TELKOMNIKA Telecommunication Computing Electronics and Control
Vol. 24, No. 4, August 2026, pp. 1121~1130
ISSN: 1693-6930, DOI: 10.12928/TELKOMNIKA.v24i4.27560

Design and evaluation of a simple load balancing prototype using the round robin algorithm in local networks

Muh. Fahmi Rustan, Wawan Firgiawan, Wiwi Nopiana
Department of Informatics, Faculty of Engineering, Universitas Sulawesi Barat, Majene, Indonesia

Article Info

Article history:

Received Sep 21, 2025
Revised Jan 23, 2026
Accepted May 25, 2026

Keywords:

Load balancing
Local network
Multi-client
NGINX
Round robin algorithm

ABSTRACT

Load balancing plays a crucial role in ensuring efficient workload distribution and maintaining stable performance in web service systems. This study presents the design and experimental evaluation of a round robin–based load balancing system implemented in a multi-client local area network (LAN) environment using NGINX as a centralized controller. The system consists of three physical machines, comprising one load balancer and two backend servers hosting identical web applications. Multiple clients generate simultaneous hypertext transfer protocol (HTTP) requests, which are distributed alternately to the backend servers using the default round robin mechanism provided by NGINX. Experimental evaluation was conducted under three workload scenarios of 50, 100, and 200 concurrent requests. The results show that the round robin algorithm consistently distributes requests evenly between the backend servers. The average response time increased from approximately 110 ms at 50 requests to 165 ms at 100 requests and 290 ms at 200 requests, indicating stable performance under light to moderate load conditions. These findings demonstrate that the proposed system is lightweight, modular, and easy to deploy in resource-limited environments. The implementation is particularly suitable for campus-scale networks and small institutional settings, serving as a practical platform for local server deployment, academic applications, and experimental learning in networking and distributed systems.

Corresponding Author:

Wawan Firgiawan
Department of Informatics, Faculty of Engineering, Universitas Sulawesi Barat
Tallu Banua Village, Sendana District, Majene Regency, Sulawesi Barat, Indonesia
Email: wawanfirgiawan@unsulbar.ac.id

1. INTRODUCTION

Computer networks play a vital role in modern information systems, with the client–server model being one of the most widely adopted architectures for delivering web-based services, data-driven applications, and cloud platforms [1]–[4]. In this model, multiple client devices access centralized services hosted on servers, enabling efficient resource sharing, centralized management, and scalable service delivery across the networked environments. The client–server paradigm provides a structured approach for handling user requests, simplifying system maintenance, and allowing services to be accessed concurrently by multiple users. Consequently, this architecture has become the fundamental foundation for modern web services and distributed computing systems.

The rapid growth of digital services, including academic information systems, e-commerce platforms, and institutional web applications, has significantly increased the number of concurrent users accessing these services. As user demand grows, servers are required to handle a high volume of simultaneous requests, which may lead to performance degradation when the system resources are not managed efficiently.

In small-to medium-scale environments, such as campus networks, laboratories, and organizational local area networks (LANs), web services are often deployed on a limited number of servers. When multiple clients simultaneously access a single server, workload concentration may occur, resulting in increased response times, resource contention, and potential service disruption [5], [6]. Figure 1 illustrates a typical scenario in which multiple clients access a single server within a LAN, highlighting the potential for performance bottlenecks under concurrent-access conditions.

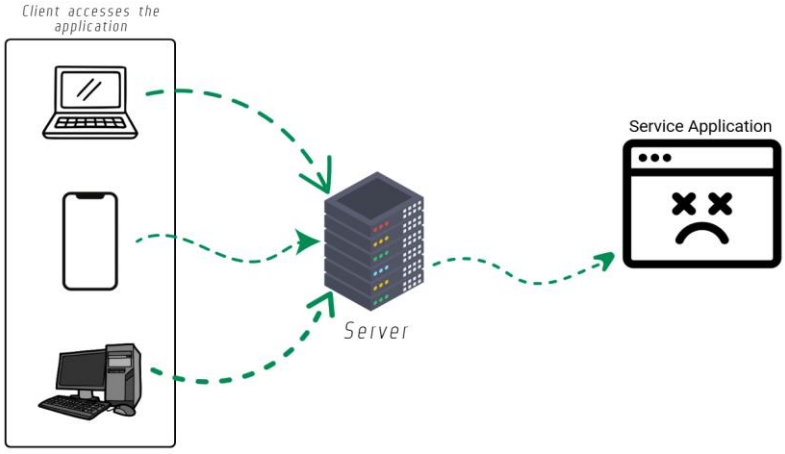

Figure 1. Illustration of multi-client access to a single-server architecture in a LAN

Load balancing is a widely used mechanism for addressing these challenges by distributing client requests across multiple backend servers. By allocating workloads more evenly, load balancing reduces the risk of server overload, improves fault tolerance, and enhances system scalability and availability [3], [7], [8]. Various load balancing strategies have been proposed in the literature, ranging from simple static algorithms to more complex dynamic and adaptive approaches. Among the most widely used static methods is the round-robin algorithm, which distributes incoming requests sequentially among the available servers [9]–[11]. Owing to its simplicity, deterministic behavior, and ease of configuration, the round robin remains a popular choice for practical implementations, particularly in environments with limited resources.

Several empirical studies have reported that deploying lightweight load balancers, such as NGINX or HAProxy, improves throughput stability and reduces service latency in multi-user access scenarios, particularly in LAN-based web service architectures. Previous studies commonly evaluate load balancing performance using metrics such as response time, request success rate, throughput, central processing unit (CPU) utilization, and memory consumption under varying numbers of concurrent connections [12]–[15]. In addition, comparative evaluations have been conducted among static algorithms (round robin, least connections, and IP hash), where round robin frequently demonstrates fair request distribution and stable performance when backend servers have comparable hardware specifications and homogeneous service profiles [9], [16], [17]. Prior research also indicates that adopting a load balancer as a single entry point improves service availability and resilience because requests may be redirected when one backend server becomes overloaded or unavailable [3], [18]. These findings suggest that simple scheduling-based mechanisms remain effective for improving service reliability in small-scale institutional networks, provided that the system is designed and evaluated using realistic LAN traffic loads, rather than purely simulated environments.

The round-robin algorithm does not consider the current server load, processing capacity, or network conditions when allocating requests [19]. Although this static characteristic limits its adaptability in heterogeneous or highly dynamic environments, previous studies have demonstrated that round robin achieves fair workload distribution and stable performance in systems with homogeneous server capacities and moderate traffic levels [5], [11], [20]. Extensions, such as weighted round-robin, have been proposed to enhance distribution by assigning predefined weights based on server capacity; however, these approaches remain static and require prior knowledge of system characteristics [3], [21].

Recent research has explored advanced load balancing approaches, including controller-based architectures [22], optimization and meta-heuristic techniques [22]–[24], and software-defined networking (SDN)-based methods [10]. Although these approaches offer improved adaptability and performance in

complex environments, they often introduce higher system complexity, additional control overhead, and reliance on cloud-based or specialized infrastructures.

Despite extensive research on load balancing algorithms, many existing studies predominantly rely on simulation-based evaluations and cloud computing platforms or SDN environments. Although these approaches provide valuable insights into scalability and adaptability, they often involve complex infrastructures that are not easily accessible in small-scale or resource-constrained settings. Consequently, experimental investigations of lightweight, easily deployable load-balancing mechanisms in real LAN environments remain relatively limited, particularly for educational institutions and campus-scale deployments [21], [22].

In response to this gap, this study presents the design and experimental evaluation of a round-robin-based load balancing system implemented in a multi-client LAN using NGINX as a centralized load balancer. The proposed system consists of one controller node and two backend servers and is evaluated under different request load scenarios to analyze the request distribution and response time behavior. Rather than introducing a new algorithm, this study demonstrates the practical effectiveness of a simple and widely used load balancing approach in a real LAN environment, while providing a lightweight, reproducible, and educational prototype suitable for academic and institutional use.

2. RESEARCH METHOD

This study applies an experimental system-based methodology to evaluate the round-robin load-balancing algorithm in a real LAN environment. The proposed implementation uses a centralized controller that is accessed by multiple concurrent clients. This section describes the research design and system architecture used for the performance evaluation.

2.1. Research design

The research design focuses on the practical implementation and experimental evaluation of a round robin–based load balancing mechanism in a small-scale LAN. This study does not propose a new algorithm, but evaluates the performance of a widely used static strategy when deployed on physical devices under concurrent client requests. The system consists of three computers connected in a LAN: one node acts as a centralized controller using NGINX, while two nodes function as backend web servers. Client hypertext transfer protocol (HTTP) requests are forwarded alternately from the controller to the backend servers following the round robin policy. The architecture is controller-based rather than an autonomous multi-agent system, making it suitable for resource-constrained environments such as campus laboratories and small institutional networks [25].

2.2. System architecture

This subsection describes the logical structure of the system and the interactions between its components. The architecture represents a simplified load-balancing topology commonly used in small-scale web service deployments. The system consists of the following components: i) clients (multi-client): multiple clients simultaneously generate HTTP requests to the controller using request simulation tools; ii) centralized controller: a centralized node running NGINX, responsible for distributing incoming requests to backend servers using the round robin algorithm; and iii) backend servers: two web servers (server 1 and server 2) that process client requests and return responses through the controller. This architecture is designed for local deployment and ease of implementation. Its primary advantages include simplicity, controllability, and suitability for laboratory-based experimentation without requiring complex virtualization or cloud infrastructure. The overall system architecture is shown in Figure 2.

2.3. System workflow

This subsection explains the operational workflow of the system, starting from when a client sends a request until a response is received by the client. The workflow demonstrates how round robin logic is applied at the controller level. As illustrated in Figure 3, the system operates according to the following sequence: i) clients send simultaneous HTTP requests to the controller; ii) the controller receives the incoming requests; iii) requests are forwarded alternately to server 1 and server 2 based on the round robin algorithm; iv) backend servers process the requests and generate responses; and v) response data and interaction logs are recorded for performance analysis. This workflow is inspired by modular task distribution concepts and is centralized and lightweight to suit small-scale experimental environments. Two primary performance metrics are observed in this study, they are workload distribution: to evaluate whether client requests are evenly distributed between the two backend servers according to the round robin principle and response time to number of requests: to analyze the impact of increasing request numbers on system response time.

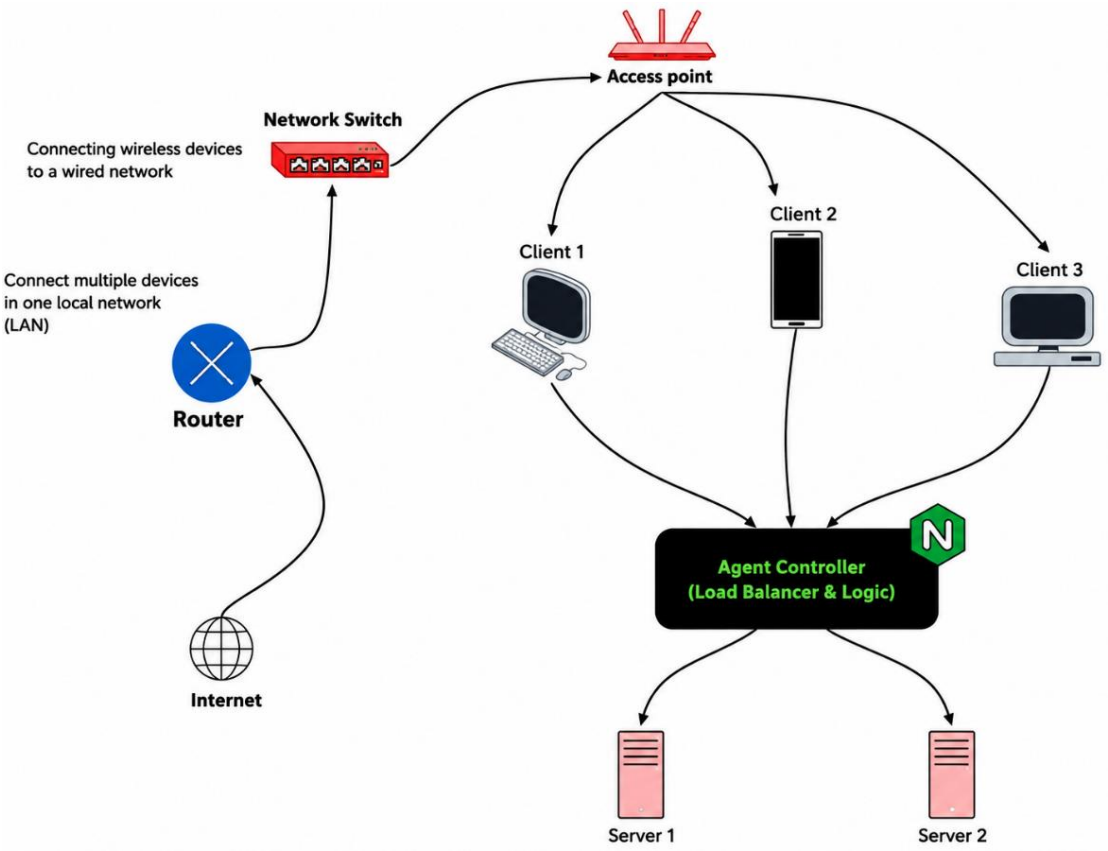

Figure 2. Load distribution diagram using two servers

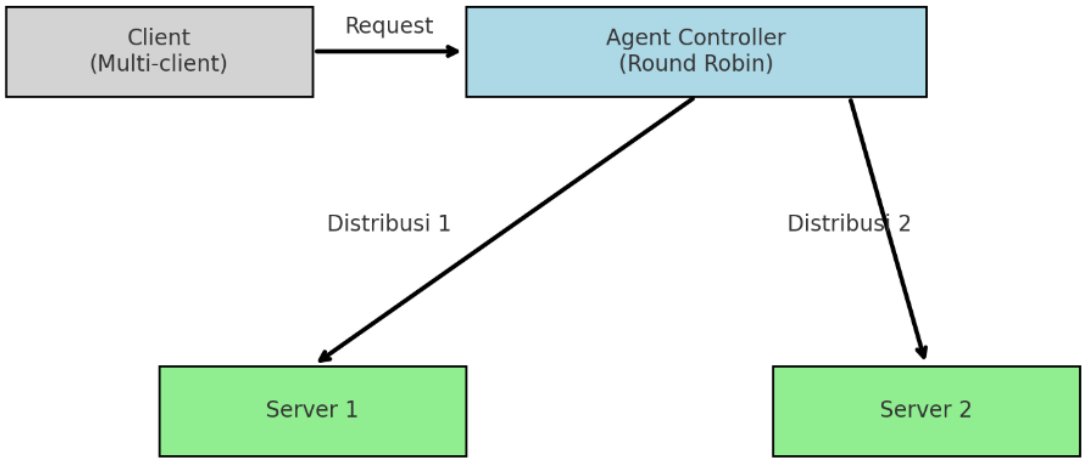

Figure 3. Distribution model of the system with centralized controller

2.4. Testing scenarios

The experimental evaluation was not intended to achieve maximum throughput but to assess the system stability in distributing workloads and maintaining acceptable response times under varying request volumes. Three testing scenarios were defined to represent light, moderate, and heavy load conditions. The detailed workload configuration for each scenario, including the number of generated requests and its corresponding load category, is summarized in Table 1. These scenarios reflect typical access patterns observed in small-scale LAN environments such as academic laboratories and internal organizational networks.

Table 1. Workload testing scenarios

Number of requests	Load category	Description
50	Light load	System tested under light request volume
100	Moderate load	System tested under medium request volume
200	Heavy load	System tested under heavy request volume

2.5. Implementation environment

Prior to conducting performance testing, the hardware and software components used in implementing the proposed load balancing system are summarized in Table 2. The system is deployed on a LAN consisting of three main nodes: 1 controller node and 2 backend servers hosting the same web application.

Table 2. System implementation components

Component	Technology used
Web server	Actual website application accessed via HTTP protocol
Load balancer	NGINX configured to run round robin logic
Request simulation	Apache benchmark (ab), Postman runner, or Python HTTP request script
Programming language	Not required for the controller, since logic is handled by NGINX
Network	LAN
Operating system	Linux (recommended)/Windows
Monitoring	NGINX logging, response time capture, and Wireshark

To support reproducibility, the hardware specifications of the devices used in the experimental setup are summarized in Table 3. All nodes were deployed on physical machines within the same LAN to ensure consistent network conditions during testing. In this configuration, NGINX functions as a centralized controller that receives all client requests and distributes them alternately to the two back-end servers using the default round-robin strategy. Each backend server hosted the same web application and could be accessed directly; however, all experimental requests were routed through the controller to ensure a controlled and observable request distribution. The integration of a physical LAN infrastructure and a real web server provides a deployment environment that closely represents those used in campus laboratory settings, small data centers and local operational environments [12], [26], [27].

Table 3. Hardware specifications of experimental nodes

Node	CPU	RAM	Storage	Operating system
Controller	Intel Core i5 (4 Cores)	8 GB	SSD 256 GB	Linux Ubuntu 20.04
Server 1	Intel Core i5 (4 Cores)	8 GB	SSD 256 GB	Linux Ubuntu 20.04
Server 2	Intel Core i5 (4 Cores)	8 GB	SSD 256 GB	Linux Ubuntu 20.04

3. RESULTS AND DISCUSSION

3.1. Overview of testing

The developed system was evaluated in a LAN environment using three main devices: one centralized controller acting as a load balancer using NGINX and two backend servers hosting identical web applications. Multiple clients, simulated through benchmarking tools, accessed the application concurrently via the controller. The primary objective of the testing was to evaluate whether the system could distribute workloads evenly across the backend servers and to analyze the impact of increasing request volumes on overall system response time. The experiments were conducted under three workload scenarios consisting of 50, 100, and 200 simultaneous requests. Each scenario was executed three times to ensure result consistency and reliability. The collected metrics included the number of requests handled by each backend server and the average response time recorded for each workload level.

3.2. Round robin working mechanism

The round robin algorithm is a simple and widely used load balancing strategy designed for environments with relatively homogeneous server resources. In the proposed system, the round robin mechanism was implemented through the default upstream configuration in NGINX, which automatically forwards incoming requests alternately to each backend server. Conceptually, the round robin logic is formulated as:

```
server_list = [Server1, Server2]
current_index = 0

function handle_request(request):
 target_server = server_list[current_index]
 forward request to target_server

 current_index = (current_index + 1) % length(server_list)
```

In the NGINX configuration, this logic is realized through the following upstream directive:

```
upstream backend_servers {
 server 192.168.1.2; # Server 1
 server 192.168.1.3; # Server 2
}

server {
 listen 80;
 location / {
 proxy_pass http://backend_servers;
 }
}
```

This configuration, each incoming request is forwarded sequentially to the backend servers in a cyclic order. The mechanism is stateless and does not consider real-time server load or resource utilization, making it lightweight and suitable for local network environments with limited complexity.

3.3. Round robin load distribution

Workload distribution is a critical metric for evaluating the effectiveness of the round robin algorithm. Based on NGINX access logs and backend server monitoring, the system consistently distributed requests evenly between the two servers across all test scenarios. Table 4 presents the number of requests received by each server under different workload conditions.

The results show that both backend servers received an equal number of requests in every scenario, which aligns perfectly with the deterministic nature of the round robin algorithm. As illustrated in Figure 4, the workload distribution remains balanced regardless of the request volume. Since the algorithm does not account for dynamic parameters such as server load or active connections, the request allocation remains predictable and consistent.

Table 4. Request distribution across backend servers

Scenario (total requests)	Server 1	Server 2	Ideal distribution
50	25	25	25 – 25
100	50	50	50 – 50
200	100	100	100 – 100

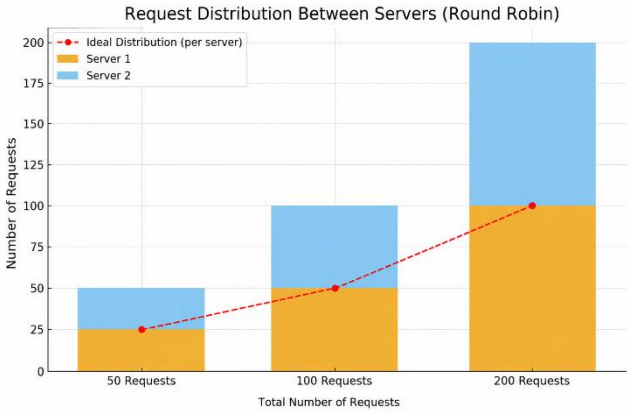

Figure 4. Request distribution between backend servers using the round robin algorithm

3.4. Response time to workload

Response time is a key indicator of web service performance. The experimental results indicate that the average response time increased proportionally with the number of concurrent requests but remained within acceptable limits for local network services. Table 5 summarizes the average response times obtained for each workload scenario. For each workload scenario, the response time values reported in this section were obtained by averaging the results of ten independent experimental runs.

Table 5. Average response time under different workloads

Scenario (total requests)	Average response time (ms)
50	110
100	165
200	290

As shown in Figure 5, the response time exhibits an upward trend as the workload increases. At 50 requests, the system achieved an average response time of 110 ms, which increased to 165 ms at 100 requests and reached 290 ms at 200 requests. The system maintained stable performance under light to moderate loads (up to 100 requests), with response times consistently below 200 ms. Under heavier load conditions, the increase in response time is primarily attributed to hardware limitations and network contention rather than the round robin mechanism itself. Similar behavior has been reported in previous studies focusing on static load balancing in environments with balanced server capacities [19].

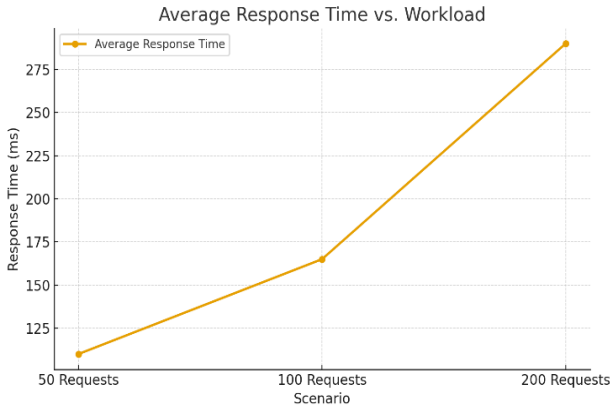

Figure 5. Average response time under varying request loads

3.5. Discussion and interpretation of results

Based on the experimental results presented in sections 3.3 and 3.4, the centralized-controller implementation of the round robin load balancing algorithm achieved an even distribution of client requests across the backend servers in all testing scenarios. The measured response times increased proportionally with the number of concurrent requests, while remaining within acceptable limits under light to moderate workloads. These results indicate that the system operates stably in a small-scale LAN environment with homogeneous server configurations.

From an analytical perspective, the observed behavior is consistent with the deterministic nature of the round robin algorithm, which forwards requests sequentially without considering real-time server load or capacity. When compared conceptually with dynamic load balancing approaches such as least connection and weighted round robin [20], [21], [28], the proposed implementation favors simplicity and predictability over adaptivity. While dynamic algorithms can achieve better performance in heterogeneous or highly variable environments, they require additional configuration, monitoring, and system overhead. In contrast, the simplicity of round robin makes it suitable for controlled environments where ease of deployment and low operational complexity are prioritized.

Despite the positive results, several limitations should be considered. The use of a centralized controller introduces a potential bottleneck, as all client requests are processed through a single node. Although this limitation was not critical in the experimental setup, it may affect scalability under higher traffic conditions. Furthermore, the assumption of homogeneous backend servers limits the applicability of

the system in environments with uneven server capacities. Therefore, the proposed system is most appropriate for educational and small institutional networks, while deployment in real production environments would require additional mechanisms such as adaptive routing, fault tolerance, and health monitoring.

4. CONCLUSION

This study implemented and evaluated a round robin–based load balancing system deployed in a LAN using a centralized controller and multiple backend servers. The evaluation focused on workload distribution effectiveness and the impact of increasing request volumes on response time. Experimental results showed that client requests were consistently distributed evenly across the two backend servers in all tested scenarios, confirming the deterministic behavior of the round robin algorithm. The response time analysis indicated stable performance under light to moderate workloads, with average response times remaining below 200 ms for up to 100 concurrent requests.

Although response time increased under heavier loads (200 requests), the system continued to operate within acceptable limits for LAN-based services, indicating that round robin remains practical for environments with relatively homogeneous server resources and predictable traffic patterns. The proposed approach offers simplicity, low implementation complexity, and ease of deployment without requiring cloud infrastructure or advanced resource management. However, due to its static nature, round robin does not adapt to real-time server conditions and may experience degradation during overloads or server failures. Future work should integrate monitoring mechanisms and adaptive strategies (e.g., least connection or weighted round robin) and extend evaluation to larger workloads and more diverse network conditions.

ACKNOWLEDGMENTS

The authors would like to express their sincere gratitude to the Faculty of Engineering, Universitas Sulawesi Barat, for providing the laboratory facilities and technical support required to carry out the experimental setup of this research. Special thanks are also extended to colleagues and staff in the Informatics Department who offered valuable discussions and assistance during the system configuration and testing phases. The authors also acknowledge the contributions of students who participated in system simulations and performance trials, whose efforts greatly enhanced the evaluation process.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Muh. Fahmi Rustan	✓		✓	✓			✓		✓	✓	✓		✓	✓
Wawan Firgiawan	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓		✓
Wiwi Nopiana	✓			✓	✓	✓			✓	✓			✓	

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

The authors declare that they have no conflict of interest.

INFORMED CONSENT

We confirm that informed consent was obtained from all individuals included in this study. The protection of privacy and confidentiality was ensured, and no personal data that could identify participants were disclosed. All participants provided written permission prior to inclusion in the study, in accordance with ethical and legal requirements.

REFERENCES

[1] X. Xu and H. Yu, "A game theory approach to fair and efficient resource allocation in cloud computing," *Mathematical Problems in Engineering*, vol. 2014, 2014, doi: 10.1155/2014/915878.

[2] J. Du, C. Jiang, A. Benslimane, S. Guo, and Y. Ren, "SDN-Based Resource Allocation in Edge and Cloud Computing Systems: An Evolutionary Stackelberg Differential Game Approach," *IEEE/ACM Transactions on Networking*, vol. 30, no. 4, pp. 1613–1628, 2022, doi: 10.1109/TNET.2022.3152150.

[3] J. Laha, S. Pattnaik, and K. S. Chaudhury, "Dynamic Load Balancing in Cloud Computing: A Review and a Novel Approach," *EAI Endorsed Transactions on Internet of Things*, vol. 10, 2024, doi: 10.4108/eetiot.5387.

[4] S. Agarwal, S. Yadav, and A. K. Yadav, "An Efficient Architecture and Algorithm for Resource Provisioning in Fog Computing," *International Journal of Information Engineering and Electronic Business*, vol. 8, no. 1, pp. 48–61, 2016, doi: 10.5815/ijieeb.2016.01.06.

[5] A. A. Adewojo and J. M. Bass, "A Novel Weight-Assignment Load Balancing Algorithm for Cloud Applications," *SN Computer Science*, vol. 4, no. 3, 2023, doi: 10.1007/s42979-023-01702-7.

[6] A. Safari and A. Rahimi, "Electrical Power Optimization of Cloud Data Centers Using Federated Learning Server Workload Allocation," *Electronics (Switzerland)*, vol. 14, no. 17, 2025, doi: 10.3390/electronics14173423.

[7] R. Diwakar, R. Sharma, D. Nayak, and H. Mohapatra, "Optimizing Load Distribution in Big Data Ecosystems," *AI and the Revival of Big Data*, pp. 177–200, 2025, doi: 10.4018/979-8-3693-8472-5.ch008.

[8] W. Deng, F. Liu, H. Jin, X. Liao, and H. Liu, "Reliability-aware server consolidation for balancing energy-lifetime tradeoff in virtualized cloud datacenters," *International Journal of Communication Systems*, vol. 27, no. 4, pp. 623–642, Apr. 2014, doi: 10.1002/dac.2687.

[9] M. E. Soklic, "Simulation of load balancing algorithms," *ACM SIGCSE Bulletin*, vol. 34, no. 4, pp. 138–141, 2002, doi: 10.1145/820127.820186.

[10] M. I. Hamed, B. M. ElHalawany, M. M. Fouda, and A. S. T. Eldien, "A novel approach for resource utilization and management in SDN," in *2017 13th International Computer Engineering Conference (ICENCO)*, Dec. 2017, pp. 337–342. doi: 10.1109/ICENCO.2017.8289810.

[11] F. Ebadifard and S. M. Babamir, "Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment," *Cluster Computing*, vol. 24, no. 2, pp. 1075–1101, 2021, doi: 10.1007/s10586-020-03177-0.

[12] H. Velayos, V. Aleo, and G. Karlsson, "Load balancing in overlapping wireless LAN cells," in *IEEE International Conference on Communications*, 2004, vol. 7, pp. 3833–3836. doi: 10.1109/icc.2004.1313270.

[13] B. Alankar, G. Sharma, H. Kaur, R. Valverde, and V. Chang, "Experimental setup for investigating the efficient load balancing algorithms on virtual cloud," *Sensors (Switzerland)*, vol. 20, no. 24, pp. 1–26, 2020, doi: 10.3390/s20247342.

[14] T. Semong *et al.*, "Intelligent load balancing techniques in software defined networks: A survey," *Electronics (Switzerland)*, vol. 9, no. 7, pp. 1–24, 2020, doi: 10.3390/electronics9071091.

[15] M. Fotoohi, S. Sirouspour, and D. Capson, "Stability and performance analysis of centralized and distributed multi-rate control architectures for multi-user haptic interaction," *International Journal of Robotics Research*, vol. 26, no. 9, pp. 977–994, 2007, doi: 10.1177/0278364907082049.

[16] J. Nazir *et al.*, "Load Balancing Framework for Cross-Region Tasks in Cloud Computing," *Computers, Materials & Continua*, vol. 70, no. 1, pp. 1479–1490, 2022, doi: 10.32604/cmc.2022.019344.

[17] W. Wang and G. Casale, "Evaluating Weighted Round Robin Load Balancing for Cloud Web Services," in *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Sep. 2014, pp. 393–400. doi: 10.1109/SYNASC.2014.59.

[18] P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," *ACM Computing Surveys*, vol. 51, no. 6, 2019, doi: 10.1145/3281010.

[19] T. Balharith and F. Alhaidari, "Round Robin Scheduling Algorithm in CPU and Cloud Computing: A review," in *2nd International Conference on Computer Applications and Information Security, ICCAIS 2019*, 2019. doi: 10.1109/CAIS.2019.8769534.

[20] I. T. Singh, T. R. Singh, and T. Sinam, "Server Load Balancing with Round Robin Technique in SDN," in *2022 International Conference on Decision Aid Sciences and Applications, DASA 2022*, 2022, pp. 503–505. doi: 10.1109/DASA54658.2022.9765287.

[21] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, M. B. Chhetri, and M. A. Salehi, "Load balancing for heterogeneous serverless edge computing: A performance-driven and empirical approach," *Future Generation Computer Systems*, vol. 154, pp. 266–280, 2024, doi: 10.1016/j.future.2024.01.020.

[22] N. S. Dey and H. K. R. Sangaraju, "A particle swarm optimization inspired global and local stability driven predictive load balancing strategy," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 35, no. 3, pp. 1688–1701, 2024, doi: 10.11591/ijeecs.v35.i3.pp1688-1701.

[23] A. M. R. Ruelas and C. E. Rothenberg, "A load balancing method based on artificial neural networks for knowledge-defined data center networking," in *LANC 2018 - Proceedings of the 10th Latin American Networking Conference*, 2018, pp. 106–109. doi: 10.1145/3277103.3277135.

[24] A. Filali, A. Kobbane, M. Elmachkour, and S. Cherkaoui, "SDN Controller Assignment and Load Balancing with Minimum Quota of Processing Capacity," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, vol. 2018-May, pp. 1–6. doi: 10.1109/ICC.2018.8422750.

[25] T. Shi, Z. Cai, J. Li, H. Gao, J. Chen, and M. Yang, "Services Management and Distributed Multihop Requests Routing in Mobile Edge Networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 2, pp. 497–510, 2023, doi: 10.1109/TNET.2022.3196267.

[26] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-based scheduling to improve web performance," *ACM Transactions on Computer Systems*, vol. 21, no. 2, pp. 207–233, 2003, doi: 10.1145/762483.762486.

[27] Y. Balan and R. Arokia Paul Rajan, "Resource Aware Weighted Least Connection Load Balancing Technique in Cloud Computing," *International Conference on Sustainable Communication Networks and Application, ICSCNA 2023 - Proceedings*, pp. 153–157, 2023, doi: 10.1109/ICSCNA58489.2023.10370307.

[28] M. A. Shahid, M. M. Alam, and M. M. Su'ud, "Performance Evaluation of Load-Balancing Algorithms with Different Service Broker Policies for Cloud Computing," *Applied Sciences (Switzerland)*, vol. 13, no. 3, 2023, doi: 10.3390/app13031586.

BIOGRAPHIES OF AUTHORS

Muh. Fahmi Rustan is a lecturer at the Department of Informatics, Faculty of Engineering, Universitas Sulawesi Barat, Indonesia. He obtained his Master's degree in Informatics Engineering and has been actively teaching and conducting research in the areas of distributed computing, data science, and intelligent systems. His academic contributions include publications in national and international venues, and his research interests involve machine learning, information systems, and computer networks. He can be contacted at email: muhfahmi@unsulbar.ac.id.

Wawan Firgiawan is a lecturer at the Department of Informatics, Faculty of Engineering, Universitas Sulawesi Barat, Indonesia. He received his Master's degree in Informatics Engineering from Universitas Hasanuddin, Makassar, Indonesia. His research interests cover artificial intelligence, optimization algorithms, disaster management systems, distributed computing and network security. He has published several works in national and international journals in the field of intelligent systems and applied computing. He can be contacted at email: wawanfirgiawan@unsulbar.ac.id.

Wiwi Nopiana is an undergraduate student at the Department of Informatics, Faculty of Engineering, Universitas Sulawesi Barat, Indonesia. Her current academic activities focus on computer networks, distributed systems, and web technologies. She is actively involved in research projects related to load balancing and system optimization in local network environments. Her research interest includes computer networks, system performance, and applied informatics. She can be contacted at email: novianawiwi0@gmail.com.