Horizontal Scalability Towards Server Performance Improvement

Ancuta-Pentronela Barzu, Mihai Barbulescu, Mihai Carabas
University POLITEHNICA of Bucharest
Computer Science Faculty
Bucharest, Romania
ancuta.barzu@stud.acs.upb.ro, mihai@roedu.net, mihai.carabas@cs.pub.ro

Abstract—**This paper presents an evaluation of an experiment conducted on a server to analyse how horizontal scalability affects its performance. This paper studies the results obtained by measuring response times and processing times when dealing with many requests by adding more machines to the system. This paper presents the technologies used to build this system of machines as well as the results obtained comparing them to a basic configuration and offering a complete analysis of how horizontal scalability affects the performance of a server.**

Keywords—server; scalability; horizontal scalability; performance; improvements; configuration; cluster; nodes; response time; processing time; Linux Virtual Server; performance improvements; load balancer; virtual machine; component;

I. Introduction

When talking about the Internet and how technologies are evolving at a fast pace, a current problem is that of scalability and the capacity of services to scale when dealing with large amount of data. In our times, Internet connection has stopped to represent a major issue when referring to the response time of the server, the real problem being the capability of the server to process many requests and answer them in a short amount of time, thus servers need to be built to support such processing.

It is a well-known fact that a machine with a better configuration can perform more tasks than one with a weaker configuration, but even that has its limitation. The costs to build a performant computer are high and the performance brought by it doesn't justify the cost. Therefore, other solutions were found, that of a cluster of computers.

The main objective of this paper is to conduct an analysis on how horizontal scalability can influence the performance of a Web Server, by analysing the response time and the processing time needed for it when dealing with many requests. Various experiments were conducted to analyse how adding more nodes to the system can affect the Web Server's throughput for many requests. For each request, the server responds with a generic response and starts processing the data., this being done on a separate thread than the one for the received request.

This paper presents the problem of scalability for a server when dealing with many requests to process large amount of data, the experiment conducted to see how horizontal scalability affects the server's performance in response time and processing time as well as an analysis of the obtained results.

II. Realted Work

Scalability is a term used to define the capacity of a system, a network or a process to handle a large amount of work. When discussing this subject, to be considered relevant, a set of requirements must be specified. Horizontal scaling refers to the process of increasing the number of machines in the system, by adding more nodes to the system. In general, this is done by adding more machines to the cluster based architecture.

Linux Virtual Server [6] also known as LVS represents an advanced load balancing solution that can be used to build highly scalable and available network services. Article [1] describes how Linux Virtual Server is implemented and how it can be used to build highly scalable network services using a cluster of servers. In this article, the authors considered that scalability of a network service is achieved when adding or removing nodes to a cluster in a transparent way.

Article [2] describes how a Linux Virtual Server can be created to obtain an incremental scalability, 24x7 availability and cost-effectiveness. The authors of this article describe two of the three methods of creating a Linux Virtual Server, and those are the Virtual Server via NAT and the Virtual Server via IP Tunneling. This article, as well as the one mentioned before, presents the advantages and disadvantages of each configuration, as well as the 4 scheduling algorithms used by the load balancer.

Article [3] describes the performance of a Linux Virtual Server by conducting a series of experiments on one to evaluate how web requests are distributed among the nodes of the cluster. This was tested on various configurations by analysing the CPU load of each machine, as well as observing the load of the LVS Director. Through their experiment, they have concluded that LVS displays a cost effectiveness in terms of price/performance compared to Hardware Load Balancer. In their article, the authors considered that their experience with Linux Virtual Server had been a positive one and that it can be a reasonable alternative to more an expensive hardware possibility.

III. Previous Work

In previous work, a basic server was tested as it is described in paper [4]. This experiment was conducted to see how many requests influences the response time of the server. The machine used was that of an Intel Core i5-4210U processor, 1.7 GHz frequency, 8 GB RAM and a SSD hard of 256 GB and a 64-bit Windows 8.1 operating system. The maximum number of requests with which the server was tested as that of 100 simultaneous requests. The conclusion of this was that the server does not scale for many requests. In this paper, the server used the MEAN [7] stack containing the following technologies: Node.JS [8], Express.JS [9], MongoDB [10] and AngularJS [11].

Article [5] continues the work of the previous described paper by analysing how vertical scalability influences the performance of it. The modification made to the server was changing the server-side technologies which were used, from Node.JS with Express.JS to Java [12] with Spring Boot [13].

The vertical scalability of the server was tested by sending multiple requests to different configurations and analysing the response time and the processing time. The base configuration used was that of a machine with 1 core with 4 GB RAM, 50 GB hard-drive and an Ubuntu 16.04 64-bit operating system. This configuration was changed, doubling the number of cores and the RAM capacity, until reaching a machine with 4 cores and 32 GB RAM. The performance was analysed by comparing obtained times. The experiment showed that by increasing the number of cores and the RAM capacity, the performance of the server also increases.

IV. Technologies Used

This chapter presents the technologies used to build the architecture of system, that of a cluster of servers.

A. *Linux Virtual Server*

Linux Virtual Server (LVS) is an advanced load balancing solution that is used to build highly scalable and available network services. LVS redirects received network connections to different servers according to a scheduling algorithm giving the impression that the parallel services of the cluster are a virtual service on a single IP address.

1) *Architecture Overview*

The standard architecture of a Linux Virtual Server consists of two elements: the load balancer and the server pool.

a) Load Balancer

The load balancer represents the front end to the service, were all requests are received. The IP address of the load balancer is the one known by the clients, thus all services are accessible through this component then redirected to the real servers. This component is also known as Virtual Server (VS).

b) Server Pool

The server pool represents the cluster of servers which offer the implemented services, such as web services, ftp, mail, etc. The servers in the server pool are hidden from the client and cannot be accessed by them, only through the load balancer. These servers are also known as Real Servers (RS).

2) *Functioning Modes*

a) Linux Virtual Server via NAT

This method is used when the real servers have private IP addresses and are in the same local network. The virtual servers and the real servers are usually interconnected through a switch. The virtual server is the only one accessible to the exterior and when requests are received it performs NAT for the real server by modifying the packages IP. In this case, all traffic is handled by the load balancer. This method is limited as the load balancer represents a bottleneck for the system.

b) Linux Virtual Server via IP Tunneling

In this case, the virtual server tunnels the requests to different real servers. The servers process the requests and send the response directly to the client, without going through the virtual server. This method scales better than the previous one, but for this method to work the real servers must support IP tunneling.

c) Linux Virtual Server via Direct Routing

This method is similar to the previous one as the real servers respond to requests directly to the client. The difference is that the virtual server doesn't tunnel the requests, it routes them directly. This method only works if each real server and the virtual server have an interface in the same local network. It also requires for the real servers to be able to respond to requests addressed to the virtual server without having to modify that the destination IP address. There are two methods for this:

- Configuring the virtual server's IP address on a loopback interface of the real server.
- Adding an iptables [14] rule so that the real server can accept packages for the IP address of the virtual server.

3) *Scheduling Algorithms*

a) Round-Robin Scheduling

This type of scheduling redirects network connections is a round-robin manner. This type of algorithm treats all servers equally ignoring the number of connections or response time.

b) Weighted Round-Robin Scheduling

This type of scheduling is useful for real servers with different processing capacities. In this case, each real server has assigned a weight represented by an integer number which indicates the processing capacity. The default weight for a real server is 1. In this type of scheduling, servers with higher weights receive new connections. If servers have equal weights then they receive the connections are distributed equally.

c) Least-Connection Scheduling

The least-connection scheduling redirects new connections based on the number of active connections on each real server. In this case, the server with the least number of active connections will receive the new connection. This type of scheduling benefits a system that contains a collection of servers with similar performance.

d) Weighted Least-Connection Scheduling

The weighted least-connection scheduling is an improvement to the least-connection scheduling where each server has assigned a weight representing its performance indicator. In this case, the server with higher weight values will receive a larger number of connections, but it also takes into consideration the number of active connections on each server.

V. Architecture

Linux Virtual Server was used to build a cluster of servers capable of processing multiple requests. The method used was that of Linux Virtual Server via Direct Routing with a least-connection scheduling. The reasons for choosing this type of architecture is because the load balancer and the real server are in the same local network. The least-connection scheduling was chosen because all servers in the system have the exact same configuration.

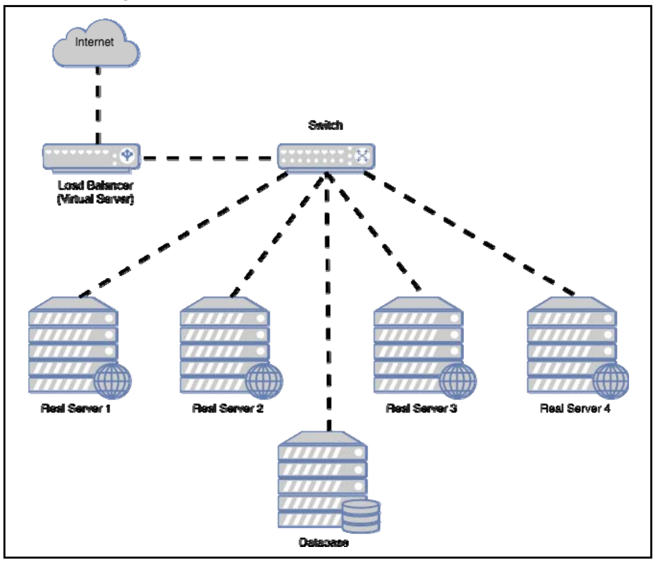

Fig. 1. Architecture of configured system using LVS

Fig. 1 represents the architecture of the system built to test the horizontal scalability of the server. This architecture has 3 major components: the load balancer, the cluster of real servers and the database. All components are connected through a switch. The load balancer is the one visible to the client, receiving requests and forwarding them to the real servers. This implements the least-connection scheduling. The cluster of servers are the ones running the services. They receive requests from the load balancer and they respond directly to the client without going through the load balancer. The real servers communicate with the database directly to store processed data or information from the client. The database is the component where all data is stored. It is separated from the servers and virtual server.

To build this architecture, the ipvsadm [15] tool was used configuring the load balancer to redirect requests to the real server based on their active connections. To resolve the problem of the real servers responding to the clients directly, an iptables rule was added on each real server to accept all requests even if the virtual server IP isn't configured on any interface interfaces. This was done with the below command.

```
iptables –t nat –A PREORUTING –d <IP_ADDR_VS> -j
REDIRECT
```

VI. Experimental Environment

An experiment was conducted to test how horizontal scaling affected the server's performance and measurements were taken and reports were generated. The environment consists of three physical machines: the server, the database and the client.

A. Server

The server's main job is to run the web application, receive tasks, process them and store the results to the database.

The server represents a set of machines, all running the same web application. The number of machines was either 2 or 4, as both cases were studied to see how the response time and processing time was influenced by a different number of machines. Apache Maven [16] was used to run the web application. Each machine is a virtual one, all running on one of the three physical machines. These were created using VirtualBox [17] using an Ubuntu [18] 16.04 64-bit image. Each machine has all necessary packages installed: Java, Apache Maven and Node.JS, this being needed by the web application to download all necessary dependencies.

Besides the 2 or 4 machines running the web application, the load balancer was also run as a virtual machine on the same physical device.

B. Database

The database runs on a separate machine to preserve the consistency of data; thus, all data is stored in the same location. In this case, when a request is received, the load balancer doesn't deal with where the information is stored, it only forwards the requests to a server and this connects to the database to retrieve, add or delete data. This database ran on a physical machine with a configuration of an Intel Core i7 with 4 cores, 10 GB RAM and a Windows 7 operating system.

C. Client

The client, also known as the test application, ran on the third physical machine, a MacBook Pro with a 2.7GHZ quad-core Intel Core i7, 16 GB RAM, 512 GB SSD and running macOS Sierra as an operating system. The test application is a Java program that creates a specific number of threads to send HTTP requests to the server and measures the response time for each request to obtain an average for the response time.

VII. Experiments and Results

To test the horizontal scalability of the server, a series of tests ran on different configurations to analyse the server's overall performance when talking about response time, processing time and number of requests processed. The base configuration for which results were compared to consists of a machine with 1 core, 4 GM rand and a 50 GB hard-drive running an Ubuntu 16.06 64-bit operating system. The configurations for which this was tested was that of 2 or 4 machines with the same configuration as the previously described one. A vertical scalability was also applied, by improving the hardware of each machine to analyse the server's performance.

The tests used for this experiment were the same as the ones used for the vertical scalability meaning a set of 9 tests with: 1, 10, 25, 50, 100, 250, 500, 1000 and 2500 requests sent. All these requests were sent simultaneously.

A. Experiments

Each experiment was conducted by doubling and quadrupling the number of machines in the system. For each experiment, the measurements were made for average response time and average processing time. Each experiment was compared to their base configuration, that of a machine with 1 machine and calculating a performance indicator as being the ratio between the measurements obtained from the base configuration and the ones obtained from the improved configuration.

1) Measurements

The first measurement conducted was regarding the average processing time. The measurements taken can be seen in figures Fig. 2, Fig. 3 and Fig. 4.

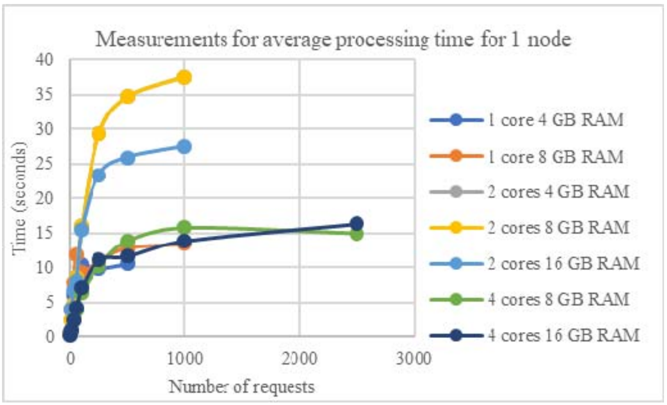

Fig. 2. Measurements for average processing time for a system containing 1 node

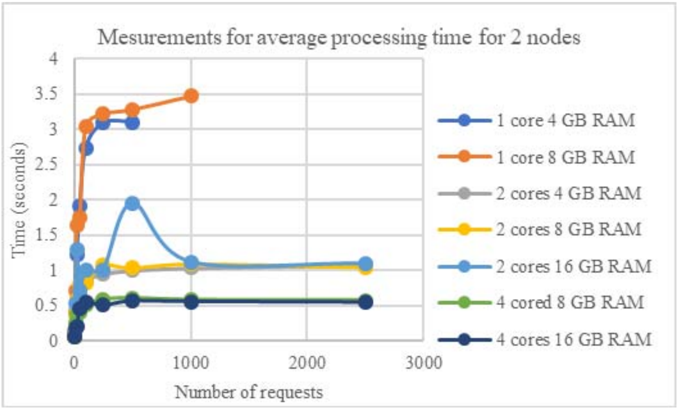

Fig. 3. Measurements for average processing time for a system containing 2 nodes

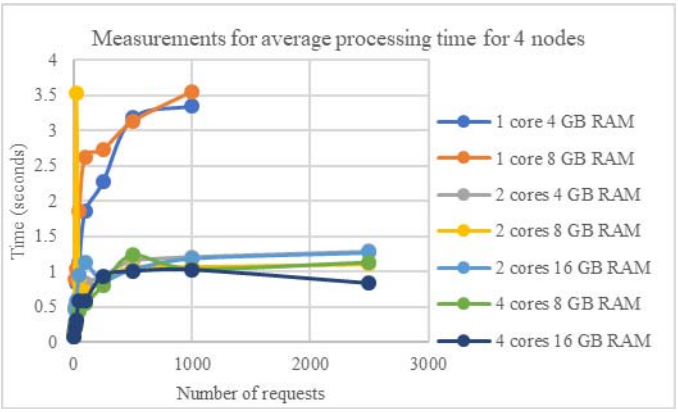

Fig. 4. Measurements for average processing time for a system containing 4 nodes

From Fig. 2, Fig. 3 and Fig. 4 we can see an improvement in performance when discussing the average processing time. For 1 node in the system, the best configuration obtained an average processing time of 15 seconds for 2500 requests, while for 2 nodes and 4 nodes, the average processing time for machines having the same configuration is between 0.5 seconds and 1 second. As it can be seen in the above figures, just by adding a second machine to the system, the processing time decreases to almost 5 seconds in worst case scenario, compared to 35 seconds for the system with one node, this being for the worst configuration, that of 1 core with 4 GB RAM.

The system containing 4 nodes appears to obtain the best measurements, but these are not far away from the system containing 2 nodes. The load received is distributed evenly on all machines, thus this may mean that the system can process more than 2500 simultaneous requests until reaching a point where a difference between 2 nodes and 4 nodes becomes significant.

The average processing time exhibits a logarithmic growth in all cases, be it a system with 1 node, 2 nodes or 4 nodes. As it can be seen from the above figures, after about 100-250 requests, the average processing time exhibits an almost constant growth.

The performance indicator for the average processing time for all configurations is:

- Above 2 for a system containing machines of 1 core with 4 GB RAM;
- Above 1.7 for a system containing machines of 1 core with 8 GB RAM;
- Above 5 for a system containing machines of 2 cores with 4 GB RAM;
- Above 6 for a system containing machines of 2 cores with 8 GB RAM;
- Above 3 for a system containing machines of 2 cores with 16 GB RAM
- Above 3 for a system containing machines of 4 cores with 8 GB RAM;

- Above 4 for a system containing machines of 4 cores with 16 GB RAM.

The other measurement taken was regarding the average response time of the system. These measurements are presented in figures Fig. 5, Fig. 6 and Fig. 7 for all tested configurations.

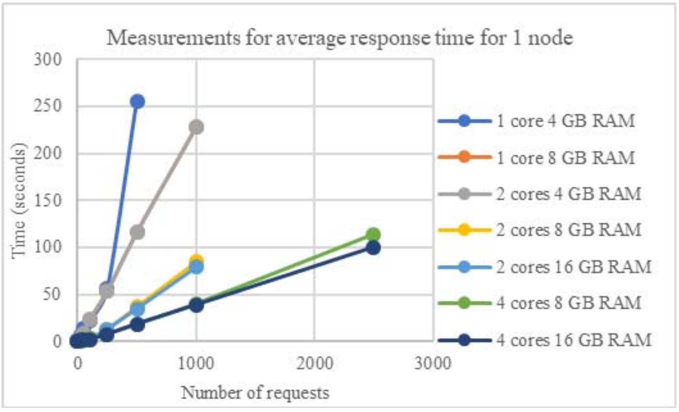

Fig. 5. Measurements for average response time for a system containing 1 node

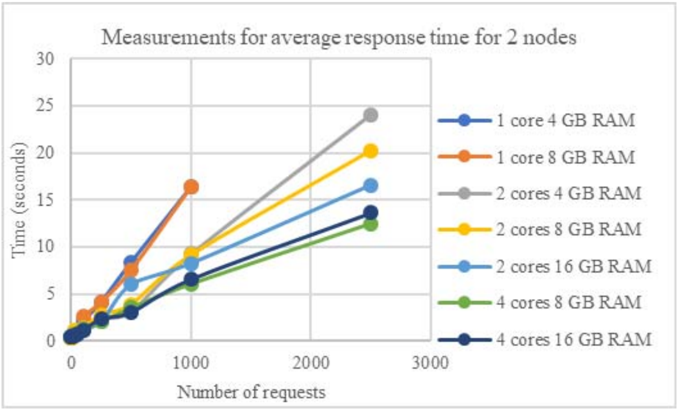

Fig. 6. Measurements for average response time for a system containing 2 nodes

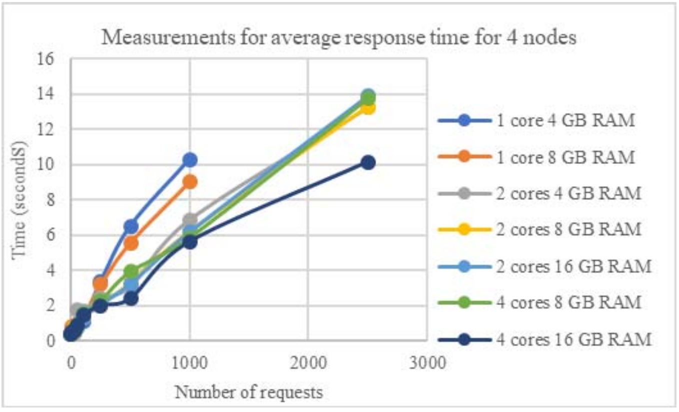

Fig. 7. Measurements for average response time for a system containing 4 nodes

From Fig. 5, Fig. 6 and Fig. 7 we can see an improvement in performance when discussing the average response time. For 1 node in the system, the best configuration obtained an average response time of 100 seconds for 2500 requests, while for 2 nodes and 4 nodes, the average response time for machines having the same configuration is between 10 seconds and 15 seconds. As it can be seen in the above figures, just by adding a second machine to the system, the response time decreases to 10-15 seconds in worst case scenario, compared to 250 seconds for the system with one node, this being for the worst configuration, that of 1 core with 4 GB RAM.

The system containing 4 nodes appears to obtain the best performance, the difference between 2 nodes and 4 nodes being an average of 5 seconds comparing each configuration, thus the system containing 4 nodes obtains a better performance when regarding response time. The highest response time obtained for a system with 2 nodes is that of almost 25 seconds, while for the system containing 4 nodes the highest response time is that of 14 seconds. The lowest response time obtained for a system containing 2 nodes is that of 13 seconds, while for a system with 4 nodes the lowest time is 10 seconds. Both times were measured for the same number of requests, that of 2500 simultaneous requests.

The average response time exhibits a linear growth in all cases, be it a system with 1 node, 2 nodes or 4 nodes. As it can be seen from the above figures, with the increase of number of requests, the average response time increases as well.

The performance indicator for the average response time for all configurations is:

- Above 4 for a system containing machines of 1 core with 4 GB RAM;
- Above 1.7 for a system containing machines of 1 core with 8 GB RAM;
- Above 1.7 for a system containing machines of 2 cores with 4 GB RAM;
- Above 1.6 for a system containing machines of 2 cores with 8 GB RAM;
- Above 1.5 for a system containing machines of 2 cores with 16 GB RAM
- Above 1.5 for a system containing machines of 4 cores with 8 GB RAM;
- Above 1.5 for a system containing machines of 4 cores with 16 GB RAM.

B. Results

While conducting these experiments it was observed that the average response time exhibits a linear growth with the increase of the number of requests. Regarding the average processing time, it can be concluded that it has a logarithmic growth, as it increases at first, but for many requests it exhibits a constant growth. By having 2 or 4 nodes in the system, the processing time exhibits similar behavior. The load balancer distributes the requests in a uniformly manner, thus all servers in the system exhibit the same behavior when monitoring them. Overall, scaling the system horizontally brings an improvement in performance for the server when discussing the response time and the processing time.

VIII. Conclusions and Future Work

This paper's objective was to observe the increase in performance of a server when applying a horizontal scalability to a server. This paper described the technologies used to build a cluster of servers as well as the architecture of the system to test the horizontal scalability of it. This experiment consisted of a series of 14 configurations, from a basic 2 nodes cluster, to a more performant 4 nodes cluster, with different types of configurations for the machines. It was observed that the server scales horizontally and better performance is obtained by adding more nodes to the system.

An improvement in the server's performance would be to analyse the web application itself and the architecture and try optimizing them. By optimizing the code, the processing time will be reduced, while optimizing the architecture will reduce the response time. To optimize the architecture, other solutions must be researched, such as EC2 from Amazon Web Services [19] or Google Cloud Platform [20]. As future work, investigations will be made regarding the current architecture and how it can be improved.

References

[1] Zhang, Wensong. "Linux virtual server for scalable network services." Ottawa Linux Symposium. Vol. 2000. 2000.

[2] Zhang, Wensong, Shiyao Jin, and Quanyuan Wu. "Creating Linux virtual servers." LinuxExpo 1999 Conference. 1999.

[3] O'Rourke, Patrick, and Mike Keefe. "Performance Evaluation of Linux Virtual Server." LISA. 2001.

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

[5] Ancuta-Petronela Barzu, Mihai Carabas. (2017, Mai). "Scalability of a Web Server: How does vertical scalability improve the performance of a server." CSCS 2017.

[6] Linuxvirtualserver.org. (2017). "The Linux Virtual Server Project - Linux Server Cluster for Load Balancing" [Online]. Available: http://linuxvirtualserver.org/

[7] Mongo Express Angular Node. (2017). "Home – Mongo Express Angular Node." [Online]. Available: http://mean.io

[8] Foundation, N. (2017). "Node.js" [Online] Nodejs.org. Available: https://nodejs.org/en

[9] Anon, (2017). [Online]. Available: https://expressjs.org

[10] MongoDB. (2017). "MongoDB for GIANT Ideas" [Online]. Available: https://www.mongodb.com

[11] Angularjs.org. (2017). "AngularJS – Superheroic JavaScript MVW Framework" [Online]. Available: https://angularjs.org

[12] Java.com. (2017). "java.com: Java + You" [Online]. Available: https://www.java.com/en

[13] GitHub. (2017). "spring-projects/spring-boot" [Online] Available: https://github.com/spring-projects/spring-boot

[14] Ipset.netfilter.org. (2017). "Man page of IPTABLES". [Online] Available: http://ipset.netfilter.org/iptables.man.html

[15] Linuxcommand.org. (2017). "ipvsadm" [Online]. Available: http://linuxcommand.org/man_pages/ipvsadm8.html

[16] Porter, B., Zyl, J. and Lamy, I. (2017). "Maven – Welcome to Apache Maven" [Online]. Available: https://maven.apache.org

[17] Virtualbox.org. (2017). "Oracle VM VirtualBox" [Online]. Available: https://www.virtualbox.org

[18] Ubuntu.com. (2017). "The leading operating system for PCs, tablets, phones, IoT devices, servers and the cloud | Ubuntu." [Online]. Available: https://www.ubuntu.com

[19] Amazon Web Services, Inc. (2017), "Amazon Web Services (AWS) – Cloud Computing Services." [Online]. Available: https://aws.amazon.com

[20] Google Cloud Platform. (2017), "Google Cloud Computing, Hosting Services & APIs | Google Cloud Platform." [Online]. Available: https://cloud.google.com