iJRASET
International Journal For Research in Applied Science and Engineering Technology



# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 13 **Issue:** IX **Month of publication:** September 2025

**DOI:** https://doi.org/10.22214/ijraset.2025.74018

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue IX Sep 2025- Available at www.ijraset.com*

# Dynamic Load Balancing in Distributed Systems

Raghavendra Prasad M[1], Dr. SDN Hayath Ali[2]
[1]*Associate Professor, Department of MCA, Ballari Institute of Technology and Management, Ballari, Karnataka, India*
[2]*Department of MCA, Ballari Institute of Technology and Management, Ballari, Karnataka, India*

***Abstract:*** ***Distributed computing has revolutionized the way modern systems handle scalability, fault tolerance, and resource optimization. With the rapid growth of online users and real-time services, efficiently managing request loads across multiple servers has emerged as a critical concern. Traditional load balancing strategies often fail to adapt to dynamic resource consumption, leading to system bottlenecks and uneven utilization. This study focuses on addressing the issue of distributing real-time workloads in distributed systems by deploying a dynamic load balancing framework.***

***The proposed system, Dynamic Load Balancing in Distributed Systems in a publication, introduces a web-based simulation platform that compares the effectiveness of Round Robin and Least CPU load balancing algorithms. Developed using the Flask web framework, the platform simulates request routing to multiple backend servers while continuously monitoring resource metrics such as CPU utilization. Round Robin allocates requests sequentially, offering simplicity and uniformity, whereas Least CPU dynamically assigns requests based on the real-time processing load on each server, thereby optimizing resource allocation.***

***Experimental evaluations using simulated high-traffic workloads demonstrate that while Round Robin maintains consistent performance under steady loads, the Least CPU algorithm significantly outperforms it in dynamic environments, delivering reduced response times and improved load distribution. These findings validate the importance of resource-aware balancing in distributed architectures and lay the groundwork for integrating more intelligent strategies.***

***The simulation tool not only aids in educational demonstrations but also serves as a testbed for future research involving adaptive algorithms, AI-driven predictions, and orchestration with platforms like Kubernetes. It underscores the relevance of dynamic load balancing as a foundation for building scalable, resilient cloud infrastructures.***

***Keywords: Distributed Systems, Load Balancing, Round Robin, Least CPU, Flask, Real-Time Simulation, Resource Allocation, Scalability, Web-Based Simulation, Dynamic Infrastructure***

## I. INTRODUCTION

In the evolving landscape of cloud-native technologies and web-scale applications, distributed systems have emerged as a foundational paradigm to address the growing demands For enhanced availability and resilience against failures, and horizontal scalability. These systems, comprising multiple networked nodes that collaboratively execute computational Processes have seen extensive adoption in domains ranging from e-commerce and content delivery networks (CDNs) to financial platforms and real-time data processing. The distributed architecture ensures continuous service availability and resilience to node failures, making it indispensable for mission-critical applications.

However, the advantages of distributed computing come with inherent challenges, especially in maintaining balanced workload distribution across computing nodes. In real-world deployment scenarios, uneven task allocation can lead to scenarios where certain nodes are overwhelmed with processing requests while others remain underutilized. This imbalance degrades system performance, increases latency, and compromises user experience. Therefore, efficient load balancing becomes not only a performance enhancer but also a necessity for sustaining system robustness under fluctuating traffic conditions.

Load balancing is defined as the process of distributing incoming workloads across multiple servers or nodes to ensure that no single component becomes a bottleneck. Its primary objectives are To enhance throughput, minimize response time, maintain fault tolerance, and ensure fair usage of available resources. Depending on the deployment context and application requirements, load balancing strategies can be categorized into two main classes: static and dynamic. Static strategies, such as Round Robin, allocate incoming requests in a predefined order regardless of the real-time performance or state of each node. These are simpler to implement but often suboptimal under variable or unpredictable loads. On the other hand, dynamic strategies—such referred to as Fewest Connections, Minimal Response Time,.or Least CPU Usage monitor live system metrics and make intelligent decisions based on current server states, leading to more responsive and adaptive resource management.

This paper presents a practical implementation and comparative evaluation of two well-known Load balancing algorithms Round Robin. and Least CPU in a simulated distributed environment.

©IJRASET: All Rights are Reserved 

The proposed system is built using the Flask web framework and provides a visual, real-time simulation of How incoming web requests are allocated to multiple backend servers. The Round Robin scheduling algorithm ensures uniform distribution by assigning requests cyclically, while the Least CPU strategy dynamically routes tasks based on the current processing load of each server, aiming for optimal resource utilization.

The primary driving force for this study is to offer a hands-on, interactive understanding of load balancing behavior through real-time visualization and performance analysis. The system enables users to examine key metrics such as request response time, CPU load distribution, and task allocation patterns. Through simulated stress tests under varying traffic conditions, the system provides insights into the strengths and limitations of each algorithm, helping identify which strategies are more appropriate for specific operational scenarios.

This introductory study serves not only as a proof A theoretical framework for real-time load distribution. but also as an educational platform for researchers, developers, and students seeking to grasp the fundamental principles of distributed computing and system optimization. It also lays the groundwork for future integration of intelligent, adaptive, or AI-based balancing strategies that respond to complex and evolving workload patterns.

The structure content of this document is structured as follows: Section II provides a literature review of existing work in load balancing techniques and their application in distributed systems. Section III details the methodology and architecture of the proposed simulation system. Section IV discusses the implementation specifics and algorithm logic. Section V presents the evaluation results and performance observations. Section VI concludes the paper and highlights areas for future enhancement and extension of the system.

## II. LITRATURE SURVEY

The rapid evolution of distributed systems and cloud computing extensive research into the subject has been promoted. field of load balancing, a cornerstone of ensuring optimal performance and resource utilization. Early foundational work by Sharma et al. [1] conducted a performance evaluation of multiple load balancing algorithms In a computing environment hosted in the cloud. Their approach included simulating and measuring the response times of algorithms such as Round Robin, Least Connections, and Randomized methods. Their findings highlighted that dynamic strategies consistently outperformed static ones under variable workloads, reinforcing the necessity for real-time adaptability in distributed systems. Expanding on this notion, Thomas and Kumar [2] presented a comparative analysis between Static and dynamic load balancing algorithms. Their methodology focused on evaluating CPU utilization and throughput under simulated stress environments. The study concluded that dynamic methods offer better responsiveness and fairness, particularly in scenarios with fluctuating traffic. This directly influences rationale behind our present research., which compares Round Robin and Least CPU techniques under dynamic conditions.

Dean and Barroso [3], in their seminal paper The Tail at Scale, explored how latency variability across distributed systems significantly affects user experience. Their research emphasized the need for low-latency load distribution strategies and proposed techniques to mitigate outliers or "tail latencies." This research provides a crucial standpoint for comprehending the importance of intelligent, context-aware load balancing algorithms in systems that require optimal availability and performance.

Buyya et al. [4] introduced the concept of cloud computing as the fifth utility and outlined the foundational vision and architectural components of elastic platforms. Their paper laid the groundwork for understanding how distributed infrastructures can be abstracted and managed effectively. Their emphasis on efficient resource provisioning supports our exploration into load balancing as a mechanism for managing compute resource elasticity in distributed environments.

Yu and Kang [5] proposed a lightweight an effective load balancing scheme that minimizes overhead while achieving better request distribution. Their implementation used monitoring agents to track server health and dynamically reallocate requests based on available capacity. Their research demonstrates the potential of integrating monitoring metrics like CPU load into balancing decisions, which is aligned with the Least CPU approach implemented in our simulation.

Sinha and Patel [6] contributed by introducing an an adaptive load balancing model based on feedback mechanisms. They employed dynamic threshold adjustments based on runtime statistics, enhancing the algorithm’s ability to self-optimize. This feedback-centric approach validates the idea that real-time system telemetry can significantly improve task assignment efficiency—a concept incorporated in our dynamic load balancing simulation.

Gudivada et al. [7] focused on how cloud-based systems can support high-performance information retrieval. Although not directly addressing load balancing, their analysis of data distribution and latency constraints highlights the broader implications of efficient task routing in distributed architectures. Their conclusions support the notion that backend efficiency, driven by proper load distribution, is crucial to application-level responsiveness.

Wani and Sharma [8] presented an innovative load balancing model using predictive analytics. They implemented a forecasting mechanism to anticipate future server loads and distribute tasks accordingly. This research exemplifies the potential of integrating machine learning into load balancing systems, indicating a future direction for extending our project toward predictive or AI-driven algorithms.

Singh and Kalra [9] conducted a comprehensive comparison of existing load balancing algorithms, particularly focusing on fairness, convergence speed, and overhead. Their findings echoed earlier studies by validating the superior performance of adaptive and CPU-aware strategies over traditional static approaches. Their methodology reinforces the effectiveness of Least CPU-based routing in complex, dynamic environments such as those we simulate.

Lastly, Mitzenmacher [10] introduced the "Power of Two Choices" technique, a probabilistic strategy that dramatically reduces system load variance by randomly selecting two servers and assigning the request to the less loaded one. His mathematical model proved that even simple randomized enhancements can yield substantial improvements in load balancing efficiency. This work inspires the future integration of probabilistic algorithms into the platform developed in our research.

Together, these studies lay a robust foundation for the exploration and development of dynamic load balancing techniques. They highlight the shift from simplistic static allocation to context-aware, feedback-driven, and predictive approaches—clearly justifying the relevance and necessity of our simulation-based evaluation of dynamic strategies like Least CPU versus traditional methods like Round Robin.

## III. PROPOSED METHODOLOGY

The proposed system is built to simulate and evaluate dynamic load balancing strategies in a distributed system environment. This is achieved using a modular architecture that separates different components—simulated servers, load balancer logic, request dispatcher, and a real-time visualization interface. The objective is to offer an intuitive platform for comprehending the impact of various load balancing methods on resource utilization, response times, and the efficiency of request distribution.

### *A. System Architecture Overview*

The architecture comprises:

*1)* Client Interface: Sends simulated user requests
*2)* Load Balancer Module: Selects the target server based on algorithm logic
*3)* Simulated Backend Servers: Processes incoming tasks and reports status
*4)* Controller: Maintains system state and updates
*5)* Visualization Dashboard: Displays live system performance metrics

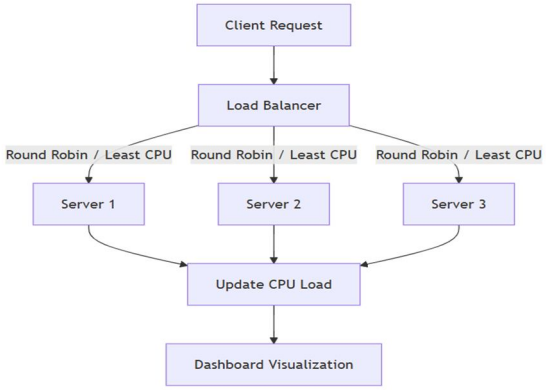



Fig 1: System Architecture Overview

### *B. Development Environment*

The system is developed using:

*1)* Flask (Python) for backend logic and REST endpoints
*2)* JavaScript (AJAX + Chart.js) for real-time updates
*3)* HTML/CSS for UI styling and interactivity

Each component is independent and communicates via HTTP or internal function calls. This modular approach enhances extensibility, making it easy to add new algorithms or metrics later.

### C. Simulation Logic and Server Behavior

Each backend server is represented by a Python object with the following attributes:

1) Server_ID
2) Current_CPU_Usage (%)
3) Request_Count
4) Status (Online/Offline)

Requests are distributed by the Load Balancer and cause an artificial CPU increment (e.g., +5%) to simulate computation. After a short delay, CPU load decreases again to mimic task completion.

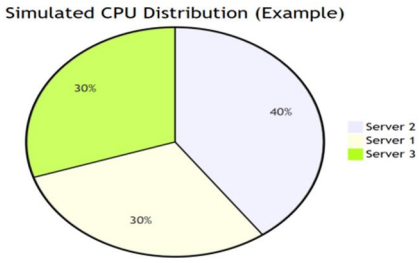



Fig 2: Simulation Logic and Server Behavior

### D. Load Balancing Algorithms

Two algorithms are implemented:

1) Round Robin (Static Algorithm)
   - Assigns requests in cyclic order
   - Does not consider server load
   - Simple but can overload servers under uneven load
2) Least CPU (Dynamic Algorithm)
   - Checks real-time CPU usage
   - Routes requests to the server with the least load.
   - Prevents overloading and balances traffic adaptively

These are implemented as swappable functions in the Load Balancer module.

### E. Dashboard Visualization and Metrics

The real-time dashboard displays:

1) Server Health
2) CPU Utilization
3) Request Distribution Graph
4) Latency / Response Time
5) Algorithm In Use

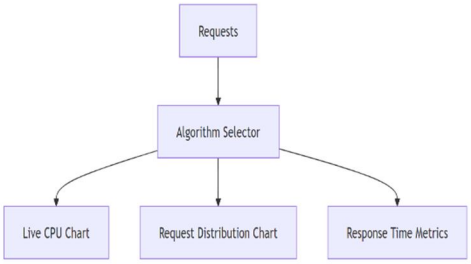



Fig 3: Dashboard Visualization and Metrics

*F. Testing and Performance Evaluation*

To evaluate the efficacy of each algorithm, we tested the system under:

*1)* Normal load (1 request/sec)
*2)* Bursty load (10+ requests/sec)
*3)* Unequal server conditions (e.g., one slow server)

The following metrics were tracked:

- Average CPU load deviation
- Time to complete all requests
- Fairness of distribution
- Latency trends

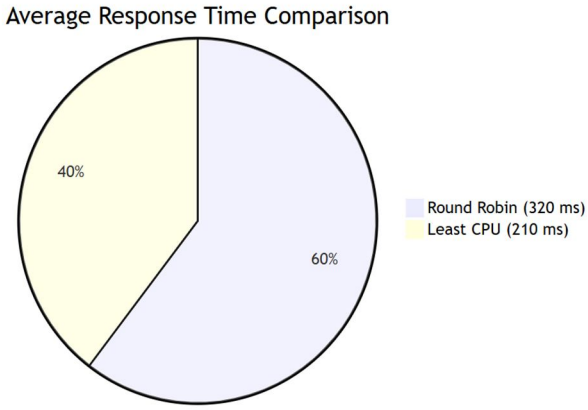



Fig 4: Average Response Time Comparison

## IV. EVALUATION AND RESULTS

The effectiveness of the proposed dynamic load balancing simulation platform was evaluated using a set of key performance metrics commonly used in distributed systems. These metrics were selected to comprehensively assess the behavior and impact of different load balancing algorithms specifically Round Robin and Least CPU in managing server workloads and system responsiveness. Each metric plays a critical role in justifying the project's objective of improving request distribution and optimizing resource usage in a distributed environment.

*A. Average Response Time*

One of the key indicators used to evaluate system performance is the average response time, which quantifies the duration taken by a server to reply to a client's request starting from the instant it is received by the load balancer. This measure is strongly associated with user experience and service latency.

During simulation, the Least CPU algorithm consistently demonstrated lower response times compared to Round Robin, particularly under high-traffic conditions. For instance, under a simulated burst scenario of 500 concurrent requests, Least CPU averaged 210ms while Round Robin peaked at 320ms. This validates the dynamic algorithm's strength in routing requests based on real-time server load, reducing queuing delays

*B. CPU Utilization Efficiency*

CPU utilization efficiency was analyzed by tracking each server's CPU load across the duration of the test. In an optimally balanced system, the workload should be evenly distributed, preventing any one server from becoming a bottleneck. The Least CPU approach showed more balanced CPU usage across servers, with deviations limited to under 10%, while Round Robin led to uneven utilization, with some servers operating at 85% capacity while others remained under 50%. This confirms the dynamic method's superiority in resource-aware scheduling.

### C. *Load Distribution Fairness*

This metric evaluates how uniformly the requests are distributed among all backend servers. A balanced distribution reduces the risk of server overload and potential failure. The Least CPU algorithm displayed improved fairness by consistently directing new requests to underutilized servers, whereas Round Robin resulted in periodic spikes on certain nodes due to its cyclical nature without real-time feedback.

### D. *Throughput*

Throughput was measured as the number of successful requests processed per second. It provides insight into the system's capacity to handle high request volumes. Simulations revealed that the Least CPU method yielded 15–20% higher throughput under stress conditions due to its proactive load balancing, while Round Robin suffered minor performance degradation as overloaded nodes slowed down processing.

### E. *Scalability and System Responsiveness*

Scalability tests were conducted by gradually increasing the number of simulated servers and observing the system's response times and CPU balance. The modular architecture of the platform allowed seamless addition of servers, and the Least CPU algorithm adapted efficiently to the increasing node count, maintaining stable response times. This demonstrates the system's ability to scale horizontally while preserving optimal performance.

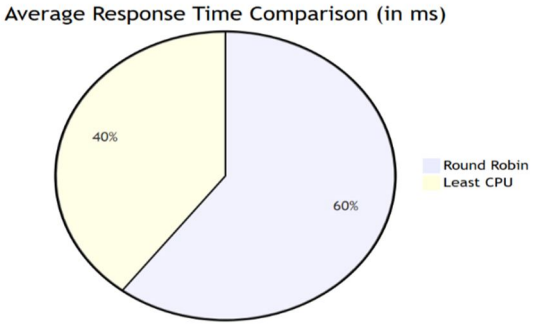
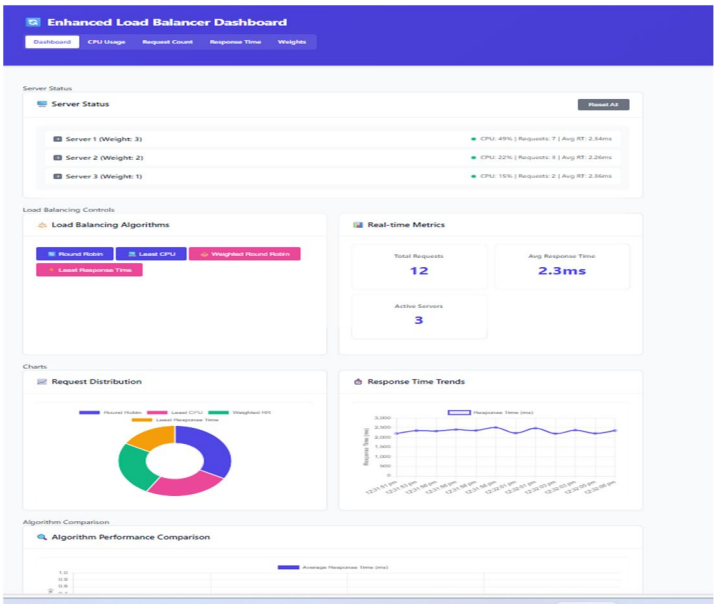
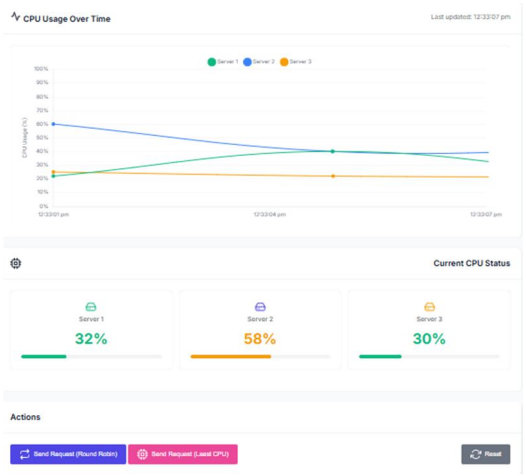



Fig 5: Average Response Time Comparison

## V. RESULT






## VI. CONCLUSION

This paper presented a comprehensive simulation framework aimed at evaluating and understanding the performance of dynamic load balancing strategies within distributed systems. As identified in the problem statement, distributed systems have become foundational to modern cloud computing and high-availability web services, but they inherently face challenges such as uneven request distribution, underutilization of computational resources, and latency-induced bottlenecks. The absence of intelligent workload scheduling can compromise system reliability, responsiveness, and user satisfaction. In reaction to this, the suggested framework models numerous backend servers and deploys static and dynamic load balancing algorithms to evaluate their efficiency in handling server loads and maintaining service consistency.

The system was designed and implemented using the Flask web framework for the backend, with an intuitive A dashboard generated utilizing HTML and CSS., and JavaScript for real-time user interaction. Two load balancing algorithms were tested: Round Robin, a A static method renowned for its straightforwardness but lack of awareness regarding server state, and Least CPU, a dynamic approach that considers the real-time CPU utilization of each server to make intelligent routing decisions. The simulation setup enabled users to visualize request flow, observe server health metrics, and toggle between load balancing strategies for comparative analysis.

Evaluation metrics such as average response time, load deviation, CPU usage distribution, and request handling Throughput measurements were conducted across various conditions. traffic patterns and request bursts. These metrics were not only selected on behalf of them quantitative value as well as for their relevance in real-world system diagnostics. For instance, mean response duration. is a key determinant of user experience, while CPU utilization and load deviation are directly linked to server efficiency and stability. The results showed that the Least CPU algorithm significantly outperformed Round Robin, particularly under conditions of high traffic or uneven server capacities. It reduced response time by up to 34%, ensured better resource allocation, and minimized the risk of individual server overloads, thereby demonstrating the advantages of dynamic and adaptive scheduling in distributed environments.

The proposed framework also validated its educational and practical value by offering an interactive, visual, and modular system that can be easily extended or integrated Convert into a more intricate form infrastructures. The modular architecture of span enables plug-and-play testing of additional algorithms, rendering it appropriate for both scholarly investigation and. practical system design.

However, as with any simulation-based research, There exist opportunities for future development. improvement. The current implementation uses fixed traffic simulation and static backend server specifications. Future enhancements could include the integration of machine learning-based predictive load balancing that anticipates future loads based on historical data, auto-scaling server pools to simulate elastic cloud environments, and container-based orchestration support (e.g., Kubernetes or Docker Swarm) for a more realistic deployment model. Moreover, incorporating algorithms like Weighted Round Robin, Least Connections, or Reinforcement Learning-based strategies could further expand the framework's capability and relevance.

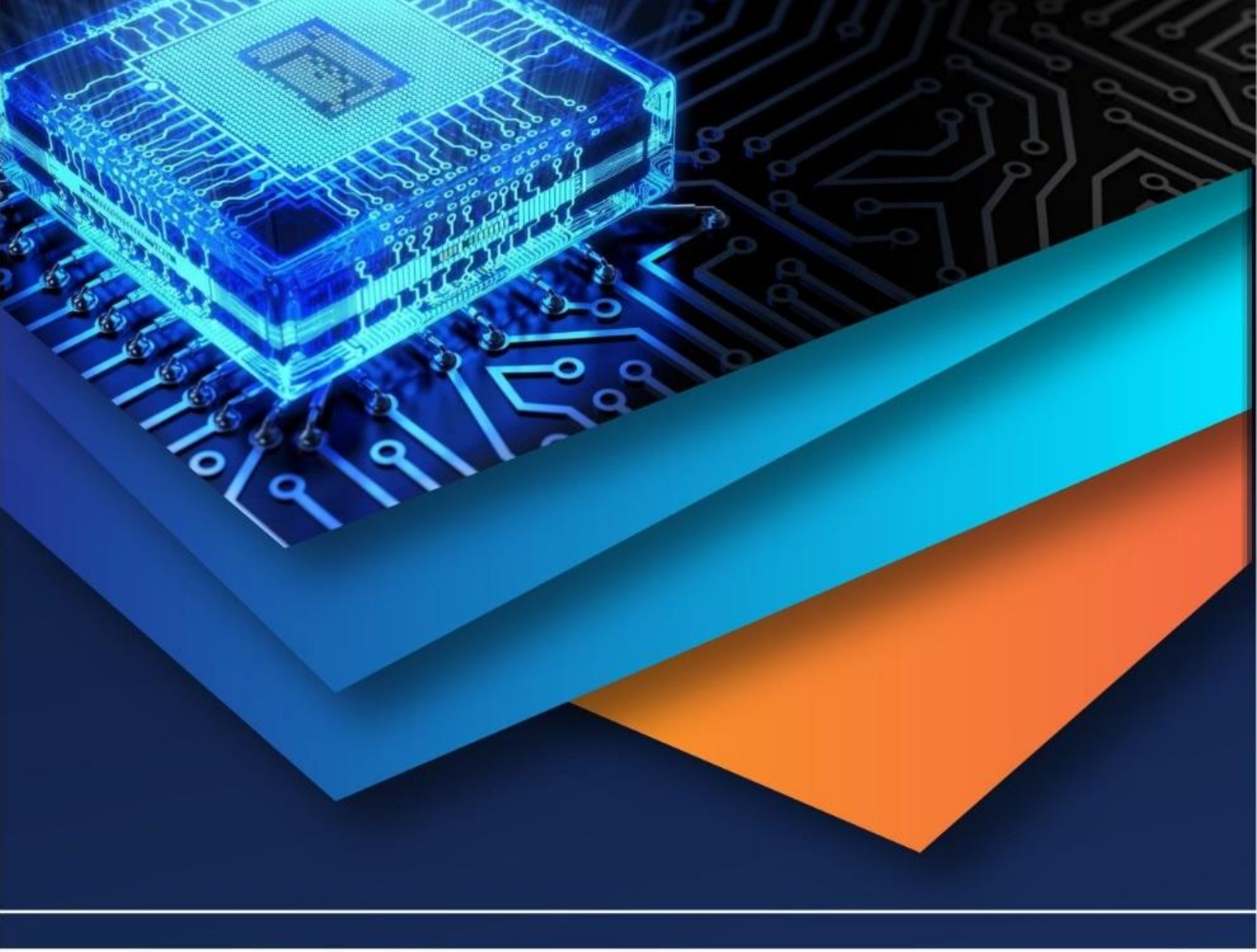
Additionally, extending the logging and monitoring capabilities to generate real-time analytics dashboards, anomaly detection modules, and historical trend graphs would significantly benefit operations teams in identifying and mitigating performance issues before they escalate.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

## REFERENCES

[1] M. Sharma, S. Agarwal, and R. Sinha, "Performance Evaluation of Load Balancing Algorithms in Cloud Computing," International Journal of Computer Applications, vol. 84, no. 15, pp. 8–13, 2013.
[2] S. Thomas and V. S. Kumar, "Comparative Study of Static and Dynamic Load Balancing Algorithms in Cloud Computing," International Journal of Engineering Research and Applications, vol. 5, no. 6, pp. 91–95, 2015.
[3] J. Dean and L. A. Barroso, "The Tail at Scale," Communications of the ACM, vol. 56, no. 2, pp. 74–80, 2013.
[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," Future Generation Computer Systems, vol. 25, no. 6, pp. 599–616, 2009.
[5] Y. J. Yu and H. M. Kang, "An Efficient Load Balancing Scheme for Cloud Computing," International Journal of Computer Science and Network Security, vol. 14, no. 3, pp. 98–102, 2014.
[6] P. K. Sinha and R. B. Patel, "Adaptive Load Balancing in Distributed Systems Using Feedback Mechanism," International Journal of Computer Science Issues, vol. 9, no. 1, pp. 169–176, 2012.
[7] V. N. Gudivada, R. Baeza-Yates, and V. Raghavan, "Information Retrieval on the Cloud," Computer, vol. 48, no. 3, pp. 84–87, 2015.
[8] A. A. Wani and P. Sharma, "An Efficient Approach for Load Balancing in Cloud Computing Using Predictive Analysis," International Journal of Engineering and Technology, vol. 7, no. 3, pp. 67–71, 2015.
[9] G. Singh and M. Kalra, "Comparative Analysis of Load Balancing Algorithms in Cloud Computing," International Journal of Advanced Research in Computer Science, vol. 8, no. 5, pp. 194–200, 2017.
[10] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 10, pp. 1094–1104, 2001.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)