ELKOMIKA: Jurnal Teknik Energi Elektrik, Teknik Telekomunikasi, & Teknik Elektronika

ISSN(p): 2338-8323 | ISSN(e): 2459-9638 | Vol. 11 | No. 2 | Halaman 439 - 450
DOI : http://dx.doi.org/10.26760/elkomika.v11i2.439 April 2023

Analysis of Load Balancing Least Connection and Shortest Expected Delay Algorithm for Web Server Using Kube-Proxy on Kubernetes

MUHAMMAD AKBAR IBNU FARHAN PUTRA SUJARWO, ISTIKMAL, ARIF INDRA IRAWAN

Fakultas Teknik Elektro, Universitas Telkom, Bandung
Email: maibnufarhan@student.telkomuniversity.ac.id

Received 8 Februari 2023 **|** *Revised* 28 Februari 2023 **|** *Accepted* 22 Maret 2023

ABSTRAK

Penelitian ini membandingkan dan menganalisis kinerja algoritma load balancing shortest expected delay dan Least Connection untuk Web Server menggunakan Kube-Proxy di Kubernetes. Peningkatan jumlah pengguna dan perubahan jumlah node pekerja yang mempengaruhi kinerja system menjadi focus penelitian ini. Hasil penelitian menunjukkan bahwa pada parameter elapsed time, shortest expected delay dengan memiliki waktu yaitu 216.295 ms. Dalam waktu pemrosesan server, shortest expected delay lebih baik dengan menghasilkan 214.257ms. Throughput saat menggunakan algoritma shortest expected delay lebih besar, rata-rata 560.256 paket/detik. Algoritma Least Connection lebih baik dengan memiliki efisiensi 35,24% dalam hal penggunaan CPU, dibandingkan shortest expected delay. Meningkatkan klaster dari dua node pekerja menjadi empat node pekerja menghasilkan pengurangan waktu pemrosesan server yang signifikan, yang berarti penyeimbangan beban menggunakan 4 node pekerja lebih efektif.

***Kata kunci**: komputasi awan, load balancing, Kubernetes, Web Server.*

ABSTRACT

This research compares and analyzes the performance of the load balancing algorithm Shortest Expected Delay and Least Connection for web server using Kube-Proxy on Kubernetes. The increasing number of users and changes in the number of worker nodes that affect system performance are the focus of this research. The results show that in elapsed time parameter, shortest expected delay has the time, which is 216.295ms. In server processing time, the shortest expected delay is better by producing 214.257ms. Throughput when using the shortest expected delay algorithm is greater, an average of 560,256 packets/second. The Least Connection algorithm is better with an efficiency of 35.24% in terms of CPU usage, compared to the shortest expected delay. Upgrading the cluster from two worker nodes to four worker nodes resulted in a significant reduction in server processing time, which meant more effective load balancing using 4 worker nodes.

***Keywords**: cloud computing, load balancing, Kubernetes, Web Server.*

1. INTRODUCTION

Cloud computing has had an increasing amount of interest in the technology industries over the past few years. Newer generation companies use cloud computing as their core technology **(Ferreira & Sinnott, 2019)**. Cloud computing provides virtualization technology that shares computing resources like hardware, software, storage space, operating system, and infrastructure over the internet **(Bhatt, 2012)**. One such core cloud computing technology gaining more traction over the years is container-based virtualization **(Portworx, 2019)**. Container-based virtualization is an operating system-level virtualization approach with less overhead than hypervisor virtualization **(Nakagawa & Oikawa, 2016)** so it can help industries by having flexibility and scalability features which create significant cost savings **(Dewi et al., 2019)**.

In container-based virtualization, instead of running a whole Virtual Machine (VM) in the operating system (OS), containers provide isolation system resources like processes, file systems, and networks to run at the host OS level **(Ferreira et al., 2019)**. Many organizations and developers around the world popularly use Kubernetes. Kubernetes helps developers by supporting scalability, which means deploying more web servers in containers is easy to do **(He, 2020) (Raj et al., 2022) (Mudrikah et al., 2022)**. According to **(Gawel et al., 2019)**, Kubernetes meets MANO specifications so that it can be widely used for many applications. One of many applications that uses container technology is web servers. Due to the high demand for the rise of information and media distribution, the website's traffic on the internet is increasing rapidly. This High traffic, combined with improper traffic load distribution, might cause the webserver to get overloaded and shut down **(Nancy et al., 2020) (Korobeinikova et al., 2022)**. Load balancing is needed to spread workloads into many web server containers to resolve this problem.

Load balancing divides the work of two or more computers so that it may be completed more quickly and efficiently using the capabilities of each machine **(Deepa & Cheelu, 2017)**. Load balancing minimizes overload by evenly distributing workloads to every node **(Sivagami & Easwarakumar, 2018)**. **(Dua et al., 2020)** suggest an alternate technique for job scheduling in load balancing that sets up clusters for specific types of tasks (real-time, data-intensive, Etc.). Labels have been assigned to each work in order to categorize them. The program is then modified to incorporate load-balancing strategies via task migration to achieve a good result. There are various load-balancing tools and methods available **(Lina et al., 2000)**. Hence they are working for the same purpose. This paper uses Kubernetes as a container orchestration tool, founded by Google in 2014. Kubernetes has its built-in load-balancing component called Kube-proxy, where all networking, including load-balancing configurations, is done at Kube-proxy **(Li, 2019).**

The implementation of Kubernetes is also carried out in many applications, such as in the telecommunications **(Rao et al., 2021)** and transportation sectors **(Choi & Kim, 2021)** so research on the development of Kubernetes continues to this day. One example is the research conducted by **(Hidayah et al., 2019)**, aiming to implement load balancing on a web server. However, it uses OpenStack's Load Balancing as a Service (LBaaS) component as a load-balancing tool for the web server. The result of previous research stated that the performance of load balancing on the web server is better than only having a single server without load balancing tools on it. Central Load Balancer, proposed by **(Kaur & Sharma, 2018)**, has the formula to compute the load-leveling of virtual products using a reasoning data center. The results show that the method performs better insert leveling in large-scale reasoning computing environments than previous balancing techniques. **(Tian et al., 2022)** proposes a Kubernetes edge-powered vision-based navigation assistance system. This system can instantly scale a sufficient number of instances to adapt to changing requested traffic of robotic vehicles so that the number of instances can be auto-scaled on demand if the requested traffic does not match the current traffic. This research aims to implement load balancing on web servers in Kubernetes containers, conduct load and scalability tests, and analyze the performance based on its throughput, elapsed time, server processing time, and CPU utilization parameters**. (Hu & Wang, 2021)** proposed the pod replica prediction, which is an auto scaler based on the HPA (Horizontal Pod Autoscaler) function. This HPA collects data from each container and compares it to a threshold to get a better auto-scalar response time. **(Muddinagiri et al., 2019)** proposed a collaboration between docker and Kubernetes on a hybrid cloud architecture system. Docker processes the container orchestration tool created to ensure that all the configuration and management for Docker containers are successfully set up on-premises before deploying to the cloud. However, previous studies have not carried out load balancing tests on changes in the number of worker nodes, especially in load balancing shortest expected delay and Least Connection algorithm for Web Server.

Therefore, in this research we upgraded the cluster from two worker nodes to four worker nodes and compare the performance of load balancing shortest expected delay and Least Connection algorithm for Web Server Using Kube-Proxy on Kubernetes. Web server often suffers overloading caused by the high number of requests it gets every second. Web servers must be always available possible. Therefore, web servers need to run on a system with reliable availability. Developers use Kubernetes as their core technology because it gives more flexibility and is much cheaper in production and development. Load balancing is needed to prevent any web server overloading, and Kubernetes provides load balancing features located in the Kube-proxy component. With various load balancing algorithms available, this research focuses on the least connection and shortest expected delay algorithm. Load tests are conducted to acknowledge which of the fore mentioned load-balancing algorithms is best to use. To perceive the effect of scalability on web servers' performance, this research scales up the number of worker node for web servers on Kubernetes and compares their performance.

2. RESEARCH METHOD

This section describes the connection between using different load balancing algorithm with web server’s performance. There are various load balancing algorithms that can be used on Kubernetes cluster, in this research, we are using least connection algorithm **(Zhu et al., 2018)** and shortest expected delay algorithm **(Lui et al., 1995)**. The two mentioned algorithms are categorized as the dynamic load balancing algorithm, where the load balancer periodically searches and chooses the web server that has designated attributes. Dynamic load balancing algorithms are considered to be complex but have better fault tolerance **(Kumar & Rana, 2015)**.

Based on **(Hidayah et al., 2019)**, it stated that web server using load balancer has better performance compared to single web server. By having load balancer and cluster, high availability is guaranteed. The previous research used two web servers operated in Openstack environment. In this research, we constructed Kubernetes cluster which consists of one master node and two worker nodes.

There are two experiment scenarios in this research; Kubernetes cluster consisting of one master node and two worker nodes to test the comparison of performance between least connection and shortest expected delay algorithm. The second scenario is comparing the performance load balancer on Kubernetes cluster between two worker nodes and four worker nodes. The topology of the Kubernetes cluster is shown in Fig 1.

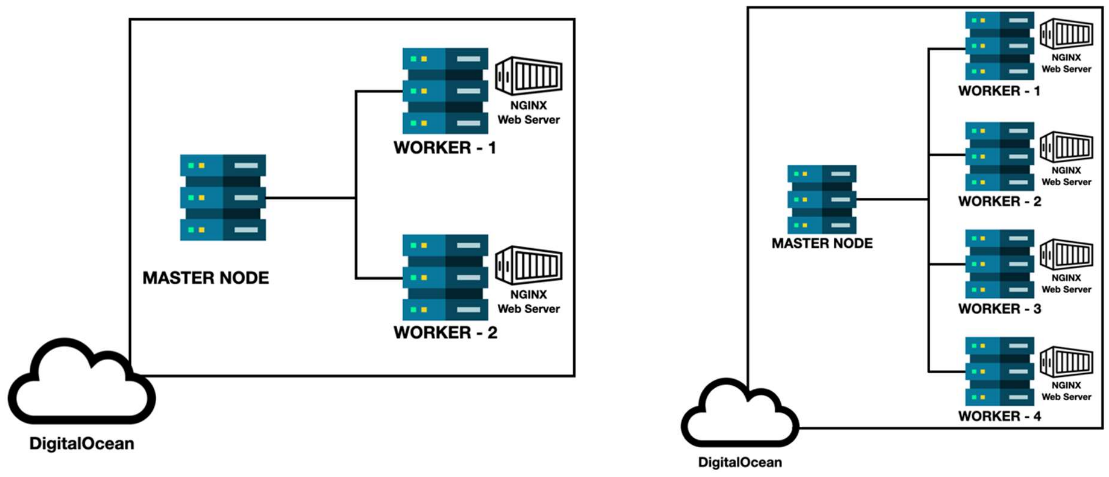

Figure 1. Topology of Kubernetes Cluster

As shown in Fig 1, the NGINX web servers are deployed by the master node into every worker node evenly. The master node focuses on controlling the worker nodes and manage the load balancing between the worker nodes. The Kubernetes clusters are operated on DigitalOcean platform, and every node is in the form of Droplets, DigitalOcean's virtual machine. The specifications of each Droplets are shown in Table 1.

Table 1. Specifications of Droplets

	CPU	Memory (GB)	Disk (GB)	IP Address
Master Node	2	4	60	128.199.131.202
Worker-1	1	2	50	178.128.50.73
Worker-2	1	2	50	178.128.85.86
Worker-3	1	2	50	167.99.72.29
Worker-4	1	2	50	178.128.105.210

Table 2. Simulation Measurements

Elapsed Time	ms
Server Processing Time	ms
Throughput	packets/second
CPU Utilization	%

In order to examine the performance of Kubernetes cluster with the fore mentioned scenarios, The Kubernetes clusters are tested by using Apache Jmeter that generates a huge number of http requests to the web server, and the result of the test is shown on the Jmeter application. The load tests are done by generating 500, 1000, 1500, 2000, and 2500 packets to the web server, where the load balancer oversees distributing all of the workloads to the web server. There are four parameters that we looked at to analyze the best load balancing algorithm, the parameters are shown in Table 2.

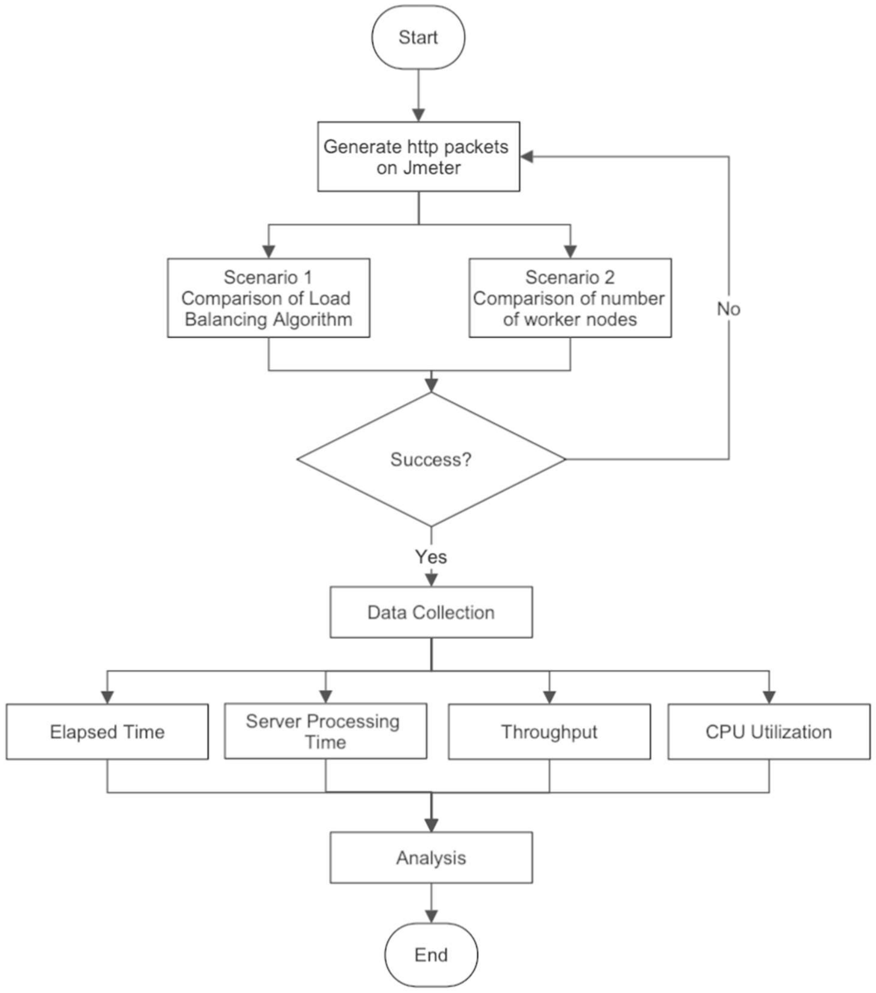
Start
Generate http packets on Jmeter
Scenario 1 Comparison of Load Balancing Algorithm
Scenario 2 Comparison of number of worker nodes
No
Success?
Yes
Data Collection
Elapsed Time
Server Processing Time
Throughput
CPU Utilization
Analysis
End

Figure 2. Research Method and Simulation Scenario

Figure 2 shows the research method and simulation scenario. We use Jmeter to generate http packet to do the load test. In the first scenario, we evaluate the load balancing algorithm in fixed number of worker with increasing the number of users. The second scenario is to see the performance of the load balancing algorithm when there is an increase in the number of users and the system makes changes by increasing the number of worker nodes. Each scenario is tested 20 times, and all the results are collected in the form of table and shown in the form of graphs.

3. RESULTS AND DISCUSSION

In this chapter, results of all the mentioned experiments are shown in graphs and the analysis are explained based on the experiment of load balancing of web server using Kube-proxy on Kubernetes cluster. This chapter highlighted the performance of the Kubernetes cluster in two different algorithms, which are least connection and shortest expected delay. On the first scenario, the Kubernetes cluster is only using two worker nodes, while comparing between least connection and shortest expected delay load balancing algorithm. The second scenario compares the performance of load balancing in Kubernetes cluster with two worker and four worker nodes.

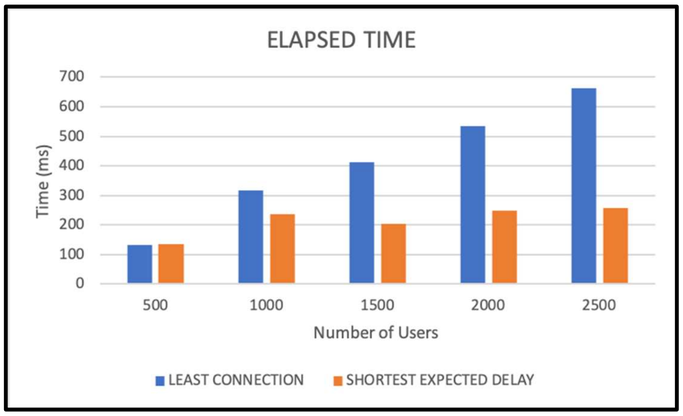

Figure 3. Elapsed Time Results on Kubernetes Cluster with Two Worker Nodes

In first scenario with elapsed parameter, the time from just before Jmeter sending the request to just after the last respond has been received. The result is shown in Figure 3, the elapsed time of shortest expected delay algorithm is faster, this is due to the mentioned algorithm selects worker nodes that has the shortest delay, therefore the time it takes for the web server to run the test is much faster. On average, the fastest algorithm is shortest expected delay, with elapsed time of 216.29 ms.

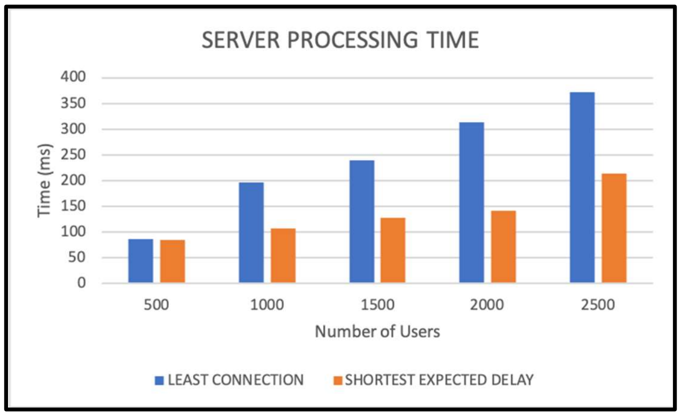

Figure 4. Server Processing Time Results on Kubernetes Cluster with Two Worker Nodes

Server processing time is the amount of time needed by the server to process each request. Server processing time is the result of subtraction between Latency time and Connect time. The average of the results is shown in Figure 4. The server processing time using shortest expected delay is significantly faster, means the time it takes for server to process request is faster. By using shortest expected delay, every request sent to the master node is distributed to the worker nodes that has low number of delays, so processing the next request is expected to be quicker. On average, shortest expected delay algorithm has faster server processing time, with 241.36 ms.

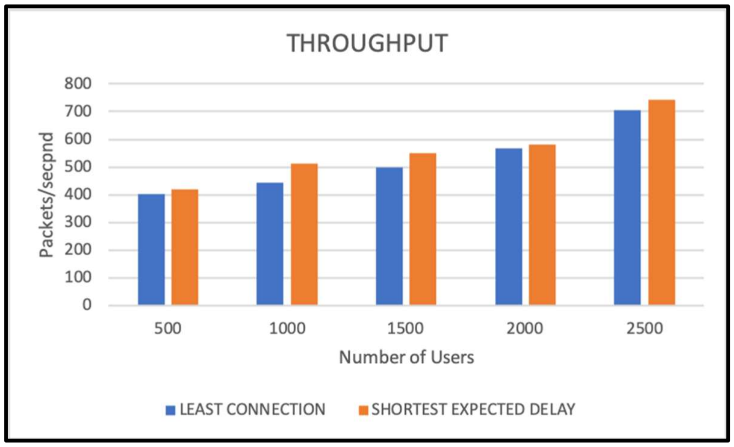

Figure 5. Throughput Results on Kubernetes Cluster With Two Worker Nodes

Throughput is calculated as number of requests per unit of time, in this case of load testing, the result is written in packets/second. The time is calculated from the start of the first sample until the end of the last sample. The results are shown in Figure 5, where the shortest expected delay algorithm has a higher amount of throughput. The high value of throughput means the amount of packet that can be transferred into the web server is high. On average, shortest expected delay algorithm has higher throughput with 560.256 packets/second.

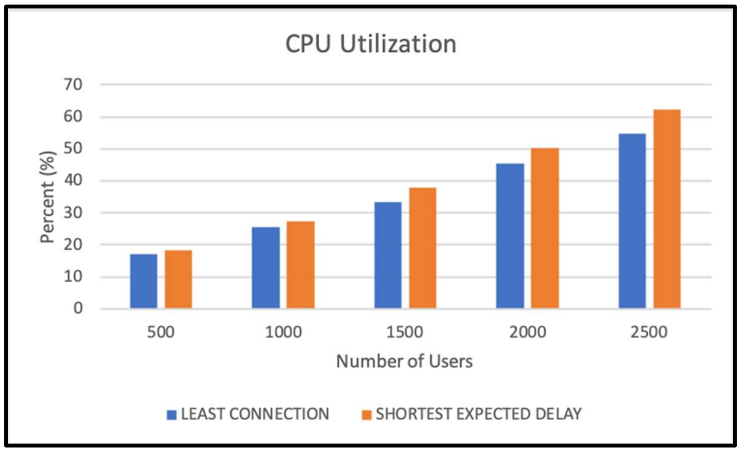

Figure 6. CPU Utilization of Kubernetes Cluster With Two Worker Nodes

In CPU utilization parameter, the performance of the web server’s CPU during load testing is being looked at. Th graph shows that by using shortest expected delay the web server is using more CPU, this caused by the web server are continuously calculating which node has the shortest delay. The results are written in percentages and shown in Figure 6, the bigger the number, the better. On average, least connection algorithm uses less CPU during load testing, with 35.24%.

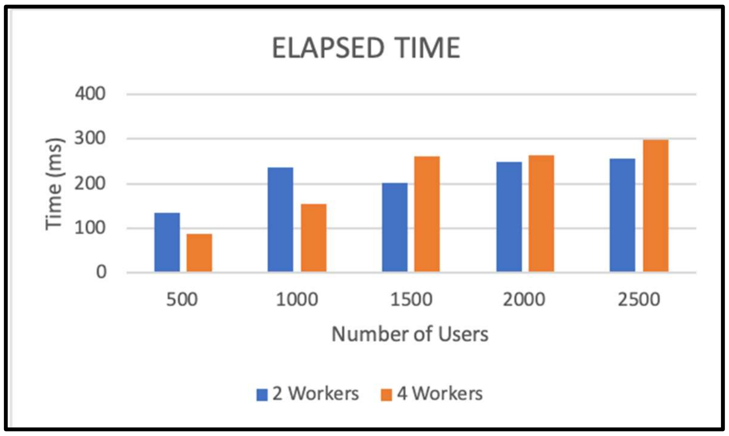

Figure 7. Elapsed Time Results on Kubernetes Cluster Using Shortest Expected Delay

The second scenario is comparing the performance of load balancing in Kubernetes cluster with two worker and four worker nodes with shortest expected delay algorithm. Based on the Figure 7, the elapsed time of having 4 worker nodes is generally higher, this is caused by the master needed to distribute every user request into 4 worker nodes which takes more time rather than distribute requests into 2 worker nodes.

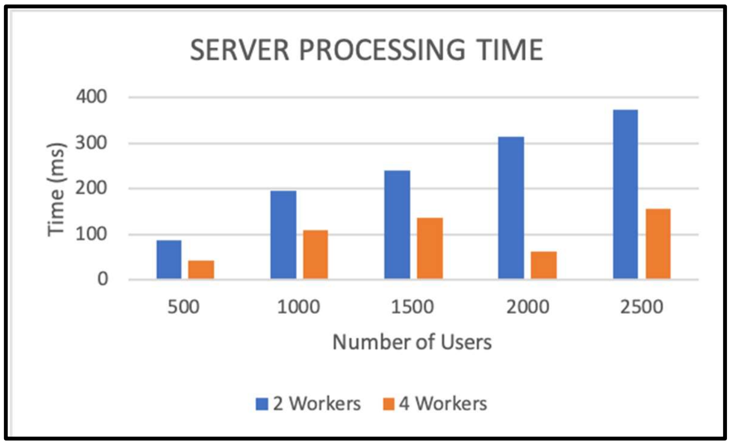

Figure 8. Elapsed Time Results on Kubernetes Cluster Using Shortest Expected Delay

The results are shown in Figure 8 The time it takes for web server with 4 worker nodes is significantly lower, this is caused by every user request are balanced into a greater number of workers compared the previous state which has only 2 worker nodes. The web server performs faster when the Kubernetes cluster has 4 worker nodes. On average, the server processing time of Kubernetes cluster with 4 worker nodes is 101.4 ms.

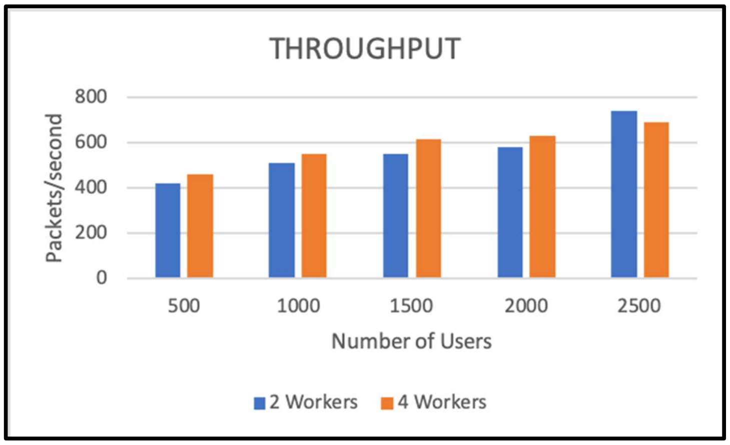

Figure 9. Elapsed Time Results on Kubernetes Cluster Using Shortest Expected Delay

The throughput of the Kubernetes cluster with 4 worker nodes is slightly higher on average compared to the cluster with 2 worker nodes. Overall, there are no significant variations of throughput between Kubernetes cluster with 2 worker nodes and with 4 worker nodes.

4. CONCLUSION

We had successfully implemented load balancing on the Kubernetes cluster for the web server and compared the performance of the least connection and shortest expected delay of the load balancing algorithm. We compared the performance of these two load balancer algorithms using two workers and four worker nodes. Based on the load test results, the shorter expected delay algorithm is the best load balancing algorithm because it gives better performance, faster elapsed time, faster server processing time, and higher throughput than the least connection algorithm. After we scaled up the Kubernetes cluster, having four worker nodes resulted in better performance in terms of faster server processing time and higher throughput. However, used four worker nodes has a slower elapsed time than used two. This happened because the master needed more time to calculate and distribute workloads into four worker nodes. This research gives a clear explanation that using the shortest expected delay algorithm and having more worker nodes results in better performance of load balancing on the Kubernetes cluster.

REFERENCES

Bhatt, D. (2011). A Revolution in Information Technology - Cloud Computing. *Walailak Journal of Science and Technology*, *9*(2), 107-113.

Choi, J. H., & Kim, S. W. (2021). Cloud-based ATC Platform Architecture Design Using Kubernetes. *IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, (pp. 1–2).

Deepa, T., & Cheelu, D. (2017). A comparative study of static and dynamic load balancing algorithms in cloud computing. *International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, (pp. 3375–3378).

Dewi, L. P., Noertjahyana, A., Palit, H. N., & Yedutun, K. (2019). Server Scalability Using Kubernetes. *Technology Innovation Management and Engineering Science*

International Conference (TIMES-ICON), (pp. 1–4).

Dua, A., Randive, S., Agarwal, A., & Kumar, N. (2020). Efficient Load balancing to serve Heterogeneous Requests in Clustered Systems using Kubernetes. *Consumer Communications & Networking Conference (CCNC)*, (pp. 1–2).

Ferreira, A. P., & Sinnott, R. (2019). A Performance Evaluation of Containers Running on Managed Kubernetes Services. *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, (pp. 199–208).

Gawel, M., & Zielinski, K. (2019). Analysis and Evaluation of Kubernetes Based NFV Management and Orchestration. *International Conference on Cloud Computing (CLOUD)*, (pp. 511–513).

He, Z. (2020). Novel Container Cloud Elastic Scaling Strategy based on Kubernetes. *Information Technology and Mechatronics Engineering Conference (ITOEC)*, (pp. 1400–1404).

Hidayah, I., Rendy Munadi, & Indrarini Dyah Irawati. (2019). Implementasi High-availability Web Server Menggunakan Load Balancing As A Service Pada Openstack Cloud. *EProceedings of Engineering*, *6* (3), 10278.

Hu, T., & Wang, Y. (2021). A Kubernetes Autoscaler Based on Pod Replicas Prediction. *2021 Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, (pp. 238–241).

Kaur, S., & Sharma, T. (2018). Efficient load balancing using improved central load balancing technique. *International Conference on Inventive Systems and Control (ICISC)*, (pp. 1–5).

Korobeinikova, T., Maidaniuk, V., Romanyuk, O., Chekhmestruk, R., Romanyuk, O., & Romanyuk, S. (2022). Web-applications Fault Tolerance and Autoscaling Provided by the Combined Method of Databases Scaling. *International Conference on Advanced Computer Information Technologies (ACIT)*, (pp. 27–32).

Kumar, S., & Rana, D. S. (2015). Various Dynamic Load Balancing Algorithms in Cloud Environment: A Survey. *International Journal of Computer Applications*, *129*(6), 14-19.

Li, R. (2019). *Load balancing strategies in Kubernetes*. ITNEXT. Retrieved from https://itnext.io/load-balancing-strategies-in-kubernetes-6213a5becd66

Lina, L., Longguo, L., & Xinyu, Y. (2000). An agent-based load balancing mechanism: PLRM using Java. *International Conference on Technology of Object-Oriented Languages and Systems*, (pp. 176–181).

Lui, J. C. S., Muntz, R. R., & Towsley, D. (1995). Bounding the mean response time of the minimum expected delay routing policy: an algorithmic approach. *IEEE Transactions on Computers*, *44*(12), 1371–1382.

Muddinagiri, R., Ambavane, S., & Bayas, S. (2019). Self-Hosted Kubernetes: Deploying Docker Containers Locally With Minikube. *International Conference on Innovative Trends and Advances in Engineering and Technology (ICITAET)*, (pp. 239–243).

Mudrikah, F. Z. A., Istikmal and B. Aditya. (2022). Design of a Geographic Information System for Forest and Land Fires Based on a Real-Time Database on Microservices Infrastructure. *International Conference on Internet of Things and Intelligence Systems* (IoTaIS), (pp. 1-6).

Nakagawa, G., & Oikawa, S. (2016). Behavior-Based Memory Resource Management for Container-Based Virtualization. *Intl Conf on Applied Computing and Information Technology-Intl Conf on Computational Science-Intelligence and Applied Informatics-Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD)*, (pp. 213–217).

Nancy, J. J., Mani S., T., Rohith, S., Saranraj, S., & Vigneswaran, T. (2020). Load Balancing using Load Sharing Technique in Distribution System. *International Conference on Advanced Computing and Communication Systems (ICACCS)*, (pp. 791–794).

Portworx, & Security Aqua. (2019). *Container Adoption Survey*. Retrieved from https://portworx.com/wp-content/uploads/2019/05/2019-container-adoption-survey.pdf?aliId=eyJpIjoiUGowN3c1dWhWN3ZFMlwvZkUiLCJ0IjoiVmFTaTFqSzZQU3B6VVNrVk1CcVBtdz09In0%253D

Raj, P., Vanga, S., & Chaudhary, A. (2022). Setting up a Kubernetes Cluster using Azure Kubernetes Service. In *Cloud-native Computing: How to Design, Develop, and Secure Microservices and Event-Driven Applications* (pp. 203–221). John Wiley & Sons, Inc.

Rao, S. K. N., Paganelli, F., & Morton, A. (2021). Benchmarking Kubernetes Container-Networking for Telco Usecases. *IEEE Global Communications Conference (GLOBECOM)*, (pp. 1–7).

Sivagami, V. M., & K.S.EaswaraKumar. (2018). Performance analysis of Load balancing algorithms using LBaaS. *International Journal Of Research And Analytical Reviews*, *9*(4), 140–150.

Tian, J. S., Huang, S.-C., Yang, S.-R., & Lin, P. (2022). Kubernetes Edge-Powered Vision-Based Navigation Assistance System for Robotic Vehicles. *International Wireless Communications and Mobile Computing (IWCMC)*, (pp. 457–462).

Zhu, L., Cui, J., & Xiong, G. (2018). Improved dynamic load balancing algorithm based on

Least-Connection Scheduling. *Information Technology and Mechatronics Engineering Conference (ITOEC)*, (pp. 1858–1862).