Journal of Communication Systems, Networks, and Security (JCSNS), Vol. 1, Issue 1
April 2026

Analysis of Load Balancing Performance in Raspberry Pi-Based Clustering Services within a Tourism System

Bintang Desta Ramadhani[1], Norma Ningsih[2]*, Haryadi Amran Darwito[3]
Politeknik Elektronika Negeri Surabaya (PENS), Indonesia
**Corresponding author: Norma Ningsih; email: norma@pens.ac.id*

Manuscript received April 08, 2026, revised April 18, 2026, accepted April 20, 2026

Abstract – *This study analyzes the performance of load balancing on a Raspberry Pi-based clustering service for a tourism e-ticketing system, focusing on the tourist destination in Trenggalek. E-ticketing systems often face challenges such as slow response times, system failures during traffic spikes, and difficulties in efficiently managing server resources. Therefore, this research aims to improve user experience, accelerate the ticket booking process, and optimize both the reliability and performance of the system. The methods used in this study include performance testing of two load balancing algorithms, Round Robin and Least Connection, implemented on a Raspberry Pi cluster server. Testing scenarios covered various load conditions: idle, normal, peak, and maximum, utilizing tools such as JMeter for load testing simulation, and Grafana and InfluxDB for real- time monitoring of system metrics. The experimental results show that under peak load (2500 requests), Round Robin produces an average response time of 3500 ms, throughput of 750 req/s, CPU usage of 25.6%, and error rate of 5%, while Least Connection achieves better performance with a response time of 2800 ms, throughput of 850 req/s, CPU usage of 15.8%, and error rate of 2%. Under maximum load, Round Robin reaches a response time of 5000 ms and error rate of 10%, whereas Least Connection maintains a lower response time of 4000 ms and error rate of 5%. Based on these findings, it is recommended that e-ticketing systems with dynamic and fluctuating loads prioritize the Least Connection algorithm. For future development, it is suggested to explore more advanced load balancing algorithms and increase server capacity to handle larger traffic loads.*

Keywords: *Load Balancing, Least Connection, Round Robin, Raspberry Pi, Clustering Service*

I. Introduction

Since early 2015, the rapid growth of internet users has significantly increased the demand for reliable and responsive online services. According to global digital reports, internet penetration continues to rise each year, with billions of users relying on online platforms for daily activities, including tourism services. In Indonesia, the tourism sector has experienced substantial digital transformation, particularly through the implementation of e-ticketing systems at popular destinations such as Pantai Pasir Putih. These systems are designed to improve service efficiency; however, they often face critical challenges during peak periods such as holidays and weekends, where sudden spikes in user access can lead to system slowdowns or even service failures.

Real-world cases show that many e-ticketing platforms experience performance degradation when handling high concurrent requests, resulting in long response times and poor user experience. This issue is further intensified by the rapid growth of smartphone users, which enables users to access booking services anytime and anywhere, thereby increasing the frequency and unpredictability of traffic loads [1], [2]. Without proper traffic management, servers can become overloaded while others remain underutilized, leading to inefficient resource usage.

To address this problem, load balancing methods such as Round Robin and Least Connection are widely implemented to distribute incoming requests across multiple servers. Round Robin assigns requests sequentially without considering server load, which may lead to uneven workload distribution under dynamic traffic conditions. In contrast, the Least Connection algorithm directs requests to the server with the fewest active connections, making it more adaptive and suitable for handling fluctuating workloads [3]. Selecting an appropriate load balancing strategy is therefore essential to ensure optimal performance, especially in real-time applications such as e-ticketing systems.

Cloud computing has emerged as a key solution for scalable service delivery, offering flexible resource management through models such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [4], [5]. However, despite its advantages, cloud computing still faces significant challenges related to workload distribution, virtualization overhead, and system scalability. Inefficient load balancing in distributed environments can result in workload imbalance, where certain nodes are overloaded while others are idle.

This imbalance directly affects system reliability and performance, potentially violating Quality of Service (QoS) standards defined in Service Level Agreements (SLA) between service providers and users [6]. Studies have shown that improper load distribution can increase response time, reduce throughput, and degrade overall system stability [7][8][9]. Therefore, effective load balancing mechanisms are crucial to ensure optimal resource utilization and maintain service continuity.

In addition to cloud-based approaches, low-cost computing infrastructure such as Raspberry Pi clusters has gained attention as an alternative solution for building scalable and energy-efficient systems. Several studies have demonstrated that Raspberry Pi clusters can be utilized to simulate distributed computing environments with lower operational costs, making them suitable for small-scale or local implementations, including tourism services.

Based on these challenges and opportunities, this study aims to analyze the performance of load balancing algorithms, specifically Round Robin and Least Connection, on a Raspberry Pi-based cluster within a beach tourism e-ticketing system. By providing experimental results and performance comparisons, this research is expected to offer practical insights into improving system reliability, efficiency, and scalability in real-world tourism applications.

II. Related Work

Previous studies on load balancing performance and the use of Raspberry Pi as a server have been conducted in various distributed system contexts, providing foundational insights for this project.

Jader et al. [10] presented a comprehensive survey on web server performance and load balancing algorithms, highlighting the importance of load balancing methods for handling dynamic workloads and improving response times in distributed systems. Haidai and Klymenko [11] proposed a load balancing method within a three-layer IoT architecture utilizing edge nodes and Raspberry Pi-based servers, demonstrating effective load distribution and improved system performance under varying task loads. El-Zoghdy and Ghoniemy [12] reviewed load balancing algorithms for high-performance distributed computing systems, emphasizing adaptive algorithms like Least Connection that respond well to fluctuating workloads by dynamically balancing server loads.

Prasad et al. [13] compared Round Robin and Equally Spread Current Execution (ESCE) algorithms, showing that while Round Robin is simple, adaptive algorithms such as ESCE provide better response times and cost efficiency in cloud environments. Siddiqi and Hossain [14] provided a detailed taxonomy and performance analysis of load balancing algorithms in cloud computing, underscoring the need for algorithms that optimize resource utilization and maintain system reliability under variable load conditions.

TABLE I
COMPARISON WITH PREVIOUS STUDY

Aspect	Previous Studies	This Study
Hardware	Raspberry Pi used mainly in IoT or server clusters [10]-[12].	Raspberry Pi cluster dedicated to tourism e-ticketing system
Load Balancing Algorithms	Round Robin, Least Connection, ESCE, and other adaptive methods [13]-[15].	Direct comparison of Round Robin and Least Connection
Testing Scenario	Simulations or IoT/cloud environment benchmarks [11],[12].	Real workload scenarios: idle, normal, peak, and maximum loads
Performance Under Light Load	Round Robin generally effective [13].	Both algorithms perform well under low/moderate load
Performance Under Heavy Load	Adaptive algorithms more stable and efficient [14],[15].	Least Connection demonstrates better stability, lower latency, and higher throughput
Performance Monitoring	Mostly simulation data or limited monitoring [10].	Real-time monitoring using Grafana, InfluxDB, and Telegraf
Limitations	Round Robin struggles with uneven or bursty loads [13].	Round Robin shows increased error rates and delays under peak and max load
Primary Contribution	Algorithm analysis and performance benchmarking [14],[15].	Practical implementation and performance evaluation in a real system

III. Methodology

A. System Design

The system architecture used in this study adopts a client– load balancer–server cluster model. The load balancers (NGINX and HAProxy) receive requests from clients (simulated using Apache JMeter) and then distribute these requests to multiple backend servers (Raspberry Pi) running a Laravel-based e-ticketing booking application.

Each backend node is containerized using Docker, facilitating easier deployment and scalability. For performance monitoring, the system is integrated with Telegraf, InfluxDB, and Grafana, which record and display key metrics such as response time, throughput, and resource usage (CPU, RAM, and network). Architecture on figure 1.

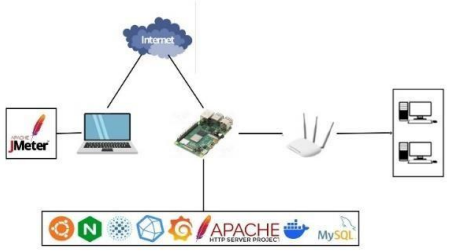

Fig 1. Design System Diagram

B. System Workflow

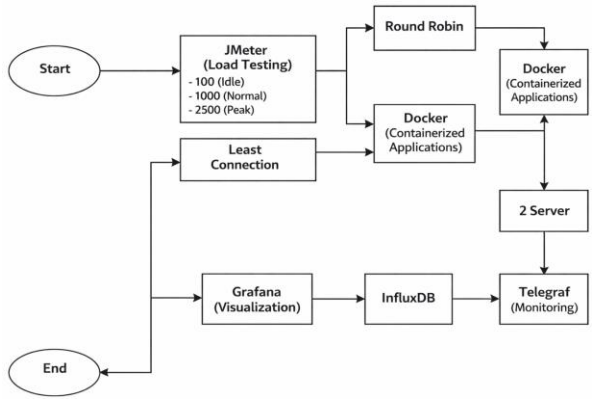

Fig 2 System Workflow

On figure 2 The testing system flowchart describes how client requests are processed by the load balancer before being forwarded to one of the server nodes:

1. The client accesses the application via a web browser or simulation from JMeter.
2. The request is sent to the load balancer (NGINX or HAProxy).
3. The load balancer selects a server node based on the algorithm used (Round Robin or Least Connection).
4. The selected server node processes the request and accesses the MySQL database.
5. The response is sent back to the client through the load balancer.
6. Telegraf collects performance data and sends it to InfluxDB.
7. Grafana displays the data as a real-time dashboard.

C. Load Balancing Method Selection

This study compares two of the most commonly used load balancing algorithms:

1) **Round Robin**: An algorithm that distributes requests sequentially to each server without considering the actual load conditions.
2) **Least Connection**: An algorithm that selects the server with the fewest active connections, enabling a more dynamic and efficient load distribution.

Both methods will be tested under identical conditions to ensure a fair and objective performance comparison.

D. Flowchart

The flowchart in figure 3 illustrates the process of distributing requests to servers using the Round Robin method. The process starts by checking the list of available servers. If no servers are available, the system will display an error or retry after a certain delay. This step ensures that requests are only forwarded to servers that are active and ready to handle them.

If servers are available, the system selects the next server in the Round Robin sequence, which means servers are chosen in a rotating order regardless of their current load. The request is then forwarded to the selected server, and after the server responds, the response is sent back to the client. This method is simple and effective for evenly distributing load, but it does not take into account differences in server capacity or current workload.

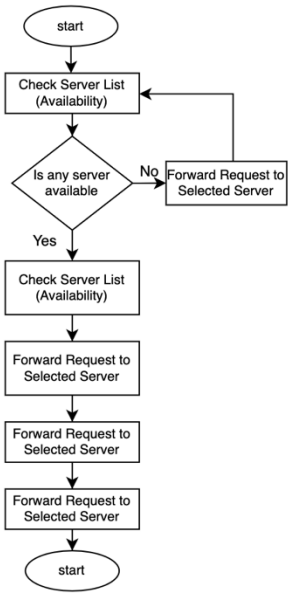

Fig 3 Round Robin Flowchart

The flowchart in figure 4 explains the process of distributing requests based on the number of active connections on each server, using the Least Connection method. The process also begins by checking the list of available servers. If no servers are available, the system will display an error or retry after a delay, similar to the Round Robin approach.

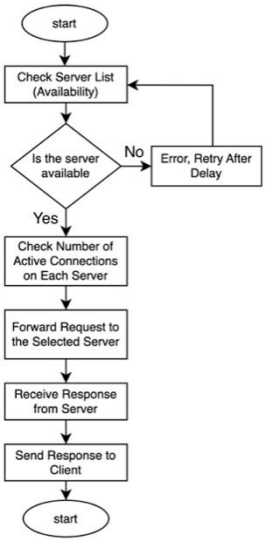

Fig 4 Least Connection Flowchart

When servers are available, the system checks the number of active connections on each server. The request is then forwarded to the server with the fewest active connections, so the load is more evenly distributed according to each server's capacity. After the server responds, the response is sent to the client. This method is more efficient at managing load

compared to Round Robin, especially when there are differences in workload among the servers.

E. Deployment and Integration Schemen

The deployment scheme uses a containerized approach on each Raspberry Pi, where every node runs the e-ticketing application service and database in isolated Docker containers. This isolation allows each service to operate independently, facilitating easier management and system scalability. The load balancer is positioned in front of the cluster, responsible for receiving all user requests simulated by JMeter and efficiently distributing the load to the Raspberry Pi nodes based on the applied load balancing algorithm (Round Robin or Least Connection).

For performance monitoring, Grafana is integrated with InfluxDB as a time-series database that stores performance metrics collected by Telegraf, a data collection agent running on each node. These metrics include CPU usage, memory usage, throughput, response time, and error rate. This architecture enables real-time visual monitoring of system performance during testing, simplifying analysis and bottleneck identification within the system. Diagram on Figure 5.

Fig 5 Beach Booking App Deployment Architecture on Raspberry Pi

F. Testing Scenario

To comprehensively evaluate the system's performance, four testing scenarios with different workloads were conducted:

1. Idle Load (100 requests) Used to observe the baseline performance of the system when there are only a few requests. This reflects the condition when there are few active users.
2. Normal Load (1000 requests) Represents the average load during a typical workday. This scenario is used to assess how the system performs under regular usage conditions.
3. Peak Load (2500 requests) Reflects user surges during weekends or holiday seasons. This test evaluates the system's stability during traffic spikes.
4. Stress Test (Max Requests Until Error) Requests are continuously sent until the system can no longer handle the load, indicated by a sharp increase in error rate. The goal is to determine the system's maximum capacity and the saturation point of both the load balancer and Raspberry Pi nodes.

Measured Parameters:

1. Response Time: Average time the server takes to respond to requests.
2. Throughput: Number of requests served per second by the system.
3. CPU and RAM Usage: Percentage of resource utilization on each server node.
4. Error Rate: Percentage of requests that failed to be processed by the system.

These varied scenarios aim to comprehensively test the system's performance and reliability under realistic conditions that may occur in the field.

IV. Result and Analysis

A. Testing and Results Overview

TABLE II
COMPARASION LOAD BALANCING ALGORITHMS

Aspect	Round Robin	Least Connection
Load Distribution	Distributes requests sequentially without load awareness	Routes requests to the server with the fewest active connections (dynamic load-aware)
Performance at Low Load	Good, stable and predictable	Good, stable and predictable
Performance at High Load	Performance degrades sharply with increased response time and errors	Maintains better stability with lower response time and fewer errors
CPU Usage	Higher CPU utilization during peak loads; uneven distribution across nodes	Lower and balanced CPU usage across servers
Throughput	Throughput declines significantly under max load	Maintains higher throughput under stress
Error Rate	Increased error rate during peak and max load	Lower error rate even under high traffic
Implementation Complexity	Simple and easy to implement	Requires active monitoring of connections
Scalability	Limited due to static distribution logic	Better scalability adapting to traffic dynamics
Recommended Use Case	Suitable for systems with light to moderate, stable loads	Suitable for dynamic systems with fluctuating and heavy loads

This research conducted performance testing of two load balancing algorithms, Round Robin and Least Connection, on a Raspberry Pi cluster used for a tourism e-ticketing system. The system was evaluated under four different workload scenarios:

1) Idle Load (100 requests)
2) Normal Load (1000 requests)
3) Peak Load (2500 requests)
4) Maximum Load (requests sent continuously until error)

Key performance indicators such as CPU usage, response time, throughput, and error rate were measured to assess the behavior of each algorithm under varying conditions. To provide a clearer understanding of the differences between these algorithms, the following section presents a detailed comparison table, followed by analysis of their performance.

B. Comparison of Load Balancing Algorithms

To clearly illustrate the strengths and weaknesses of the Round Robin and Least Connection algorithms, Table 2 summarizes their key characteristics and observed performance based on the experimental results.

C. System Load Balancing Test Result

This study tested the performance of the Raspberry Pi-based clustering system using two load balancing algorithms: Round Robin (implemented with NGINX) and Least Connection (implemented with HAProxy). The evaluation was conducted under four different load scenarios: idle, normal, peak, and maximum load.

To provide a clear comparison of both algorithms under these conditions, Table 3 summarizes the key performance metrics collected during testing, including CPU usage, response time, throughput, and error rate.

TABLE III
PERFORMANCE SUMMARY OF LOAD BALANCING ALGORITHMS UNDER DIFFERENT LOADS

Load Condition	Algorithm	CPU Usage (%)	Response Time (ms)	Throughput (req/s)	Error Rate (%)
Idle (100)	Round Robin	2.1	55.4	950	0
Idle (100)	Least Connection	1.2	55.4	950	0
Normal (1000)	Round Robin	5.4	100	880	0
Normal (1000)	Least Connection	3.2	100	880	0
Peak (2500)	Round Robin	25.6	3500	750	5
Peak (2500)	Least Connection	15.8	2800	850	2
Max (>5000)	Round Robin	48.9	5000	600	10
Max (>5000)	Least Connection	35.2	4000	700	5

D. Performance Comparison

1. CPU Usage: Least Connection consistently shows lower CPU usage than Round Robin across all load conditions, indicating a more efficient load distribution.
2. Response Time: The response time under Round Robin increases dramatically under peak and maximum loads, reaching up to 5000 ms. Least Connection maintains significantly lower response times in the range of 2800 to 4000 ms.
3. Throughput: Least Connection achieves higher throughput rates during peak and maximum loads, which reflects better system stability.
4. Error Rate: The error rate for Round Robin rises sharply under heavy loads, while Least Connection maintains a substantially lower error rate.

E. Performance Monitoring Visualization

Real-time monitoring data collected via Grafana provided further insights into the system behavior under different load balancing algorithms. The visualization results are summarized as follows:

1. CPU Usage per Node: Visualizations indicated a more balanced CPU load across server nodes when using Least Connection, whereas Round Robin caused uneven load spikes.
2. Response Time Trends: Least Connection maintained more stable and consistent response times over time compared to Round Robin, which exhibited large spikes during high traffic.
3. Throughput and Error Rates: Throughput under Round Robin declined sharply at maximum loads with a corresponding increase in errors, while Least Connection sustained better throughput and lower error rates.

V. Conclusion

This This study evaluated the performance of Round Robin and Least Connection load balancing algorithms on a Raspberry Pi-based clustering system for a tourism e-ticketing application. The results show that Round Robin performs adequately under low to moderate loads but experiences significant degradation under high traffic, characterized by increased response time, reduced throughput, and higher error rates. In contrast, Least Connection demonstrates better adaptability by distributing requests based on active connections, resulting in more efficient resource utilization, lower latency, and improved system stability, particularly under peak and maximum loads.

These findings indicate that Least Connection is more suitable for dynamic and resource-constrained environments such as tourism e-ticketing systems. The use of Grafana and InfluxDB also proved effective for real-time monitoring and performance analysis.

Future work should explore advanced load balancing techniques, including weighted and machine learning-based approaches, as well as scalability testing on larger clusters, fault tolerance mechanisms, and energy efficiency optimization to further enhance system performance and reliability

References

[1] McKinsey & Company, "Digital McKinsey," [Online]. Available: https://www.mckinsey.com/business-functions/mckinsey-digital/our- insights. [Accessed: 14-Jun-2024].

[2] Forrester Research, "The Future of Mobile Ticketing in the Travel Industry," 2018.

[3] H. Nasser and T. Witono, "Analisis Algoritma Round Robin, Least Connection, dan Ratio pada Load Balancing Menggunakan Opnet Modeler," *Jurnal Informatika*, vol. 12, no. 1, 2016. [Online]. Available: https://doi.org/10.21460/inf.2016.121.455. [Accessed: 14-Jun-2024].

[4] S. Suresh and S. Sakthivel, "A novel performance constrained power management framework for cloud computing using an adaptive node scaling approach," *Computers & Electrical Engineering*, vol. 60, pp. 30- 44, 2017.

[5] A. Adhikari and T. Amgoth, "Heuristic-based load-balancing algorithm for IaaS cloud," *Future Generation Computer Systems*, vol. 81, pp. 156- 165, 2018.

[6] S. Afzal and G. Kavitha, "Load Balancing in Cloud Computing: A Hierarchical Taxonomical Classification," *Journal of Cloud Computing: Advances, Systems & Applications*, vol. 8, no. 22, pp. 1-24, 2019.

[7] S. K. Mishra, B. Sahoo, and P. P. Parida, "Load balancing in cloud computing: a big picture," *J. King Saud Univ. Comput. Inform. Sci.*, pp. 1-32, 2018.

[8] R. Achar, P. S. Thilagam, N. Soans, P. V. Vikyath, S. Rao, and A. M. Vijeth, "Load balancing in cloud based on live migration of virtual machines," *in Proc. 2013 Annu. IEEE India Conf. (INDICON)*, pp. 1-5, 2013.

[9] D. Magalhaes, R. N. Calheiros, R. Buyya, and D. G. Gomes, "Workload modelling for resource usage analysis and simulation in cloud computing," *Comput. Electr. Eng.*, vol. 47, pp. 69-81, 2015. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2015.08.016. [Accessed: 14-Jun-2024].

[10] S. Suresh and S. Sakthivel, "A novel performance constrained power management framework for cloud computing using an adaptive node scaling approach," *Computers & Electrical Engineering*, vol. 60, pp. 30- 44, 2017.

[11] O. H. Jader, S. R. M. Zeebaree, and R. Zebari, "A State Of Art Survey For Web Server Performance Measurement And Load Balancing Mechanisms," *International Journal of Scientific & Technology Research*, vol. 8, pp. 535-543, 2019. [Online]. Available:

[12] A. Haidai and I. Klymenko, "Method of Load Balancing in Distributed Three-Layer IoT Architecture," *Information, Computing and Intelligent Systems*, 2024.

[13] S. F. El-Zoghdy and S. Ghoniemy, "A Survey of Load Balancing In High- Performance Distributed Computing Systems," 2014.

[14] R. Prasad, P. G. P. Rao, and T. Taibi, "Comparison of load balancing algorithmsin a Cloud," 2012.

[15] F. S. Prity and M. M. Hossain, "A comprehensive examination of load balancing algorithms in cloud environments: a systematic literature review, comparative analysis, taxonomy, open challenges, and future trends," *Iran J. Comput. Sci.*, vol. 7, pp. 663–698, 2024.