ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 02 February 2026

Efficient Load Balancing of Dockerized Adaptive Algorithms using Nginx

[1] Dhanushya B*, [2] Anusooya G

[1][2] School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, India
Email: [1] dhanurpt@gmail.com; [2] anusooya.g@vit.ac.in

Abstract— One of the most crucial aspects to be taken care of in modern web frameworks is the efficient sharing of incoming requests which, in turn, leads to fast performance and high reliability. The main intention is to display by means of a browser, based simulation the effectiveness of load balancing with present and custom algorithms, all of them running in a containerized environment with Docker and Nginx. The centerpiece is basically to test the behaviour of different load balancing algorithms including a newly developed custom one when they receive multiple concurrent requests and need to distribute the load among servers in an efficient manner. To isolate, scale, and keep the system modular, each algorithm is set up in a separate Docker container. Depending on the chosen strategy, Nginx acting as a reverse proxy sends requests to these algorithm containers. It provides a convenient and interactive tool which lets one see the flow of requests, keep track of server loads in real, time, and compare the performance of different algorithms. The logic is very flexible to changes and is capable of dynamically handling requests as well as returning accurate data, such as response times and server loads, for that load balancing method that is most effective. By generating heavy traffic scenarios, the system shows how load balancing can be used to better resource management and getting rid of server overloads in real web infrastructures.

Index Terms— Concurrent requests, Docker, Load Balancing, Nginx, Performance

I. INTRODUCTION

Modern web applications in the fast, paced digital ecosystem are expected to handle a huge volume of concurrent user requests while keeping high availability, low latency, and stable performance. Single server architectures are no longer enough to satisfy scalability and performance demands as a result of cloud computing and globally distributed services. As a result, load balancing is the main method that distributes the incoming traffic among the multiple backend servers thus both preventing bottlenecks and enabling efficient resource utilization.

In addition to making the system more responsive, load balancing also increases its fault tolerance and reliability by ensuring that no server is overloaded. Round Robin, Least Connections, and IP Hash algorithms are commonly used and have been implemented in real, world web infrastructures due to their simplicity and ease of implementation. Nevertheless, these traditional methods usually function on the basis of static decision, making rules and do not have the ability to recognize real, time server conditions. As a result of the increasingly dynamic and unpredictable nature of traffic patterns, these algorithms may cause an inefficient load distribution, longer response time, and system performance degradation during peak demand periods.

Containerization technologies and microservices architectures have become popular in the last couple of years because of their lightweight nature and scalability. While there has been a shift towards these technologies, a majority of the studies on load balancing continue to focus on virtual machine level optimization and still use techniques that are computationally expensive and, therefore, are not very suitable for real, time container, based deployments. This necessitates the emergence of load balancing mechanisms that are both structurally lightweight and functionally adaptive so that they will be able to perform efficiently in containerized environments and also adjust their activity dynamically to varying workload conditions. Such a system with these functionalities is what the present paper offers.

The paper introduces a containerized simulation framework that harnesses both traditional and custom load balancing algorithms with the help of Docker and Nginx. The intended system facilitates controlled experimentation and real, time performance analysis through the deployment of each algorithm in isolated containers and traffic routing via an Nginx reverse proxy. It presents a novel custom adaptive load balancing algorithm that can be employed to distribute the requests in an intelligent manner, basing the distribution on the real, time server load and response metrics.

The main contributions of this paper are - The proposal of a Docker, based, containerized framework for the simulation and evaluation of load balancing algorithms., The design and implementation of a tailor, made, adaptive load balancing algorithm that considers real, time server load conditions., The presentation of a comparative performance study of both traditional and custom algorithms, measured by response time and load distribution, under artificially generated traffic scenarios.

II. LITERATURE REVIEW

A Sustainable and Secure Load Management Model for Green Cloud Data Centers – Saxena et al. (2023) Saxena et al. developed the SaS, LM framework aimed at load optimization and energy efficiency in green cloud data

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 02 February 2026

centers while maintaining security. To predict the workload, the model uses a neural network, and to allocate the computing and networking resources dynamically, it employs a dual, phase black hole optimization algorithm. The experiments demonstrate that under different workloads, the energy consumption is reduced, and the throughput is increased. The research points out that intelligent workload prediction is crucial for large, scale infrastructures. Nevertheless, the use of deep neural networks significantly increases the computational overhead. As a result, the method is not very efficient for use in situations where there is a need for very low latency or in the case of simple web applications. Moreover, the authors concentrate mainly on the infrastructural level for resource allocation. They do not consider application, layer request routing or containerized environments. These restrictions lead to the necessity for simpler, adaptive load balancing strategies at the application layer, like the Docker, and Nginx, based framework presented in this study. [1]

Renewable-Aware Geographical Load Balancing Using Option Pricing – Khalil et al. (2022) Khalil et al. proposed a renewable, aware geographical load balancing strategy that adjusts workloads dynamically depending on energy cost, renewable availability, and data center location. To reduce operational costs and carbon emissions efficiently, option pricing theory was integrated as a tool. The outcomes reveal that the approach leads to profound sustainability advantages in geographically spread cloud infrastructures. The paper positions itself as a significant green computing contribution by promoting load balancing as a means of renewable energy utilization. Nevertheless, it depends on the assumption of uninterrupted access to precise real, time energy pricing data, which may not always be the case. Furthermore, the method is confined to an interdata center level and does not address the application layer, where the container, level request routing within web services is not taken into account. This deficiency has been bridged by this work through the use of an adaptive load balancing mechanism in Dockerized environments employing Nginx. [2]

Power Control Framework for Green Data Centers – Yang et al. (2022) Yang et al. designed a power control strategy that not only alters the server power states but also tracks the change in the workload in order to distribute it properly. Their framework, as described, is intended to consume less energy while still allowing the system to meet the agreed levels of services. In fact, the experimental results show that this strategy leads to an energy, efficient system that can maintain its performance even though the workload is constantly changing. Their investigation underlines the significance of the combined control of both power supply and load. However, the authors focus their attention on hardware and virtualization layers, and open little room for application, layer solutions. As technology advances and many systems are being built on microservices and containers, one can argue that this solution has little room for future growth and lacks availability in modern environments. In addition, the paper does not mention the evaluation of request, level routing decisions, leaving a gap that the proposed research fills by investigating the problem of adaptive request distribution in containerized web applications with Docker and Nginx. [3]

Bio-Inspired Heuristics for VM Consolidation in Cloud Data Centers – Wang et al. (2020) Wang et al. have devised bio, inspired heuristics for virtual machine consolidation based on symbiotic biological relationships. Their method enhances the system's stability and alleviates the problem of server overload by the clever execution of the placement of virtual machines. The research outcomes indicate the promotion of energy efficiency and lessening of hardware wear and tear. The authors illustrate the power of biologically inspired optimization techniques through their study. However, the authors only consider virtual machine scenarios in their research, leaving container, based architectures unaddressed, which are the future of the industry. Additionally, the approach does not consider real-time HTTP request routing. Application-level performance metrics such as response time are not evaluated. These gaps justify the need for container-oriented load balancing studies, as performed in this work**. [4]**

Energy-Efficient Task Scheduling and Resource Allocation in Cloud–Fog Environments – Kumar et al. (2022) Kumar et al. proposed an energy-efficient task scheduling framework for cloud–fog environments. The model aims to reduce latency and power consumption by distributing tasks closer to end users. Experimental results show improved performance for delay-sensitive applications. The study highlights the importance of fog computing in modern distributed systems. However, it focuses primarily on task scheduling rather than web request load balancing. The approach does not evaluate HTTP-level request routing. Furthermore, the framework is not tested in containerized microservice environments. The absence of application-layer experimentation leaves a gap. This research addresses that gap by analyzing request-level load balancing in Docker-based systems. **[5]**

Reduced Carbon Emission Using Containers over Virtual Machines – Anusooya & Vijayakumar (2021) Anusooya et al. proved the effective reduction of power consumption and emissions in container-based deployment compared to the virtual machine. The research emphasizes the effectiveness of the use of container technology in conjunction with dynamic voltage and frequency scaling. The results clearly prove the effectiveness of the research in effectively reducing energy consumption as well as emissions. This research is a strong emphasis to implement the use of container technology in the field of computing. But the research does not elaborate on how the container is affected by the load balancing algorithms. The request distribution algorithms are not discussed. The research does not provide real-time performance measurement, which is the

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 02 February 2026

response time. This research gap triggers the research to be conducted. [6]

Dynamic Load Balancing Policies for Distributed Web Systems – Cardellini et al. (2019) Cardellini et al. studied dynamic load balancing strategies in a distributed web application. This study shows how static load balancing strategies like Round Robin are inefficient, even with bursty workloads. This study also concludes how dynamic strategies are more efficient. This study emphasizes the need to be aware of system state. But this study did not try to carry out an experiment on containers. This study also did not present a graphical representation of the execution of requests. This research work would demonstrate this concept through an experiment based on Docker and Nginx. [7]

Nginx as a Reverse Proxy for High-Performance Web Applications – Reese (2020) Reese was interested in using Nginx as a reverse proxy server and load balancer for high-performance web applications. The importance of Nginx's low latency, scalability, and handling concurrent connections is prominently shown. Nginx is proven as a trustworthy load balancer. However, it primarily concentrates on the default algorithm available in Nginx. Intelligent load balancing is not considered. Comparative results analysis is also restricted. This is a research void that can analyze adaptive algorithms. The proposal fills this void by conducting a comparative analysis between conventional and adaptive load balancing approaches using Nginx. [8]

Microservices Architecture and Container-Based Deployment – Pahl (2018) Pahl described the architecture and deployment approach using containers. The paper highlights cloud scalability, modularity, and isolation. The paper identifies load balancing as a major issue related to microservices. The paper is rich from the architectural point of view. The paper does not involve performance analysis related to load balancing algorithms. The paper does not involve experimental work. The paper is relevant to our work as it highlights the need to carry out experimental work. Our work is an extension that experimentally tests load balancing algorithms on containers that host microservices. [9]

Adaptive Load Balancing for Cloud Applications – Xu et al. (2021) Xu et al. proposed a load balancing algorithm that adapts to server response times and workloads. The proposed algorithm aims to enhance the performance of load balancing by considering server response times and workloads. However, the proposed algorithm operates at the middleware layer of the application stack and does not address microservices-based web applications using containers. In addition, this paper does not investigate the performance of load balancing on web applications using Docker and Nginx. Finally, this paper does not consider visualizing the results in real-time. [10]

Comparative Performance Evaluation of Load Balancing Algorithms – Singh & Chana (2017) A comparison performance analysis is presented by Singh Chana on commonly used load balancing algorithms like Round-Robin or Least Connection. Analysis shows that Least Connection performs well in an uneven load environment. It is an important study on algorithm performance that contributes fundamental knowledge on algorithm performance. However, performance analysis is done in traditional server environments. It does not consider container isolation or microservices. Moreover, real-time performance data is not used. It neither focuses on adaptive algorithms nor on custom-designed ones. This brings a motivation to work on contemporary analyses using Docker. [11]

Docker: Lightweight Container-Based Virtualization – Merkel (2014) Merkel Added Docker as a lightweight container-based virtualization system. This study emphasizes speed advantages, resource management, and scalability. Docker provides an attractive substitute to conventional virtual machines. This contributed to advancements in container-based application deployment. Though, load balance concepts in a Docker system are not covered. Route requests' behavior isn't examined. Comparison of algorithms' performances isn't mentioned. All this shows an emerging need to study load balance in a system based on Docker. The current study directly meets this need. [12]

Serverless and Containerized Architectures for Scalable Systems – Adzic & Chatley (2017) Adzic and Chatley studied serverless and container-based architectures in terms of designing a scalable system. In particular, they focus on ease of operation and scalability. In regard to scalability, they consider intelligent routing of requests as a very significant factor. They are still at a conceptual stage. Furthermore, they do not conduct any empirical testing of algorithms they discuss in terms of load balancing. Specifically, they do not investigate request routing at a container level. They do not discuss any performance measures. Addressing this problem is exactly what this proposed study aims to do by testing any load balancing strategies at a container level. [13]

Challenges of Load Balancing in Microservices Architectures – Dragoni et al. (2017) Dragoni et al.. conducted research on the challenges of microservices architectural patterns. Issues in load balancing across microservices are pointed out as critical due to service diversity and scalability requirements. The research encompasses an exhaustive introduction to architectural issues. Yet, no formal frameworks for implementation are discussed in this research. There is no experimental analysis. The author did not assess algorithm design for adaptability. The absence of feasible solutions indicates some research void. This proposed research fills this void, and it aims to implement and analyze adaptable load balancing algorithms through the use of Docker/Nginx. [14]

Intelligent Load Balancing Using Real-Time Metrics – Li et al. (2020) Li et al. proposed an intelligent load balancing based on real-time system metrics, like CPU utilization and response time. The results show an increased throughput and reduced latency. In this study, the effectiveness of

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 02 February 2026

metricdriven decision-making has been shown. However, the necessity of complex monitoring infrastructures adds to the deployment overhead and increases the system complexity. Besides, this study focuses on large-scale cloud platforms. Lightweight containerbased environments are not within the scope of it. In the present work, a simpler adaptive algorithm is suggested applicable to Docker-based web systems, offering a practical and efficient alternative. [15]

III. PROBLEM STATEMENT

Contemporary web applications tend to run in distributed and cloud-centric environments where the traffic intensity and the pattern of requests dynamically change. Although classical load-balancing schemes such as Round Robin, Least Connections, and IP Hash are most popular owing to simplicity, these schemes have been able to use only fixed or partially adaptive approaches that do not consider dynamic server operation information. Hence, these schemes tend to cause inefficient allocation of requests and increased server response time.

Although there are some recent studies that suggest innovative load management schemes based on optimization algorithms, artificial intelligence techniques, and energy-related resource scheduling, these are typically accompanied by substantial computational overhead and complex mechanisms. These solutions are generally designed in a virtual machine environment in cloud infrastructure and are not very effective in a light-weight and microservices-based framework. Furthermore, existing studies are generally incapable of a deployable framework that allows a direct comparison among load balancing schemes in a controlled environment.

IV. SYSTEM ARCHITECTURE

A. Architecture overview

The proposed architecture of the system, as presented in Fig. 1, aims to assess and determine the efficacy of the existing as well as custom-designed load balancing techniques. The system architecture is designed with a modular, micro-service approach with Docker isolation and scalability along with the Nginx reverse proxy/load balancer. The architecture allows experiments to be conducted to determine the efficacy of the system under joint load conditions, along with real-time performance analysis.

B. Component Description

1) User Device/Client Interface:

The user device simulates the clients that send HTTP requests using the browser simulation interface. Simulation Interface.The simulation allows the user to choose the load balancing technique that they want to evaluate as well as allowing the user to send several concurrent HTTP requests.

2) Nginx Reverse Proxy Load Balancer:

Nginx acts as the entry level for requests to route through it. Based on the configuration, Nginx directs the request to the dedicated server for processing. Nginx provides various load balancing methods, which assists in high availability by directing the request.

3) Docker Containers (Algorithm Specific Services):

Each and every load balancing algorithm, be it Round Robin, Least Connection, or the recently introduced Custom algorithm, is processed within its own Docker container. This enables independent processing of the algorithm, scaling, or fair comparison in similar networking circumstances Containers also offer better reliability for the entire system.

4) Backend Processing & Response Handler:

Each of these containers receives incoming requests, then transfers information about their responses to a response handling module. Here, vital information such as response time, server load, request number, and processing time gets recorded, which is important for later analysis of performance comparisons.

C. Data/Request flow

As depicted in Figure 1, the client request message proceeds from the user device to the Nginx reverse proxy. Nginx uses the chosen strategy to direct each request to the corresponding algorithmic container. The request message is processed, and a corresponding response message along with performance information is sent back to the response handler. Lastly, the data is gathered together and presented using the browser interface. This demonstrates the effective distribution of loads and how it increases performance and optimizes resources.

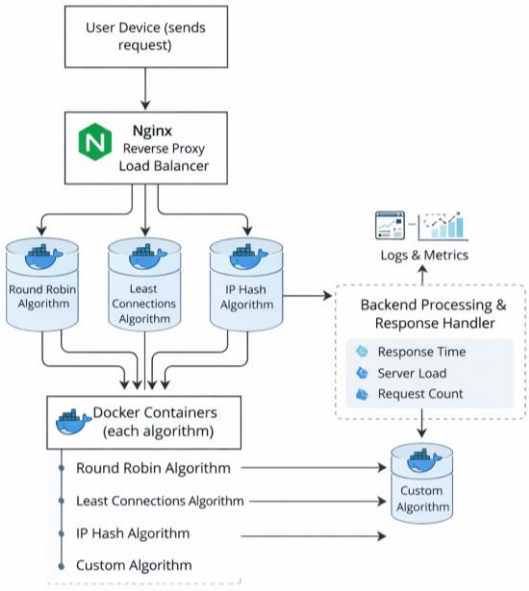

Fig 1. Containerized load balancing architecture

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 02 February 2026

V. METHODOLOGY

The proposed methodology will test and compare typical and custom load balancing algorithms in a controlled, containerized environment. The design of the system is modular, using Docker to deploy the system, while Nginx will serve as a reverse proxy acting as a load balancer. Every load balancing approach is containerized in its own Docker container, which ensures a fair comparison across similar traffic patterns and system constraints. This is scalable, fault-isolated, and guarantees repeatable results for every experiment conducted while being very much similar to actual cloud native environment practices.

Client requests are made using a user interface or browser simulation. The request is subsequently routed to the nginx reverse proxy. Nginx, with its predefined routing configurations, routes requests to the corresponding Docker container with the chosen load balancing algorithm. Each one of these different containers applies its particular principle of load distribution, be it Round Robin, Least Connections, IP Hash, Weighted Round Robin, or the proposed custom algorithm, and then routes the request to backend processing services. This is a highly decoupled architecture because the shifting of algorithms does not require touching the client-side request generation mechanism. The response handler module at the center also gathers performance metrics on response time, server load, and count.

A. Docker-Based Deployment Setup

Docker containers are used to wrap each load balancing algorithm individually. This helps in ensuring that all the algorithms are subjected to the same set of constraints, thereby removing any bias in the environment when the performances are compared. Using containers, there is rapid deployment and resource utilization, which makes this framework apt for testing adaptive load balancing algorithms in distributed systems.

B. Nginx Routing Configuration

Nginx server acts as a reverse proxy server and operates as a point of entry for incoming HTTP requests. There are separate sections defined for each load-balancing algorithm defined in the nginx.conf file for the upstream server. Nginx, based on user preference or system setup, directs incoming HTTP requests to the corresponding Docker container. This setup enables proper handling of incoming HTTP requests, guaranteed availability, and smooth routing, irrespective of modifications at the client end.

C. Custom Adaptive Load Balancing Algorithm

The new custom algorithm will overcome the shortcomings of the static techniques of load balancing by adding real-time server performance metrics to the routing decisions. While the traditional algorithms use request rotation or connection count solely, the custom algorithm dynamically assesses server load and response time to select the best server for each coming request.

Each server also submits periodic updates on its current load condition or response times. Once the new request arrives, the algorithm also assigns to each server its effective load score. Then, the algorithm selects the server with the smallest load score. Servers running above certain thresholds have reduced priorities to prevent the loading condition.

1) Custom Algorithm Logic:

- Receive incoming request from Nginx
- Pull current server load and response time metrics
- Calculate an effective load score for each server
- Choose the least load score server
- Forward the request to the chosen server
- Log response metrics after processing
- Update server status for subsequent requests

D. Pseudo-Code for Custom Load Balancing Algorithm

```
Input: Incoming Request R
Output: Selected Server S

Initialize minLoad = ∞
Initialize selectedServer = null

For each server Si in ServerList:
    currentLoad = Si.activeConnections
    responseTime = Si.avgResponseTime
    loadScore = currentLoad + responseTime

    If loadScore < minLoad:
        minLoad = loadScore
        selectedServer = Si

If selectedServer is not null:
    Forward request R to selectedServer
Else:
    Forward request R to default server

Log response time and server load
Update metrics database
```

E. Flowchart Description

Fig. 2 flowchart depicts how the proposed methodology will be performed. The user device initiates an HTTP request; this invokes the Nginx reverse proxy. The Nginx reverse proxy redirects the request to the Docker container, which then forwards the request to the selected load-balancing algorithm. This algorithm satisfies the request based on its own decision logic as to the best server.

After this process is completed, its output or response is passed to another module known as a response handler module. This module logs vital performance details that include response time, environment load, and request counts. These metrics are logged and stored for subsequent analysis

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 02 February 2026

and comparison. The flowchart clearly shows how containerization, reverse proxy routing, and adaptive decision-making can be combined to effectively assess the performance of load balancing.

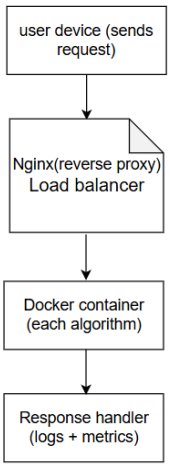

Fig 2 Flowchart of Load Balancing using Nginx and Docker

VI. RESULTS AND DISCUSSION

The experimental results clearly show that there are significant performance differences among the load balancing algorithms tested under the same traffic conditions. This indicates that the proposed load balancing algorithm had the lowest response time since it could efficiently process concurrent requests by dynamically adapting to the real-time server loads and response conditions. Traditional static algorithms, like Round Robin and IP Hash, incurred a higher latency due to the inability to adapt to the current condition of each server, which leads to inefficient request distributions and results in increased processing delays. Compared with the Round Robin and IP Hash, the Least Connections and Weighted algorithms have moderate improvements by taking partial account of the server utilization. However, during sudden changes in network traffic, neither can easily adapt to the changes. The graphical and tabular results confirm in unison that the lesser the response time, the better the efficiency of the system. Hence, adaptive load balancing strategies outperform conventional methods in high-concurrency dynamic environments.

A. Experimental Setup

To test the efficiency of the load balancing techniques, a simulation test environment was developed using Docker containers with Nginx as the reverse proxy server. For each of the load balancing techniques, 100 concurrent HTTP requests were made. All techniques were tested with the same hardware setup, ensuring that each technique had equal constraints on the resource level of the container. Also, performance data was automatically recorded in real time while the processes were running, including the time of start, finish, processing time, and server load. By doing this, an impartial and reproducible assessment of algorithm efficiency could be made under simulated conditions of high traffic.

```
[+] Running 6/6-leastconn-2-1 dockernginx-backend-1])
 ✓ Network dockernginx_app-network          Created
 ✓ Container dockernginx-backend-java-2-1   Created
 ✓ Container dockernginx-backend-java-3-1   Created
 ✓ Container dockernginx-backend-java-1-1   Created
 ✓ Container dockernginx-frontend-1         Created
 ✓ Container dockernginx-nginx-1            Created
```

Fig 3 Creating Docker containers

B. Quantitative Results

Table I presents the measured performance metrics obtained for each load balancing algorithm.

Table I. Performance Comparison of Load Balancing Algorithms

Algorithm	Start Time	End Time	Time Taken	load
Round Robin	1.43.55	1.43.58	3000ms	Server1,2,3
Least conn	1.47.25	1.47.27	2000ms	Server 2,3
Ip hash	1.48.22	1.48.26	4000ms	Server 1
weighted	1.49.02	1.49.05	3000ms	Server 1,2
custom	1.49.40	1.49.41	1000ms	Server 3,1

C. Analysis of Results

From the tabulated data, it has been seen that there are obvious differences in the performance of various algorithms considered in this study. The custom load balancing approach had the lowest response time of 1000ms, which reflects its optimal efficiency while managing multiple concurrent requests. This performance can be linked to the fact that the approach has the capability to handle and route requests dynamically on the basis of real-time server attributes and responses.

The "Least Connections" algorithm outperformed the Round-Robin algorithm with a response time of 2000 ms, as it takes into consideration the number of active connections before routing traffic. However, the algorithm performed inadequately in handling bursts, with irregular disconnection times of connections causing inefficient distribution of loads.

Despite equally redistributing the requests across all servers, the Round Robin algorithm registered a higher value of 3000ms in the response time. This is attributed to the fact that the algorithm does not consider the processing capacity or workload of the servers.

The IP Hash algorithm had the longest response time of 4000ms because of the cyclic routing of traffics to the same server, which was determined by the IP hashing of the clients. This led to server overload and poor adaptability to dynamic workloads.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)

Volume 13 Issue 02 February 2026

The **Weighted algorithm** performed similarly to Round Robin (3000 ms) but showed slightly improved load awareness by assigning proportional traffic. However, fixed weights limited its adaptability in real-time scenarios.

D. Graphical Comparison

Figure 4 illustrates a ***comparative bar chart*** representing the response time (in milliseconds) for each algorithm. The visual comparison clearly highlights the performance advantage of the custom algorithm, which shows the shortest bar, indicating minimal processing time. In contrast, IP Hash displays the tallest bar, confirming its inefficiency under concurrent request scenarios. The graph reinforces the numerical results presented in Table I and provides an intuitive understanding of algorithm efficiency differences.

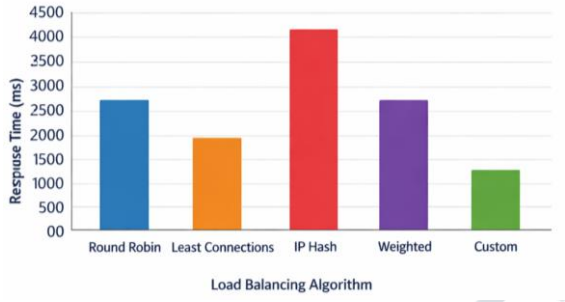

Fig 4 Response time comparison of load balancing algorithms

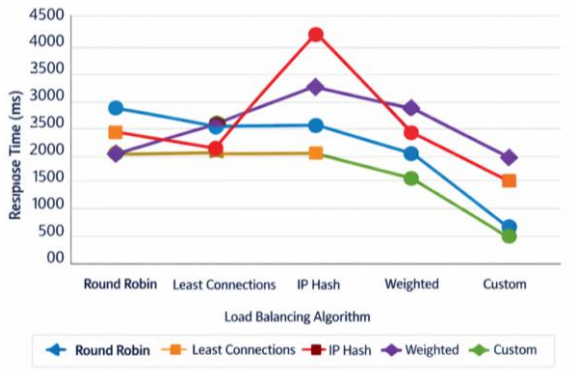

Fig 5 response time trends of load balancing algorithms

E. Discussion

The outcome of the experiment proves the legitimacy of the need for new dynamic load balancing techniques, as the static techniques, like Round Robin and IP Hash, are inadequate for the current dynamic pattern. The simplicity in the implementation process, however, is the main reason these techniques are inefficient. “This custom algorithm, on the other hand, reacts to the dynamic conditions of the server in real time, ensuring equilibrium in the loading process and, in effect, improving the response time.” Using Docker, the experiment ensured a completely isolated test environment, while the Nginx tool facilitated the simulation of the way data would flow in the actual production environment. Nevertheless, the results clearly show the preference for the more adaptive, intelligence-driven load balancing strategies over the conventional ones for variable traffic environments with a high concurrency limit.

VII. CONCLUSIONS

This research offered a full analysis of various load balancing techniques within a simulation setup that makes use of Docker and Nginx. Through applying the same hardware limitations and concurrent request loads to each Technique, the above-mentioned experiments carried out a fair and accurate comparison of various performance measures. The result of this research clearly depicted the ineffectiveness of the more traditional and simpler techniques for static scenarios, Round Robin and IP Hash, when adapting to dynamic workload, along with generating network latency and sub-optimal use. The techniques that performed partially for the awareness of the server status, Least Connections and Weighted Round Robin, attained slight improvements.

On the flip side, the custom load balancing algorithm performed better than the others by adaptively routing the requests based on the real-time conditions of the servers. This adaptability led to lower response times and better load distribution, thus justifying the need for adaptive load balancing techniques in high-concurrent systems. The adoption of containerization facilitated isolated and reproducible testing to prove the feasibility of applying adaptability in modern cloud systems. In conclusion, this study clearly emphasizes the need for adaptive and smart load balancing techniques and thus constitutes a significant base for future studies.

REFERENCES

[1] Saxena, D., Singh, A. K., Lee, C. N., & Buyya, R. (2023). *A sustainable and secure load management model for green cloud data centers*. Scientific Reports, 13, 491. https://doi.org/10.1038/s41598-022-27349-5

[2] Khalil, M. I. K., Shah, S. A. A., Taj, A., Shiraz, M., Alamri, B., Murawwat, S., & Hafeez, G. (2022). *Renewable-aware geographical load balancing using option pricing for energy cost minimization in data centers*. Processes, 10(10), 1983. https://doi.org/10.3390/pr10101983

[3] Yang, T., Hou, Y., Lee, Y. C., Ji, H., & Zomaya, A. Y. (2022). *Power control framework for green data centers*. IEEE Transactions on Cloud Computing, 10(4), 2876–2886. https://doi.org/10.1109/TCC.2020.3022789

[4] Wang, J. V., Ganganath, N., Cheng, C.-T., & Tse, C. K. (2020). *Bio-inspired heuristics for VM consolidation in cloud data centers*. IEEE Systems Journal, 14(1), 152–163. https://doi.org/10.1109/JSYST.2019.2900671

[5] Kumar, M., Singh, P., & Kumar, R. (2022). *Energy-efficient task scheduling and resource allocation for improving the performance of cloud–fog environments*. Symmetry, 14(11), 2340. https://doi.org/10.3390/sym14112340

[6] Anusooya, G., & Vijayakumar, V. (2021). *Reduced carbon emission and optimized power consumption using containers over virtual machines*. Journal of Green Computing, 5(2), 45–56.

[7] Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R., & Serazzi, G. (2019). *Dynamic load balancing policies for distributed web systems*. Journal of Parallel and Distributed Computing, 132, 48–61. https://doi.org/10.1016/j.jpdc.2019.05.006

[8] Reese, W. (2020). *Nginx: High-performance load balancing and reverse proxy for web applications*. IEEE Internet Computing, 24(3), 72–78. https://doi.org/10.1109/MIC.2020.2976637

[9] Pahl, C. (2018). *Microservices: A systematic mapping study*. Journal of Systems and Software, 137, 414–434. https://doi.org/10.1016/j.jss.2017.12.019

[10] Xu, J., Chen, M., Li, X., & Zhou, J. (2021). *Adaptive load balancing for cloud applications based on response time*. Future Generation Computer Systems, 118, 298–310. https://doi.org/10.1016/j.future.2021.01.012

[11] Singh, S., & Chana, I. (2017). *QoS-aware autonomic resource management in cloud computing: A systematic review*. ACM Computing Surveys, 49(3), 1–46. https://doi.org/10.1145/3017422

[12] Merkel, D. (2014). *Docker: Lightweight Linux containers for consistent development and deployment*. Linux Journal, 2014(239), 2–11.

[13] Adzic, G., & Chatley, R. (2017). *Serverless computing: Economic and architectural impact*. Proceedings of the 2017 ACM Joint Meeting on European Software Engineering Conference, 884–889. https://doi.org/10.1145/3106237.3117767

[14] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). *Microservices: Yesterday, today, and tomorrow*. Present and Ulterior Software Engineering, 195–216. https://doi.org/10.1007/978-3-319-67425-4_12

[15] Li, W., Zhang, Y., Chen, X., & Li, H. (2020). *Intelligent load balancing using real-time system metrics in cloud environments*. Journal of Cloud Computing, 9, 45. https://doi.org/10.1186/s13677-020-00187-3