Ilkogretim Online - Elementary Education Online, 2020; 19 (4): pp. 744-751
http://ilkogretim-online.org
doi:10.17051/ilkonline.2020.04.178

A tutorial: Load balancers in a container technology system using docker swarms on a single board computer cluster

***Marischa Elveny,** *Faculty of Computer Science and Information Technology (Fasilkom-TI), Universitas Sumatera Utara, Medan, Indonesia, marischaelveny@usu.ac.id*
Ari Winata, *Faculty of Computer Science and Information Technology (Fasilkom-TI), Universitas Sumatera Utara, Medan, Indonesia.*
Baihaqi Siregar, *Faculty of Computer Science and Information Technology (Fasilkom-TI), Universitas Sumatera Utara, Medan, Indonesia.*
Rahmad Syah, *Faculty of Computer Science and Information Technology (Fasilkom-TI), Universitas Sumatera Utara, Medan, Indonesia.*

Abstract. Overload requests on a webserver are one of the important issues in web server management. The main cause of overload requests on the webserver is the high demand by the user / client for the webserver that exceeds the maximum load limit borne by the webserver. One of the ways used to handle overload requests on the webserver is by sharing the requests evenly among the available web servers. The webserver management system still uses a single server by the webserver management developer which results in the possibility of overloading requests on the webserver. The method proposed in this study is a haproxy load balancer on a single board computer cluster using a docker swarm. Before the performance testing analysis stage is carried out, the researcher will build a single board computer cluster, using 1 raspberry pi master server which will become the load balancer and 3 raspberry pi worker servers to become a webserver which will respond to users alternately according to the algorithm that will be used on the hap Roxy load balancer. After testing in this study, it was concluded that using the load balancing method with the least connection algorithm resulted in an accuracy rate of 95.47%, while testing using the load balancing method with the round robin algorithm was able to produce a greater level of accuracy, namely 97%. Use 1 raspberry pi master server which will be the load balancer and 3 raspberry pi worker servers to become the webserver which will respond to the user alternately according to the algorithm that will be used on the haproxy load balancer. After testing in this study, it was concluded that using the load balancing method with the least connection algorithm resulted in an accuracy rate of 95.47%, while testing using the load balancing method with the round robin algorithm was able to produce a greater level of accuracy, namely 97%. use 1 raspberry pi master server which will be the load balancer and 3 raspberry pi worker servers to become the webserver which will respond to the user alternately according to the algorithm that will be used on the haproxy load balancer. After testing in this study, it was concluded that using the load balancing method with the least connection algorithm resulted in an accuracy rate of 95.47%, while testing using the load balancing method with the round robin algorithm was able to produce a greater level of accuracy, namely 97%.

Keywords: Load balancer, IOT, Docker swarms, Round robin algorithm.

Received: 04.10.2020	Accepted: 11.11.2020	Published: 26.12.2020

BACKGROUND

A server is a computer system that provides a certain type of service in a computer network. The server is a place that contains various types of information or data. The server is supported by a processor that is scalable and large RAM, also equipped with a special operating system, which is called a network operating system [1]. In service processing by the server, there is a condition that the number of services requested by the client exceeds the maximum load that the server cannot bear [2] [3]. So this can result in heavy traffic on a cross-network, and the workload on the server also increases. When a system experiences an increase in the number of requests to exceed the maximum threshold of a server per day / hour / minute / second, this can cause system performance on the server to be very slow and cause problems, one of which is a complaint felt by the client or user [4]. The problem that arises in this research is the increasing workload on the web server in line with the increasing number of web-based applications and clients or users that must be handled or served. The data used by web-based applications is usually stored in a database server [5] [6]. This allows corruption of data from the entire system to the application when the database server experiences problems that might occur. This can

cause system performance on the server to be very slow and cause problems, one of which is the complaints felt by clients or users.

The increasing demand for information needs on the internet has caused traffic on the internet to also become denser. The increasing amount of traffic on the internet will cause the workload of an information service provider, namely the web server to be overloaded or better known as Overload, which can cause the server to go down [7]. To solve the problem of overloading the server, the concept of cluster computing can be applied, which is a technology that combines several computers or servers that work together as if they are a single system. In its implementation, the cluster computing system is built using the load balancing method, which is a technique for distributing traffic loads to several server clusters in a balanced manner, so that traffic can run optimally and optimally. The implementation of load balancing on the web server cluster is very important and can be a solution in handling server loads that are too heavy due to the number of requests or requests so that it can increase scalability in distributed systems [8].

Research on the Container Technology System and related to Docker has been carried out by several previous researchers. Based on the analysis conducted [9], the right technology for realizing a high availability webserver system is to adopt a container-based virtualization technology instead of a virtual machine-based. This technology does not build your own virtual machine, it saves more memory, processor and storage. The startup time required for Docker is also very fast, even much faster than the physical server. This happens because docker shares or shares the Linux kernel from a physical server [9].

Another study, namely the Implementation of Load Balancers with Lightweight Virtualization Using Docker for Video on Demand Services discusses the application of load balancers with lightweight virtualization to dockers. This study aims to determine the performance of the load balancer on video on demand services. From the results of the research conducted it is known that the server performance using load balancing is better than the single server, because the workload and traffic load are no longer served by one server anymore, but the load is divided into three servers. In this study it is also known that the best algorithm to use for load balancing is the least connection, because there is a decrease in CPU utilization by 5.17% [10].

Other research, namely the Implementation of Server Clusters on the Raspberry Pi using the Load Balancing Method, discusses conducting server clusters that are applied to the Raspberry Pi. The method used in the construction of this server cluster uses the load balancing method. The test is applied by comparing the performance of the Raspberry Pi which handles data traffic singly without using a load balancer and testing the Raspberry Pi using a load balancer as a load balancer between server cluster members. Based on the results of tests that have been done [14], it can be proven that the Raspberry Pi as a load balancer is able to minimize the performance load of webserver data traffic. Aside from that.

RESEARCH OBJECTIVES

The purpose of this study was to determine the results of implementation testing and load balancer analysis in handling requests to the webserver in the container technology system network using a docker swarm on a single board computer cluster.

System Analysis and Design

General Architecture

The general architecture section describes the process flow of the system to be built. The general architecture of the system to be built can be seen in Figure 3.1.

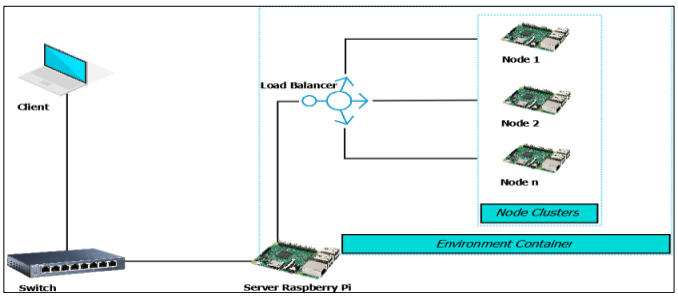

Figure 1. *General Architecture*

The following is an explanation of the working principle of the system in Figure 3.1.

Client

In this section, the client will make a service request to the server via a network that has gone through the network routing stage.

Switch

In this section the switch will forward the service request that the client has requested to the server.

Server

In this section the server will receive a service request and then immediately process the request and forward it to the load balancer that has been created in such a way.

Load Balancer

In this section the load balancer will use the Round Robin scheduling algorithm and the Least Connection scheduling on the HAProxy Load Balancer will then be forwarded to the single board computer cluster.

Single Board Computer Clusters Using Docker

In this section Docker will use the container technology system as a tool that will handle a number of requests that have been forwarded by the previous stage, and return them to the server, and the server will respond to the client.

Load Balancer Implementation Analysis

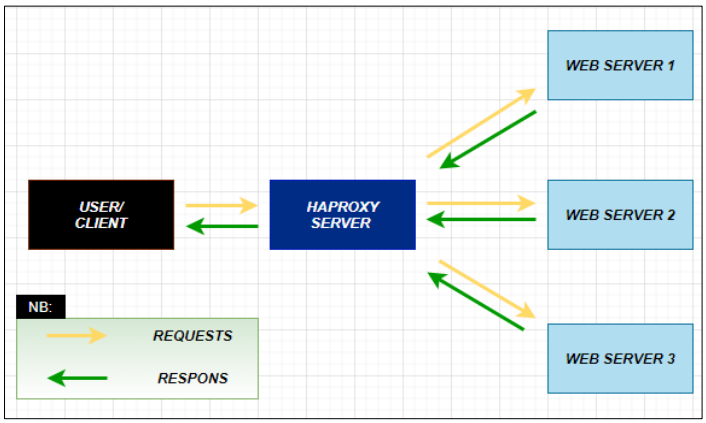

Figure 2. *Load Balancer Implementation Analysis*

The following is an explanation of the working principle of the system in Figure 3.2.

User / Client

In this section the user / client will send several requests to the haproxy server and will receive feedback in the form of the desired response.

Haproxy Server

In this section haproxy will receive several requests from users and then distribute the requests to nodes that have service availability or service availability that are still able to receive these requests.

Webserver - 1

In this section Webserver - 1 will accept requests that have been distributed by haproxy. If this webserver is unable to provide the service, it will give a null or error value to the haproxy server and will distribute the request to other web servers.

Webserver - 2

In this section Webserver - 2 will receive requests that have been distributed by haproxy. If this webserver is unable to provide the service, it will give a null or error value to the haproxy server and will distribute requests to other web servers.

Webserver - 3

In this section, Webserver - 3 will receive requests that have been distributed by haproxy. If this webserver is unable to provide the service, it will give a null or error value to the haproxy server and will distribute requests to other web servers. If other web servers have reached the maximum threshold value, the haproxy server will send null or error to the users.

System Flowchart

Flowchart design aims to make it easier for users or readers to more easily understand the workflow of a system so that the system can be used as well as possible. The flowchart design for requesting service requests from users and sending responses by the server can be seen in Figure 3.3.

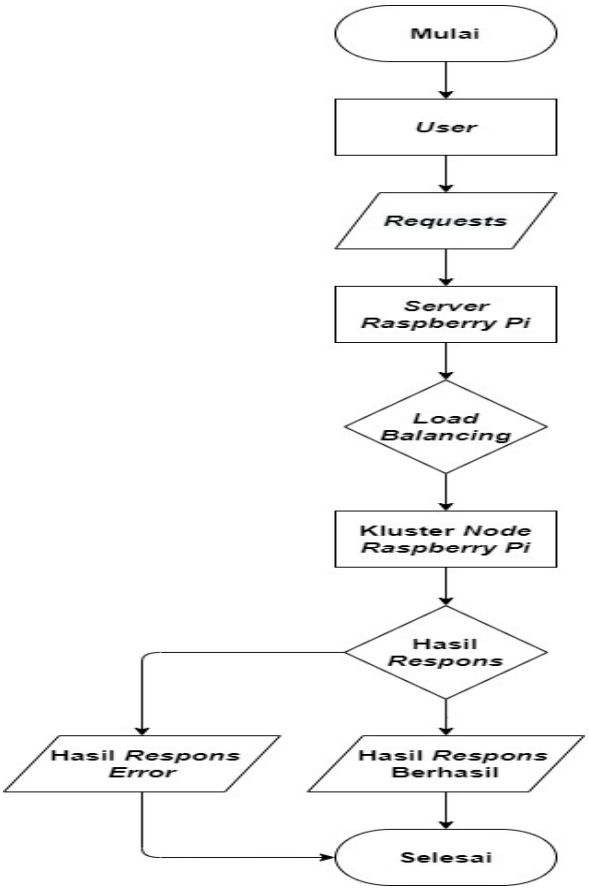

Figure 3. *Flowchart system*

The following is an explanation of the working principle of the system in Figure 3.3.

User

In this section the user will make a request.

Request

In this section, the user will receive a request and then proceed to the raspberry pi server section.

Raspberry Pi Server

In this section the raspberry pi server will receive requests by the user and then proceed to haproxy as a load balancer in this study.

Load Balancer

In this section the haproxy load balancer will continue to the raspberry pi node cluster.

Raspberry Pi Cluster Node

In this section the raspberry pi node cluster will respond whether it still has service availability or service availability that can execute requests by the user. If node 1 is willing it will give results, if node 1 is not willing it will notify haproxy as a load balancer.

Result Response

In this section the results will distinguish between the results of a successful response and the result of an error response, and will be sent back to the user.

Requests Testing Scenario Data

The data used is using 1,680,000 requests made by virtual users using the JMeter application.

Table 1. *Requests Testing Scenario Data*

No.	Users	Ramp-Up Period	Loop Count	Request
1	2,000	10	30	60,000
2	2,000	20	40	80,000
3	2,000	30	50	100,000
4	3,000	10	30	90,000
5	3,000	20	40	120,000
6	3,000	30	50	150,000
7	4,000	10	30	120,000
8	4,000	20	40	160,000
9	4,000	30	50	200,000
10	5,000	10	30	150,000
11	5,000	20	40	200,000
12	5,000	30	50	250,000
Result				**1,680,000 Requests**

RESULTS AND DISCUSSION

Testing Parameters

Throughput

Throughput is the actual bandwidth or the actual measured bandwidth at a certain time in the internet network routing process. Throughput testing is a variable parameter of Quality Of Sevice with the aim of seeing network performance in terms of speed of packet delivery that can be sent by utilizing the available bandwidth.

$$T = \frac{u}{r} * loop \quad (1)$$

Where:

T = Throughput r = ramp up

u = users requests $loop$ = loop count

Response Time

Response Time is the time it takes to complete a request and send it back to the client. Response time testing aims to measure how fast a webserver can receive requests from clients. The results of this test can be seen using the JMeter application in seconds.

$$Response\ Time\ (ms) = Average\ (ms) \quad (2)$$

Request Per Second

Requests Per Second is a measure of the scalability of throughput handled by a system. Testing requests per second is done by performing a number of services per second from the client which will be sent to the webserver to determine the performance of a webserver that uses a load balancing system. This aims to see how many requests can be handled by the webserver running on the docker swarm container cluster. The results of this test can be seen using the JMeter application in seconds.

$$Requests\ per\ second = hits\ per\ second\ (hits/s) \quad (3)$$

Request Loss

Requests Loss are requests sent by the client to the overloaded webserver because the webserver is unable to accommodate more than the maximum capacity of the webserver so that it cannot be accepted by the webserver. The request loss parameter can be measured using the JMeter application by overloading the webserver with requests until the resources on the webserver reach the maximum limit.

$RL = u * e$ (4)

Where:

RL = Requests Loss
u = Users
e = Error Rate

Test Result

Graph of the Test Results of the throughput Test Parameters

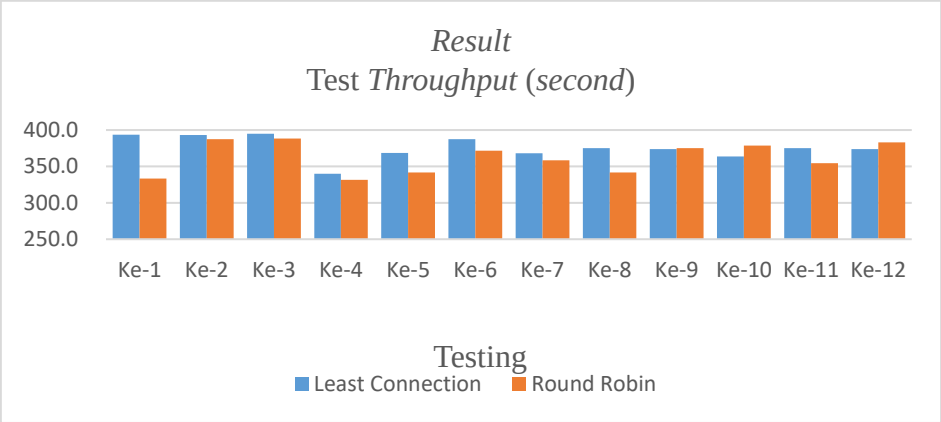

Figure 4. *Graph of the test results of the throughput test parameters*

Based on the test results as shown in Figure 4.1 where in the 1st to 12th test the test value obtained using the least connection algorithm is 4347.6 / s, while the test value obtained using the round robin algorithm is smaller, namely 4345.3 / s.

Based on the test results as shown in Figure 4.1, it can be concluded that the more connections or requests from the user, the greater the resulting throughput. This indicates that the more user connections, the better the server's ability to serve user requests.

The test results can be seen that the system using the least connection algorithm tends to have a higher value than the round robin algorithm where the load sharing process is based on the number of users on the server. Thus, it can reduce queues and can serve more requests so that the throughput obtained is greater.

Graph of Test Results for Response Time Test Parameters

Based on the test results as shown in Figure 4.2 where in the 1st to 12th test the test value obtained using the least connection algorithm is 8,236 / ms, while the test value obtained using the round robin algorithm is smaller, namely 7,861 / ms.

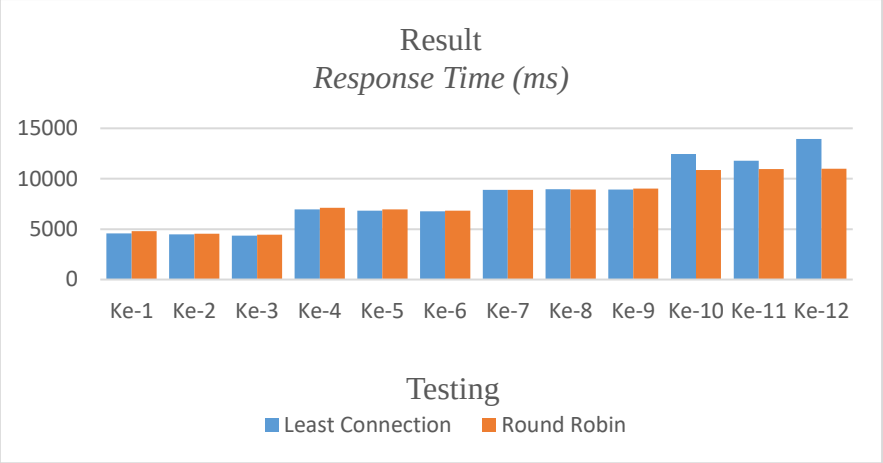

Figure 5. *Graph of test results for response time test parameters*

From the results of the response time testing carried out, the average response time value obtained tends to increase along with the increase in the throughput value obtained, the greater the throughput value obtained during testing, the time it takes for a server cluster to send an http response to the client will be getting slower because the requests are coming more and more.

From the graph shown in Figure 4.2, it can be seen that the smaller response time value is when the server cluster uses a load balancing system with a round robin algorithm, this is because the resulting transmission speed and throughput have better results than the least connection algorithm.

Comparison Chart of Load Balancer Accuracy

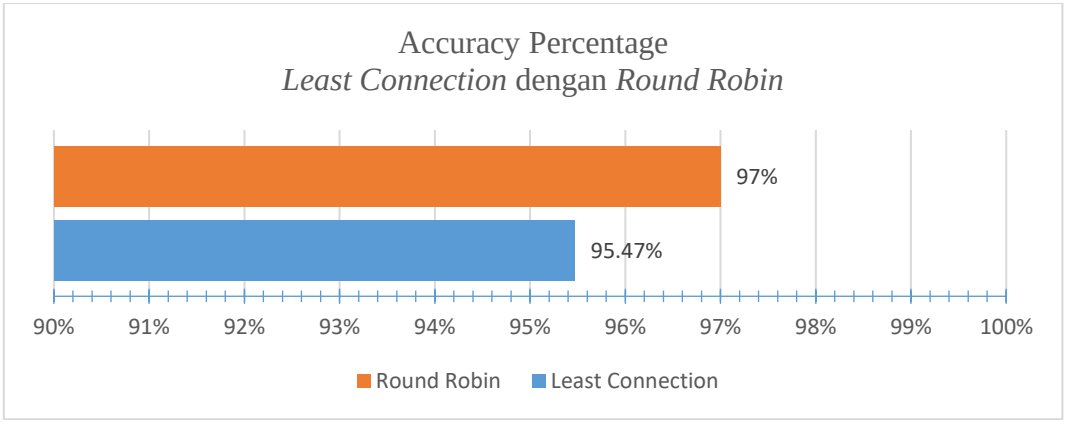

Figure 6. *Accuracy percentage graph*

Based on the total test results of all parameters, the accuracy value obtained using the least connection algorithm is 4.53%, while the error accuracy value obtained using the round robin algorithm is smaller, namely 3.00%.

From Figure 4.3 it can be seen that the loading of incoming requests can be divided by the server server to the three available worker servers. In round robin load balancing the load sharing is evenly divided into three servers, while at the least load balancing connection the load sharing is divided based on the least active connections. There is a difference in the percentage level of accuracy up to 1.46% between testing using the round robin algorithm and testing the least connection. It can be seen in Figure 4.3 that the load balancing system with the round robin algorithm has a better performance in distributing the load.

CONCLUSION

Based on the results of this study, the application of the hap Roxy load balancer method using the least connection algorithm resulted in an accuracy rate of 95.47%, while testing using the round robin algorithm was able to produce a greater level of accuracy, namely 97%.

REFERENCES

Adiputra, F. (2015). Containers and dockers: virtualization techniques for managing multiple web applications. *Simantec Journal, 4*(3), 167-176.

Bik, Fadhulloh Romadlon., Asmunin. 2017. Docker Implementation For Management of Multiple Web Applications. *Journal of Informatics Management,* 7(2), 46 - 50.

Novia, I. (2019). Load Balancer Implementation With Lightweight Virtualization Using Docker For Video On Demand Services. *E-Proceeding of Engineering,* 6(1), 802-809.

Kahanwal, B., Singh, T.P. (2012). The Distributed Computing Paradigms: P2P, Grid, Cluster, Cloud, and Jungle. *International Journal of Latest Research in Science and Technology,* 1(2), 183-187.

Kaur, K., & Rai, A.K. (2014). A Comparative Analysis: Grid, Cluster and Cloud Computing. *International Journal of Advanced Research in Computer and Communication Engineering,* 3(3), 5730- 5734.

Tanjung, P.K. (2017). Implementation and Analysis of Computer Cluster System Using Docker Virtualization. *e-Proceeding of Engineering,* 4(3), 3548.

Moilanen, M. (2018). *Deploying an Application using Docker and Kubernetes.* Thesis, Oulu University of Applied Sciences, Isandia.

Ruest, D., & Ruest, N. (2009). *Virtualization a Beginner's Guide:* McGraw Hill. English.

Prime, Cape. (2017). *Implementation and Analysis of Computer Clustering System Using Docker Virtualization,* Journal, Telkom University, Bandung.

Primary, Rivaldy Arif. (2018). Implementation of a Cluster Webserver Using Load Balancing Methods on Docker, Lxc, and Lxd Containers. *e-Proceeding of Engineering,* 5(3), 5028.

Son, Ridho Habi., & Sugeng, W. 2016. Implementation of Server Clusters on the Raspberry Pi Using the Load Balancing Method. *Journal of Education and Informatics Research (JEPIN).*

Shinde, S.S., & Chavan, A.R. (2014). Isolation of Mangiferin from Different Varieties of Mangifera Indica Dried Leaves. *International Journal of Scientific and Engineering Research,* 5(6), 928-934.