Web Server Load Balancing Mechanism with Least Connection Algorithm and Multi-Agent System

Afiyah Rifkha Rahmika[1*], Zulkifli Tahir[2], Ady Wahyudi Paundu[3], and Zahir Zainuddin[4]
[1−4]Informatics Engineering Department, Universitas Hasanuddin
Makassar 90245, Indonesia
Email: [1]rahmikaar19d@student.unhas.ac.id, [2]zulkifli@unhas.ac.id, [3]adywp@unhas.ac.id,
[4]zahir@unhas.ac.id

***Abstract*—Demands for information over the Internet massively increase through the continuous expansion of website applications. Therefore, generating powerful and efficient server architecture for web servers is a must to satisfy Internet users and avoid the overloaded system. The research focuses on developing a new mechanism for load balancing to distribute incoming HTTP requests in website applications by combining the Least Connection algorithm and Multi-Agent System (LC-MAS). The proposed mechanism distributes the request based on load condition and the fewest number of active connections. The research applies virtualization technology to build servers on this proposed mechanism. The architecture is built inside a physical server with Proxmox as virtualization management and Linux Debian 7.11 as an operating system. Then, the research is tested in two scenarios (LC-MAS and LC) using 500, 1,000, and 1,500 requests. The performance of this proposed mechanism is measured through the values of average response time, throughput, and error percentage. The results show that the proposed mechanism (LC-MAS) distributes the workload more equally than LC, with an average response time for 1,500 requests of 1338.8 milliseconds, 20.07% error, and 125 transactions per second. The LC-MAS makes the website application performance much better when the request increases. The LC-MAS helps in the utilization of system resources and improves system robustness.**

***Index Terms*—Web Server, Load Balancing Mechanism, Least Connection Algorithm, Multi-Agent System**

I. Introduction

THE massive use of the Internet has become a new habit in this digital era. A website is one of the Internet services that is widely involved in every aspect of human daily lives, with millions of users accessing it from everywhere. The website presents various kinds of information content needed by children to adults. Unfortunately, the number of users accessing websites worldwide causes a web server that provides a website to down or overload [1].

Websites, such as e-commerce services, even receive millions of connections from Internet users simultaneously in one day. Due to some conditions, the website is frequently visited. For example, it happens when e-commerce provides big promotions at the end of the year or on certain dates or new students register in the information system platform.

As a result, web servers have become a crucial part of Internet infrastructure. Building a proper, good, and reliable web server architecture is important to manage a huge increase in traffic and requests to protect the business from the risk of system failure, transactional data loss, and error messages process. Being down for a brief moment can make the website application lose a lot of money. It becomes a serious thing [2]. The workload of a web server increases as the number of website visitors increases, leading to an overloading condition. This situation triggers the need for load balancing, which is a method that equally distributes the loads among the available virtual machines [3].

Load balancing can be applied to balance the load between servers. It is a technology to distribute the workload of service on a cluster of servers or network devices in a balanced way when there is a request from the user. This technique can increase the throughput, reduce response time, and give better resource utilization so the system can achieve the best performance. Besides, if one server fails, the system can still run using other available servers because many servers work to handle the requests [4]. Load balancing has methods and algorithms for balancing the load on the system to maintain and consume the available system capabilities. Hence, it optimizes service performance

Received: Aug. 16, 2022; received in revised form: Jan. 10, 2023; accepted: Jan. 11, 2023; available online: Sept. 18, 2023.
*Corresponding Author

to satisfy the demand. Load balancing algorithms are categorized as static and dynamic [5, 6].

However, one of the problems in load balancing is that it does not provide any method to monitor the condition of server resources by default. Resource monitoring is needed when developing a system with a large number of requests [7]. Monitoring can be done by checking the condition of server resources, e.g., memory usage, CPU percentage, disk performance metrics, and others. Additionally, server resource monitoring needs to use other methods that combine with load balancing algorithms to distribute requests equally and reach better resource utilization.

Monitoring server conditions in developing a load balancing mechanism has been carried out in the research model of load balancing using a reliable algorithm with a Multi-Agent System (MAS) [8]. The model is tested by sending a total of 1,800 HTTP requests for 10 s to be distributed to only three web servers. The total time interval for each request is 0.005 seconds (10/1800) which is unreasonable for a web server to handle all requests simultaneously. The results of the test parameters are still not good because they sacrifice response time and the number of errors.

Previous research proposes a network design to measure the performance of web server load balancing using LC and Round-Robin algorithm to each load balancer for performance comparison [4]. HAProxy with default KeepAlive (KA) configuration and NGINX is applied as the load balancer. Then, the environment is built using three computers as the web server and one computer as a load balancer and tested with several examinations, i.e., load and stress test, time test, click test, and benchmark test. The LC algorithm produces better results for tests like request per second and transfer rate (KB/sec).

Another previous study discusses the efficiency of three different load balancing algorithms, such as Round Robin, LC, and Least Loaded [9]. It uses the OPNET tool to simulate the proposed system architecture that consists of 1 load balancer, 8 HTTP servers, and 112 users. The result shows that the Least Loaded algorithm consumes more CPU load than others. The LC algorithm distributes the request more fairly than others to balance the load among servers.

In previous research, the developed load balancing algorithm addresses the load imbalance issue among agents in MAS using Software Performance Engineering (SPE) approach [10]. The proposed algorithm is built using Java Agent Development Framework (JADE) as a MAS development tool, and the results are obtained by considering the proposed algorithm and using the first-come, first-serve method. It is observed that the response time of the agents has improved. Agents work with their maximum capability. The agents in MAS utilize the system resources, such as CPU capacity, memory, and bandwidth.

Previous research investigates the performance of several load balancing tools like HAProxy and NGINX over a cloud computing environment and finds that congestion is one of its main problems [11]. The LC algorithm is tested through HAProxy and NGIN tools to examine its behavior. The results show that HAProxy is faster than NGINX in terms of handling requests, but NGINX provides less CPU utilization.

Previous research also builds a web server service using Desktop Personal Computer (PC) to replace the server computer task carried [12]. The environment is built using three Desktop PCs as a load balancer. It uses two different algorithms applied to the load balancer. It is tested twice. The first test uses a common algorithm, i.e., the Round Robin algorithm, for the load balancer. The second test is done by using the LC algorithm. Several metrics measure performance, such as request loss, successful transactions, throughput, transaction rate, longest transactions, and response time. The results show that the task of a computer server can be replaced by a Desktop PC at a cheaper cost. The LC algorithm is more reliable than others in terms of throughput, successful transaction, request loss, and longest transaction.

The model of load balancing using a reliable algorithm with MAS proposes a reliable load balancing system using the Least Time First Byte (LFB) algorithm combined with the resource information from a mobile agent [8]. There are two testing scenarios in the previous research. The first scenario applies the Weighted Least Connection (WLC) algorithm, and the second uses the LFB algorithm to distribute the requests. The system is tested with 1,800 requests for 10 seconds. The results show that the LFB algorithm is more stable in dealing with many requests. The WLC algorithm is successfully completed within 78.99% of requests, while the LFB algorithm reaches 100% over 1,800 requests. LFB algorithm sacrifices the response time, throughput, and other parameters.

Previous research also proposes the Self-Based Load Balancing (SGA_LB) algorithm in a cloud environment [13]. It aims to reach proper balancing of workload on virtual machines by maximizing the availability, throughput, scalability, and reliability of the system. The algorithm comprises three agents: in-house, external, and migration. The agent concept is built with the JADE library in the CloudSim framework. The results deal with distributing the workload in terms of memory and utilization of resources in the cloud environment. When the MAS concept is used, it produces a better workload and considerable

improvement in overall performance and efficiency.

Another previous research proposes a dynamic load balancing algorithm by considering the server's static and dynamic load factor to calculate the weight of the closest server real-time performance parameters to calculate the server's load [14]. The composite load is calculated based on response time and number of connections of each node. The larger the weight of the node is, and the smaller the composite is, the greater probability it will be assigned in the request. It can be concluded that NGINZ's built-in algorithm has problems balancing the distribution load. The result shows that the proposed algorithm is better than NGINX's built-in two algorithms, such as Round Robin and LC, regarding response time and requests processed value.

Moreover, in previous studies, the number of virtual machines used is very small. Hence, the research adds more virtual machines so that more nodes can work together to handle HTTP requests. The Least Connection (LC) algorithm is chosen to eliminate the calculation process on the first byte of incoming HTTP requests as it becomes a trigger in increasing the response time of previous research.

The research proposes a load balancing mechanism to balance requests on the server by considering server resource conditions, i.e., a load of CPU and memory to produce better system performance. This mechanism combines the Least Connection algorithm and MAS (LC-MAS) method to distribute the requests equally to the server and monitor the server resource condition periodically to achieve maximum resource utilization. The proposed mechanism will distribute the requests based on load condition and the fewest number of active connections as the existing research does not consider resources condition in developing load balancing mechanisms. In the research, the performance of the proposed mechanism is compared with a system that uses LC only.

II. Research Method

A. Research Stages

The first stage collects some references related to problems on the previous web server load balancing mechanism and determines the focus on the performance parameters. The stages of research are described in Fig. A1 in Appendix. Parameter values are analyzed to determine whether the proposed mechanism is running. If not, it will be back to the second stage. The second stage prepares the mechanism environment to make sure that there are no deficiencies in the hardware and software that prevents the system from running properly. However, if the proposed mechanism runs properly, the next step is to take the parameter values to analyze and prepare reports.

Algorithm 1 Least Connection Algorithm Pseudocode

```
1.for (j=0; j<n; j++) {
2.    for (i=j+1; i<n; i++) {
3.     if (C(Si) < C(Sj))
4.         m=i;
5.     }
6.  return Sj;
7.}
8.return NULL;
```

B. Least Connection (LC) Algorithm

The load balancing mechanism proposed in the research uses the LC algorithm to distribute the request load to the servers. This algorithm divides the request load based on the number of active connections in real-time. It chooses the servers with the smallest active connections. Pseudocode for the LC algorithm is shown in Algorithm 1.

The algorithm works with two loops. The first loop repeats the process as much as the number of incoming requests based on line 1. Meanwhile, the second loop repeats the process to count the number of active connections from each server based on line 2. Moreover, S represents a group of servers, and C is the number of connections from S at the moment. As for line 3, the algorithm compares the number of active connections and stores the result. Then, it chooses the fewest number of active connections from server S and returns the value to Sj. The Sj value is the current server with the fewest number of active connections. When there are no incoming requests, the value returned is NULL, or no process is running. However, the number of connections the server handles is inversely proportional to the load conditions, so this algorithm makes uneven load distribution. The researchers do not use NGINX's built-in LC algorithm.

C. Multi-Agent System (MAS)

The researchers use two types of agents with different functions in the research. The main agent functions to request information on backend server resources. Meanwhile, a reporting agent functions to receive the request and provide resource information to the main agent. Agents are built using JADE which is a Java library. The workflow of two types of agents can be seen in Fig. A2 in Appendix.

The main agent checks the number of active agents when it starts to run on the system. This behavior loops until the main agent stops running. Reporting agents on each backend server register themselves to the Directory (Yellow Pages). Yellow Pages information is then sent to the main agent to check how many

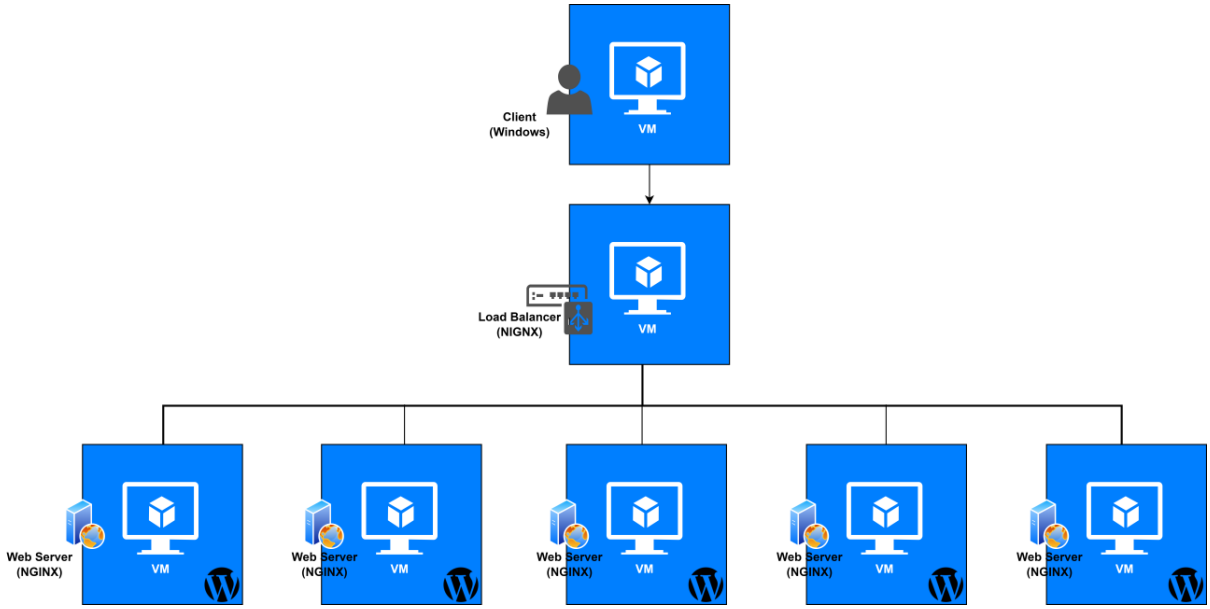

Fig. 1. Load balancing architecture.

reporting agents are active. Then, the main agent sends a request for resource information while the reporting agent is in charge of providing that information every 500 milliseconds to the main agent. Reporting agent starts to collect information about the percentage of CPU and memory usage from web servers and calculate the number of active connections. The number of active connections calculated by the reporting agent uses the LC algorithm, described in Algorithm 1, to reduce the delay when that process has to move to other virtual machines. After the information is complete, the reporting agent starts to calculate the load of a web server by multiplying the CPU and memory percentages and dividing those by the smallest number of active connections. The formula is written in Fig. A2 in Appendix on the reporting agent's side.

D. Architecture

The researchers use virtualization technology to build servers in this proposed mechanism. The architecture is built inside a physical server with Proxmox as virtualization management and Linux Debian 7.11 as an operating system. Then, the researchers deploy virtual machines as servers.

Inside the virtual machines, the researchers create two types of service: one service for the load balancer and five services for the web server. Web servers are built using NGINX. Web server service contains standard Content Management System (CMS) WordPress application content. Figure 1 shows the architecture in the research. The researchers also use five virtual machines to add nodes that work in parallel to handle incoming requests simultaneously to shorten execution time. Existing research mostly only uses a relatively small number of virtual machines, such as three or four.

E. Testing Scenarios

The proposed load balancing mechanism is tested with two scenarios. The researchers use Apache JMeter as a tool to create testing plans. Apache JMeter generates HTTP traffic to the web server cluster. Apache JMeter sends 500, 1,000, and 1,500 HTTP requests to both scenarios. Each scenario has a ramp-up time of 10 seconds for each number of requests.

The first scenario tests the load balancing mechanism using the LC algorithm combined with MAS (LC-MAS). This scenario uses six agents: one main agent and five reporting agents. The main agent is located on the load balancer server while the reporting agent is on each backend server. Figure A3 in Appendix shows the flowchart for the LC-MAS scenario.

The main agent categorizes the resource load conditions as normal or overloaded based on the backend server resource information sent by the reporting agent. It is the first process in this scenario. The load is calculated based on CPU and memory percentage values and divided by the number of incoming requests. The

formula is as follows [15].

$$\text{load} = \frac{\%\text{CPU} \times \%\text{Memory}}{\text{Nrequest}}.$$

It must not exceed the threshold value that has been set, i.e., 80%. This value is agreed as a "sweet spot" for monitoring the server resource when it comes to load problems. Load is categorized as normal when the value is below or equal to 80%. It is an overload when the value is higher than 80%. Then, backend servers with normal categories are grouped to count the number of active connections they are currently handling. The backend server IP address with the smallest number of active connections is sent to the load balancer server to execute the requests.

The load balancer server is only tasked to forward the requests to the elected backend server. The entire calculation and decision-making process is carried out by the agents. It aims to reduce the waiting time caused by several processes when calculating the number of active connections that must move to the load balancer virtual machine.

The second scenario tests the load balancing mechanism using the LC algorithm only to distribute the requests. The LC algorithm distributes requests to the backend server with the smallest number of active connections. The load balancer counts the number of active connections from five backend servers and compares those with each other. The smallest number of active connections is selected to execute the request. Figure 2 shows the flowchart for the LC scenario.

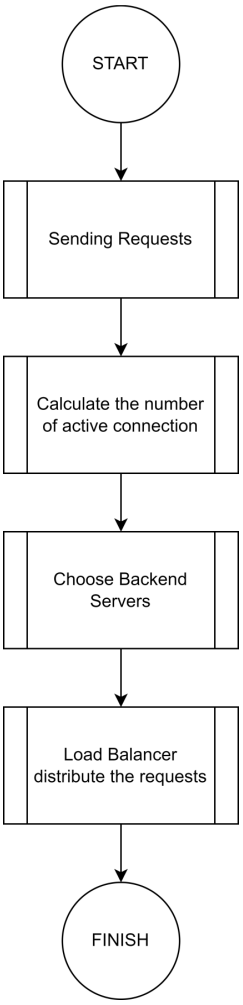

Fig. 2. Least Connection (LC) scenario.

III. Results and Discussion

Testing parameters in the research are the values of response time, throughput, and error percentage. These parameters analyze the performance of the proposed mechanism and represent the effectiveness of the systems after performing load balancing. The result of testing parameters can describe the performance of the system in terms of resilience in dealing with a large number of requests.

The research focuses on load testing. A load test is a performance test that checks how systems function under a large number of concurrent virtual users performing transactions over a certain period. In other words, the test measures how systems handle heavy load volumes.

A. Average Response Time

Response time is calculated starting when the request is sent until it is finished to execute. It represents the minimum time of a system that applies a specific load balancing algorithm to respond. The response time value should be small for better performance of the system. Average response time is the total response time divided by the number of incoming requests. This parameter is calculated in milliseconds. Figure 3 shows the graph of the average response time from two testing scenarios.

The average response time for LC is smaller than LC-MAS when executing 500 and 1,000 requests. The difference is quite large. It is 306.61 milliseconds for 500 requests and 484.62 milliseconds for 1,000 requests. Several factors affect higher average response time in LC-MAS. First, the backend server that executes requests always changes based on load conditions and the smallest number of active connections. This activity makes the system very busy, so a delay affects the time for the following request to be executed. Second, log data show that requests have response times above 1,000 milliseconds executed by the backend server

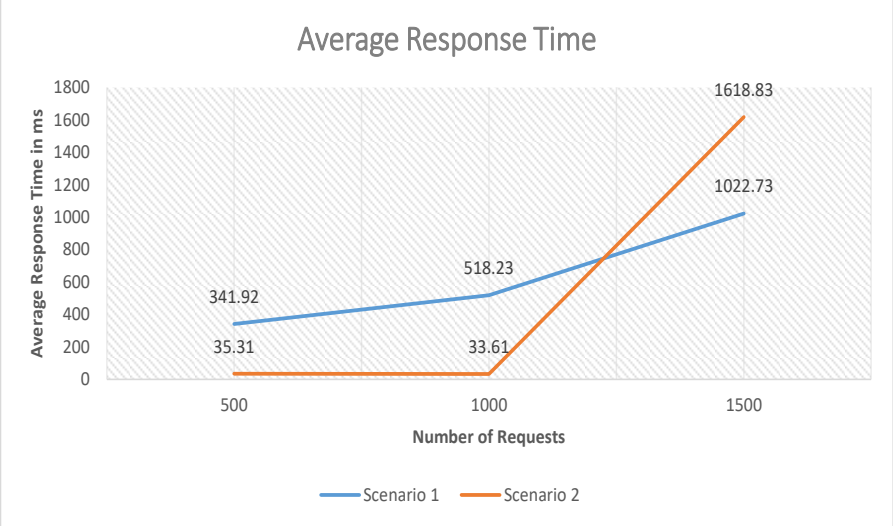

Fig. 3. Average response time graph.

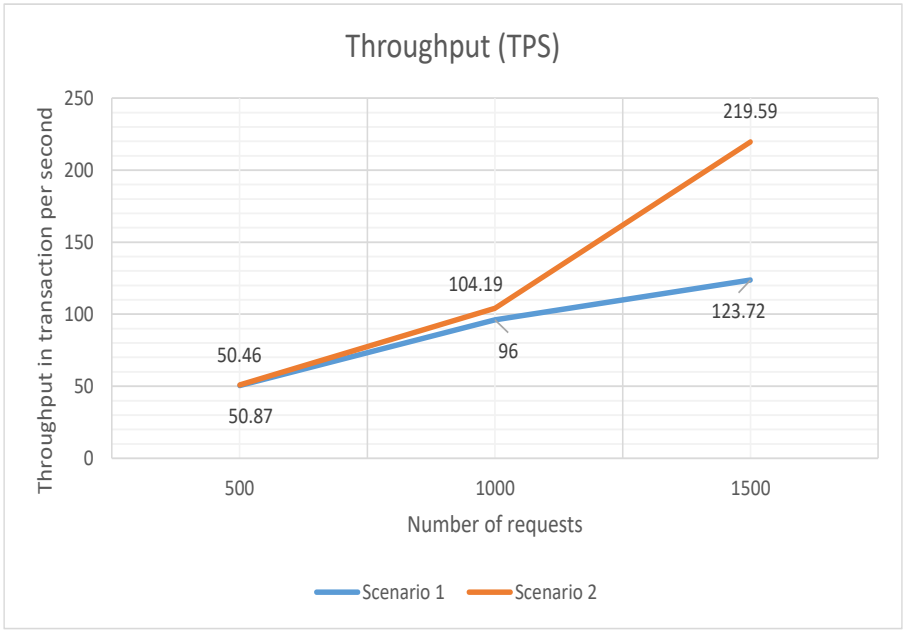

Fig. 4. Throughput graph.

with the highest load. Meanwhile, requests that have response times below 500 milliseconds are executed by a new backend server with a small load and a number of connections.

Nevertheless, the average response time of LC-MAS is smaller than LC when executing 1,500 requests. A higher average response time for the LC scenario caused by the load balancer distributes the requests only based on the currently active connections. It is without considering whether the load condition is normal or overloaded.

B. Throughput (TPS)

Throughput testing aims to find out how many transactions or requests can be processed by the server's amount of time. In the research, throughput value information provides knowledge of how tough the servers are in processing requests in one second. The time is calculated from the beginning of the first request to the end of the last request. It also includes the

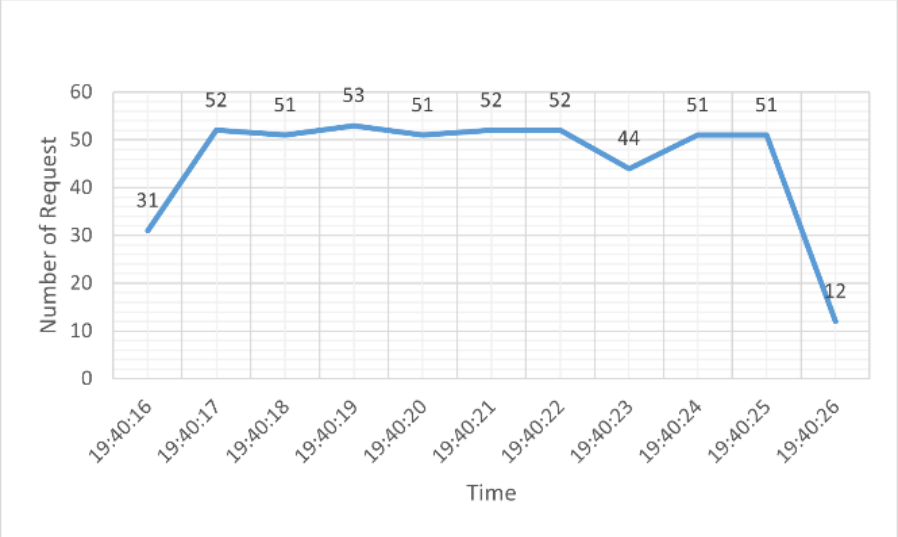

Fig. 5. Least Connection algorithm and Multi-Agent System (LC-MAS) transaction per second for 500 requests.

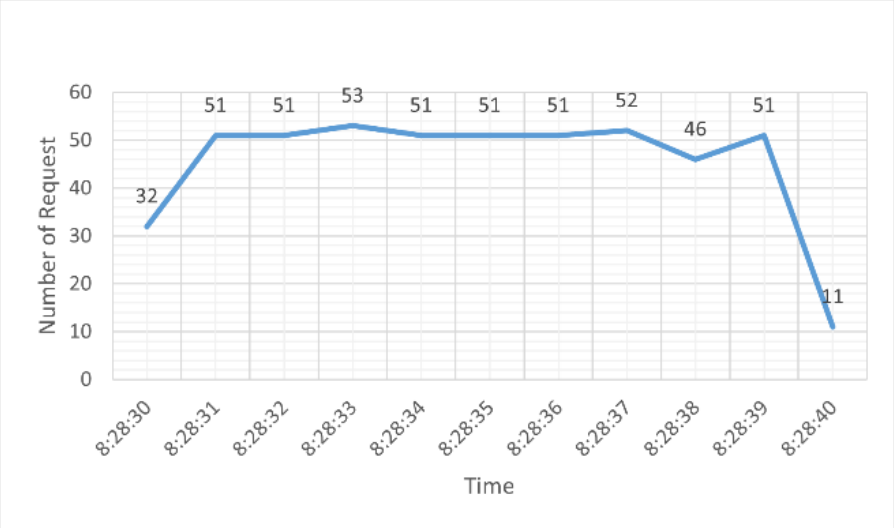

Fig. 6. Least Connection (LC) transaction per second for 500 requests.

interval between the requests. A greater throughput value indicates that the system works well. Figure 4 shows the throughput graph of both scenarios.

The throughput value of LC is indeed greater than LC-MAS. Backend servers can serve more requests in one second, so the duration needed to execute entire requests is shorter than LC-MAS. However, high throughput values lead to higher response time. The number of requests per second for each scenario is shown as follows. First, it is for 500 requests. The ramp-up time set to send 500 requests is 10 seconds, where the time interval for each request is 0.02 seconds (10/500). Figures 5 and 6 show the number of requests per second that the backend server of each scenario can execute. Based on the graphs, the average number of requests executed by the two scenarios is 50 per second.

Second, the ramp-up time set to send 1,000 requests is 10 seconds, where the time interval for each request is 0.01 seconds (10/1,000). Figures 7 and 8 show the number of requests per second that the backend server of each scenario can execute. Based on the graphs, the average number of requests executed by the two scenarios is 100 per second.

Third, it has 1,500 requests. The ramp-up time set to send 1,500 requests is 10 seconds, with the

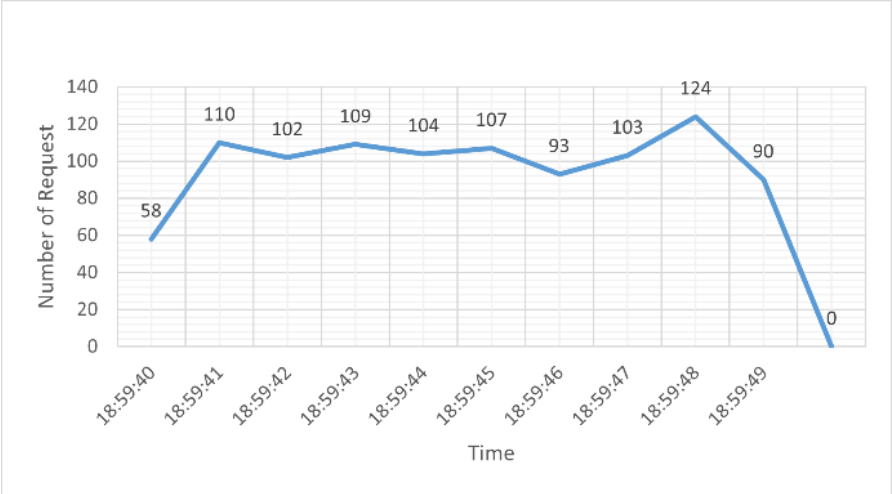

Fig. 7. Least Connection algorithm and Multi-Agent System (LC-MAS) transaction per second for 1,000 requests.

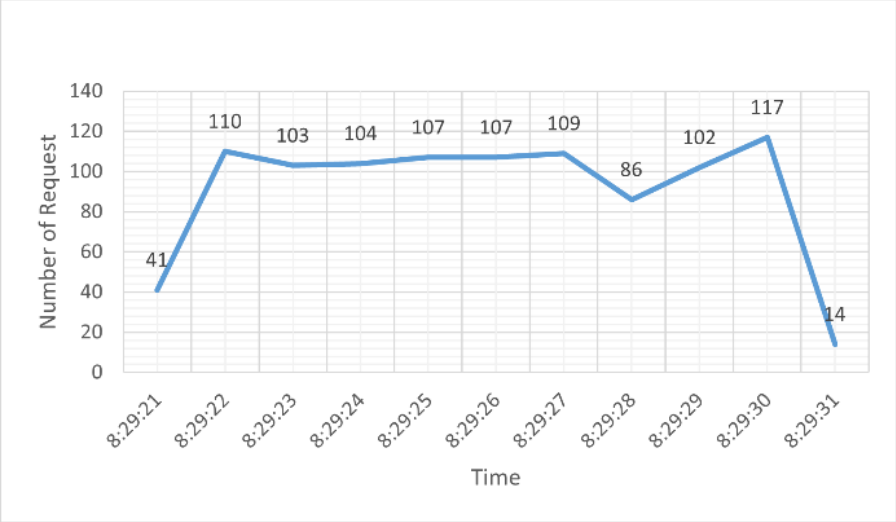

Fig. 8. Least Connection (LC) transaction per second for 1,000 requests.

time interval for each request being 0.007 seconds (10/1,500). Figures 9 and 10 illustrate the number of requests per second that the backend server of each scenario can execute. Based on the graphs, the average number of requests executed by LC-MAS scenario is 125 requests per second. Meanwhile, LC scenario is 250 requests per second.

The backend server's ability to execute requests increases along with the increased number of requests. The distribution of requests handled by each server per second is more equal in both scenarios for 500 and 1,000 requests. It proves that the presence of a load balancing mechanism has succeeded in distributing requests equally without any of the servers being overloaded. However, when executing 1,500 requests, the distribution of requests in scenario 2 (LC) is unequal. The number of requests executed in the first, second spikes high and decreases drastically in the next second. It is caused by the distribution process based only on the number of active connections without considering the condition of server resources. The server that executes the request becomes burdened because only one node works to execute all requests in the first second. This imbalance causes some requests to fail to be executed and leads to higher error values.

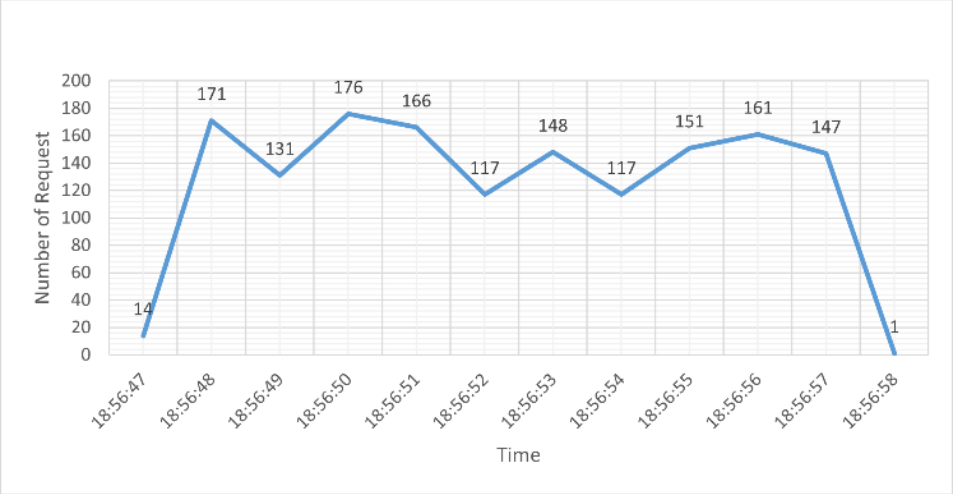

Fig. 9. Least Connection algorithm and Multi-Agent System (LC-MAS) transaction per second for 1,500 requests.

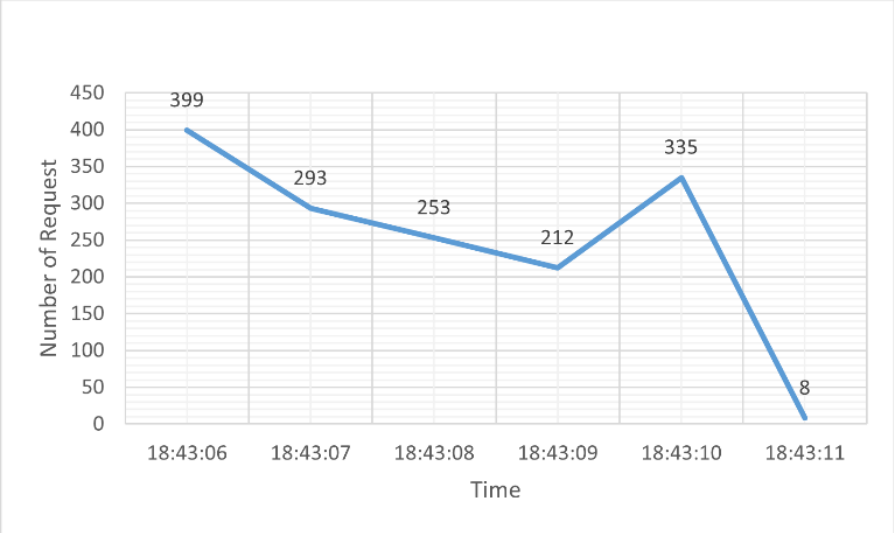

Fig. 10. Least Connection (LC) transaction per second for 1,500 requests.

C. *Error (%)*

Error testing checks whether the system can handle errors that may occur in the future. Error testing refers to the number of failed requests or not processed by the system. The error value is obtained from the number of failed requests to be executed divided by the total requests multiplied by 100%. There is no error in both scenario when executing 500 and 1,000 requests. However, errors in both scenarios occur when the system handles 1,500 requests. Figure 11 shows the error percentage in the LC-MAS scenario.

The total number of failed requests to be processed by backend servers is 301 over 1,500 requests. The type of error is 502 Bad Gateway, which occurs in HTTP protocol. It is caused by several conditions, such as overloaded servers due to a high number of visitors trying to access the website simultaneously and agent activity, affecting CPU and memory usage. Agent activity in every 500 milliseconds communicates with each other to monitor the load condition, affecting the percentage of CPU and memory usage. A high percentage value can produce a large response time and lead to system failure in handling requests because it takes too long to respond to requests. Reporting agents must report the backend server resources condition every three seconds, and the main agent must receive that continuously, so the server is too busy when a request comes. It causes the requests to be unable to

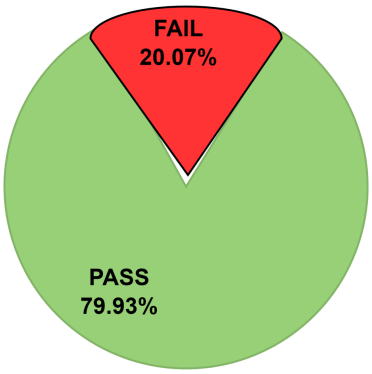

Fig. 11. Least Connection algorithm and Multi-Agent System (LC-MAS) error percentage.

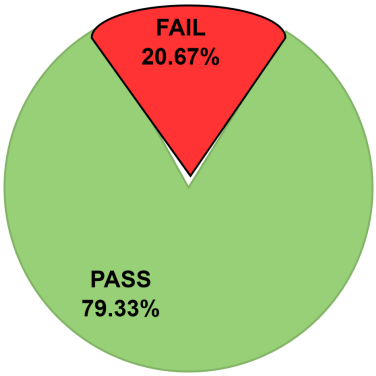

Fig. 12. Least Connection (LC) error percentage

be executed, and response time will become higher.

Figure 12 shows the error percentage in the LC scenario. The total number of failed requests to be processed by backend servers is 310, over 1,500 requests. The type of errors is categorized into three types, as shown in Table I. First, the load balancer address fails to respond because the server is very busy serving incoming requests. The load balancer is busy mapping incoming requests, which should be handled next. The load balancer also has to count the smallest number of active connections, so it has too many tasks. Second, the connection reset is caused by Apache JMeter. It fails to receive responses to requests sent. Connection times are out while requests are processed, so it needs to be repeated to complete the entire request. Third, it occurs when the network connection is not stable. This error message indicates that the server does not wait for any data from a client. It is rare and cannot be predictable. In some cases, rebooting or restarting the system is one of the options to solve this problem.

TABLE I
ERROR TYPES IN LEAST CONNECTION (LC) SCENARIO.

No	Error Message	Number of Error	%
1	Non-HTTP response message: 10.163.12.122:80 failed to respond	268	17.87
2	Non-HTTP response message: Connection reset	39	2.60
3	Non-HTTP response message: Software caused connection abort: receive failed	3	0.20
	Total	310	20.67

IV. CONCLUSION

The research develops a new mechanism for balancing server workloads on website applications by combining the LC algorithm and MAS (LC-MAS), and the result is compared with the LC algorithm. The proposed mechanism is tested by sending several requests to determine the performance of the mechanisms. Parameters used to measure the performance are the average value of response time, throughput, and error percentage.

Based on the results and analysis, it can be concluded that the new mechanism developed using the LC-MAS has successfully distributed the workloads and resources more than the LC algorithm. The combination is proven to perform better regarding response time and error percentage when the number of requests increases. The number of requests that the servers execute is also equally distributed every second. Additionally, the error value is much smaller than in other scenarios. It indicates that it is more resilient in handling increasing requests. Utilization of system resources becomes more efficient than before without burdening any of the servers.

The LC mechanism produces better response times and higher throughput when executing 1,000 requests. However, the performance of this mechanism decreases as the number of requests increases. LC-MAS scenario provides an average response time for 1,500 requests with 1338.8 milliseconds. It has a 20.07% error and 125 transactions per second. However, the LC-MAS makes the website application performance much better when the requests increase. When distributing requests, the server resource condition is not considered, so the servers become burdened. This mechanism is more suitable for small environments with the same specification.

There are some limitations to the research. The environment is still local so the testing results are not in real-time condition and the hardware capacity must be increased to get better storage sharing. The research can be improved by testing the mechanism

with more parameters, such as scalability and master and slave concept. It is better to add more testing scenarios and increase the number of users tested. For the next challenge, it is recommended to implement the cloud environment for a better experience.

REFERENCES

[1] D. Arnaldy and T. S. Hati, "Performance analysis of reverse proxy and web application firewall with telegram bot as attack notification on web server," in *2020 3rd International Conference on Computer and Informatics Engineering (IC2IE)*. Yogyakarta, Indonesia: IEEE, Sept. 15–16, 2020, pp. 455–459.

[2] M. R. M. Bella, M. Data, and W. Yahya, "Web server load balancing based on memory utilization using Docker swarm," in *2018 International Conference on Sustainable Information Engineering and Technology (SIET)*. Malang, Indonesia: IEEE, Nov. 10–12, 2018, pp. 220–223.

[3] N. K. C. Das, M. S. George, and P. Jaya, "Incorporating weighted round robin in honeybee algorithm for enhanced load balancing in cloud environment," in *2017 International Conference on Communication and Signal Processing (ICCSP)*. Chennai, India: IEEE, April 6–8, 2017, pp. 0384–0389.

[4] L. H. Pramono, R. C. Buwono, and Y. G. Waskito, "Round-Robin algorithm in HAProxy and Nginx load balancing performance evaluation: A review," in *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. Yogyakarta, Indonesia: IEEE, Nov. 21–22, 2018, pp. 367–372.

[5] P. Geetha and C. R. R. Robin, "A comparative-study of load-cloud balancing algorithms in cloud environments," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. Chennai, India: IEEE, Aug. 1–2, 2017, pp. 806–810.

[6] K. S. Chaudhury, S. Pattnaik, H. S. Moharana, and S. Pradhan, "Static load balancing algorithms in cloud computing: Challenges and solutions," in *Soft Computing and Signal Processing: Proceedings of 2nd ICSCSP 2019*. Hyderabad, India: Springer, June 21–22, 2020, pp. 259–265.

[7] S. Jain and A. K. Saxena, "A survey of load balancing challenges in cloud environment," in *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*. Moradabad, India: IEEE, Nov. 25–27, 2016, pp. 291–293.

[8] M. Afriansyah, M. Somantri, and M. A. Riyadi, "Model of load balancing using reliable algorithm with multi-agent system," in *IOP Conference Series: Materials Science and Engineering*, vol. 190, no. 1. IOP Publishing, 2017, pp. 1–8.

[9] M. E. Mustafa, "Load balancing algorithms Round-Robin (RR), leastconnection, and least loaded efficiency," *Computer Science & Telecommunications*, vol. 51, no. 1, pp. 25–29, 2017.

[10] S. Ajitha, "Methodology for load balancing in multi-agent system using SPE approach," in *Security issues and privacy concerns in Industry 4.0 applications*. Wiley Online Library, 2021, ch. 11, pp. 207–227.

[11] S. K. Saeid and T. A. Yahiya, "Load balancing evaluation tools for a private cloud: A comparative study," *ARO-The Scientific Journal of Koya University*, vol. 6, no. 2, pp. 13–19, 2018.

[12] I. K. A. and Y. Rosmansyah, "Web server farm design using Personal Computer (PC) desktop," in *2018 10th International Conference on Information Technology and Electrical Engineering (ICITEE)*. Bali, Indonesia: IEEE, July 24–26, 2018, pp. 106–111.

[13] J. M. Faustina, B. Pavithra, S. Suchitra, and P. Subbulakshmi, "Load balancing in cloud environment using self-governing agent," in *2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. Coimbatore, India: IEEE, June 12–14, 2019, pp. 480–483.

[14] L. Zhu, J. Cui, and G. Xiong, "Improved dynamic load balancing algorithm based on least-connection scheduling," in *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*. Chongqing, China: IEEE, Dec. 14–16, 2018, pp. 1858–1862.

[15] J. Cao, Y. Sun, X. Wang, and S. K. Das, "Scalable load balancing on distributed web servers using mobile agents," *Journal of Parallel and Distributed Computing*, vol. 63, no. 10, pp. 996–1005, 2003.

APPENDIX

The Appendice can be seen in the next page.

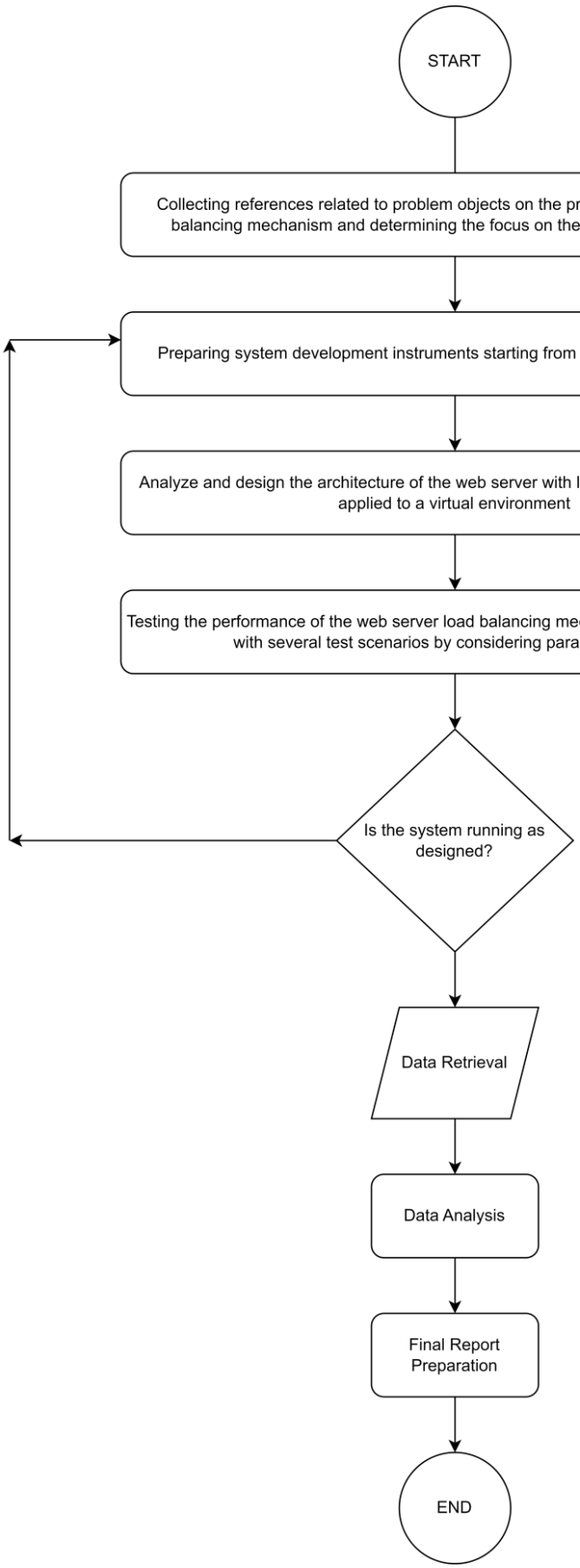

Fig. A1. Research stages.

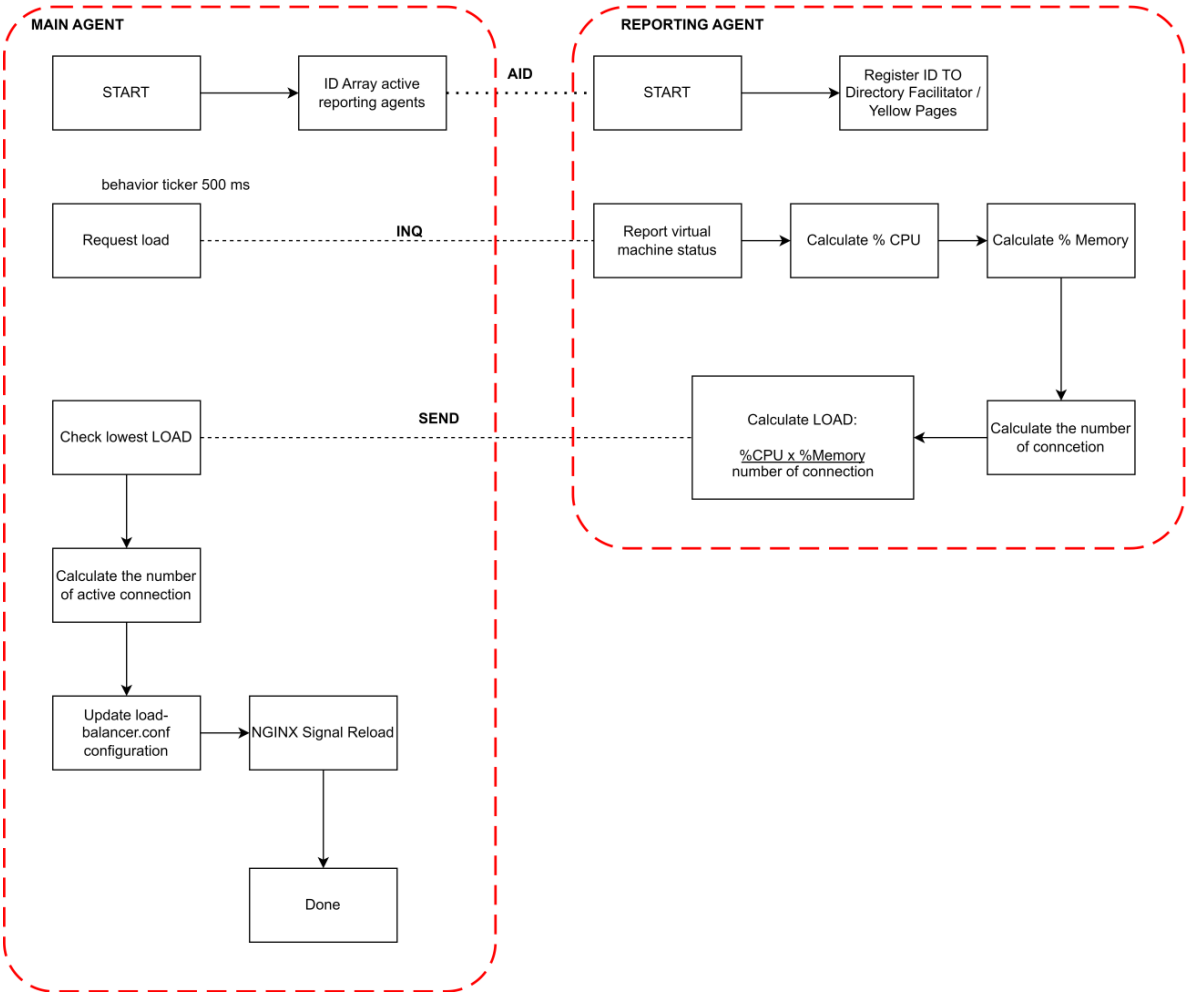

Fig. A2. Multi-Agent System workflow.

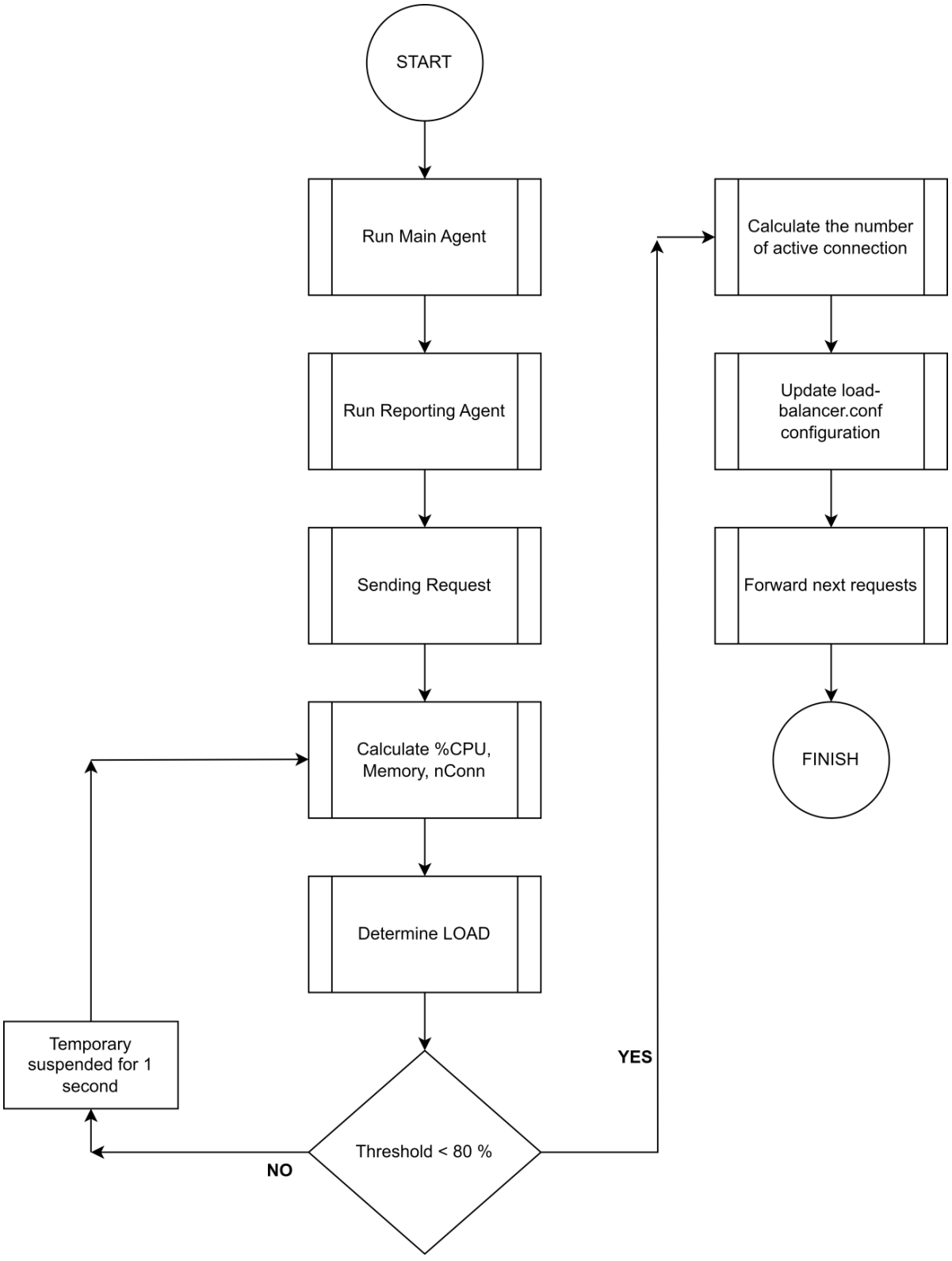

Fig. A3. Least Connection algorithm and Multi-Agent System (LC-MAS) scenario.