experiment was to process a large amount of data and return the result to the user. The processing is done by sending a request to the server in order to process a large file.

The server was tested on a machine with the following configuration, an Intel Core i5-4210U processor, 1.7 GHz frequency, 8 GB RAM and a SSD hard of 256 GB. The operating system used is a 64-bit Windows 8.1 operation system.

The server was tested with a maximum of 100 simultaneous requests, resulting in an average of 11 seconds spent per each request, with the response time increasing with the growth of the number of requests to process.

In the following chapters the results obtained from performing a vertical scalability on the existing server will be presented, as well as the modifications made to the server's configuration, as well as the server application.

III. Server Changes

Taking into account previous tests, and the fact that Node.js does not permit creating threads natively, only by using various third-party modules, but that defeats the purpose of actually using Node.js, the server suffered a few modifications.

The biggest modification the server has suffered was the technologies used for the back-end. These were changed from Node.js to Java [20] for the back-end programming language, and from Express.js to Spring Boot [21] for the framework used to develop the server. By using Java as the back-end technology, threads can be created in order to process requests. The Spring Boot framework was chosen as it is the most popular framework when developing Java server-side applications.

The main core of the server, the basic server structure, was generated using jHipster [22], a tool used to generate web applications that use Java with Spring Boot as back-end technologies, MongoDB for databases and AngularJS as front-end technologies.

Another modification that was brought to the server was the fact that the files for generating tasks are located on the server side, the client only has to send the number of task needed and the operation for this task. This change was made in order to bypass the sending of a file from the client side, and it is being used only for test purposes.

The last modification made to the server is the fact that each request generates a thread in order to process the received task and immediately send a response to the client saying that the processing of the task has started. Thus, the server does the process in background and the client does not receive a Connection Timeout error.

IV. Experimental Environment

In order to test the vertical scalability of the web application an experiment was devised so that proper measures could be taken and generate reports based on the obtained results.

The testing environment consists of two machines, one machine used to deploy the server and the other machine used as a client in order to send multiple requests to the server.

A. Server

The server's main job was to run the web application developed, receive tasks and process them in order to obtain a better performance when dealing with a large amount of requests. This web application is described by the Java server generated using jHipster presented above.

Another important task for the server is running a small monitoring script in order to obtain valuable information regarding its physical components, such as CPU load and RAM usage. This script was made in order to analyze how the server is affected by the large number of requests.

In order to conduct the experiment of testing the vertical scalability of the server, these two applications had to run at the same time, the server in order to process requests, and the monitoring script in order to obtain information about the machine. In order to build the server, the project used Apache-Maven [23].

The server was deployed on a virtual machine. Using VirtualBox [24], a basic virtual machine was created with an Ubuntu [25] 16.04 64-bit image. This virtual machine had a basic configuration of 1 core with 4 GB RAM and a 50 GB hard-drive. The machine has installed all necessary packages: Java, MongoDB, Apache Maven and Node.js which was needed for the jHipster generator.

B. Client

The client, or the test application for this experiment, ran on a different machine than that of the server. The test application represents a Java program that can generate a large number or requests and send them to the server in order to process them.

In order to send a number of N requests simultaneously, the test application generated different threads for each request, so that each thread can handle its own task.

This application ran on a machine with a configuration of Intel Core i7-3520M processors, with 2 cores of 2.9 GHz frequency, 8 GB RAM capacity and an operating system of Windows 10 64-bit. The test application was ran using the IntelliJ [26] editor designed for building Java applications.

V. Experiments And Results

In order to test the vertical scalability of the server, a few experiments were conducted. In order to conduct these experiments a series of tests were executed on the same machine that suffered hardware improvements to test the vertical scalability.

The server ran on a virtual machine which was improved for each series of tests. The virtual machine's initial configuration was that of a 1 core with 4 GB RAM, 50 GB hard disk and an Ubuntu 16.04 operating system. For further tests this machine was improved by doubling the RAM capacity as well as the number of cores the machine has.

The experiments consisted of running 9 test suits, each with: 1 request sent, 10 requests sent, 25 requests sent, 50 requests sent, 100 requests sent, 250 requests sent, 500 requests sent, 1000 requests sent and 2500 requests sent.

A. *Experiments*

Each experiment was conducted by altering the initial configuration of the initial virtual machine, running the set of 9 tests and measuring how the machine reacted to these tests. The measurements that were taken into consideration were the average response time, the average processing time of a task, the total processing time of all tasks, the CPU load and how much RAM has been used while conducting the experiment on each machine.

The average response time was measured in the test client application, while the total processing time and average processing time were measured on the server. To see how the CPU and memory usage was affected, while the server was up, a monitoring script was running in parallel that gathered information of CPU load and memory usage in percentages, at every second.

In order to measure the performance of improved servers, a ration between the measurements obtained from the first experiment and the measurements obtained from improved configurations had been calculated to determine how well the server has improved. Each experiment was compared to the base experiment, the 1 Core with 4 GB RAM configuration. If the performance coefficient has a value over 1 then we can say that the server has obtained an improvement in performance.

The following subchapters describe the observations that were made while conducting the experiment for each configuration.

1) 1 Core with 4 GB RAM configuration

While testing the first configuration, what was observed was the fact that for the 500 requests test, for the first try, the server stopped processing requests after 490 requests. After running the test again, it did not fail offering a result. Another observation was made when running the 1000 requests test. This test was executed 3 times, each time the server stopped processing after a number of requests. The maximum number of requests the server managed to process was that of 824. Seeing as how after three trials the server couldn't manage to process al 1000 requests, no further tests were executed. The results are described in the following table.

TABLE V. THE MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 1 CORE AND 4 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.416	1.771	0.428
10	3.948	0.799	4.572
25	6.124	7.405	12.338
50	8.583	13.991	24.849
100	10.359	22.382	45.956
250	9.871	56.372	112.333
500	10.541	225.524	226.825

From the obtained measurements, we can conclude that the average processing time has a logarithmical growth, the total time to process all requests has a linear growth, while the average response time tends to have an exponential growth but after a number of requests this becomes linear.

While monitoring the server's activity in order to obtain data about the CPU load and RAM usage, it was observed that these characteristics increased with the number of requests, the more requests the server had to process resulted in high CPU load and RAM usage. The CPU load does not increase more that 45%, while the percentage of used RAM has a linear growth, but never exceeding 35%. The percentage of used RAM by the Java process almost has a constant value below 10%.

2) 1 Core with 8 GB RAM configuration

One thing to observe while running the tests for this configuration was the fact that for 500 requests, while running the tests, the server stopped processing requests after the 464th request, thus a new run was made. On the second try it stopped at the 440th request. For the third run of this test case all requests were processed. Continuing to the 1000 requests test, the experiment was conducted four times. The first three experiments stopped after a number of requests while the forth test ran without problem. The 2500 requests tested ended with three trials, all exhibiting a sudden stop after a number of requests. The first time the test was run, the server stopped processing after 2170 requests, the second time after 1860 requests and the third time after 1174 requests. Seeing as how the maximum number of requests the server was able to process started to decrease drastically, no more experiments were conducted for this configuration. The results for this configuration are presented in the following table.

TABLE VI. THE MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 1 CORE AND 8 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.608	0.695	0.614
10	4.071	0.749	4.596
25	7.943	4.384	12.031
50	11.795	8.458	22.026
100	9.333	23.864	46.686

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
250	10.983	54.002	108.503
500	12.850	116.218	239.010
1000	13.441	227.582	458.613

The same observation as the previous configuration regarding the average processing time and total processing time can be made to this experiment as well. The difference between this configuration and the previous one is that the average response time has a linear growth.

While calculating the ratio between the two measurements, it was observed that the performance coefficient has a value around 1, meaning that the increasing of the RAM capacity did not bring a big improvement.

The CPU load in this case has almost reach full capacity, being above 80% for the last test run. The percentage of used RAM by the machine and by the Java process almost have a constant growth, never exceeding 25% and 10%.

3) 1 Core with 16 GB RAM configuration

While testing this machine, it was observed that everything went smoothly until the test involving 1000 requests when after 3 trials, the server still didn't manage to process all requests. For the first trial the server stopped processing after 948 requests, the second one stopped at 799 requests and the third one stopped at 725 requests. After seeing that the number of requests processed kept decreasing, no further tests were made. The results can be seen in the following table.

TABLE VII. THE MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 1 CORE AND 16 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.517	0.713	0.526
10	3.9673	0.7938	4.284
25	9.0574	1.5454	11.456
50	17.33258	4.49798	22.056
100	14.37559	19.24522	46.714
250	14.552752	48.302796	106.581
500	14.774252	102.667852	214.242

The observation for how these measurements tent to grow is the same as the previous experiment, the average processing time having a logarithmical growth, while the average response time and the total processing time have a linear growth.

The ratio obtained for the average response time in half of the test cases is above 2 and overall above 1. The ratio for the total processing time and the average processing time have values around 1.

The CPU load and RAM usage increased as the experiment kept progressing, with the CPU having a tendency to reach its maximum capacity, as it reached 80% when running the last test. The percentage of used RAM rarely exceeds 10%, keeping a constant growth.

4) 2 Cores with 4 GB RAM configuration

When testing this configuration, what was observed was the fact that until the 1000 requests test, all tests ran smoothly. For the 1000 requests test, the server stopped processing after a number of requests (992, 865, 998, 998). Seeing as how these numbers were close to 1000, the test was run for a fifth time with the result of processing all requests. For the 2500 requests tests, though, after 5 trials, the server still didn't manage to process all requests, stopping at 2311 requests, 2418 requests, 2062 requests, 2144 requests and 2004 requests. Since these numbers are far from 2500, the testing has stopped. The result of this experiment can be observed in the following table.

TABLE VIII. THE MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 2 CORES AND 4 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.328	0.689	0.353
10	2.441	0.746	2.843
25	4.649	1.255	5.586
50	8.821	1.379	10.672
100	16.212	3.887	21.94
250	25.544	15.770	54.698
500	28.746	40.319	105.726
1000	36.476	91.703	218.201

The same observation can be made for this configuration as well, the average processing time has a logarithmical growth, while the average response time and total processing time have a linear growth.

The ratio obtained for the average response time, compared to the base configuration was in most cases above 2. The average processing time ratio still maintains its value around 1, but the total processing time ratio is over 2 in most cases.

The CPU load is a bit high as it reaches 80%. The percentage of used RAM by the machine has a linear to constant growth, never exceeding 60%, while the percentage of the used RAM by the Java process is closer to a constant growth by never exceeding 20%.

5) 2 Cores with 8 GB RAM configuration

What was observed for this configuration during the tests was that until the 500 requests test no error occurred, but for the 1000 requests test, the first time it ran it stopped at 792 requests. This was a small number so the test was rerun. After the rerun on this test, all requests were successful. While running the 2500 requests tests, it was observed that the server stopped after a certain number. For the first trial the server stopped processing after 2356 requests. For the second trial, it

stopped after 2490 requests. For the third one the server stopped at 2360 requests and for the forth one it stopped at 2338 requests. Seeing as how, no matter how many times the tests were run, the server stopped at a point, no further tests were done. The measurements made for this experiment can be found in the table below.

TABLE IX. MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 2 CORE AND 8 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.394	0.621	0.395
10	2.3455	0.845	2.642
25	4.796	0.876	5.513
50	8.655	1.544	10.649
100	16.042	3.176	21.183
250	29.287	12.784	52.365
500	34.669	36.715	104.167
1000	37.552	84.606	201.942

From the obtained measurements, we can conclude that the average processing time has a logarithmical growth, while the average response time tends to have a linear growth as well as the total processing time.

The ratio of the average response time has a value over 2 in most cases, as well as the total processing time ratio, but the average processing time ratio has a remained constant, having a value around 1.

The CPU load characteristic can be considered high as it reaches 80% capacity. The percentage of used RAM as well increases with the number of requests but it is never above 40%. The percentage of used RAM by the Java process increases as well but it is never above 20% of the total RAM.

6) 2 Cores with 16 GB RAM configuration

For this experiment, what was observed was the fact that, until the 1000 requests tests, no error occurred. For the 1000 requests test the server stopped processing after 860, but after a retrial of the test, it succeeded in completing all requests. For the 2500 requests test, though, 5 retrials were made, each of them ending with the server stopping at a point in time. It stopped after 2490 requests at first, thus being close to 2500 the test was rerun, but the server never managed to process all requests, thus the testing stopped. The results of this experiment can be found in the following table.

TABLE X. MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 2 CORE AND 16 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.419	0.607	0.426
10	3.966	0.776	4.48
25	6.699	1.133	7.615
50	8.056	1.484	9.657
100	15.568	3.041	19.832
250	23.406	13.226	46.436
500	25.937	34.910	93.05
1000	27.507	79.238	181.966

This configuration also presents a logarithmical growth for the average processing time and a linear growth of the average response time and total processing time, similar to the first experiment.

The ratio obtained for the average response time has a value above 2 in most cases. The ratio for the total processing time in most cases is above 2. The ratio for the average processing time, though is still constant around the value of 1.

When the machine was monitored, it was notable that the percentage of how much RAM is being used by the machine never exceeds 20%, while the CPU load tends to reach 80%.

7) 4 Cores with 8 GB RAM configuration

When testing this configuration, what was observed was the fact that for the 1000 requests test the server stopped processing after a number of requests. It took 5 trials in order for the server to finish processing these requests. When running the 2500 requests test the server had a similar behavior as to the previous one. It stopped processing requests after around 2200 requests, but after 5 trials the server managed to process them all. The measurements obtained for this configuration can be found in the following table.

TABLE XI. MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 4 CORE AND 8 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.365	0.634	0.366
10	0.735	0.833	1.32
25	2.336	0.8047	2.788
50	3.871	1.111	4.791
100	6.498	2.599	9.668
250	10.243	7.835	22.788
500	13.917	18.055	45.813
1000	15.799	40.056	94.02
2500	15.010	113.571	220.25

These measurements present the same growth as previous experiments, logarithmical growth for the average processing time, and linear growth for the average response time and total processing time.

The ratio of the average response time was observed to have a value over 2, in most cases having a high value, even that of 12. The ratio for the average processing time has improved as well, having mostly values over 1.5 while the ratio for the total processing time is above 3 in most cases.

The CPU load never reaches full capacity, the maximum value being that of 70%. The load on each CPU is similar to the average CPU load. The RAM capacity that is never above 40% while the RAM used by the Java process never increases above 20%.

8) 4 Cores with 16 GB RAM configuration

When conducting this experiment, it was observed that it encountered problems only when dealing with the 2500 requests test. There were 5 trails for this test, in the end the server managed to process all requests. The following table best describe de obtained results.

TABLE XII. MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 4 CORE AND 16 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.323	0.597	0.323
10	0.983	0.666	1.225
25	2.402	1.148	2.959
50	4.249	1.165	4.992
100	7.106	1.917	9.273
250	11.175	7.573	22.719
500	11.665	18.694	44.193
1000	13.904	38.909	87.946
2500	16.401	100.132	210.457

The same observation can be made regarding the growth of the three measurements, the average processing time has a logarithmical growth, while the average response time and total processing time have a linear growth.

The ratio for the average response time was calculated and in most cases had a value over 3, even reaching the value of 11 in some cases. The ratio for the total processing time has a value above 3 in most cases and the ratio for the average processing time has a value above 2 in half of the cases.

The CPU load on each core is similar to the average CPU load and it never exceeds 60%. The used RAM also increases but the maximum value it reaches is that of 20%, while the RAM used by the Java process grows as well but it is a bit over 10%.

9) 4 Cores with 32 GB RAM configuration

The last configuration tested was that of 4 Cores with 32 GB RAM. When running the tests, it was observed that, until reaching the 2500 requests test, the server managed to process al requests. For the 2500 requests, though, the test had to be run a number of 5 times in order to process all requests. The results obtained are described in the following table.

TABLE XIII. MEASUREMENTS OBTAINED FOR THE CONFIGURATION OF 4 CORE AND 32 GB RAM

No. req.	Avg. proc. time (seconds)	Avg. resp. time (seconds)	Server proc. time (seconds)
1	0.347	0.602	0.348
10	3.2038	0.6736	4.495
25	6.609	1.00828	7.537
50	4.79868	2.12062	6.907
100	8.19832	2.50827	11.414
250	10.481604	9.111256	24.649
500	14.373798	18.951246	49.192
1000	17.74672	42.237994	96.843
2500	20.14575	102.145215	209.451

We can conclude, from the measurements made, that the average processing time has a logarithmical growth, while the average response time and total processing time tend to have a linear growth.

The ratio for the average response time is above 2 in all cases and mostly above 6. The ratio for the total processing time is above 3 while the ratio for the average response time is mostly above 1. The server exhibits an improvement.

With the number of requests the server had to process, the CPU load and the used RAM capacity have increased. Each core has a CPU load similar to that of the average CPU load and it never exceeds 60%. The percentage of used RAM has a more constant growth and never exceeds 20%, while the percentage of used RAM by the Java process never exceeds 11%.

B. Results

While conducting these experiments, it was observed that the average processing time tends to have a logarithmical growth in all cases, while the average response time and total processing time have a linear growth, thus with the increasing number of requests, the response time and total processing time increase as well.

The comparison made with the base configuration has proven that, with the improvements of certain components of the server, the response time and processing time improve as well, the server being able to process more requests in a shorter period of time and offer a response as well.

When monitoring the activity of the server it was clear that the CPU load and memory usage increased with the number of requests the server had to process. The load was distributed evenly on each CPU, in cases of configurations having 2 ore 4 cores, while the RAM usage tends to grow but in a more constant manner.

With the help of these 9 experiments we can conclude that the increase in RAM capacity has little effect on the performance of the server, while increasing the number of cores the server has exhibits a growth in performance, mainly

the response time of a request, which is the most important aspect for a client as well as the total processing time of all requests.

VI. Conclusions

This paper's objective was to observe the increase in performance of a server while executing a series of tests on various configurations. The method to study this increase in performance was that of vertical scalability, meaning improving one, or more components of the machine in order to bring a better response.

This paper tested the server on a number of 9 configurations, starting from a configuration of 1 Core with 4 GB RAM and increasing the RAM capacity and the number of cores by doubling them until reaching a configuration of a machine with 4 Cores and 32 GB RAM.

By running the set of tests of sending a large number of requests to the server, it was observed that with the increase in the number of Cores the server exhibits a significant improvement in its response time and total processing time, decreasing the two mentioned characteristics to half each time the number of Cores were increased.

By increasing the RAM capacity, though, the server showed small improvements in time, almost insignificant, but it reduced the number of failed tests, the server managing in the end to process all given requests without the necessity of rerunning a series of tests to determine the response time.

VII. Further Work

Besides vertical scalability, there is horizontal scalability as well which implies increasing the number of machines in order to process requests to improve the server's performance in response time and processing time.

An improvement in the server's performance as further work would be to use horizontal scalability to monitor how the server responds to a large number of requests.

Another method of improving the server's performance is to use vertical scalability with horizontal scalability to see how the server reacts to a large number of requests when increasing the number of machines as well as improving the machines by improving various components of the machines (number of cores, RAM capacity, storage capacity, etc.).

Another way to improve the performance of the server would be to use existing solutions offered by various companies, such as using the EC2 offered by Amazon Web Services [8], or to use the existing solution offered by Google, namely Google Cloud Platform [9]. An issue with these solutions is that they require finances [7] in order to build a distributed solution but the advantage would be that these platforms offer management of the servers as well.

As further work, an implementation of this server by scaling it horizontal and vertical will be made. The results of these modifications will be discussed in future papers.

References

[1] Guitart, Jordi, et al. "Characterizing secure dynamic web applications scalability", Parallel and Distributed Processing Symposium, 2005 Proceedings. 19th IEEE International. IEEE 2005.

[2] Arlitt, Martin, Diwakar Krishnamurthy, and Jerry Rolia. "Characterizing the scalability of a large web-based shopping system." ACM Transactions on Internet Technologies 1.1 (2001): 44-49.

[3] Garcia, Daniel F., et al. "Experimental evolution of horizontal and vertical scalability of cluster-based application servers for transactional workloads." 8th International Conference of Applied Informatics and Communications (AIC'08). 2008.

[4] Amza, Cristina, et al. "Specification and implementation of dynamic web site benchmarks." 5th Workshop on Workload Characterizations. No. LABOS-CONF-2005-016. 2002.

[5] Mosberger, David and Tai Jin, "httperf – a tool for measuring web server performance." ACM SIGMETRICS Performance Evaluation Review 26.3 (1998): 31-37.

[6] Ancuta-Petronela Barzu. (2015, September). "Scalability of a Web Server: How does a server scale when dealing with large amounts of data." unpublished

[7] Aviv Kaufmann, Kerry Dolan. (2015, June) "Price Comparison: Google Cloud Platform vs. Amazon Web Services. ESG." [Online]. Available: https://cloud.google.com/files/esg-whitepaper.pdf

[8] Amazon Web Services, https://aws.amazon.com

[9] Google Cloud Platform, https://cloud.google.com

[10] Jakarta Tomcat Servlet Container, http://jakarta.apache.org/tomcat

[11] RUBiS: Rice University Bidding System, http://rubis.ow2.org

[12] Httpperf(1) – Linux man page, https://linux.die.net/man/1/httperf

[13] MySQL, http://www.mysql.com

[14] MEAN Stack, http://mean.io

[15] Node.js, https://nodejs.org/en

[16] Express.js, https://expressjs.org

[17] MongoDB, https://www.mongodb.com

[18] AngularJS, https://angularjs.org

[19] JavaScript, https://www.javascript.com

[20] Java, https://www.java.com/en

[21] Spring Boot, https://github.com/spring-projects/spring-boot

[22] jHipster generator, https://jhipster.github.io

[23] Apache Maven, https://maven.apache.org

[24] VirtualBox, https://www.virtualbox.org

[25] Ubuntu, https://www.ubuntu.com

[26] IntelliJ, https://www.jetbrains.com/idea