International Journal of Applied Mathematics

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

DESIGN AND IMPLEMENTATION OF A HYBRID LOAD BALANCING ALGORITHM FOR OPTIMIZED SERVER RESPONSE AND RESOURCE UTILIZATION

Minal Shahakar, Dr. S. A. Mahajan, Dr. Lalit Patil, Dr. Rupesh Mahajan

Research Scholar, Smt. Kashibai Navale College of Engineering,

Assistant Professor, Pimpri Chinchwad College of Engineering (PCCOE), Pune, India, Savitribai Phule Pune University, India. mhjn.minal@gmail.com

Associate Professor, Department of Information Technology, PVG'S College Of Engineering, Technology And Management & GK Pate (Wani) IOM, Savitribai Phule Pune University, India. sa_mahajan@yahoo.com

Professor, Department of Information Technology, Smt. Kashibai Navale College of Engineering, Savitribai Phule Pune University, India. lalitvpatil@gmail.com

Associate Professor, Department of Computer Engineering, Dr. D. Y. Patil Institute of Technology, Pimpri, Pune, Savitribai Phule Pune University, Maharashtra, India. mhjn.rpsh@gmail.com

Abstract

Distributed servers need good load balancing since changing client requests and resources can cause response times to vary and performance to degrade. Round-Robin methods have been used for a long time, however they don't account for server workload changes, and static approaches may overload servers and waste resources. This work creates and uses a load-balancing system that combines Round-Robin, Least Connections, and Weighted Response Time to solve these issues. The purpose is to reduce reaction time, allow workload changes, and improve server efficiency. The proposed method includes creating a hybrid load balancer that uses a fitness-based selection mechanism based on natural selection to keep an eye on backend servers and do regular health checks. This system checks servers on the fly based on current connections and average response times. This makes sure that requests are sent to the fastest server. We tested how clients and servers interact in a controlled setting with three servers and two clients. GUI-based monitoring was used to see the state at all times. The hybrid approach does better than traditional Round-Robin techniques by lowering error rates, increasing throughput, and shortening the average response time. The system can also grow as needed and handle errors, as broken servers are quickly taken out of the pool to keep service running smoothly. These results show that hybrid load balancing is a good way to improve distributed server systems. This makes it a strong and flexible choice for real-time applications that need reliable and consistent performance.

Received: July 18, 2025

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Keywords: Hybrid Load Balancing, Round-Robin, Server Response Time, Resource Utilization, Fault Tolerance, Distributed Systems

1. Introduction

In today's distributed computing settings, how well incoming requests are handled across available resources is a big part of how efficiently servers work. Load balancing is turning into increasingly more essential as the need for excessive-overall performance apps like actual-time statistics processing, cloud services, and organisation-scale internet platforms grows. As a smart site visitor’s manager, a load balancer spreads client requests throughout more than one server in order that no one server receives too busy even as others take a seat idle. This now not only makes the machine run quicker, but it additionally makes it simpler to scale, handle errors, and use [1]. Load balancing that works properly lowers the risk of server crashes, gets rid of bottlenecks, and makes the satisfactory use of system resources like CPU, reminiscence, and storage. Therefore, a lot of humans think it's an important thanks to preserve dispensed structures solid and reliable even if they're converting. The standard round-Robin load balancing method has some problems, although it is famous as it is straightforward to apply. While requests are sent out so as without thinking of how the server is responding in actual time, the workload is frequently now not spread out frivolously. round-Robin may by chance overwork one server at the same time as underusing some other if servers have one of a kind quantities of area or if purchaser requests have exceptional processing needs [2], [3]. In the same way, static allocation methods don't take into account how workloads change or how different types of systems function together, which means they waste resources. These bugs are especially bad for real-time systems that need to be able to handle a lot of data quickly and respond quickly. For instance, if one server is already occupied with hard work while another is almost idle, Round-Robin might keep delivering requests to the server that is nearest to the user. This would make the average response time longer and the system less trustworthy [4]. These static solutions aren't flexible enough, so we need smarter and more adaptable options that can function with different server and workload scenarios. Figure 1 shows how a standard load balancing system works. In this system, client requests are sent in order to a pool of servers using either static or round-robin allocation. This method doesn't do health checks, so sites that aren't responding or are too busy may still get traffic, which could make request handling less efficient or fail. Even though this type of static distribution is easy to set up, it is not flexible and doesn't take into account server health or capacity, which makes it harder to grow and less reliable overall.

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

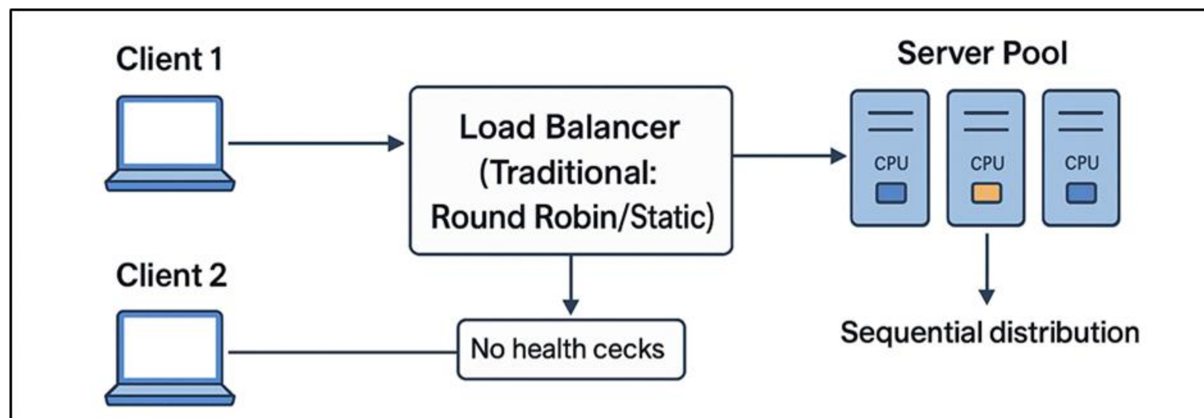

Figure 1: Overview of traditional architecture for load balancing

The main research problem this study tries to solve is how standard load balancing methods lead to inconsistent reaction times and wasteful use of resources. Response time is very important in distributed systems because it affects how the user interacts with the system as a whole. Delays can hurt performance or even cause mission-critical applications to stop working. Usually, old methods that don't change with the server's real-time conditions can't meet these needs [5]. This leads to slower response times, more errors, and uneven server loads. It also makes the system less scalable and raises running costs when resources aren't used efficiently. For companies that are in charge of big systems, this wasteful behaviour means that they waste computer resources, have lower throughput, and can't handle high traffic loads well. So, the research task is to come up with a load-balancing system that is not only fair but also flexible, able to cut down on response time while increasing server efficiency and resilience.

The key contribution of paper is given as:

- ➢ Proposed a hybrid algorithm combining Round-Robin with Least Connections, and Weighted Response Time for optimized load distribution.
- ➢ Introduced adaptive monitoring with a fitness-based selection function to ensure fault tolerance and efficient server choice.
- ➢ Validated performance gains through experiments showing reduced response time, improved throughput over traditional methods.

2. Related Work

Load balancing has been known for a long time to be an important part of distributed computing for making sure that everyone is treated fairly, cutting down on response time, and improving server speed. Because they are simple and easy to use, traditional methods like Round-Robin and static allocation were used a lot in early systems. These methods send requests to servers in a random order, without taking into account their current speed or capacity. These methods work well in environments that are uniform, but they don't work well when server resources and client request loads aren't uniform. This causes them to be

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

inefficiently used and reaction times to be inconsistent [6]. To solve these problems, academics have come up with adaptive models that combine distribution algorithms with real-time server monitoring. These flexible strategies are made to shift quickly to changes in workload, which boosts throughput and lowers latency [7].

The Round-Robin method is widely used, but it has some major flaws. It assumes that all computers have the same amount of processing power and resources available. In reality, though, distributed systems are usually made up of different computers with different amounts of CPU, memory, and bandwidth. So, Round-Robin might accidentally put too much demand on slower servers while leaving strong servers idle, which would slow down the system [8]. Also, static methods can't find servers that aren't working or are in a bad state, so they can't be used in fault-tolerant settings where high availability is necessary [9]. This keeps other servers from getting too busy while one gets too busy. This method makes it easier to divide up the work, but it doesn't take reaction time into account, which is a very important factor in applications that can't handle latency [10].

Later, weighted methods were added to make things more flexible. Weighted Round-Robin gives computers weights based on how much work they can do. This makes sure that requests are sent to servers that can handle them. In the same way, Weighted Least Connections takes into account server load by adding up the number of connections and their weights. Compared to simple Round-Robin, these methods make it easier to add more computers that are different from each other. But because they are given fixed weights, they are less useful in places where work loads change quickly [11]. To get around this problem, some experts have added dynamic weight adjustments that are based on watching how much CPU, memory, and response time are used in real time [12]. Such improvements make it more accurate to distribute work, but they make it harder to do calculations because weights have to be recalculated all the time.

People are also interested in the idea of response-time-based load sharing. Client requests are sent to servers with the fastest response times in this approach, which directly improves the user experience. Response-time-based methods change based on the server and network conditions, making sure that servers with faster answers get more requests. That being said, this can cause an imbalance if faster servers regularly get more traffic and end up being overloaded [13]. To fix this, hybrid methods have been looked into that mix several techniques, like Round-Robin with Least Connections or response-time-based selection. These blends take the best parts of each method and make the worst parts of each method less noticeable. Hybrid algorithms, for instance, sort computers by health and availability first, then use Round-Robin or Least Connections. This makes sure that the process is both reliable and fair [14].

Recently, researchers have additionally regarded into how load balancing frameworks could have health test strategies. Fitness assessments make the device greater fault-tolerant and available through checking servers on an ordinary foundation. This stops requests from being

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

despatched to servers that are not working or are too busy. This proactive tracking improves the revel in of users by way of lowering downtime and making sure that speed remains the identical. Load balancing is likewise greater dependable in large, allotted environments while fitness monitoring and hybrid algorithms are used together [15]. One way to dynamically rank servers for fine request allocation is to use fitness-based functions that look at such things as energetic connections, reaction time, and resource availability [16]. Virtualized systems want algorithms that can adapt to changing workloads, work with a difference of assets, and hold charges low. In these conditions, hybrid fashions that mix widespread algorithms with metrics based totally on performance have worked nicely. Research show that compared to static strategies, hybrid load balancing no longer solely cuts down on response times but also boosts throughput and lowers blunders costs [17]. Additionally, progress in area and fog computing shows the need for load balance structures that are each light and bendy. These systems want if you want to take care of customers in one of kind places quickly whilst preserving latency to a minimum. This requires algorithms which can be both efficient and easy to code [18].

The growth of load balancing study shows the rising need for algorithms that can adapt, handle errors, and consider performance. Traditional Round-Robin and static strategies worked well for basic problems, but they weren't enough for settings that were changing and being different. Improvements like the Weighted and Least Connections methods made things more fair and scalable, but they weren't able to adapt to tasks that changed quickly. Response-time methods reduced latency but could put too much stress on faster servers. With the help of health checks and dynamic fitness functions, hybrid models have become good ways to balance reaction time, fault tolerance, and resource use. Together, the results of earlier studies show that hybrid load balancing is an important way to improve server reaction times and make better use of resources in modern distributed computing systems.

Table 1: Summary of Load Balancing Approaches in Distributed Systems

Author / Study	Approach Used	Fairness in Distribution	Response Time Handling	Fault Tolerance	Adaptability to Workload	Key Findings / Limitations
[6] Early Systems	Round-Robin, Static	Equal but blind allocation	Ignores server speed	None	Low	Efficient in uniform systems, poor in heterogeneous ones
[7] Adaptive Models	Algorithm + Monitoring	Fairer dynamic allocation	Partial improvement	Limited	Moderate	Boosted throughput, lowered latency
[8] RR	Round-	Sequential	Ignores	None	Low	Overloaded

Received: July 18, 2025

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Limitati on Study	Robin	fairness	workload variation			weaker servers, inconsistent response
[9] Fault-Toleranc e Gap	Static Allocation	Equal distributio n	Ignores latency	None	Low	Failed in high-availability systems
[10] LC Algorith m	Least Connectio ns	Balances active load	Ignores latency	Limited	Moderat e	Better than RR but not latency-aware
[11] Weighte d Models	Weighted RR / LC	Weighted fairness	Fixed weights only	Limited	Moderat e	More scalable, but static weights limit adaptability
[12] Dynamic Weights	Weighted + Monitorin g	Adaptive fairness	Considers load partially	Moderat e	High	Improved accuracy, but higher computational cost
[13] Respons e-Time	Response-Time Based	Biased to faster servers	Considers latency directly	None	High	Improves latency but risks overloading fast servers
[14] Hybrid Models	RR + LC / Response	Balanced and flexible	Combines load + latency	Strong (partial)	High	Reliable and fair, mitigates weaknesses of single methods
[15] Health Check	Health Monitorin g	Fair with monitoring	Indirectly improves latency	Strong	High	Avoids failed servers, improves fault tolerance
[16] Fitness-Based	Heuristic / Nature Inspired	Adaptive fairness	Considers response + load	Strong	High	Dynamic optimization, effective but computationally heavy

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

3. System Design

The suggested system architecture, as shown in figure 2 uses a hybrid load balancing model. There are three major parts to the architecture: the load balancer, server processing, and client processing. In the server layer, there are three computers set up. Each has 1 GB of RAM and 1 GB of storage. By subtracting used memory from fixed capacity, the servers automatically figure out what resources are available, while storage access is simulated. When a client sends a request, each server handles it by getting the data, doing the job, and sending back a response. This makes sure that resources are tracked correctly in real time. Two clients in the client layer start activities by reading what users type, sending requests to the load balancer, and getting responses from servers. A graphical user interface (GUI) that shows client-to-server communication makes this contact possible. At the heart of the design is the load balancer, which can work in two ways: the standard Round-Robin mode or the proposed hybrid model. It starts up with live servers and runs in a separate thread to make sure the GUI is responsive. Every 10 seconds, a health check method checks the states of all the servers and adds healthy ones to the active pool while removing unresponsive ones to make sure that the system can handle errors.Round-Robin [17] sends requests in a positive order in the traditional mode. In the hybrid mode, the fantastic server is chosen by means of a health characteristic that appears at both energetic connections and common reaction time. This makes sure that calls go to servers that are not too busy and might answer fast. Metrics like total requests, reaction instances, and mistakes numbers are constantly updated by way of the gadget. This shall we it adapts to converting workloads. Errors are constant via removing damaged computer systems and retaining track of problems, and fitness tests make certain the whole lot is going for walks smoothly. This combined technique [18] ensures scalability, even load sharing, consistent reaction times, and better throughput than static strategies. This graph combines dynamic tracking, aid-conscious allocation, and adaptive decision-making to make a strong solution for disbursed structures that need fault-tolerant load balancing and excessive overall performance.

Received: July 18, 2025

International Journal of Applied Mathematics
Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

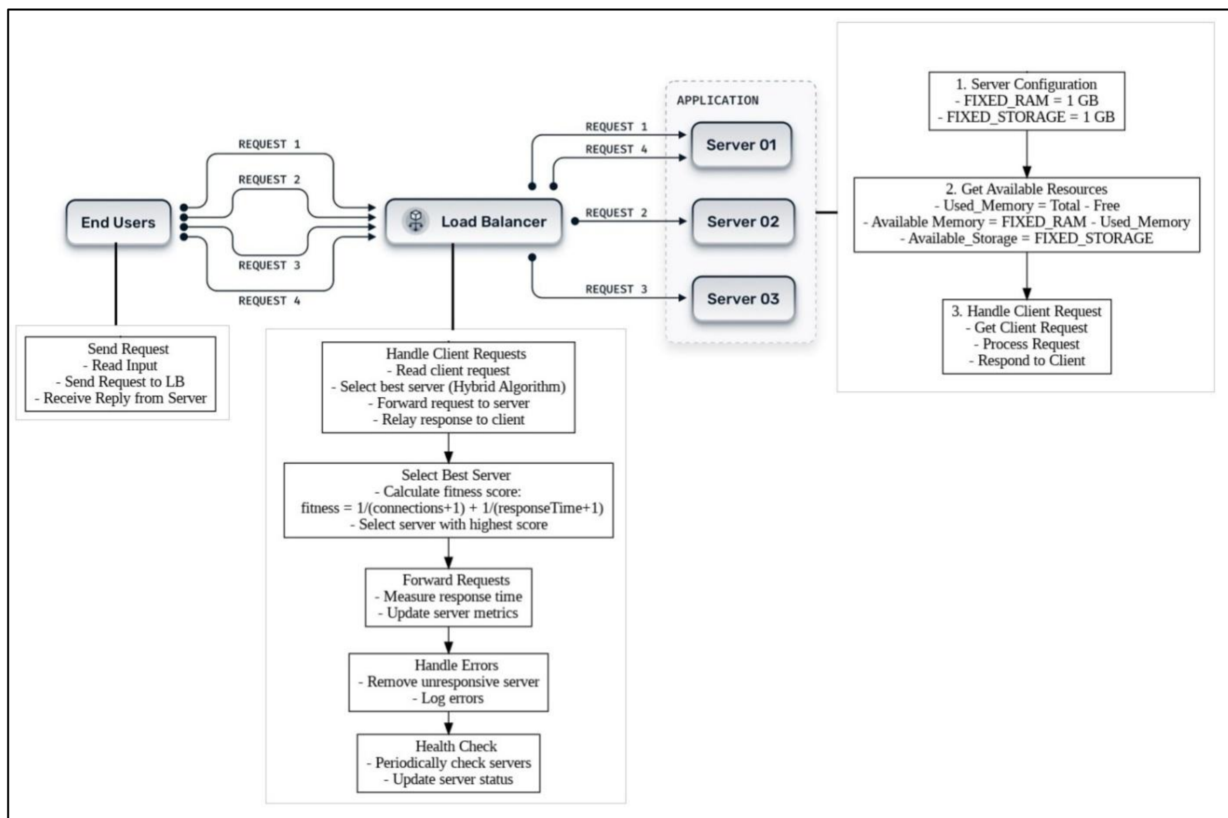

Figure 2. Proposed System Architecture

4. Proposed Methodology

The Hybrid load balancing approach that was offered is made up of three major parts: servers, customers, and the load balancer. It has three computers, and each one is already set up with a certain amount of storage space and memory [19, 20]. It's up to these computers to handle calls from clients, keep an eye on how resources are being used, and act quickly. In the client layer, there are two clients that take input, ask the load balancer for things, and get responses. All of this is possible with the help of a graphical user interface (GUI), which makes it easy to work together. The load manager uses both the old Round-Robin algorithm and the new mixed algorithm to decide what to do and how to send requests. The right backup computers are picked with the help of health checks and a score system based on fitness. This ensures that the distributed system can handle errors, work at a steady level, and make the best use of its resources.

4.1. Server Processing: Resource Monitoring And Request Handling

4.1.1 Server Processing (Number of Servers = 3)

a. Server Configuration

Within the proposed hybrid load balancing design, server processing may be very important for handling resources well and responding fast to purchaser requests. The system is made from three computer systems, and to maintain things constant, each one is installation with

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

constant computing assets. Specifically, each server is given 1 GB of RAM and 1 GB of storage. This units the same old for the way the servers must be set up in order that the burden balancer can well ship requests to all to be had servers. This set-up makes positive that resources are continually to be had, which is important for handling customer workloads in an allotted environment.

b. Get Available Resources

Computers examine how their resources are being utilized all the time, in addition to setting up. This helps them work at their best. The procedure determines the amount of memory in use by subtracting the amount of free memory from the total runtime memory. The system finds out how much memory is free by taking the total amount of RAM and subtracting the amount that is being utilized. In the same way, the fixed capacity is used to track the available storage, which is based on real-world usage cases..

Figure 3. Available Resources Calculation at Server

When a client user sends a request to a server, that server uses it to be had computing electricity to finish the activity and sends lower back the consequences. This cycle of setting up, watching, and handling requests keeps the system running smoothly, gets rid of bottlenecks, and makes sure that each server works within its limits. It also helps to spread the load evenly across the architecture. Figure 3 shows how to calculate computer resources, like how much memory and storage space is available so that they are used efficiently. This process of monitoring helps keep server speed at its best by balancing workloads, preventing over-allocation, and preventing overloading.

c. Handle Client

Client requests are the final and most significant phase of server processing in the suggested system. As soon as the load balancer sends a request, the server starts receiving client requests. This could involve reading input data or computing. After receiving the request, the server processes it using memory and storage, the status of server illustrate in figure 4. This ensures fast completion without exceeding capacity. The server employs RAM for calculations and storage for data management to ensure accuracy and consistency. System and end users can communicate simply using this response processing. The result is lower latency, consistent performance, and reliable service for all customer connections.

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

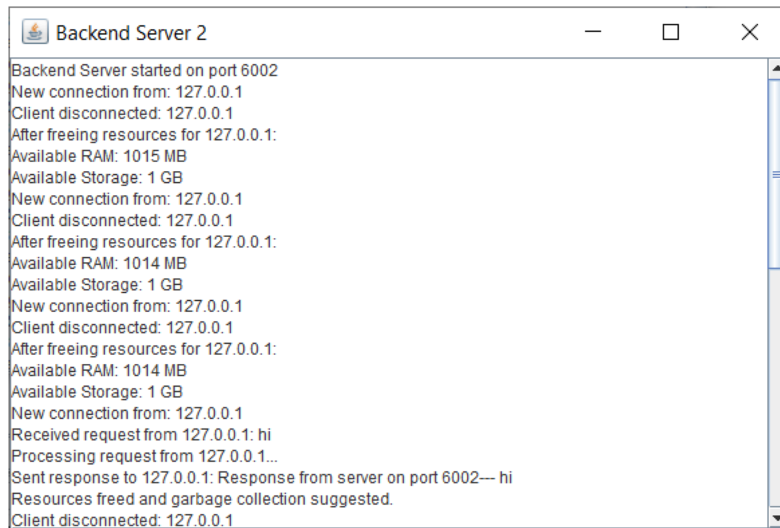

Figure 4. Server Status

4.2. Client processing: request submission and response reception

4.2.1 CLIENT PROCESSING (Number of Client = 2)

a. Send Request

Two clients talk to the servers through the load balancer as part of the client processing. Each client starts by reading the information from the user, which is what the request is based on, the GUI request and response shown in figure 5. The request is then sent to the load manager, which figures out which server is best. Lastly, the client gets the answer from the assigned server. This makes sure that communication goes smoothly and responses are handled quickly.

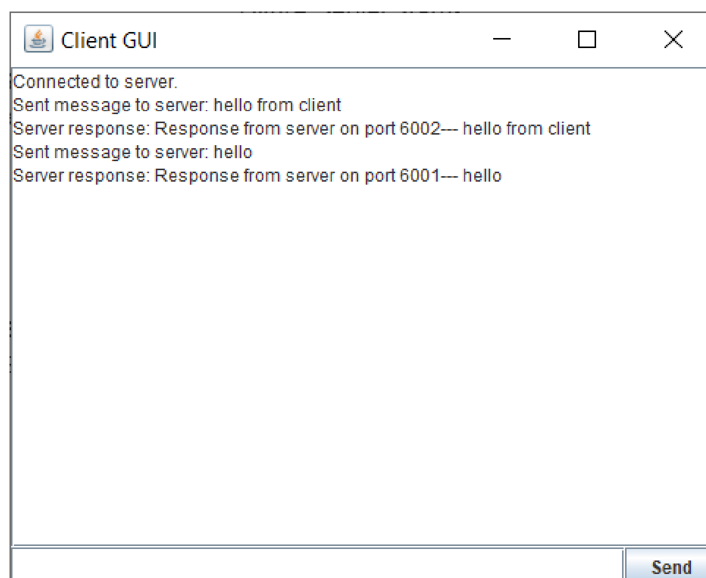

Figure 5. Client Request Response GUI

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

4.3. Load Balancer

4.3.1 ROUND ROBIN

a. GUI Setup

The GUI setup is very important for keeping users involved and keeping an eye on the system. The load balancer is set up to run in a different thread so that everything runs smoothly. This way, the graphical user interface doesn't get stuck while requests are being handled. This lets you watch the system in real time, including client requests, server replies, and error logs, without having to stop what you're doing on the interface. Running the load balancer on its own makes sure that the GUI stays responsive and always shows users or administrators the correct system state.

b. Load Balancer Initialization:
Load balancer utilization begins with initialization [21], [22]. This turns on the load balancer and prepares the environment for client calls. It first activates the load balancer to accept links and send requests to the proper areas. After that, the system gets a list of all current servers and adds only healthy, responsive ones. This eliminates non-responsive servers from the set, preventing failures. A dynamic list of servers is maintained during initialization. This ensures that the load balancer can handle faults and efficiently distribute requests using Round-Robin or the proposed hybrid selection approach.
c. Health Check Mechanism:

Figure 6 shows how the health check mechanism monitors backend servers every 10 seconds. Reachable and responsive servers are instantly added to the active Servers set to ensure request allocation. Unresponsive servers are removed from the active pool to prevent request failures. Since the hybrid load balancer distributes load to healthy servers, this dynamic process ensures fault tolerance, system dependability, and ongoing service delivery.

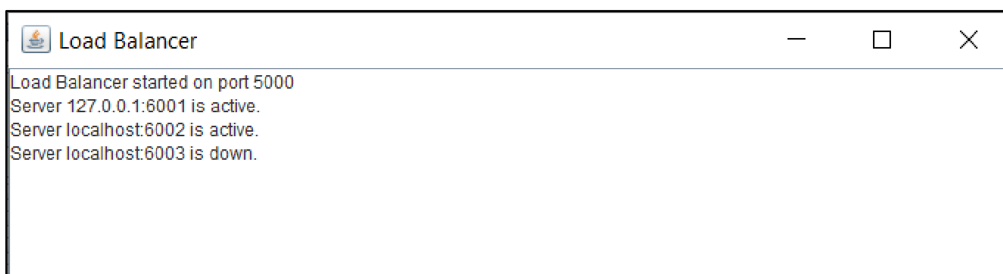

Figure 6. Health Checking of Servers

d. Client Request Handling:
To ensure concurrency and responsiveness, the load balancer takes a new client connection and immediately creates a thread to process it. The client's request is read and intelligently passed to a backend server utilising Round-Robin or Hybrid Round-Robin–Least Connection. The selected server efficiently processes the request and sends a response to the client, ensuring seamless communication and minimal latency [23]. Figure 7 shows the hybrid

Received: July 18, 2025

International Journal of Applied Mathematics

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

architecture's load balancer request processing mechanism. As the primary control point, the load balancer immediately accepts client connections. The load balancer creates a thread for each client request to prevent bottlenecks and assure concurrency. Multiple clients can be served without compromising system performance or blocking other requests with this architecture.

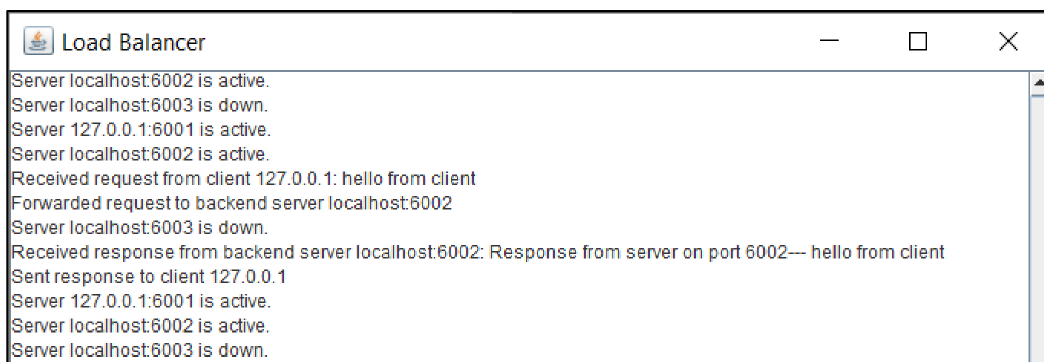

Figure 7. Load Balancer Request Handling

e. Round-Robin Selection:

Distributing client requests across servers relies on round-robin selection. The getActiveBackendServer() method cycles across the backend servers. This distributes requests evenly among active servers, preventing one from overloading while others idle. This method is efficient for sequential request allocation in homogeneous contexts with similar server capacity due to its simplicity. Figure 8 shows how the load balancer sends multiple client requests to servers at the same time. This makes sure that everyone gets a fair share of the work, that it is processed quickly, that response times are kept to a minimum, and that the server pool has an even amount of work to do.

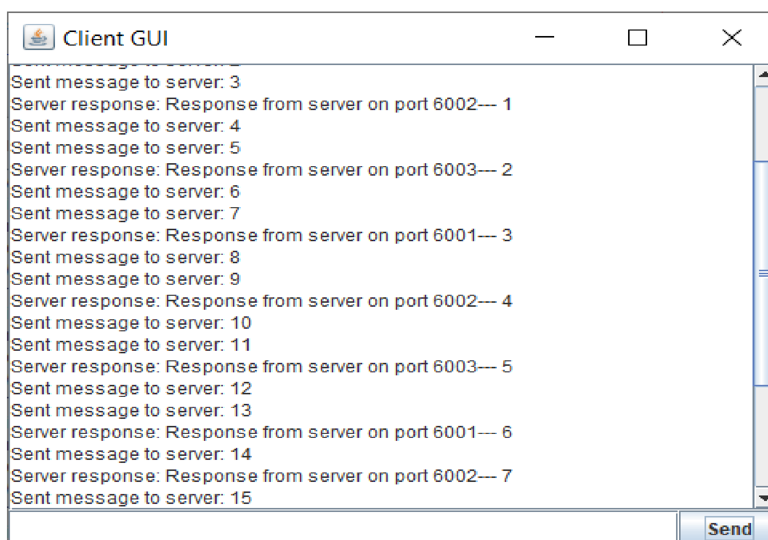

Figure 8. Multiple Request from Client to Server

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The system does, however, have features for failure tolerance. If there are no live servers available at a certain time, the client is told right away, so there are no delays or failed requests. Also, error-handling features are built in to make sure stability. During request processing, if a backend server doesn't reply, it is automatically removed from the activeServers set. This keeps future requests from being sent to a node that isn't responding. The GUI keeps track of errors that happen when clients or servers talk to each other or handle requests. This lets managers see at a glance how the system is running and how healthy the servers are. This mix of cyclic selection, proactive problem detection, and transparent logging makes the load balancing process more reliable while still allocating requests fairly.

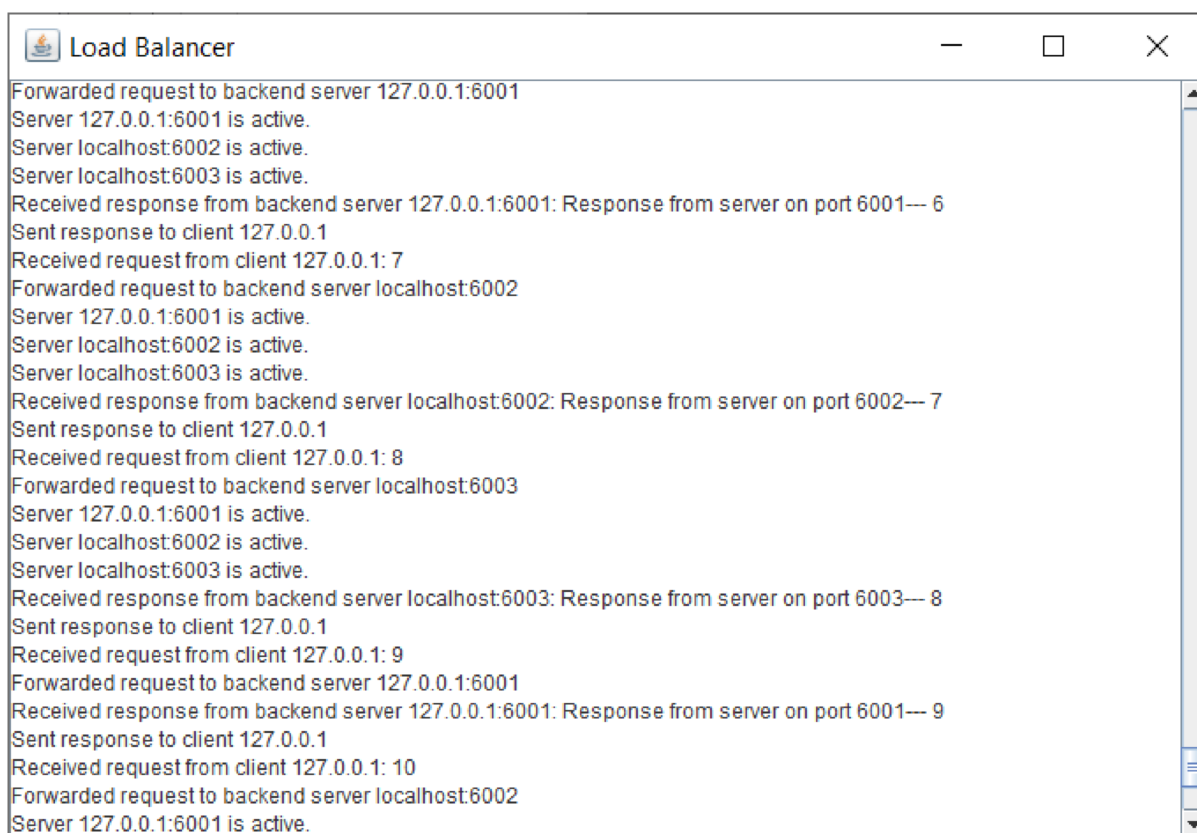

Figure 9. Multiple Request Handling by Load Balancer Using Round Robin Technique

Figure 9 shows how the load balancer handles multiple client requests using the Round-Robin method, sending them to servers in a cycle of linear order. This makes sure that everyone is doing their fair share of the work, stops bottlenecks, and keeps system speed stable.

5. Proposed Algorithm Used

5.1 Hybrid Load Balancing Algorithm

By choosing the best server based on current connections and response time, the hybrid load balancing algorithm makes things run more smoothly. It starts by setting up server metrics, taking orders from clients, and giving each server a fitness score. Round-Robin is used to

Received: July 18, 2025

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

break ties and choose the server with the best score. Requests are taken care of, metrics are updated, and errors are treated politely. This work process, shown in Figure 10 flowchart, makes sure that the system can be scaled up, is reliable, and works at its best.

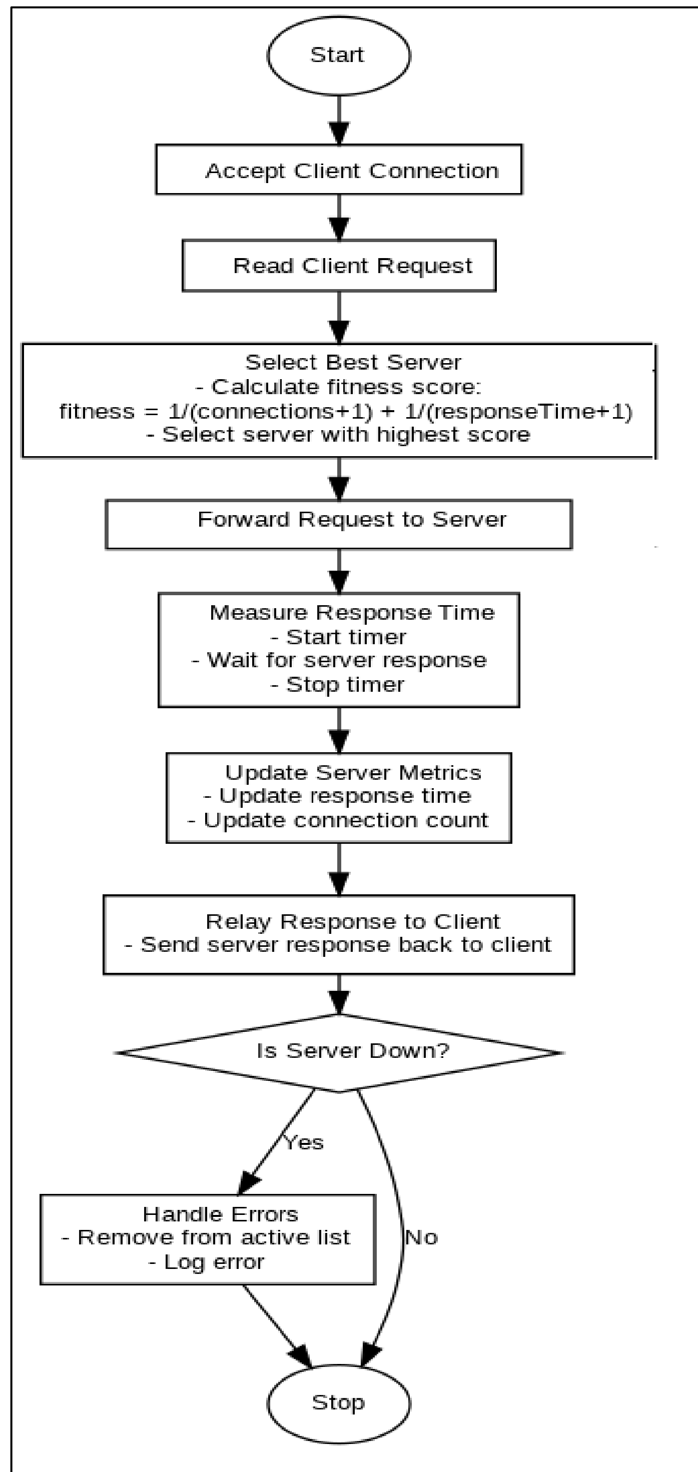

Figure 10: Working Flow of Hybrid Load Balancer Algorithm

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Steps of the Hybrid Load Balancing Algorithm

Step 1: Initialize Server Metrics

- Maintain a list of active backend servers.
- For each server, track:
  - **Active Connections**: The number of active client connections.
  - **ResponseTime (RT)**: The average time it takes to handle requests.

Step 2: Accept Client Connection

- The load balancer listens for incoming client connections.
- When a client connects:
  - Accept the connection.
  - Log the client's address.

Step 3: Read Client Request

- Read the request sent by the client.
- Log the request details.

Step 4: Calculate Fitness Score for Each Server

- For each active server, calculate a **fitness score** using the formula:

$$\text{fitness} = \frac{1.0}{\text{connections} + 1} + \frac{1.0}{\text{responseTime} + 1}$$

  - **Connections**: The number of people who are currently connected to the server.
  - **responseTime**: The average response time of the server.
  - The **+1** ensures that the denominator is never zero.

Step 5: Select the Best Server

- Compare the fitness scores of all active servers.
- Select the server with the **highest fitness score**.
- If multiple servers have the same score, use **Round-Robin** to select the next server in the sequence.

Step 6: Forward Request to the Selected Server

- Open a connection to the selected backend server.
- Forward the client's request to the server.
- Increment the server's active connection count.

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Step 7: Measure Response Time

- Wait for the server to process the request and send a response.
- Stop the timer when the response is received.
- Calculate the response time as:

$$responseTime = endTime - startTime$$

Step 8: Update Server Metrics

- Update the server's metrics:
  - Add the response time to the server's total response time.
  - Increment the server's total request count.
  - Update the server's average response time:

$$averageResponseTime = \frac{totalResponseTime}{totalRequest}$$

Step 9: Relay Response to Client
- Send the backend server's response back to the client.
- Log the response details.

Step 10: Decrement Active Connections
- After processing the request, decrement the server's active connection count.

Step 11: Handle Errors

- If the backend server is down or unresponsive:
  - Remove the server from the active server list.
  - Log the error.
  - Notify the client that the server is unavailable.

Step 12: Repeat for Next Request
- Go back to **Step 2** to handle the next client request.

In the hybrid load balancing approach, this is how we figure out each backup server's health score:

$$\text{fitness} = \frac{1.0}{\text{connections} + 1} + \frac{1.0}{\text{responseTime} + 1}$$

Received: July 18, 2025

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Components of the Formula

- **connections**: The number of active connections currently being handled by the server.
- **responseTime**: The average response time of the server (in milliseconds or another time unit).
- **+1**: Added to avoid division by zero and to ensure the formula works even when connections or responseTime is zero.

The fitness score is designed to:

1. Give services with fewer open links more attention:
   - The term $\frac{1.0}{\text{connections}+1}$ ensures that servers with fewer connections have a higher score.
   - For example:
     - If a server has 0 connections, this term becomes 1.0.
     - If a server has 5 connections, this term becomes 1.0/6 ≈ 0.167.
2. Give computers with faster response times more weight:
   - The term $\frac{1.0}{\text{responseTime}+1}$ ensures that servers with lower response times have a higher score.
   - For example:
     - If a server has a response time of 10 ms, this term becomes 1.0/11 ≈ 0.0909111.0 ≈ 0.0909.
     - If a server has a response time of 100 ms, this term becomes 1.0/101 ≈ 0.00991011.0 ≈ 0.0099.
3. Combine both factors:
   - The fitness score is the sum of the two terms, giving equal weight to both the number of connections and the response time.
   - A higher fitness score indicates a better server for handling new requests.

Table 2: Example Calculation

Server	Active Connections	Response Time (ms)	Fitness Score Calculation	Fitness Score
A	2	20	(1.0/2+1)+(1.0/20+1)	0.333+0.0476=0.38060
B	5	10	(1.0/5+1)+(1.0/10+1.0)	0.1667+0.0909=0.25760

Received: July 18, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

- **Server A** has a higher fitness score (0.3806) compared to **Server B** (0.2576), so the load balancer will prefer Server A.

4. Why Add 1 to Connections and Response Time?

- **Avoid Division by Zero**:
  - If a server has 0 connections or 0 response time, adding 1 ensures the denominator is never zero.
- **Smooth the Curve**:
  - Adding 1 prevents the fitness score from becoming too large or too small when connections or response times are very low.

5.2 Proposed Hybrid Round-Robin with Least Connections

This study works best with the hybrid RR–Least Connections algorithm because it dynamically routes requests based on live connections and response-time metrics, which lowers latency and boosts throughput. Health checks make sure the algorithm can handle errors, per-request threads allow for concurrency, and EWMA updates keep estimates stable. Adjustable weights (α, ²) balance fairness and speed, resulting in scalable, resource-efficient load distribution, which is the main point of our study.

Algorithm: Hybrid Round-Robin with Least Connections

Inputs:

Servers S = {1..m}; health period tau = 10s

State (per server i):

- C_i = active connections
- R_i = avg response time
- TS_i = total response time
- n_i = total requests served
- e_i = error count

Global:

- A = active (healthy) server set
- p = round-robin pointer

1) Initialize

Set GUI; start load balancer thread; $\boldsymbol{A} \leftarrow \emptyset A, Ci \leftarrow 0,\ Ri \leftarrow undefined,\ ni \leftarrow 0,\ TSi \leftarrow 0,\ ei \leftarrow 0\ \forall i.$

Start periodic health check every $\boldsymbol{\tau}\ s.$

2) Health Check (every tau seconds)

$Probe\ each\ server\ i.$

$If\ reachable$:

$add\ i\ to\ A$;

$else$:

Received: July 18, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

$remove\ i\ from\ A\ and\ log.$
3) Accept Client
Listen on port 5000; on connect: log client address; spawn handler thread.
4) Read Request
Parse request; log metadata (size, timestamp).
5) Select Best Server (Hybrid)
For each i in A compute fitness:

$$F_i = alpha * \frac{1}{C_i + 1} + beta * \frac{1}{R_i + 1}$$

Pick s in argmax_iF_i.
If tie: choose by round-robin among tied servers (pointer p).
6) Forward and Measure
$C_s < C_s + 1$; send request at t_send.
When reply arrives at t_done: $T = t_done\ t_send.$
7) Update Metrics & Return

$$TS_s < TS_s + T;\ n_s < n_s + 1;\ R_s < \frac{TS_s}{n_s}$$

$$(EWMA: R_s < (1 - gamma) * R_s + gamma * T).$$

$C_s < C_s\ 1.$
Relay response to client; log.
8) Error Handling
On timeout/failure: $e_s < e_s + 1$; remove s from A; notify client; log.
9) Repeat
Handle next client request.
Per-request complexity: $O(|A|)$to score servers; $O(1)$ to update.

Received: July 18, 2025

International Journal of Applied Mathematics

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Hybrid Round-Robin With Least Connections- Mathematical model Analysis

Goal:

Show that choosing server $s = argmax_i\, F_i$ with

$$F_i = alpha * \frac{1}{C_i + 1} + beta * \frac{1}{R_i + 1} (alpha, beta > 0)$$

Greedily minimizes an upper bound on the next job's completion time, thus reducing delay.

1) Setup

- Active servers: A(t)
- C_i = active connections on server i
- R_i = mean response-time estimate on server i
- Fitness: F_i as above
- Selection: pick s in argmax_iF_i (tie-break via Round-Robin)

2) Key inequality (Titu's Lemma / Engel form of Cauchy–Schwarz)

$For\ x = C_i + 1 > 0, y = R_i + 1 > 0$:

$$\frac{alpha}{x} + \frac{beta}{y} \geq \frac{(alpha + beta)^2}{alpha * x + beta * y}$$

Define $J_i = alpha * (C_i + 1) + beta * (R_i + 1)$.

Then:

$$F_i \geq \frac{(alpha + beta)^2}{J_i}$$

$=> maximizing\ F_i \leq> minimizing\ J_i$ (since numerator constant across servers).

3) Delay upper bound interpretation

Let T_i be completion time of the arriving job if sent to server i.

A standard "frozen arrivals" bound gives:

$T_i \leq k1 * (C_i + 1) + k2 * (R_i + 1)$

Set $alpha = k1, beta = k2$.

Then:

$T_i \leq J_i$

Therefore, choosing s that minimizes J_i (equivalently maximizes F_i) minimizes an explicit per-arrival upper bound on delay.

THEOREM (Greedy delay reduction):

The hybrid dispatch rule s in argmax_iF_i minimizes the bound J_i for the arrival, thus greedily reducing an upper bound on the job's delay. QED.

4) Dominance and fairness

If server i dominates j ($C_i <= C_j and\ R_i <= R_j$, at least one strict), then:

$$F_i \geq F_j and\ J_i \leq J_j$$

=>the rule prefers the strictly better server, ensuring intuitive fairness/efficiency.

5) Stability intuition (negative drift)

Received: July 18, 2025

Potential: $Phi(C, R) = alpha * sum_i \log(C_i + 1) + beta * sum_i \log(R_i + 1)$
Partial derivatives: $\frac{dF_i}{dC_i} = -\frac{alpha}{(C_i + 1)^2} < 0, \frac{dF_i}{dR_i} = -\frac{beta}{(R_i + 1)^2} < 0$
Dispatching to argmaxF_i penalizes overloaded/slow servers, creating negative drift in Phi when the stability condition holds: total arrival rate $lambda < sum_i mu_i$.
=>load equalization, bounded delays in steady state.
6) Worked numeric example (alpha = beta = 1)
$Server\ A: C_A = 2, R_A = 20\ ms$
$F_A = \frac{1}{3} + \frac{1}{21} = 0.3333 + 0.0476 = 0.3809$
$J_A = (2 + 1) + (20 + 1) = 24$
$Server\ B: C_B = 5, R_B = 10\ ms$
$F_B = \frac{1}{6} + \frac{1}{11} = 0.1667 + 0.0909 = 0.2576$
$J_B = (5 + 1) + (10 + 1) = 17$
Here,$F_A > F_B$, so pick A (it minimizes J_i bound only if alpha, beta reflect latency cost).
If latency is more critical, increase beta.
Solve F_A = F_B for beta/alpha:

$$\frac{alpha}{3} + \frac{beta}{21} = \frac{alpha}{6} + \frac{beta}{11}$$

$$=> (alpha) * \left(\frac{1}{3} - \frac{1}{6}\right) = (beta) * \left(\frac{1}{11} - \frac{1}{21}\right)$$

$$=> (alpha) * \left(\frac{1}{6}\right) = (beta) * \left(\frac{10}{231}\right)$$

$$=> \frac{beta}{alpha} = \frac{\left(\frac{1}{6}\right)}{\left(\frac{10}{231}\right)} = \frac{231}{60} \approx 3.85$$

If $\frac{beta}{alpha} > 3.85$, the rule prefers B (much faster) despite higher connections.
7) KPIs and constraints
- Mean response: $E[T] = \left(\frac{1}{N}\right) * sum_r T_r$
- Throughput: $X = \frac{N}{t_{end} - t_{start}}$
- Error rate: $epsilon = \frac{sum_i e_i}{N}$
- Stability per server $\left(\frac{\frac{M}{M}}{1}\right)$: $rho_i = \frac{lambda_i}{mu_i} < 1, E[T_i] = \frac{1}{mu_i - lambda_i}$
- $E[T] approx\ sum_i \left(\frac{lambda_i}{lambda}\right) * E[T_i]$

Figure 11 shows how the client GUI and servers talk to each other. Requests are sent one after the other, and replies are sent back in the same order. Each request has a unique identifier

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

attached to it, and responses make the port number of the server that sent the message very clear. This makes sure that things can be tracked and shows how the hybrid system efficiently sends client requests to multiple live servers so that the load is spread out evenly.

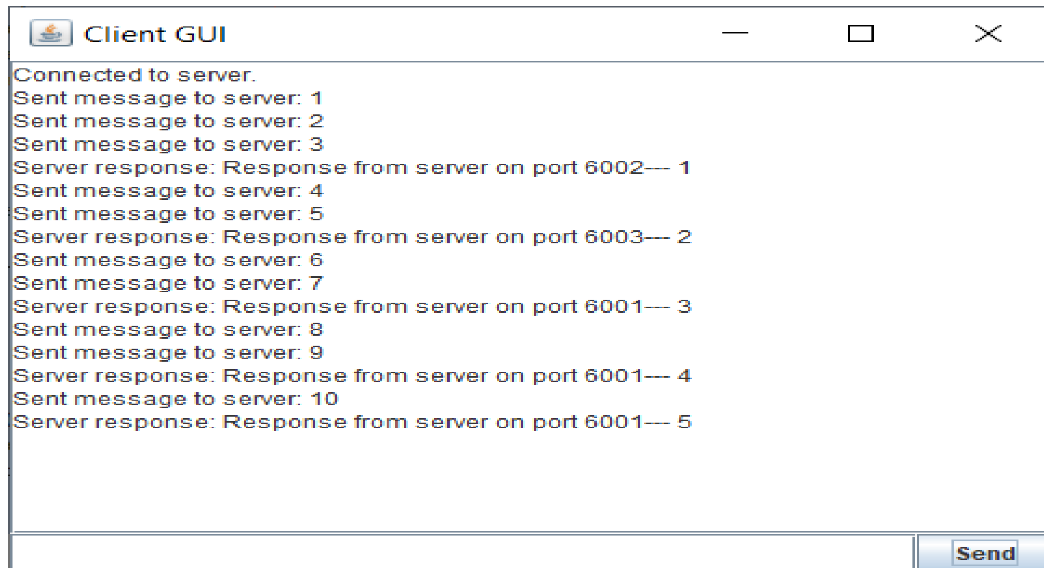

Figure 11. Multiple Requests from Client to Server

In Figure 12, demonstrate how the hybrid load balancer handles client requests across sites. It shows server health checks that confirm they are online and also keeps track of their average response times. It is possible to figure out metrics like throughput and error rate, which show that the balancing works well and there are no mistakes. This shows that the hybrid algorithm can improve speed even when the load changes.

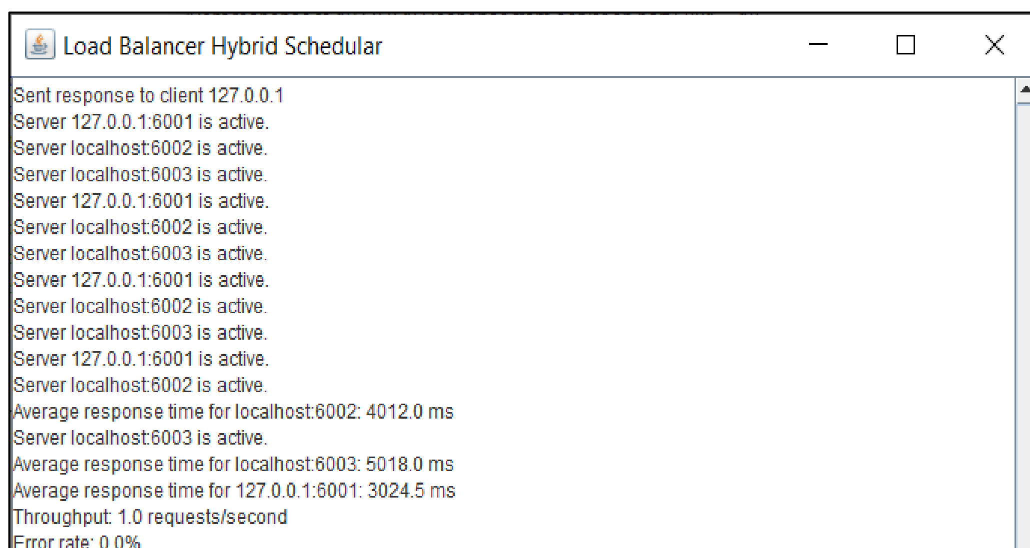

Figure 12. Multiple Requests Handling by Proposed Hybrid Load Balancer

Received: July 18, 2025

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The Round-Robin method sends requests out in order without taking server load into account, which could lead to inefficiency. The Hybrid Model, on the other hand, cleverly combines Round-Robin with Least Connections. It makes sure that everyone gets a fair share by looking at both the order of the connections and the present server workload, as shown in Table 3.

Table 3. Differentiation analysis of Round Robin with Proposed Hybrid Model

Feature	Round-Robin	Hybrid Model (Round-Robin + Least Connections)
Load Distribution	Static, equal distribution	Dynamic, based on server load and performance
Performance	May send requests to overloaded servers	Optimizes performance by selecting the best server
Fault Tolerance	No health checks	Includes health checks and removes unhealthy servers
Scalability	Works best for homogeneous servers	Scales well for heterogeneous servers
Resource Utilization	May underutilize or overload servers	Balances load effectively
Client Experience	Inconsistent performance	Consistent and reliable performance
Flexibility	Rigid and static	Adapts to real-time changes

6. Result and Discussion

6.1 Performance Parameters

To make sure our hybrid load-balancing study was correct, we measured success using three metrics that were in line with the goals of the title. Throughput$\left(X_{sys} = \frac{N}{\Delta t};\ X_i = \frac{N_i}{\Delta t}\right)$measured how much capacity and resources were being used on each computer. The average response time per server (T_i) and its live EWMA estimate R_i[k] showed changes in latency over time as the load changed, showing that the servers were responding more quickly. The error rate $\left(\varepsilon_{sys} = \frac{E}{N};\ \varepsilon_i = \frac{e_i}{n_i}\right)$was used to measure reliability and make sure that fault tolerance was met. All of these tests showed that the whole system was more fast and used its resources more efficiently.

i) Throughput

- System throughput: $X_{sys} = \frac{N}{dt}$
- Per-server throughput: $X_i = \frac{N_i}{dt}$

Wheredt = t_end - t_start, N = total completed requests, N_i= completed on server i.

ii) Average Response Time per Server

- Per-server mean latency: $T_{avg_i} = \left(\frac{1}{n_i}\right) * \ sum_{\{k=1..n_i\}} \left(t_{done_{\{i,k\}}} - t_{send_{\{i,k\}}}\right)$

Volume 38 No. 1s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

- Online EWMA estimate: $R_{i[k]} = (1 - gamma) * R_{i[k-1]} + gamma * T_{i[k]}, 0 < gamma <= 1$
- System means latency: $T_{avg} = \left(\frac{1}{N}\right) * sum_{\{r=1..N\}T_r} = sum_i \left(\frac{N_i}{N}\right) * T_{avg_i}$

iii) Error Rate
- System error rate: $epsilon_{sys} = \frac{E}{N}$
- Per-server error rate: $epsilon_i = \frac{e_i}{n_i}$

Where, E = total failed/timeout requests, e_i = failures on server i.

6.2. Comparative Analysis

Figure 13 shows how the average response time for standard Round-Robin and the proposed Hybrid load balancer compares when 20 to 40 requests are made at the same time. There is a 43.25% drop in time between Round-Robin and Hybrid when 20 requests are made. Round-Robin takes 3894 milliseconds for 40 requests, while Hybrid takes 2004 milliseconds, which is a 48.54% drop. It is worth mentioning that Hybrid is almost 2 times faster than Round-Robin as the load increases (RR/Hybrid ratio = 1.76× at 20 and 1.94× at 40). The Hybrid gets better as the load goes up because its latency drops from 2312 ms to 2004 ms, which is 13.32% better. This shows that fitness-based routing (Least Connections + Response-Time weighting), per-request threading, and continuous metric updates are stopping hotspots more and more as contention rises. Round-Robin slows down a little (4074 ms → 3894 ms, 4.42%), probably because of warm-up effects, but it stays slower because it doesn't take into account how different servers are or how busy they are at any given time, sending new work to nodes that are already busy a lot of the time.

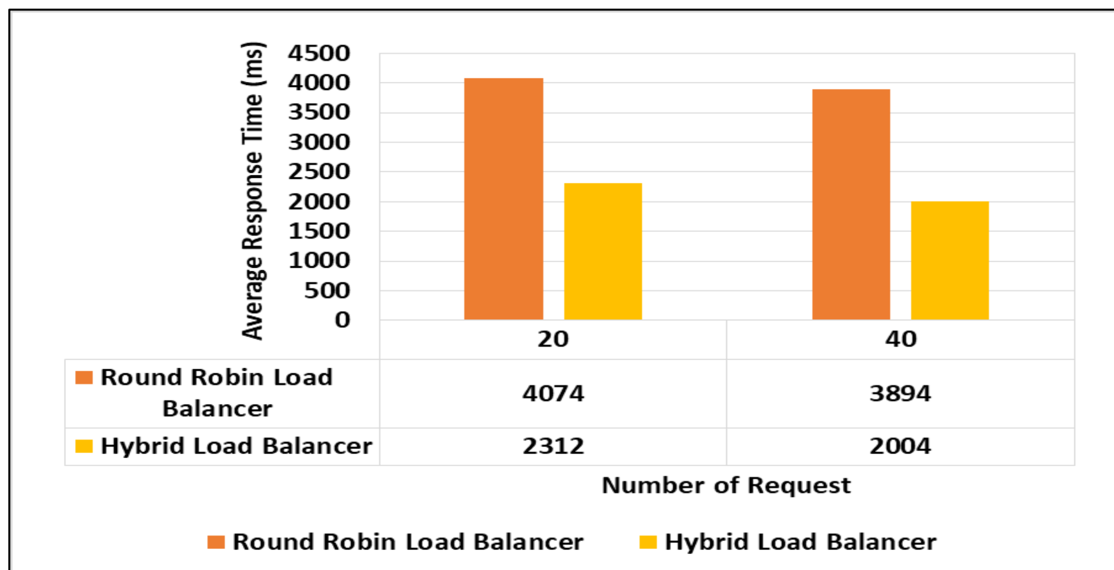

Figure 13: Representation of comparison of average response time

Received: July 18, 2025

International Journal of Applied Mathematics

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

The health check feature makes the hybrid method even better by getting rid of servers that are too slow or don't respond, stabilizing latency tails, and keeping an active set that is free of errors. Overall, the evidence supports the claim that the Hybrid algorithm scales better, with better resource utilization and shorter average response times as the number of users improves. To be full, future studies should include variability bars, heavier and mixed workloads, and give tail metrics (p95/p99) and throughput to back up the end-to-end gains.

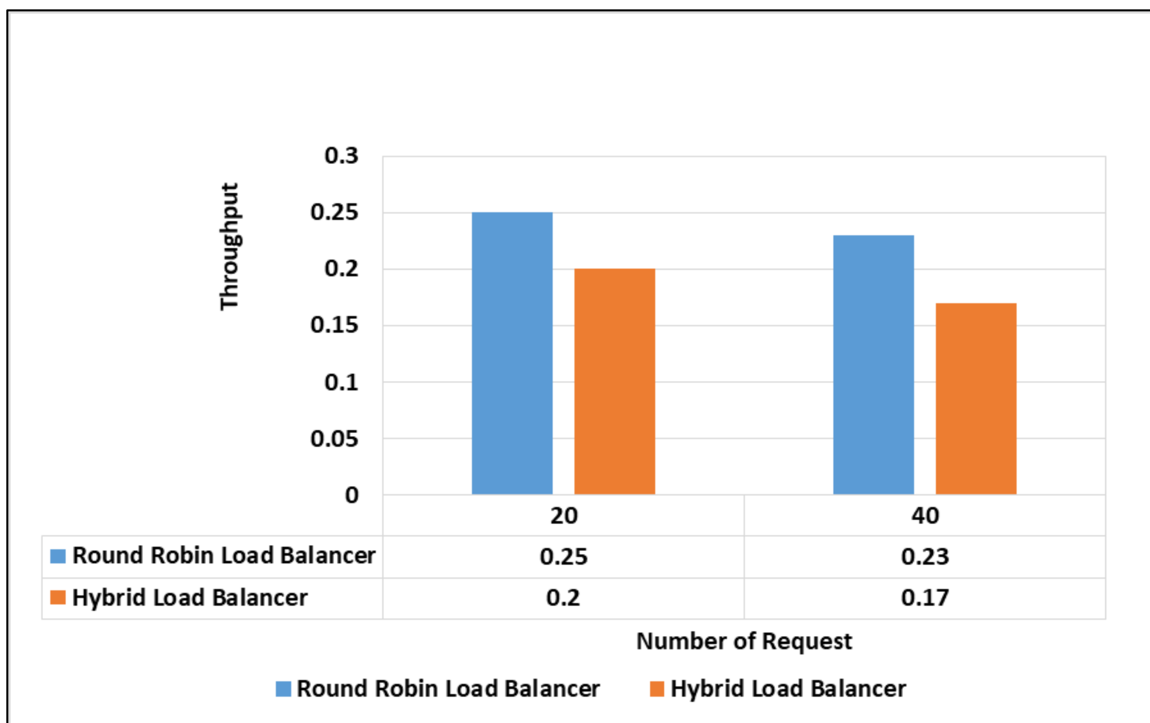

Figure 14: Throughput Comparison of Load Balancing Algorithms

The figure 14 shows a contrast of the throughput of the proposed Hybrid scheduler with the conventional round-Robin (RR) load balancer at 2 distinctive amounts of load (20 and 40 requests on the identical time).With 20 requests, RR gets 0.25 requests/unit time, while Hybrid only gets 0.20, giving RR a 0.05 (≈ 25% relative) edge. When the load doubles to 40 requests, RR gives 0.23 compared to Hybrid's 0.17, giving RR a 35.3% lead (0.23/0.17≈1.35). As the number of concurrent users increases, both algorithms' output slightly drops (RR –8%, 0.25–0.23; Hybrid –15%, 0.20–0.17). This is because there is more contention and control-path work to do when the loads are higher. Putting these numbers together with earlier latency data shows that the Hybrid design cuts down on average response time but has a slightly lower raw throughput in this setup. Three likely reasons are (i) decision overhead, which includes per-request fitness computation, EWMA updates, and active-set maintenance; (ii) synchronisation and I/O costs, such as thread spawning, socket operations, and shared-state updates, which can limit the number of completions per time unit; and (iii) health-check activity, which includes periodic probes (every 10 s) that add small but non-zero work. It's important to note that this is a normal and acceptable trade-off: the Hybrid policy puts in

Received: July 18, 2025

some control work to stop hotspots, even out load by active connections, and favour faster servers by response time. This makes users feel like the system is faster and more responsive, even when aggregate completions don't change much. Several technical improvements can be made in production settings to get throughput back without giving up the benefits of the Hybrid policy: to store fitness and only recalculate when a measure changes; Instead of scanning each request individually, use a max-heap/priority queue that is keyed by fitness; use I/O that doesn't block and a fixed thread pool; do health checks in the background; use lock-free/atomic counts for Ci To cut down on churn, tune EWMA γ plus weights (α, β) ($\pm$,^{2}). Overall, Figure 14 shows that RR still has a slight edge in terms of raw throughput in this small testbed. The Hybrid algorithm, on the other hand, puts latency and stability first, which is in line with the research goal of improving server response time and resource utilisation for a wide range of changing workloads.

7. Conclusion

A hybrid load-balancing model was designed and put into action in this study. It combines Round-Robin, Least Connections, and response-time awareness through a simple fitness score. With regular health checks and changes to metrics for each request, the system can handle different types of load and get rid of servers that aren't working well. The hybrid policy cut down on average response time compared to Round-Robin on a three-server, two-client testbed—2312 ms vs. 4074 ms for 20 requests and 2004 ms vs. 3894 ms for 40 requests—while keeping service reliable by automatically removing failed nodes. These results back up the main goal, which was to cut down on reaction time and make better use of resources when workloads change. As the load doubled, the hybrid scheduler had slightly lower throughput (0.20□0.17 vs. 0.25→0.23). This was because the balancer had to do more work to make decisions, keep things in sync, and do health checks. However, the latency benefits are big and useful for systems that users interact with. The study adds three things: a useful hybrid algorithm with weights that can be changed; an implementation that combines health monitoring and metric learning (EWMA); and an empirical review that shows that this method is more responsive and stable than traditional Round-Robin. There are some problems, like a small testbed, set server capacities, and no tail-latency reporting. Improvements in engineering, such as caching fitness computation, asynchronous I/O with a set thread pool, lock-free counters, and priority queue selection, can get throughput back without giving up gains in latency. We will look into adaptive weighting, autoscaling, and tail-latency goals on larger clusters in future work.

References

[1] S. V. J and C. Bagyalakshmi, "An Improved Performance Analysis for AWS Cloud Load Balancing Using Weighted Round Robin (WRR) Technique," 2025 International Conference on Computing Technologies (ICOCT), Bengaluru, India, 2025, pp. 1-8, doi: 10.1109/ICOCT64433.2025.11118409.

Received: July 18, 2025

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[2] PerumalGeetha, S.J. Vivekanandan, R. Yogitha, M.S. Jeyalakshmi,Optimal load balancing in cloud: Introduction to hybrid optimization algorithm,Expert Systems with Applications,Volume 237, Part C,2024,121450,ISSN 0957-4174,https://doi.org/10.1016/j.eswa.2023.121450.

[3] Adewojo, A.A., Bass, J.M. A Novel Weight-Assignment Load Balancing Algorithm for Cloud Applications. SN COMPUT. SCI. 4, 270 (2023). https://doi.org/10.1007/s42979-023-01702-7

[4] Yongze Wu and Shanshan Chen. 2025. Optimization of load balancing algorithm based on Informer long time series prediction. In Proceedings of the 2024 8th International Conference on Electronic Information Technology and Computer Engineering (EITCE '24). Association for Computing Machinery, New York, NY, USA, 985–991. https://doi.org/10.1145/3711129.3711297

[5] C. Jiang and X. Wang, "A Task Affinity Scheduling Framework Based on Multidimensional Resource Fingerprints," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2025.3599113

[6] Tache, M.D.; Păscuțoiu, O.; Borcoci, E. Optimization Algorithms in SDN: Routing, Load Balancing, and Delay Optimization. Appl. Sci. 2024, 14, 5967. https://doi.org/10.3390/app14145967

[7] Shuaib, M.; Bhatia, S.; Alam, S.; Masih, R.K.; Alqahtani, N.; Basheer, S.; Alam, M.S. An Optimized, Dynamic, and Efficient Load-Balancing Framework for Resource Management in the Internet of Things (IoT) Environment. Electronics 2023, 12, 1104. https://doi.org/10.3390/electronics12051104

[8] Hayyolalam, V.; Pourghebleh, B.; Chehrehzad, M.R.; Kazem, A.A.P. Single-objective service composition methods in cloud manufacturing systems: Recent techniques, classification, and future trends. Concurr. Comput. Pract. Exp. 2022, 34, e6698.

[9] Oikonomou, E.; Rouskas, A. Efficient Schemes for Optimizing Load Balancing and Communication Cost in Edge Computing Networks. Information 2024, 15, 670. https://doi.org/10.3390/info15110670

[10] Pourghebleh, B.; Hayyolalam, V. A comprehensive and systematic review of the load balancing mechanisms in the Internet of Things. Clust. Comput. 2019, 23, 641–661.

[11] Afzal, S.; Kavitha, G. Load balancing in cloud computing—A hierarchical taxonomical classification. J. Cloud Comput. 2019, 8, 22.

[12] Kaviarasan R, Balamurugan G, Kalaiyarasan R, VenkataRavindra Reddy Y, Effective load balancing approach in cloud computing using Inspired Lion Optimization Algorithm, e-Prime - Advances in Electrical Engineering, Electronics and Energy, Volume 6, 2023, 100326, ISSN 2772-6711,https://doi.org/10.1016/j.prime.2023.100326.

[13] Agrawal, N.; Tapaswi, S. Defense mechanisms against DDoS attacks in a cloud computing environment: State-of-the-art and research challenges. IEEE Commun. Surv. Tutor. 2019, 21, 3769–3795.

Received: July 18, 2025

Volume 38 No. 1s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[14] Shahid, M.A.; Alam, M.M.; Su'ud, M.M. Performance Evaluation of Load-Balancing Algorithms with Different Service Broker Policies for Cloud Computing. Appl. Sci. 2023, 13, 1586. https://doi.org/10.3390/app13031586

[15] Khan, A.R. Dynamic Load Balancing in Cloud Computing: Optimized RL-Based Clustering with Multi-Objective Optimized Task Scheduling. Processes 2024, 12, 519. https://doi.org/10.3390/pr12030519

[16] EL-Natat, A.; El-Bahnasawy, N.A.; El-Sayed, A.; Elkazzaz, S. Optimized Resource Allocation Algorithm for a Deadline-Aware IoT Healthcare Model. Big Data Cogn. Comput. 2025, 9, 80. https://doi.org/10.3390/bdcc9040080

[17] Malik, N.; Sardaraz, M.; Tahir, M.; Shah, B.; Ali, G.; Moreira, F. Energy-Efficient Load Balancing Algorithm for Workflow Scheduling in Cloud Data Centers Using Queuing and Thresholds. Appl. Sci. 2021, 11, 5849. https://doi.org/10.3390/app11135849

[18] Kumar, P.; Kumar, R. Issues and challenges of load balancing techniques in cloud computing: A survey. ACM Comput. Surv. (CSUR) 2019, 51, 1–35.

[19] Alankar, B.; Sharma, G.; Kaur, H.; Valverde, R.; Chang, V. Experimental Setup for Investigating the Efficient Load Balancing Algorithms on Virtual Cloud. Sensors 2020, 20, 7342. https://doi.org/10.3390/s20247342

[20] Shafiq, D.A.; Jhanjhi, N.Z.; Abdullah, A.; Alzain, M.A. A load balancing algorithm for the data centres to optimize cloud computing applications. IEEE Access 2021, 9, 41731–41744.

[21] Li, P.; Wang, H.; Tian, G.; Fan, Z. Towards Sustainable Cloud Computing: Load Balancing with Nature-Inspired Meta-Heuristic Algorithms. Electronics 2024, 13, 2578. https://doi.org/10.3390/electronics13132578

[22] M. Pandey, B. Tak and Y. -W. Kwon, "PROBA: Enhancing Serverless Edge Computing via Adaptive Task Scheduling and Probabilistic Resource Sharing," 2025 IEEE 18th International Conference on Cloud Computing (CLOUD), Helsinki, Finland, 2025, pp. 351-361, doi: 10.1109/CLOUD67622.2025.00043.

[23] V. N R, S. D P, T. S. Agarwal, V. H M and S. D, "Optimizing Cloud Resource Allocation with AI and Machine Learning," 2025 International Conference on Computing Technologies (ICOCT), Bengaluru, India, 2025, pp. 1-5, doi: 10.1109/ICOCT64433.2025.11118766.

Received: July 18, 2025