A Relative Load Balancing (RLB) Method for Efficient Multi-Server Request Handling

Ameer Akram Mousa
Faculty of Information Technology, University of Babylon, Iraq
ameerakru@gmail.com (corresponding author)

Mahdi S. Almhanna
Faculty of Information Technology, University of Babylon, Iraq
mahdialmhanna@gmail.com

Received: 2 June 2025 | Revised: 23 July 2025 | Accepted: 2 August 2025

 DOI: https://doi.org/10.48084/etasr.12453

ABSTRACT

Load balancing is a critical technique in distributed systems and network infrastructures, designed to efficiently distribute incoming workloads or network traffic across multiple servers or resources. By dynamically allocating tasks, load balancing maximizes resource utilization, minimizes response times, and prevents any single server from becoming a bottleneck. This paper focuses on building a Relative Load Balancing (RLB) system between server groups to process client requests and avoid problems in static and dynamic load balancing systems, such as round robin and fewer connections. The requests were divided proportionally according to the capacity of a group of root servers. A system for processing text images and extracting data was developed to implement the principle of relative load balancing between servers, controlled by a master server called the backbone. This system handles the testing and measurement mechanism for the RLB request processing. The results showed that the proposed RLB system achieved a throughput of 101,000 B/s, a PDR of 97 %, average latency of 600 ms, and average response time of 2 s, with RAM usage of 2100 MB, main memory usage of 52% and CPU usage of 50%. In addition, the total processing time (Makespan) was 10 s, with a loss of time in waiting of 1 s and a gain of 8 s.

Keywords-backboned server; distributed systems; load balancing; resource utilization

I. INTRODUCTION

A decentralized computer network system consists of many nodes that have significantly different loads, which leads to congestion in data traffic and processing tasks, causing congestion and loss of packets during transmission and reception, and leading to a reduction in data flow rates and slow system responses [1]. As enterprise cloud environment applications grow increasingly complex and user demands escalate, traditional load balancing approaches often fail to optimally utilize available server resources. This inefficiency results in extended processing times, uneven server utilization, and ultimately affects user experience [2]. Load balancing techniques have evolved from simple static approaches, such as Round Robin (RR), where requests are distributed sequentially regardless of server capacity, to more sophisticated dynamic methods that consider server metrics, such as connection counts or response times [3]. However, these approaches still often result in suboptimal resource allocation, particularly in heterogeneous environments where server processing capabilities vary significantly [4]. Static load balancers do not adapt to changing server conditions, while many dynamic approaches react too slowly or rely on metrics that do not accurately reflect true processing capacity [5]. The problem statements for load balancing are as follows:

- Traditional load balancing methods suffer from critical limitations in modern multi-server environments [6].
- Resource underutilization: Static load balancing approaches distribute requests without considering the varying processing capabilities of different servers, leading to faster servers idling while slower servers continue processing [7].
- Response time inefficiencies: When requests are not distributed according to actual server processing capabilities, client response times are dictated by the slowest server in the cluster, creating unnecessary bottlenecks [8].
- Adaptability challenges: Most dynamic load balancing approaches fail to adapt effectively to changing server performance characteristics over time, resulting in suboptimal request distribution [9].

The proposed Relative Load Balancing (RLB) method addresses these limitations by introducing a proportional workload distribution mechanism based on empirically measured server group capacities. Rather than making distribution decisions based on simplistic predefined rules, the proposed RLB approach continuously evaluates actual processing performance and adjusts request distribution

accordingly, ensuring that server resources are utilized optimally according to their demonstrated processing efficiency.

Several load balancing strategies have been proposed to optimize resource utilization and reduce processing latency in distributed systems. In [10], a hybrid model was proposed to classify the number of files in the cloud according to specific types, such as audio, video, text maps, and images, based on the Ant Colony Optimization (ACO) algorithm to balance network load and solve the problem of slow and aggravating data traffic. The results showed good performance, offering scalability and robustness in the network. In [11], a set of load balancing algorithms, such as classic and hybrid load balancing, to improve cloud computing performance was reviewed. A brief explanation of the performance metrics, literature, and their effects was presented using a simulator. In [12], a model based on the use of fog computing resources and load balancing was proposed, based on the available resources in the network and relying on the PSW-Fog clustering algorithm to reduce energy consumption and execution time during the implementation of fog cloud applications and services. This model was simulated using iFogSim, and the results showed that it contributed to reducing the time delay and energy consumption costs.

In [13], a new method for dynamic load balancing was proposed based on a modified Particle Swarm Optimization (PSO) algorithm and machine learning. The algorithm simulated the best procedure to build a training model that contributes to improving performance after balancing the load on virtual nodes in the network, maintaining its balance, and reducing the waiting time for task execution. The results showed that the proposed algorithm outperformed its competitors in the field of load balancing and reduced the time required to perform the task. In [14], a modified genetic algorithm was proposed to improve the task scheduling process and find the optimal solution by reducing the number of iterations to select nodes in the network. This algorithm was compared with others, and the results showed that the total completion time of the operations and the average response time were low, improving the quality of service. In [15], two algorithms, ACO and PSO, were used to achieve effective load balancing for IoT tasks and reduce communication cost and response times. The proposed algorithm was compared with the RR algorithm, and the results showed that it improved the response time of the IoT application and achieved effective load balancing in fog nodes.

II. METHODOLOGY

The proposed method is based on the implementation of the RLB method in the context of text image processing, a computationally intensive task with varying processing requirements depending on image complexity. By dividing text image processing requests into parts and distributing them proportionally across server groups, the system demonstrates the practical benefits of relative load balancing while providing a useful text extraction service. The centralized backbone server architecture manages request distribution, collects performance metrics, and dynamically adjusts load balancing parameters to maintain optimal system performance.

The proposed RLB method introduces a dynamic, performance-based approach to load distribution. By proportionally allocating requests according to empirically measured server group capacities, the system optimizes resource utilization and minimizes overall processing time. The backbone server architecture enables centralized coordination with minimal overhead, while the continuous feedback mechanism ensures that the system adapts to changing performance characteristics. The implementation within a text image processing context demonstrates both the practical application of the RLB method and provides a robust testing framework for performance evaluation. This study aimed to:

- Develop a relative load balancing system that dynamically distributes request processing based on empirically measured server group capacities.
- Minimize the total completion time of the client request by avoiding scenarios where some servers complete processing while others continue to process.
- Create a self-adjusting load balancing mechanism that continuously optimizes the request distribution based on the observed processing performance.
- Implement and test the RLB approach within a practical text image processing application to validate its effectiveness.
- Quantify the performance improvements of the RLB method compared to traditional load balancing approaches.

The key novel contributions of this study include:

1. Proportional distribution: Develops a new algorithm that distributes request parts proportionally according to demonstrated server group processing capabilities rather than theoretical or static metrics.
2. Dynamic adaptation mechanism: Creates a feedback loop system that continuously refines load balancing parameters based on actual processing times, ensuring optimal distribution even as server performance characteristics change.
3. Centralized coordination architecture: Designs a backbone server architecture that efficiently manages request distribution, result aggregation, and load balance information updates with minimal communication overhead.
4. Application to text-image processing: Implements the RLB method in the context of text extraction from images, demonstrating its effectiveness in a computationally intensive real-world application.

The proposed RLB system architecture consists of three main components:

- Client: Initiates requests by uploading text image files through a web interface.
- Backbone server: Functions as the central coordinator and load balancer, responsible for receiving and registering

client requests, determining request distribution based on relative load balancing information, and distributing request parts to appropriate root server groups. In addition, it collects and aggregates the processed results and analyzes processing times to update the load balancing parameters, and then returns the combined results to the client.

- Root server groups: Consists of multiple server clusters (two in the current implementation), each containing one or more servers that receive request parts from the backbone server. They process assigned request portions (text-image processing) and return the results to the backbone server.

This architecture enables centralized control while allowing parallel processing across distributed server groups. Each root server within a group processes the same request parts, enabling fault tolerance, although the research implementation uses a single server per group for evaluation purposes, as shown in Figure 1.

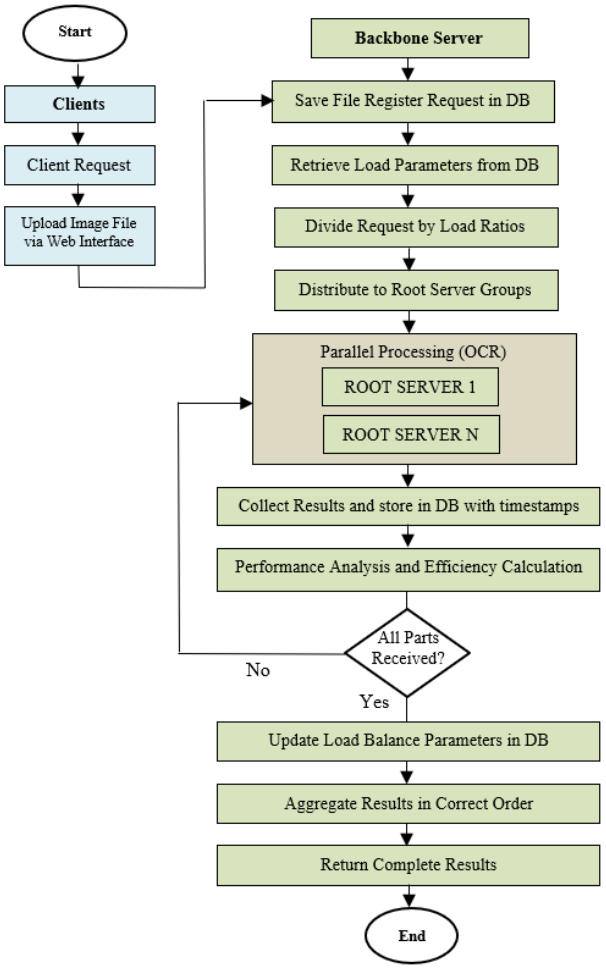

Fig. 1. The main block of the proposed system.

In addition, the proposed RLB method employs a continuous feedback and adjustment approach consisting of the following key elements:

- Request Registration: Client requests are received by the backbone server and stored in a database for tracking and analysis.
- Proportional Distribution: Request parts are assigned to root server groups based on their demonstrated processing efficiency from previous performance measurements.
- Parallel Processing: Root server groups process their assigned request parts simultaneously, utilizing specialized tools (Tesseract OCR) for text extraction from images.
- Result Aggregation: The backbone server collects and combines processed results from all root server groups.
- Performance Analysis: Processing times are analyzed to calculate the relative efficiency ratio between server groups, which is then used to adjust the future request distribution.
- Dynamic Adjustment: Load balancing parameters are continuously updated based on actual processing performance, ensuring optimal distribution over time.

The system implementation utilizes PHP for web interfaces and request handling, MySQL for data storage, and Python with Tesseract for image processing operations.

The detailed steps of the RLB method start with the reception of the request from a client through the web interface, and the backbone server saves the file and registers the request in a database. In the second stage, load balance parameters are retrieved by the backbone server from the database, including relative processing capacity ratios between server groups. In the third stage, the backbone server divides the request into parts based on load balancing parameters, and the request parts are distributed to the appropriate root server groups. In the fourth stage, root servers process in parallel the assigned request parts using Tesseract OCR, including text extraction and any additional analysis. The processed results from all root server groups are submitted to the backbone server and stored in the database with timestamps. The backbone server calculates processing times for each server group. In the last stage, the updated load balance efficiency ratios are stored in the load balance information table in the database, and the updated parameters will be used for subsequent request distribution and response aggregation.

```
Pseudocode of the Proposed Algorithm
Client Request Handling
  function handleClientRequest(file):
1.  Save uploaded file
    move_uploaded_file(file)
    registerRequestInDatabase(file)
2.  Retrieve load balancing information:
    servers = query("SELECT * FROM
    LoadBalanceTable")
3.  Distribute the request based on load
    balance parameters
    distributedRequests =
    divideRequestByLoadRatio( file,
    servers)
4.  Send requests to server groups
```

```
    for each server in servers:
      curl_multi_add_handle( request,
      server)
      curl_multi_select(servers)
    End for
5.  Collect responses
    for each server in servers:
      responses[server] =
        curl_multi_getcontent(server)
      storeResponse(responses[server])
    End for
7.  Calculate and update load balance
    values
    startTime =
      getFirstServerResponseTime()
    endTime = getLastServerResponseTime()
    processingTimeGap = endTime –
      startTime
    singlePartProcessingTime =
      getAveragePartProcessingTime()
    newLoadBalanceValue =
      floor(processingTimeGap /
    singlePartProcessingTime)
    updateLoadBalanceTable(
    newLoadBalanceValue)
8.  Return aggregated response to client
    result = query
      ("SELECT * FROM ResponseTable ORDER
      BY ID ASC")
    return result
```

A system was designed to collect network information, especially servers and the path leading to them, using ping and traceroute. The measurement was performed repeatedly, with a period of 15 minutes between each repetition, over a period of 24 hours. Then, the most efficient servers and paths that can work together in a specific period were determined using the mathematical coefficient of variation.

The performance of the RLB method is evaluated using the following metrics:

1. Total request completion time: The end-to-end time from request submission to final response delivery.
2. Server utilization efficiency: Percentage of time each server spends actively processing versus waiting.
3. Load distribution accuracy: How closely the actual processing time ratio matches the expected ratio based on load balancing parameters.
4. Adaptation responsiveness: How quickly the system adjusts to changes in server processing capabilities.
5. Processing time gap: The time difference between when the first server group finishes processing and when the last server group finishes processing (ideally minimized).
6. Scalability: System performance characteristics as the number of server groups and request complexity increases.
7. Total processing time, or Makespan (s), refers to the total time required to complete all tasks in a system. In the context of load balancing, it is defined as the maximum completion time among all processors or servers.

The backbone server contains several key components:

- Request handler: Receives and registers client requests.
- Response collector: Gathers processed results from root servers.
- Load balance calculator: Analyzes processing times and updates load parameters.
- Request distributor: Divides and assigns request parts based on load information.
- Response aggregator: Combines processed parts into a complete response.

The flow of operations follows this sequence:

- The client sends a request to the backbone server.
- The backbone server registers the request and retrieves load balance information.
- The request is divided and distributed to the root server groups.
- Root servers process their assigned parts.
- Processed results are returned to the backbone server.
- The backbone server analyzes processing times and updates load balance parameters.
- The aggregated results are delivered to the client.

This architecture enables continuous optimization through the feedback loop between request processing and load balance parameter adjustment. Specifically, the proposed RLB method adopts a relative load-balancing strategy to proportionally divide client requests of varying sizes among server groups based on real-time evaluation of server and route conditions. The system is built around a three-phase process.

A. Phase 1: Collect and Analyze Server and Route Information

Collect network information about servers and routes, where ping represents the servers' response time and traceroute represents information about servers and their routes, and then analyze the information using the coefficient of variation to extract the best available servers and routes and ignore the less efficient ones. Data on servers and the routes are collected and analyzed using the mathematical coefficient of variation, and the values are transformed into a binary array combined with AND logic.

B. Phase 2: Setup and Programming Servers

It was observed that traceroute requires a long time to collect information on the path leading to the server. This time increases as the number of routers on the way increases, as it collects information for all routers, often exceeding the time frame specified to measure this information for the network,

offering inaccurate information. To avoid this problem, the task of collecting information was distributed to more than one server (slave servers) controlled by a master server to complete the work. Slave servers are responsible for collecting information for a specific number of servers that need to be verified following this mechanism.

C. Phase 3: Master Server Configuration (Backbone)

1. Configure and install the main server with Windows 10.
2. Install Python version 3.11.9 (other versions can also be used). Note that path permission is given at the beginning of the installation.
3. Install XAMPP and run Apache and MySQL servers.
4. Install Python-related libraries.
5. Create a database to store information received from the slave server (ipdb).
6. Create a table (ipping1) to store ping information.
7. Create a table (tra) to store trace route information.
8. Create a table (servers) to store IP addresses, states, and maximum requests for slave servers.
9. Create a table (ippingb) to store the binary result of analyzing the ping information by the coefficient of variation.
10. Create a table (iptracb) to store the binary result of analyzing the traceroute information by the coefficient of variation.
11. Create a table (lgcary) to store the binary result of AND logic gate information.
12. Create a PHP file (addserver), represented as an HTML page in the client's browser, to add the information of the slave server.
13. Create a Python file (MSTserver.py) to manage the distribution of tasks between slave servers, receive the data collected about the real servers, analyze them by the coefficient of variance, convert them to binary arrays, and link them together using AND logic. Then identify the target servers that can be dealt with in a special table.
14. Create a PHP file (chn.php) to receive the data from the slave servers and store it in the master server database.

III. IMPLEMENTATION OF THE PROPOSED SYSTEM

The implementation of the RLB system was based on the following: Windows 10 OS installed on all servers (backbone and root servers), Tesseract OCR installed on root servers, backbone server implemented with PHP for client request upload and MySQL database for request/response storage and load balance parameters, and PHP scripts for request distribution, response collection, and load balance calculation. Root servers use Python scripts integrated with Tesseract for image processing and an HTTP interface for receiving request parts and returning results. The request table stores information about client requests, and the response table stores processed results from root servers. The load balance table stores efficiency ratios for server groups. Figure 2 shows the setup of the backbone and root servers.

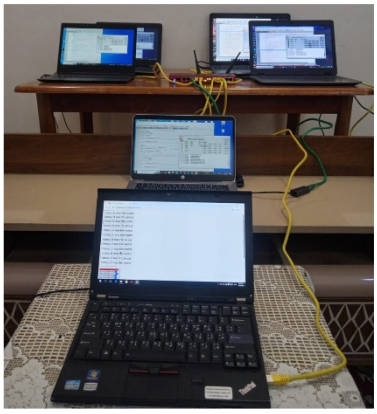

Fig. 2. Backbone and root servers.

IV. RESULTS

The results of the proposed system are based on a relative load balancing method (TGsrv) and two baseline approaches: single server (Ssrvr) and RR (TRsrv). In the Ssrvr configuration, all client requests are processed by a single designated server, often resulting in bottlenecks and increased processing delays under high load. The TRsrv model distributes requests cyclically across all available servers, offering better load distribution than Ssrvr but failing to account for real-time server performance or capacity. In contrast, the TGsrv approach dynamically assigns processing tasks based on the relative load and performance metrics of each server, including response time and historical workload data. This method enables more efficient utilization of resources and significantly reduces the risk of overloading any single server. Experimental results demonstrate that TGsrv consistently outperforms both Ssrvr and TRsrv in terms of task completion time, system throughput, and scalability, especially in scenarios with varying request sizes and server capacities. System configuration is as follows:

- Packet Size = 1024 Bytes
- Number of Packets = 500 (representing split tasks or chunks of image data)
- Image files = 512 KB each (split across packets)
- Time measured = total for upload, processing, response
- Throughput: Total number of bytes processed per second (higher is better).
- Latency: Time between request and response initiation (lower is better).
- Average response time: Time from upload to final result delivery.
- Transfer volume: Total image data processed (500 packets $\times$1024 bytes=512 KB$\approx$4 Mbits).

- Bandwidth usage: Indicates how efficiently the network is utilized per second.

Table I shows the efficiency of the RLB system due to its superior performance across a number of evaluation characteristics. RLB outperformed all other algorithms tested, demonstrating its ability to process large volumes of image text requests in real time with little loss of data.

TABLE I. THROUGHPUT, PDR, AND AVERAGE LATENCY

Algorithms	Throughput (Bytes/sec)	PDR (%)	Avg Latency (ms)
Without load balancing	42,000	52%	1450
RR	74,000	71%	900
Classic weighted RR	88,000	89%	720
Interleaved weighted RR	90,000	91%	700
Random static	66,000	67%	1000
Least Connections	88,000	89%	720
Weighted Least Connections	97,000	96%	630
Sticky RR	76,000	74%	850
Geographic Geo-1	50,000	55%	1300
Geographic Geo-2	70,000	70%	950
RLB	101,000	97%	600

Table II shows that RLB handled traffic most well under concurrent request conditions, as seen by its lowest average response time (2.0 s) and maximum bandwidth utilization (1720 Kbps). Conventional methods, such as Interleaved WRR and Weighted Least Connections, did well but fell short of RLB by a small margin. The response time was much slower (6.4 s) without load balancing, underscoring the significance of intelligent distribution techniques. The dynamic, feedback-driven request distribution technique of the proposed RLB is largely responsible for its outstanding performance in achieving the highest bandwidth utilization and the lowest average response time. In contrast to static or RR-based approaches, RLB adjusts task assignments based on real-time server load monitoring, offering faster response times and more effective use of available bandwidth by preventing idle servers from being underutilized and minimizing overloading on active servers.

TABLE II. AVERAGE RESPONSE TIME, TRANSFER VOLUME, AND BANDWIDTH USAGE

Algorithms	Avg resp. time (s)	Transfer volume (Mbits)	Bandwidth usage (Kbps)
Without load balancing	6.4	4.0	625
Round Robin	3.5	4.0	1150
Classic weighted Round Robin	2.9	4.0	1520
Interleaved weighted Round Robin	2.7	4.0	1570
Random static	4.1	4.0	1025
Least Connections	2.9	4.0	1520
Weighted Least Connections	2.1	4.0	1680
Sticky Round Robin	3.4	4.0	1190
Geographic Geo-1	5.1	4.0	730
Geographic Geo-2	3.6	4.0	1100
RLB	2.0	4.0	1720

As shown in Tables III and IV, the context-aware design of the proposed RLB is responsible for its remarkable resource efficiency, which is demonstrated by its lowest CPU usage (50%) and main memory usage (52%). RLB works by continuously assessing each server's load and allocating resources to reduce unnecessary calculations and idle wait periods. RLB makes it possible for workloads to be distributed more evenly, reducing processing overhead per server by preventing any node from becoming a bottleneck. Furthermore, RLB had the lowest makespan (10 s) and the biggest gain of time (8 s) in terms of processing efficiency. This speed results from its capacity to give server responsiveness and availability top priority, enabling tasks to be finished more quickly without the delays that queuing or rebalancing frequently introduces in less adaptable systems. All these characteristics show that RLB is resource-conscious and performance-efficient, which makes it ideal for scaled, real-time settings. Out of all the algorithms compared, RLB exhibits the shortest waiting time. Tasks begin processing nearly instantly after delivery when waiting times are shorter, which minimizes delays and boosts responsiveness. Besides, the amount of processing time saved in comparison to a baseline (in this case, without load balancing) is indicated by the gain of time metric. By allocating the workload as efficiently as possible, RLB attains the most time gain, completing tasks much more quickly.

TABLE III. RAM, MAIN MEMORY, AND CPU USAGE

Algorithm	RAM Usage (MB)	Main Memory Usage (%)	CPU Usage (%)
Without load balancing	1800	72%	89%
Round Robin (RR)	2200	65%	76%
Classic Weighted RR	2300	59%	63%
Interleaved Weighted RR	2250	56%	61%
Random static	2400	69%	82%
Least Connections	2300	59%	62%
Weighted Least Connections	2150	54%	53%
Sticky Round Robin	2350	61%	67%
Geographic Geo-1	2500	71%	85%
Geographic Geo-2	2450	66%	75%
RLB	2100	52%	50%

TABLE IV. TOTAL PROCESSING TIME (MAKESPAN), LOSS OF TIME IN WAITING AND GAIN OF TIME

Algorithm	Total processing time (Makespan) (s)	Loss of time in waiting (s)	Gain of time (s)
Without load balancing	18	7	0
Round Robin	15	6	3
Classic weighted Round Robin	13.5	3.5	4.5
Interleaved weighted Round Robin	13	3	5
Random static	16	5	2
Least Connections	13.5	3.5	4.5
Weighted Least Connections	11.5	1.5	6.5
Sticky Round Robin	14.5	4.5	3.5
Geographic Geo-1	17	6	1
Geographic Geo-2	14	4	4
RLB	10	1	8

So, compared to others, RLB minimizes bottlenecks and prevents any server from being overloaded by dynamically adjusting to server capacity and current loads. This equitable allocation improves the system's capacity to perform tasks more quickly and reduce waiting times. Other algorithms that allocate jobs without taking into account the server's capacity or load in real time, particularly static or simpler scheduling methods, have longer waiting times. Time is saved because of RLB's awareness and real-time feedback, which reduces queue buildup. In addition, low waiting time and high time gain together show that RLB uses available computational resources more efficiently, maximizing throughput by avoiding wasted time on overloaded or idle servers.

V. SYSTEM COMPARISONS

Several additional evaluation metrics can be used to highlight the contribution of the proposed RLB (TGsrv) system, highlighting improvements in efficiency, scalability, reliability, and system behavior under load. Table V shows a comparison of the best results from the proposed and other load balancing algorithms. With a 97% PDR and 100% throughput, RLB once again took the lead, surpassing cutting-edge tactics, such as DEELB, DEERA, and IQSLB, as shown in Table VI. These findings demonstrate that RLB is a better option for dynamic, distributed, and failure-prone computing systems. RLB achieves the highest throughput, which effectively makes use of the server's resources to handle the greatest number of requests. On the other hand, conventional techniques such as Weighted Least Connections and RR fall behind, demonstrating inefficient use of resources. RLB processes more concurrent requests per minute, illustrating how well it scales in response to demand and making it ideal for dynamic settings with varying workloads. RLB adapts dynamically to the real capacity of each server, optimizing performance and avoiding bottlenecks by allocating more load to stronger servers and less to weaker ones. In addition, the proposed RLB can automatically redistribute the load among servers when a server goes down. This fault-tolerance feature protects against performance deterioration and ensures constant service availability. Static methods are inefficient because they arbitrarily assign requests without taking into account the state of the server. Compared to simpler static methods, the design of the proposed RLB allows distributed, failure-prone, and real-world systems to be managed more effectively.

TABLE V. COMPARISONS OF TOP ALGORITHMS

Metric	RR	RLB	Weighted Least Connections	Interleaved Weighted RR
Throughput (%)	44.21%	100%	94.7%	92.6%
Scalability (Max requests/Min)	16	24	23	22
Adaptability (to server capacity)	No	Yes	No	No
Fault Tolerance (server down)	No	Yes (Rellocation)	No	No
Load Awareness / Feedback-based Adjust	No	Yes	No	No

TABLE VI. COMPARISON WITH RELATED WORKS

Study	Method	Number of tasks	PDR	Max throughput (%)
[16]	Energy-Efficient Opportunistic (EEFO)	50	/	81.27%
	Dynamic Energy-Efficient Resource Allocation (DEERA)			89.2 %
	Efficient Load-Balancing Security (ELBS)			74.21%
	Delay Energy Balanced Task Scheduling (DEBTS)			85.3 %
	Dynamic Energy-Efficient Load Balancing (DEELB)			93.65%
[17]	Improved Queue-based Scheduling with Load Balancing (IQSLB)	50	56%	60%
	Dynamic and Resource-Aware Load Balanced Approach (DRALBA)		68%	72%
	Dynamic Scheduling Strategy		85%	89%
[18]	RR and Least Connections	50	/	81.11%
Proposed RLB	Weighted Least Connection	50	96%	94.7%
	Relative Load Balancing		97%	100%

VI. CONCLUSIONS

This study presented a novel approach to load balancing in multi-server environments through the introduction of an RLB method. The proposed system dynamically distributes client requests between server groups based on relative processing capabilities, demonstrating significant improvements over traditional static and dynamic load balancing techniques. By implementing a proportional distribution of request processing and continuous performance evaluation, the RLB method optimizes resource utilization while minimizing client wait times. A practical implementation focused on text image processing, providing both a valuable application case and a testbed for validating the effectiveness of the proposed load balancing approach. This study contributes valuable insights to the field of distributed computing and provides a practical framework for implementing more efficient load balancing strategies in enterprise systems. Future work could extend the approach to more complex application scenarios by adding machine or deep learning enhancements and investigating additional parameters to optimize request distribution.

REFERENCES

[1] T. Akhtar, N. G. Haider, and S. M. Khan, "A Comparative Study of the Application of Glowworm Swarm Optimization Algorithm with other Nature-Inspired Algorithms in the Network Load Balancing Problem," *Engineering, Technology & Applied Science Research*, vol. 12, no. 4, pp. 8777–8784, Aug. 2022, https://doi.org/10.48084/etasr.4999.

[2] G. Goel and A. K. Chaturvedi, "Multi-Objective Load-balancing Strategy for Fog-driven Patient-Centric Smart Healthcare System in a Smart City," *Engineering, Technology & Applied Science Research*, vol. 14, no. 4, pp. 16011–16019, Aug. 2024, https://doi.org/10.48084/etasr.7749.

[3] E. Suganthi and F. Kurus Malai Selvi, "Weight factor and priority-based virtual machine load balancing model for cloud computing," *International Journal of Information Technology*, vol. 16, no. 8, pp. 5271–5276, Dec. 2024, https://doi.org/10.1007/s41870-024-02119-y.

[4] S. Jadon, P. K. Kannan, U. Kalaria, K. R. Varsha, K. Gupta, and P. B. Honnavalli, "A Comprehensive Study of Load Balancing Approaches in Real-Time Multi-Core Systems for Mixed Real-Time Tasks," *IEEE*

Access, vol. 12, pp. 53373–53395, 2024, https://doi.org/10.1109/ACCESS.2024.3388291.

[5] S. Pramanik, "Central Load Balancing Policy Over Virtual Machines on Cloud:," in *Advances in Marketing, Customer Relationship Management, and E-Services*, A. J. Nair, S. Manohar, A. Mittal, and W. Ahmed, Eds. IGI Global, 2024, pp. 96–126.

[6] A. Kaur and A. Garg, "A review on load balancing algorithms in cloud computing," *International Journal for Modern Trends in Science and Technology*, vol. 7, no. 07, pp. 25–30, Feb. 2022, https://doi.org/10.46501/IJMTST050328.

[7] A. K. K. Baniya, S. S. Pant, D. Paudel, A. Gupta, S. Singh, and H. Mohapatra, "Load Balancing in Cloud Computing Ensuring Fault Tolerance, High Availability, and Security," in *Risk-Based Approach to Secure Cloud Migration*, Minakshi and T. Kumar, Eds. IGI Global, 2025, pp. 253–284.

[8] P. Ajay, A. Sharma, D. V. Gowda, A. Sharma, S. Kumaraswamy, and M. R. Arun, "Priority Queueing Model-Based IoT Middleware for Load Balancing," in *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*, Madurai, India, May 2022, pp. 425–430, https://doi.org/10.1109/ICICCS53718.2022.9788218.

[9] C. Dong, X. Xu, A. Liu, and X. Liang, "Load balancing routing algorithm based on extended link states in LEO constellation network," *China Communications*, vol. 19, no. 2, pp. 247–260, Feb. 2022, https://doi.org/10.23919/JCC.2022.02.020.

[10] M. Junaid, A. Sohail, A. Ahmed, A. Baz, I. A. Khan, and H. Alhakami, "A Hybrid Model for Load Balancing in Cloud Using File Type Formatting," *IEEE Access*, vol. 8, pp. 118135–118155, 2020, https://doi.org/10.1109/ACCESS.2020.3003825.

[11] S. K. Mishra, B. Sahoo, and P. P. Parida, "Load balancing in cloud computing: A big picture," *Journal of King Saud University - Computer and Information Sciences*, vol. 32, no. 2, pp. 149–158, Feb. 2020, https://doi.org/10.1016/j.jksuci.2018.01.003.

[12] M. Kaur and R. Aron, "An Energy-Efficient Load Balancing Approach for Scientific Workflows in Fog Computing," *Wireless Personal Communications*, vol. 125, no. 4, pp. 3549–3573, Aug. 2022, https://doi.org/10.1007/s11277-022-09724-9.

[13] U. K. Jena, P. K. Das, and M. R. Kabat, "Hybridization of meta-heuristic algorithm for load balancing in cloud computing environment," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2332–2342, Jun. 2022, https://doi.org/10.1016/j.jksuci.2020.01.012.

[14] Z. Zhou, F. Li, H. Zhu, H. Xie, J. H. Abawajy, and M. U. Chowdhury, "An improved genetic algorithm using greedy strategy toward task scheduling optimization in cloud environments," *Neural Computing and Applications*, vol. 32, no. 6, pp. 1531–1541, Mar. 2020, https://doi.org/10.1007/s00521-019-04119-7.

[15] M. K. Hussein and M. H. Mousa, "Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020, https://doi.org/10.1109/ACCESS.2020.2975741.

[16] M. Shuaib *et al.*, "An Optimized, Dynamic, and Efficient Load-Balancing Framework for Resource Management in the Internet of Things (IoT) Environment," *Electronics*, vol. 12, no. 5, Feb. 2023, Art. no. 1104, https://doi.org/10.3390/electronics12051104.

[17] N. Albalawi, "Dynamic Scheduling Strategies for Load Balancing in Parallel and Distributed Systems." In Review, Sep. 25, 2024, https://doi.org/10.21203/rs.3.rs-4916145/v1.

[18] S. Agarwal, "Optimized Load Balancing Using Adaptive Algorithm In Cloud Computing With Round Robin Technique," *Educational Administration: Theory and Practice*, pp. 1328–1335, Feb. 2024, https://doi.org/10.53555/kuey.v30i2.6847.