Eduvest – Journal of Universal Studies
Volume 6 Number 1, January, 2026
p- ISSN 2775-3735- e-ISSN 2775-3727

A Comparative Study of Dynamic Load Balancing Algorithms for Microservices in Heterogeneous Multi-Cloud Environments

Domma Uli Sitinjak, Galura Muhammad Suranegara
Universitas Pendidikan Indonesia Kampus Purwakarta, Indonesia
Email: dommauli@upi.edu, galurams@upi.edu

ABSTRACT

Microservices-based application architectures in cloud environments require load balancing mechanisms that can adapt to differences in server capacity and workload fluctuations. This study aims to evaluate the performance of dynamic load balancing algorithms—Least Connection, Weighted Least Connection, and Least Response Time—in heterogeneous server environments using Amazon Web Services (AWS) and Google Cloud Platform (GCP). The evaluation was conducted through staged testing by observing application performance based on p95 latency, throughput, error rate, and load distribution patterns. The results indicate that no single algorithm consistently outperforms the others across all scenarios and platforms. Weighted Least Connection tends to produce a more proportional load distribution according to server capacity, while Least Connection and Least Response Time are more influenced by the number of active connections and initial response time. Overall, both AWS and GCP are able to maintain application performance stability across all load levels. These findings confirm that the effectiveness of dynamic load balancing algorithms in heterogeneous cloud environments is influenced by workload characteristics and server capacity, indicating that algorithm selection should be tailored to the specific system objectives.

KEYWORDS *Load balancing Dynamic, Heterogen, Least Connection, Weighted Least Connection, Least Response Time, AWS, GCP.*

INTRODUCTION

The rapid advancement of information technology has brought significant transformations to modern computing systems, enabling the delivery of services that are faster, more efficient, and more reliable. Dependence on digital services continues to increase, making the need for server infrastructures that can ensure high availability and low latency increasingly critical. Dynamically fluctuating user demand often creates bottlenecks on specific servers, which may lead to overload conditions, delayed responses, and even operational service disruptions (Akerele et al., 2024). Consequently, load balancing mechanisms have become an essential strategy for maintaining system stability and consistent performance (G A & Pawar, 2024; Riskiono & Pasha, 2020). The implementation of effective load balancing strategies allows systems to sustain optimal performance despite high demand

fluctuations and unstable traffic conditions, while also ensuring service continuity and improving the efficiency of computing resource utilization (Shafiq et al., 2021).

Load balancing refers to the distribution of workloads evenly among multiple servers in order to increase throughput and reduce latency (Kaviarasan et al., 2023). In practice, incoming requests are allocated to available servers and resources, with the allocation process governed by specific algorithms. These algorithms are generally classified into two main categories: static and dynamic load balancing, each employing different approaches to achieve optimal load distribution (Trivedi et al., 2023). Static load balancing algorithms distribute workloads without considering the current condition of servers, whereas dynamic algorithms adjust resource allocation based on real-time server status, such as the number of active connections, memory usage, and CPU utilization (Oyediran et al., 2024). Both approaches possess distinct characteristics and advantages, depending on workload uniformity and the availability of system resources (Kumar et al., 2023).

Dynamic load balancing algorithms have proven to be more suitable for heterogeneous environments, as they can adapt load distribution based on real-time server conditions, resulting in higher performance and accuracy compared to static algorithms (Maurya & Sinha, 2022; Sharma & Sharma, 2021). In contrast, static algorithms perform optimally in homogeneous environments where workloads are predictable and execution times are stable, with uniform configurations across all machines. Examples of dynamic algorithms include Least Connection, Weighted Least Connection, and Least Response Time. Least Connection directs new requests to the server with the fewest active connections, while Least Response Time routes requests to the server with the fastest response time combined with the lowest connection load at a given moment. Meanwhile, Weighted Least Connection distributes workloads by taking into account predefined processing capacities assigned to each server (Rahman & Hadiwandra, 2023; Fawwazi et al., 2025; Arifin et al., 2024). These dynamic approaches are more effective in heterogeneous systems due to their flexibility in accommodating differences in resource capacity.

Several studies have explored the effectiveness of load balancing algorithms in handling high server workloads. Sinlae et al. (2022) compared the Round Robin and Least Connection algorithms to evaluate throughput across different demand levels, demonstrating that Least Connection is more efficient under initial load conditions due to more balanced connection distribution, although its performance declines as the load increases. Ariestiandy et al. (2023) conducted a comparison of the same algorithms in a clustered server architecture within a cloud computing environment and found that variations in connection levels significantly affect average response time, with Least Connection providing more stable responses than Round Robin. Furthermore, Herdiansyah et al. (2025) evaluated the performance of Least Connection and Round Robin algorithms on a Microsoft Azure–based web server using response time, throughput, and connection stability as performance metrics. Their results indicated that Least Connection performs better under

moderate to high workloads due to its ability to dynamically adjust load distribution, while Round Robin performs more effectively under low workloads by distributing requests evenly without considering server conditions. These foundational studies are essential for understanding the efficiency limits and performance characteristics of each algorithm before applying them to more complex scenarios, such as heterogeneous environments or the use of advanced dynamic algorithms (Alkhatib et al., 2021).

Despite extensive prior research, several limitations remain in existing load balancing studies. Some research has not clearly classified load balancing algorithms into static and dynamic categories or used such classifications as a basis for algorithm selection, resulting in comparative outcomes that do not fully reflect algorithm characteristics or behavior under the tested system conditions. Additionally, several studies restrict their evaluations to a single cloud computing platform, limiting the generalizability of findings across cloud environments with differing infrastructure characteristics.

Based on these gaps, this study focuses on a comparative analysis of dynamic load balancing algorithms—Least Connection, Weighted Least Connection, and Least Response Time—implemented on a microservices-based web application deployed in a heterogeneous server environment. Experiments are conducted on two major cloud computing platforms, Amazon Web Services (AWS) and Google Cloud Platform (GCP), to evaluate system performance using throughput, latency, error rate, and load distribution as key performance metrics.

RESEARCH METHOD

This study adopts a quantitative approach using a quasi-experimental method to evaluate the performance of dynamic load balancing algorithms in a microservices-based web application deployed in a heterogeneous server environment. The experiments are conducted on two cloud computing platforms, namely Amazon Web Services (AWS) and Google Cloud Platform (GCP), employing a non-equivalent group design due to differences in infrastructure characteristics across the platforms. To maintain result comparability, traffic load parameters and application configurations are consistently controlled, although the specifications of the virtual machines are not entirely identical (Abraham & Supriyati, 2022).

The research workflow illustrating the experimental stages and data analysis process is presented in Figure 1.

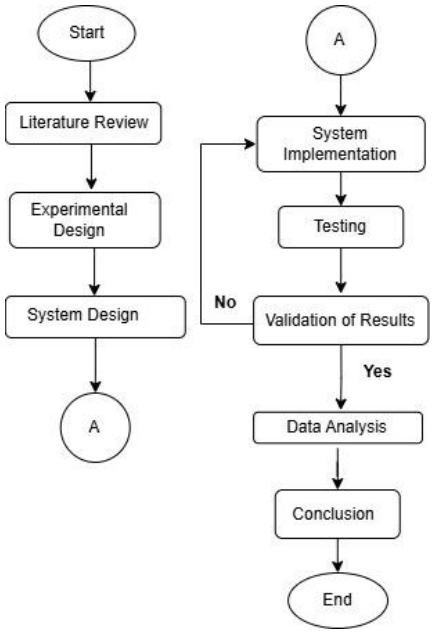

Figure 1. Research Flow Diagram

Literature Review

At the literature review stage, the researchers examined theories, methods, and findings from previous studies related to dynamic load balancing algorithms, microservices-based web application architectures, and performance evaluation in cloud computing environments. The literature review served as a foundation for determining the research approach, selecting appropriate algorithms, and formulating relevant evaluation metrics.

Experimental Design

The experimental design stage focused on developing testing scenarios that include variations in load balancing algorithms, traffic load levels, and performance evaluation parameters. This design aims to ensure that the experiments accurately represent system conditions in a controlled and consistent manner across both tested platforms.

System Design and Implementation

The system design and implementation stage involves constructing the architecture and deploying the testing environment for a microservices-based web application on two cloud platforms, namely AWS and GCP. The test environment consists of a single load balancer node and multiple backend servers with varying resource configurations to represent heterogeneous server conditions, as detailed in Tables 1 and 2. The operating system used is Ubuntu 22.04 LTS (Jammy Jellyfish), selected for its ease of installation and maintenance, as well as its broad compatibility with cloud computing hardware on both AWS and GCP. All instances are deployed within the same region to minimize the impact of geographical latency, while differences in cloud platform characteristics remain a key aspect of the analysis.

Table 1. Heterogeneous Server Specifications on AWS

Server	Machine Type	RAM (GiB)	Virtual CPU	Storage	Region
AWS-Server-1	T3 micro	1	2	10 GiB SSD (*gp3*, 3000 IOPS default)	AP-Southeast-1
AWS-Server-2	T3 small	2	2	10 GiB SSD (*gp3*, 3000 IOPS default)	AP-Southeast-1
AWS-Server-3	T3 medium	4	2	10 GiB SSD (*gp3*, 3000 IOPS default)	AP-Southeast-1
AWS LoadBalancer	T3 micro	1	2	10 GiB SSD (*gp3*, 3000 IOPS default)	AP-Southeast-1

Table 2. Heterogeneous Server Specifications on GCP

Server	Machine Type	RAM (GiB)	Virtual CPU	Storage	Region
gcp-Server-1	e2-cutom-2-1024	1	2	10 GB SSD *(Balanced Persistent Disk*)	Asia-Southeast-1
gcp-Server-2	e2-custom 2-2048	2	2	10 GB SSD *(Balanced Persistent Disk)*	Asia-Southeast-1
gcp-Server-3	e2-custom 2-4096	4	2	10 GB SSD *(Balanced Persistent Disk)*	Asia-Southeast-1
gcpLoadBalancer	e2-custom-2-1024	1	2	10 GB SSD *(Balanced Persistent Disk*)	Asia-Southeast-1

The microservices-based web application was implemented using publicly available source code obtained from a GitHub repository licensed under the MIT License (Vithanage, 2018). This license grants permission for use, modification, and distribution for academic purposes, provided that proper attribution is given to the original developer (Open Source Initiative, 2024). The application was deployed using a container-based architecture by leveraging Docker and Docker Compose to simplify service management and orchestration. The application was executed without modifying its core functional structure; only network configurations and IP addressing were adjusted to meet the requirements of cloud-based testing environments.

Nginx Plus was employed as a load balancer (reverse proxy) with the implementation of dynamic load balancing algorithms, namely Least Connection, Weighted Least Connection, and Least Response Time, to distribute incoming requests across backend servers with heterogeneous resource characteristics. Nginx Plus was consistently deployed on both AWS and GCP by downloading the license package from a temporary IAM-protected repository, installing the official

packages, and verifying the service. The service achieved an *active (running)* status on both platforms, indicating successful installation and license validation.

Load distribution across backend servers was monitored using Netdata, with CPU utilization serving as the primary observation parameter. In addition, supporting metrics such as platform-level CPU utilization were obtained through the native monitoring services of each cloud provider, namely Amazon CloudWatch and Google Cloud Monitoring. Figure 2 presents an overview of the system implementation architecture and the testing scenarios applied in this study.

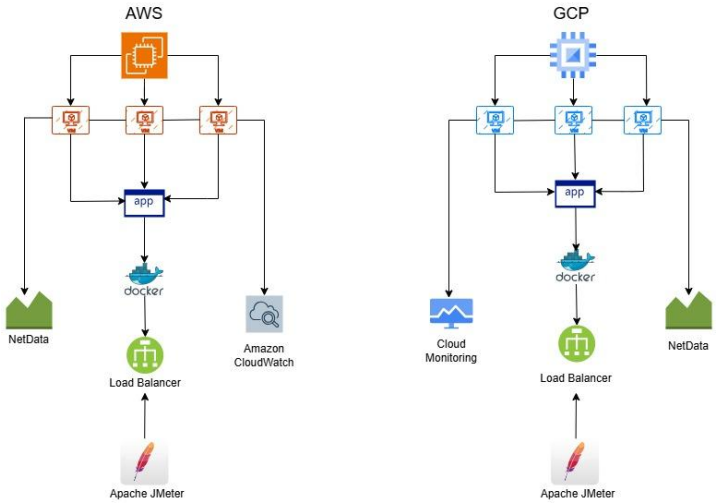

Figure 2. Diagram System

Testing

The performance evaluation of the load balancing algorithms was conducted through load simulation using Apache JMeter, which executed HTTP GET requests to the frontend to ensure that testing variables remained controlled. The evaluated aspects included latency, throughput, error rate, and load distribution across servers, with real-time CPU monitoring performed using Netdata and the native cloud monitoring tools, namely Amazon CloudWatch and Google Cloud Monitoring. CPU utilization values were used as indicators of server load conditions and served as the basis for assessing load distribution throughout the testing process.

The testing scenarios are presented in Table 3, which outlines the load parameters based on varying traffic levels. These scenarios were designed with consideration of each server's capacity to prevent overload conditions on any individual server during testing, thereby ensuring reliable evaluation of system performance and load balancing effectiveness.

Table 3. User Load Testing Scenarios

Scenario	Number of Threads (users)	Ramp-Up Period (sec)	Loop Count	Total request
Low	50	20	20	1000
Medium	200	20	20	4000
High	500	20	20	10000

Data Analysis and Conclusion

Each testing scenario on both platforms and for each load balancing algorithm was executed with five repetitions to ensure result consistency and reliability. The reported values represent the average of these five runs to minimize transient variations and to ensure that the data accurately reflect the stable performance of each algorithm. System analysis was conducted by comparing latency, throughput, error rate, load distribution, and average CPU utilization across all scenarios. These results serve as the basis for evaluating the effectiveness of the load balancing algorithms as well as for comparing platform performance under equivalent testing conditions.

RESULT AND DISCUSSION

This section presents the experimental results and performance analysis of the load balancing algorithms implemented in heterogeneous server environments across the evaluated platforms. The experiments were conducted to assess the effectiveness of each algorithm in distributing request workloads, maintaining system performance stability, and ensuring resource efficiency under varying load levels, namely low, medium, and high. The primary evaluation metrics include throughput (RPS), latency with a focus on the 95th percentile, error rate (%), and load distribution among servers.

The discussion of experimental results is divided into two main sections based on the cloud platform. The first section examines the performance of the algorithms in the AWS environment, while the second section discusses the results obtained in the GCP environment. In addition to the primary metrics, CPU utilization is included as a supporting indicator of resource efficiency. However, CPU usage is not used as the primary metric for cross-platform comparison because the two cloud environments differ in machine types, architectures, and default configurations that cannot be fully standardized. Therefore, the comparative analysis across platforms focuses on latency and throughput, while CPU utilization data are presented as supplementary information to provide insights into computational load trends during the experiments.

Each testing scenario on both platforms was executed with five repetitions to ensure result consistency and reliability. The values reported in Tables 4 and 5 represent the average of these five runs to reduce transient variations and to ensure that the data accurately reflect the stable performance of each algorithm. The repeated trials exhibited minimal variation across runs, indicating that the final values reliably represent actual system conditions on each platform rather than anomalies from single executions. This approach ensures that the interpretation of algorithm effectiveness is based on metrics validated through repeated measurements.

Performance Analysis of Load Balancing Algorithms on AWS

The experimental results obtained in the AWS environment are presented in Table 4, which summarizes the throughput, latency (95th percentile), error rate, and load distribution for each load scenario.

Table 4. Load Balancing Algorithm Performance Results on AWS

Load Balancer Algorithm	Traffic	Throughput (kb/sec)	Avg Latency (ms)	Error Rate (%)	Load Distribution (%)		
					Server 1	Server 2	Server 3
Least Connection	Low	49,86	49,60	0	1,80	1,50	2,11
	Medium	195,58	49,40	0	2,53	1,84	2,04
	High	489,24	48,80	0	5,71	2,80	3,83
Weighted Least Connection	Low	49,70	56,80	0	1,01	1,60	2,69
	Medium	194,68	50,20	0	1,62	2,32	3,62
	High	486,72	48,80	0	2,65	3,58	4,97
Least Response Time	Low	49,68	51,80	0	2,20	1,50	1,33
	Medium	195,76	50,60	0	1,86	2,13	2,29
	High	486,72	48,80	0	2,94	1,94	2,85

Testing in the AWS environment demonstrates consistent system behavior across all load balancing algorithms, particularly in terms of scalability and performance stability. Across the three load levels, throughput increases linearly with the number of incoming requests, reaching approximately 49 kb/sec under low load, increasing to around 195 kb/sec under medium load, and approaching 490 kb/sec under high load. This linear growth pattern indicates that the heterogeneous server architecture is able to maintain good performance scalability. No throughput degradation is observed for any of the algorithms, suggesting that all three load balancing mechanisms operate stably despite differences in server capacity.

The p95 latency metric is used to represent response quality under worst-case conditions experienced by the majority of requests and is considered more representative than average latency alone. The experimental results show that p95 latency remains within the range of 48–56 ms for all algorithms and load levels. Weighted Least Connection records the highest latency under low traffic conditions (56.80 ms), which decreases to 48.80 ms as the load increases. In contrast, Least Connection and Least Response Time maintain nearly identical latency values in the range of 48.80–51.80 ms, indicating more consistent response behavior. These latency patterns are influenced by how each algorithm utilizes heterogeneous resources. For Weighted Least Connection, higher initial latency occurs because high-weight servers are not yet fully utilized under low load; however, as traffic increases, the algorithm effectively leverages the capacity of larger servers, resulting in lower latency. Conversely, Least Connection and Least Response Time exhibit nearly identical latency values, indicating that although these algorithms do not account for server capacity, the applied workload remains within instance limits and does not trigger significant latency increases.

Error rate serves as a key indicator of system reliability, and all algorithms achieve a 0% error rate across all testing scenarios. The absence of failed requests indicates that the system operates within a safe processing capacity, preventing

overload conditions or packet loss during request handling. This outcome is supported by workload levels that remain within instance capabilities and by load distribution decisions that prevent any single server from becoming excessively burdened.

The most pronounced differences among the algorithms appear in the load distribution parameter. Under the Least Connection algorithm, the load distribution pattern shows a tendency to select lower-capacity servers more frequently than higher-capacity ones. In high-traffic scenarios, smaller servers handle 5.71% of requests, while larger servers receive only 2.80%. This pattern occurs because Least Connection prioritizes servers with fewer active connections, without considering their processing capacity. A similar pattern is observed in the Least Response Time algorithm, where smaller servers handle 2.94% of the workload under high traffic, indicating that Least Response Time is more influenced by faster initial response times than by actual server capacity. This behavior may lead to bottlenecks on smaller servers, particularly in large-scale scenarios.

In contrast, the Weighted Least Connection algorithm does not exhibit these issues. Across all traffic levels, it distributes workloads more proportionally according to each server's capacity. This is reflected in a more balanced load distribution, where higher-capacity servers receive a slightly larger share of the workload while remaining within controlled limits. This behavior occurs because Weighted Least Connection normalizes the number of active connections based on each server's assigned weight, ensuring that even though larger servers handle more connections, the connection-to-weight ratio remains low. As a result, higher-capacity servers absorb greater workloads in a controlled manner, while lower-capacity servers are protected from overload.

Performance Analysis of Load Balancing Algorithms on GCP

The experimental results obtained in the GCP environment are presented in Table 5, which summarizes throughput, latency (95th percentile), error rate, and load distribution across all load scenarios.

Table 5. Load Balancing Algorithm Performance Results on GCP

Load Balancer Algorithm	Traffic	Throughput (kb/sec)	Avg Latency (ms)	Error Rate (%)	Load Distribution (%)		
					Server 1	Server 2	Server 3
Least Connection	Low	49,56	54,20	0	3,20	2,93	2,61
	Medium	195,90	52,20	0	3,61	3,22	2,52
	High	487,90	46,20	0	4,78	5,68	2,63
Weighted Least Connection	Low	49,64	51,20	0	2,89	3,03	4,83
	Medium	195,28	47,80	0	2,09	3,43	5,25
	High	487,66	47,20	0	3,70	4,35	5,38
Least Response Time	Low	49,24	56,20	0	3,60	4,45	2,56
	Medium	196,20	50,80	0	4,62	3,12	3,15
	High	487,82	47,20	0	3,36	3,43	2,93

Testing in the GCP environment demonstrates consistent system performance patterns across all load balancing algorithms, with stable throughput increases observed at each load level. Throughput for all algorithms increases linearly as request volume grows, from approximately 49 kb/sec under low load, to around 195 kb/sec under medium load, and approaching 488 kb/sec under high load. The consistency of this pattern indicates that the server architecture is capable of maintaining performance scalability without degradation for any algorithm. This behavior further suggests that workloads are effectively distributed, preventing processing capacity degradation despite increasing request volumes.

In terms of p95 latency, all algorithms also exhibit relatively stable behavior, although the differences are more pronounced than those observed in the AWS environment. All three algorithms demonstrate similar latency reduction trends as load increases. Least Connection records a latency of approximately 54 ms under low load, which decreases under medium load and reaches around 46 ms under high load. Weighted Least Connection follows a similar trend, decreasing from approximately 51 ms under low load to around 47 ms under high load. Meanwhile, Least Response Time produces the highest latency under low load at approximately 56 ms, then declines to around 50 ms under medium load and stabilizes at approximately 47 ms under high load. This decreasing latency trend is driven by increasingly stable workload characteristics as traffic increases. Under low load, performance disparities among servers are more pronounced, making algorithms sensitive to initial metrics—such as Least Response Time—which results in higher initial latency. As the load increases, all servers are forced to operate more consistently, reducing response time variability and leading to lower latency values.

All algorithms again record an error rate of 0% across all load levels, indicating stable system reliability under all testing scenarios. The absence of failed requests confirms that the load distributions generated by each algorithm remain within server capacity limits and that the distribution decisions effectively prevent overload conditions on any individual server.

Differences in algorithm characteristics are most evident in the load distribution parameter. Under the Least Connection algorithm, the load distribution pattern shows a tendency that is not fully proportional to server capacity. Lower-capacity servers continue to receive a relatively larger share of the workload, accounting for approximately 3.20% under low load and increasing to 4.78% under high load, while higher-capacity servers receive smaller portions in certain scenarios. This pattern arises because Least Connection considers only the number of active connections without accounting for server processing capacity.

A similar pattern is observed with the Least Response Time algorithm, where smaller servers handle approximately 3.60% of the workload under low traffic and continue to receive comparable shares under medium and high load, at approximately 4.62% and 3.36%, respectively. This behavior indicates that the

algorithm prioritizes faster initial response times, even when computational capacity is lower.

In contrast, the Weighted Least Connection algorithm produces a more proportional workload distribution based on server capacity. Under low traffic conditions, higher-capacity servers receive the largest share of requests at approximately 4.83%, and this pattern remains consistent under medium and high loads, ranging from approximately 5.25% to 5.38%. This behavior aligns with the algorithm's mechanism of normalizing active connections by server weight, ensuring that higher-capacity servers handle more requests while maintaining a balanced connection-to-weight ratio.

Performa Platform

The performance evaluation of Amazon Web Services (AWS) and Google Cloud Platform (GCP) in this study is not intended to determine which platform is inherently superior, but rather to observe system performance characteristics based on application testing results within each cloud environment. The evaluation focuses on application-level performance indicators, represented by p95 latency and throughput, which directly reflect the quality of service experienced by end users. CPU utilization is included as a supporting parameter to provide insight into system stability during the testing process. All experiments were conducted within the same region, Singapore, to minimize the impact of inter-regional network latency differences.

This approach was adopted because the study does not employ identical virtual machine types across AWS and GCP, although the number of vCPUs was aligned. As a result, resource utilization–based comparisons do not fully represent direct platform performance, and the analysis therefore emphasizes application performance behavior rather than raw infrastructure metrics.

Visualizations of the p95 latency results for AWS and GCP are presented in graphical form in Figures 3 and 4.

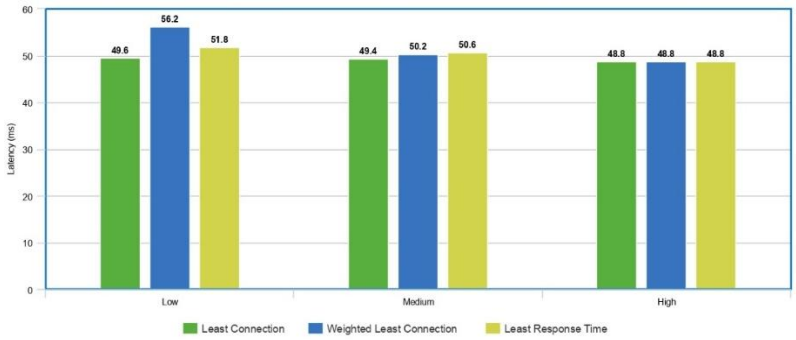

Figure 1. Latency versus Traffic Level on AWS

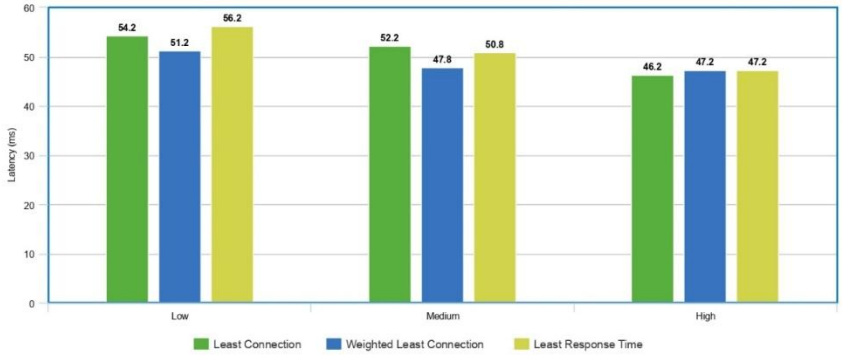

Figure 4. Latency versus Traffic Level on GCP

Based on the p95 latency graphs for AWS and GCP shown in Figures 3 and 4, both platforms exhibit relatively stable response patterns across all load levels. On AWS, p95 latency values remain consistent across algorithms, with only minor fluctuations as the workload increases. This condition indicates that the AWS instances used are still able to efficiently handle increasing workloads, such that changes in load balancing algorithms do not result in significant variations in response time.

In contrast, GCP shows slightly greater p95 latency variation under low load conditions, particularly for the Least Response Time algorithm, which records higher initial latency compared to the other algorithms. However, as the load increases, latency values for all algorithms on GCP demonstrate a declining trend and become more uniform under high load. This pattern indicates that increased workload leads to more consistent resource utilization, thereby reducing disparities in initial response times across algorithms.

Visualizations of throughput test results for the AWS and GCP platforms are presented in graphical form in Figures 5 and 6.

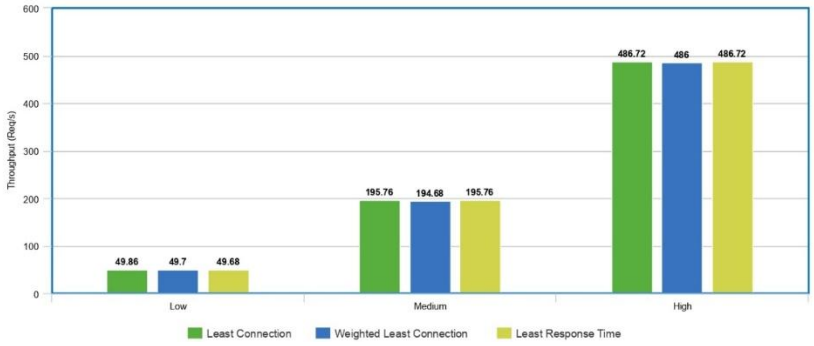

Figure 5. Throughput versus Traffic Level on AWS

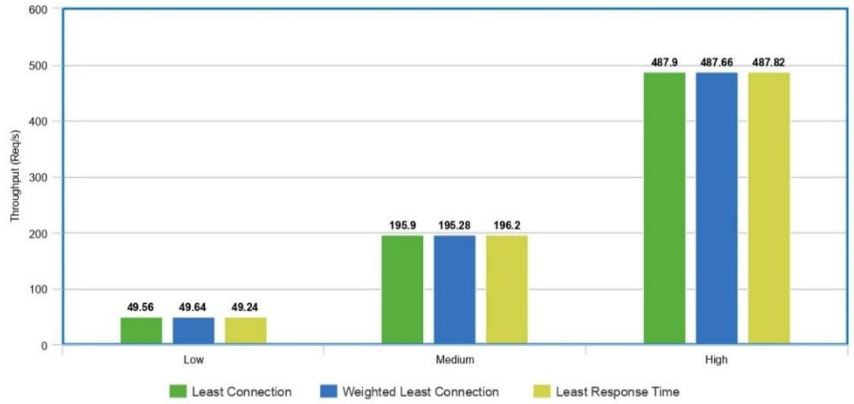

Figure 6. Throughput versus Traffic Level on GCP

Based on the throughput graphs shown in Figures 5 and 6, both AWS and GCP are able to maintain linear throughput growth as the workload increases. No throughput degradation is observed on either platform or for any algorithm, indicating that backend processing capacity remains sufficient across all testing scenarios. The similarity in throughput patterns suggests that differences in virtual machine characteristics do not significantly affect the system's aggregate request processing capability, as long as the configuration and workload remain within the capacity limits of the instances.

As a complementary aspect of platform performance analysis, CPU utilization was also recorded during the testing process and is summarized in Table 6.

Table 6. Comparison of Average CPU Utilization on AWS and GCP

Platform	Traffic	Least Connection	Weighted Least Connection	Least Response Time
AWS	Low	13.89	13.49	14.53
	Medium	17.00	14.27	14.37
	High	17.20	15.50	19.47
GCP	Low	27.33	24.69	26.70
	Medium	29.64	25.43	27.10
	High	31.64	29.40	28.81

CPU utilization on both AWS and GCP shows a consistent increasing trend as the workload grows, without causing any system failures across all testing scenarios. On AWS, CPU utilization values remain relatively lower, while on GCP they tend to be higher at each traffic level. This difference reflects variations in virtual machine characteristics and resource management mechanisms inherent to each platform. Therefore, CPU utilization in this study is treated as a supporting parameter to demonstrate system stability during testing rather than as a primary metric for direct platform comparison.

CONCLUSION

After conducting a series of experiments, this study concludes by evaluating the performance of dynamic load balancing algorithms—Least Connection,

Weighted Least Connection, and Least Response Time—in heterogeneous server environments for microservices-based web applications deployed on Amazon Web Services (AWS) and Google Cloud Platform (GCP). The experimental results indicate that no single algorithm consistently outperforms the others across all scenarios and platforms, as each algorithm exhibits distinct load distribution characteristics and system response behaviors. Weighted Least Connection tends to produce a more proportional workload distribution based on server capacity, while Least Connection and Least Response Time are more influenced by the number of active connections and initial response times.

At the platform level, performance comparison is not intended to determine which cloud provider is inherently superior, given the differences in virtual machine types used. Consequently, the evaluation focuses on application-level performance metrics, namely p95 latency and throughput, which demonstrate that both AWS and GCP are capable of maintaining service stability across all load levels. These findings provide empirical insights into the behavior of dynamic load balancing algorithms in heterogeneous, cross-platform cloud environments and emphasize that the selection of algorithms and platforms should be aligned with workload characteristics and the specific objectives of the target system.

Despite its contributions, this study is subject to limitations in terms of testing scale and infrastructure configuration diversity. Therefore, future work is recommended to explore larger-scale scenarios, including an increased number of instances, more extreme workload patterns through stress testing, and the use of virtual machine types with more standardized vCPU characteristics to further enrich comparative performance analysis.

REFERENCES

Abraham, I., & Supriyati, Y. (2022). Desain kuasi eksperimen dalam pendidikan: Literatur review. Jurnal Ilmiah Mandala Education (JIME), 8(3). https://doi.org/10.36312/jime.v8i3.3800

Akerele, J. I., Uzoka, A., Ojukwu, P. U., & Olamijuwon, O. J. (2024). Optimizing traffic management for public services during high-demand periods using cloud load balancers. Computer Science & IT Research Journal, 5(11), 2594–2608. https://doi.org/10.51594/csitrj.v5i11.1710

Alkhatib, A., Alsabbagh, A., Maraqa, R., & AlZu'bi, S. (2021). Load balancing techniques in cloud computing: Extensive review. Advances in Science, Technology and Engineering Systems Journal, 6(2), 860–870. https://doi.org/10.25046/aj060299

Ariestiandy, D., Suhery, L., Jusmawati, & Yanuardi. (2023). Evaluasi load balancing: Studi komparatif least-connection dan round-robin dalam konteks cloud computing. Jurnal Fasilkom, 13(3), 424–430. https://doi.org/10.37859/jf.v13i3.6236

Arifin, S., Nugraha, A. W., Mukti, F. S., & Jatmika, S. (2024). MQTT broker optimization: Comparative analysis of round robin and least response time. Jurnal Nasional Teknik Elektro, 13(3), 127–136. https://doi.org/10.25077/jnte.v13n3.1260.2024

Fawwazi, M. M., Putra, E., & Putri, N. A. (2025). Evaluation and comparison of load balancing algorithm performance in the implementation of weighted least connections and round robin in cloud computing environment. Journal of Computer

Science, Information Technology and Telecommunication Engineering (JCoSITTE), 6(1), 741–747. https://doi.org/10.30596/jcositte.v6i1.21731

G A, A., & Pawar, R. (2024). Optimizing cloud application performance: A survey on load balancing techniques. International Journal of Scientific Research in Engineering and Management, 8(5), 1–5. https://doi.org/10.55041/ijsrem34983

Herdiansyah, R. N., Iqbal, M., & Aulia, S. (2025). Perbandingan performa load balancing pada web server di Microsoft Azure menggunakan algoritma least connection dan round robin. e-Proceeding of Applied Science, 11(4), 1037. Fakultas Ilmu Terapan, Universitas Telkom

Kaviarasan, R., Balamurugan, G., Kalaiyarasan, R., & Reddy, Y. V. R. (2023). Effective load balancing approach in cloud computing using inspired lion optimization algorithm. e-Prime – Advances in Electrical Engineering, Electronics and Energy, 6, 100326. https://doi.org/10.1016/j.prime.2023.100326

Kumar, P. R., Rajagopalan, S., & Charles, J. C. P. (2023). Lightweight native edge load balancers for edge load balancing. Global Journal of Information Systems and Applications, 3(1), 48–55. https://doi.org/10.53623/gisa.v3i1.256

Maurya, S. K., & Sinha, G. (2022). Load balancing in cloud computing: An analytical review and proposal. Indonesian Journal of Electrical Engineering and Computer Science, 26(3), 1530–1537. https://doi.org/10.11591/ijeecs.v26.i3.pp1530-1537

Open Source Initiative. (2024). The MIT License. https://opensource.org/license/MIT

Oyediran, M. O., Ojo, O. S., Ajagbe, S. A., Aiyeniko, O., Obuzor, P. C., & Adigun, M. O. (2024). Comprehensive review of load balancing in cloud computing system. International Journal of Electrical and Computer Engineering, 14(3), 3244–3255. https://doi.org/10.11591/ijece.v14i3.pp3244-3255

Rahman, S. A., & Hadiwandra, T. Y. (2023). Perbandingan algoritma weighted least connection dan weighted round robin pada load balancing berbasis Docker Swarm. Jurnal Inovtek Polbeng – Seri Informatika, 8(2).

Riskiono, S. D., & Pasha, D. (2020). Analisis metode load balancing dalam meningkatkan kinerja website e-learning. Jurnal Teknoinfo, 14(1), 22–26. https://doi.org/10.33365/jti.v14i1.466

Shafiq, D. A., Jhanjhi, N. Z., & Abdullah, A. (2021). Load balancing techniques in cloud computing environment: A review. Journal of King Saud University – Computer and Information Sciences, 33(9), 1129–1141. https://doi.org/10.1016/j.jksuci.2021.02.007

Sharma, V., & Sharma, H. C. (2021). A review of cloud computing scheduling algorithms. International Journal of Innovative Science and Research Technology, 6(12).

Sinlae, A. A. J., Bagir, M., & Prayitno, M. H. (2022). Analisis perbandingan algoritma round-robin dengan least-connection terhadap peningkatan throughput layanan web server. JURIKOM (Jurnal Riset Komputer), 9(5), 1584–1591. https://doi.org/10.30865/jurikom.v9i5.4995

Trivedi, D., Parmar, N., & Rahevar, M. (2023). Methodological assessment of various algorithm types for load balancing in cloud computing. In S. Awasthi et al. (Eds.), Sustainable computing: Transforming Industry 4.0 to Society 5.0 (pp. 269–277). Springer. https://doi.org/10.1007/978-3-031-13577-4_16

Vithanage, K. (2018). Web-based microservice architecture example [Computer software]. GitHub. https://github.com/kasvith/simple-microservice-example