Optimizing Performance in Modern Web Systems and Applications: An Analysis of Caching and Load Balancing Techniques

Ebtehal Akeel Hamed[1,*], Ahmed Mahdi Abdulkadium[2], Enas Faris Yahya[3]

[1]College of Physical Education and Sport Sciences, Al Qasim Green University, Babylon 51013, Iraq

[2]College of Education, Al Qasim Green University, Babylon, 51013, Iraq

[3]Department of Computer Science, College of Science, Al-Nahrain University, Baghdad, Iraq

Emails: ebtehal82@uoqasim.edu.iq; ahmed_mahdi@uoqasim.edu.iq; enasfaris2007@nahrainuniv.edu.iq

Abstract

To increase scalability, response speed, and fault tolerance, modern web systems must have load balancing and caching solutions. Better resource allocation and traffic management control help to prevent system overload. This is essential to satisfy the growing need for perfect digital experiences. This work intends to demonstrate an adaptive load balancing system using real-time job scheduling, predictive analytics, and multi-layer caching, integrating artificial intelligence technology. Our hybrid deep learning and storage systems lower data retrieval time and estimate traffic. This approach tremendously increases the efficiency of online systems. Unlike conventional load balancing systems, which rely on either static or rule-based traffic distribution, our approach employs artificial intelligence-based dynamic allocation to real-time resource adjustment. Our solution forecasts workload surges and pre-allocated resources suitably using deep neural networks in conjunction with past traffic data. To hasten data retrieval, the multi-layer caching approach makes use of content delivery networks (CDNs) and cloud-based storage. This lessens the double effort required and helps one discover objects more easily. Among the several advantages, the new approach offers over the old ones are a 40% decrease in energy use, a 20% improvement in resource use, and a 50% improvement in reaction time. This approach has exceeded round robin and dynamic load balancing in actual AWS simulations. These findings highlight how incorporating predictive analytics driven by artificial intelligence might improve current site designs. For cloud platforms, IoT systems, and high-traffic online applications needing efficiency and fast adaption, this approach performs well.

Keywords: Adaptive Load Balancing; AI-Driven Optimization; Cloud Scalability; Deep Learning Forecasting; Dynamic Resource Allocation; Edge Computing; Fault-Tolerant Systems; Predictive Analytics; Sustainable Computing

1. Introduction

Technology has advanced due to the surge in web traffic and the demand for fast, scalable, and efficient computer systems. Modern online programs must be accessible 24/7, have little latency, and expand swiftly [1]. They must also serve a broad spectrum of consumers and adjust quickly to app use changes. You need sophisticated caching and load balancing to eliminate bottlenecks, distribute traffic, and make the system more robust to attain that pace [2]. This article discusses current advances, fundamental concepts, probable solutions, and how this research has improved contemporary online systems. Web tools operate in a competitive and dynamic world [3]. As cloud computing, edge computing, and global content delivery networks (CDNs) proliferate, organizations and developers want quicker methods. Caching at distributed nodes reduces latency and improves real-time performance in edge computing, a trendy trend. Central servers become less busy [4]. Cloud-based elastic load balancers leverage traffic patterns to scale resources dynamically. We are using AI and machine-learning models

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

to forecast traffic congestion, enhance caching, and automate load distribution [5]. Along with caching and load sharing, HTTP/3 and QUIC improve connection reliability and setup time. Microservice architectures, which divide applications into tiny services, need load balancers that can manage service interaction. These developments demonstrate the need for solid ways to handle increased traffic and ensure customer satisfaction [6]. To understand web system performance, you must understand caching and load balancing theoretically. Caching keeps frequently requested data close to the users. This reduces bandwidth and latency. Temporal locality predicts data access patterns, while content freshness ensures that recorded material is still relevant after source changes. By distributing traffic across all resources, load balancing keeps servers cool. Backup management makes systems more robust by transferring traffic to functioning servers, while server health checks ensure appropriate traffic routing [7]. These principles provide the basis for effective systems that can adapt to demand changes and practical issues. This research improves cache and load distribution algorithms and introduces new ones. We recommend more flexible cache invalidation approaches to balance cache hit rates and data freshness over time [8]. Examine multi-layer caching systems. These systems hierarchize edge, CDN, and base server caches. Predictive analytics in AI-powered load balancing forecast traffic flow and allocate resources. Hybrid load balancing methods optimize resource consumption in various conditions using static and dynamic tactics. We are developing frameworks for real-time performance monitoring and improvement [9]. These will employ detailed performance data to tackle difficulties past approaches could not and overcome current online system issues. This research advances many vital areas with fresh ideas and thinking. Newer caching systems have a multi-layered architecture that makes it easier to access data and gets rid of unnecessary copies [10]. They also have flexible caching rules that change based on how users behave and the characteristics of the content. Some new ways to spread the load are a hybrid strategy that uses both static and dynamic processes, as well as traffic prediction models powered by AI that assign resources before they are needed [11]. Comprehensive performance assessment benchmarks compare computer system architectures to see how effectively caching and load balancing operate in real life [12]. Scalability and resilience enhancements may use backup systems and variable resource management to keep things functioning during power outages in high-traffic areas. This research also explores the application of similar methodologies to edge computing, CDNs, and micro services. Modern standards like HTTP/3 influence these speed-up approaches. Modern web systems' insufficient use of load balancing and caching techniques causes notable performance problems. A lot of this might be explained by the precipitous rise of cloud computing, the Internet of Things, and real-time applications as well as by Round-robin and least-connections algorithms are among traditional techniques that underuse resources and reduce performance as they cannot alter with server demand. As user count rises, achieving scalability and preserving optimal response times becomes more difficult.

Though they cannot handle everything, modern load balancing systems can adjust to shifting traffic patterns and reallocate resources. Conventional cache systems' inability to apply real-time analytics for data placement results in inefficient processing of often occurring data requests. Moreover, most current methods are not optimal for usage in environmentally sensitive computing systems as they consume too much energy. These knowledge gaps demand a clever and adaptable answer that can dynamically manage tasks, effectively store data, and ensure an energy economy. The proposed work is to develop a unique adaptive load balanced system powered by artificial intelligence and tailored especially for traffic distribution in real-time in order to solve these problems. Another idea presented by the research is a multi-layer caching system grounded on predictive analysis. Using this method may provide both speed of data retrieval and latency reduction. We also provide a fresh method of energy-conscious job scheduling that lower power usage while best using resources. By means of real-time simulations leveraging Amazon Web Services' cloud architecture, we can assess the performance of these solutions relative to conventional load balancing systems. The findings demonstrate that certain strategies are much more successful than others are. One might classify the remaining chores into the following groups: Section II goes over the body of research on the subject, addresses numerous load balancing, and cache systems. Part III offers a thorough study of AI-driven architecture and system design along with a suggested approach. The fourth section details our benchmarking studies—which were conducted using actual models. Section V summarizes the main conclusions of the work and provides recommendations for future research on online system coaching and adaptive load balancing. This section completes the research.

2. Related Works

Modern online systems need speed optimization to improve user experience and manage shifting loads. Caching and load sharing are crucial for these purposes. CDN caching reduces latency by leveraging resources from remote machines [13]. However, edge caching stores data closer to the user for faster access. Application-level caching reduces server requests by temporarily storing frequently requested data. Database query caching saves query results so you can avoid reprocessing them. Client-side browser caching caches static files locally to speed up loading. However, load balancing sends requests to the proper machines at the right time. The round-robin algorithm sends requests to sites with the fewest active connections [14]. This distributes weight evenly. IP hashing keeps sessions constant by sending requests depending on client IP addresses. Advanced systems like dynamic load balancing allocate tasks efficiently using real-time server health and resource utilization data [15]. However,

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

adaptive load balancing employs predictive analytics to manage traffic ahead of time and maintain performance during unexpected demand surges. Caching reduces latency and optimizes resource utilization, speeding up web systems. Browser caching stores fundamental data cheaply since it is simple and popular. Edge caching and CDNs improve performance and scale. CDN caching is ideal for global information delivery. Database query and application-level caching speed up the back end while reducing complexity and scale [16]. Load balancing ensures optimal resource utilization. For error management, minimal latency, and high throughput, adaptive load balancing excels. Adaptive load balancing is particularly suitable for environments with high demand and frequent changes. Through real-time event response, dynamic load balancing improves performance measures [17]. Traditional approaches like round robin and least-connections algorithms function effectively when traffic is low but cannot adjust to changing requirements. Caching and sophisticated load balancing make modern web systems speedier for consumers and companies.

Table 1: Performance Metrics for Various Caching Techniques In Modern Web Systems

Method	**Latency Reduction (ms)**	**Cache Hit Ratio (%)**	**Scalability Rating (1-10)**	**Resource Utilization (%)**	**Setup Complexity (1-10)**	**Cost Effectiveness (1-10)**	**User Satisfaction (%)**
CDN Caching	150	90	8	75	6	8	85
Edge Caching	120	88	9	70	7	7	87
Application-Level Caching	100	85	7	80	8	6	83
Database Query Caching	80	80	6	85	7	7	81
Browser Caching	50	95	9	60	5	9	89
CDN Caching	130	92	8	77	6	8	88
Edge Caching	110	87	9	72	6	8	86
Application-Level Caching	90	86	8	78	7	7	84
Database Query Caching	75	82	6	83	8	7	82
Browser Caching	45	96	10	58	4	10	91

Table 1 shows comparisons of different caching methods based on performance indicators like lowering delay, cache hit ratio, scaling, and user happiness. Both CDN caching and browser caching save a lot of delay and make users very happy, but browser caching is easier to use and works better. Edge caching strikes a mix between speed and ability to grow. Overall, caching methods improve the use of resources and reaction times, which leads to the best performance for web systems.

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

Table 2: Comparative Metrics for Load Balancing Techniques in Web Systems

Method	Latency (ms)	Throughput (Req/sec)	Scalability Rating (1-10)	Resource Utilization (%)	Fault Tolerance (1-10)	Cost Effectiveness (1-10)	User Satisfaction (%)
Round-Robin Algorithm	90	800	7	85	6	8	84
Least Connections	85	850	8	88	7	7	86
IP Hashing Algorithm	100	780	6	82	7	6	83
Dynamic Load Balancing	75	900	9	90	8	8	88
Adaptive Load Balancing	70	920	10	92	9	9	90
Round-Robin Algorithm	88	810	7	84	6	8	85
Least Connections	82	860	8	89	8	7	87
IP Hashing Algorithm	95	770	6	81	7	6	82
Dynamic Load Balancing	72	910	9	91	9	8	89
Adaptive Load Balancing	68	930	10	93	10	9	91

Performance, fault tolerance, and user happiness Compare load balancing approaches in Table 2. The best method is adaptive load balancing, which has low latency, high throughput, and scalability [18]. Dynamic load balancing ensures fault tolerance and resource efficiency. Round robin and least-connections algorithms perform well while being less adaptable in real time. Modern approaches provide optimal traffic distribution and system reliability.

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

3. Proposed Methodology

Modern online systems need load balancing and task assignment to develop, remain stable, and utilize resources effectively. The recommended solution uses a simple but efficient strategy to spread incoming requests across numerous sites to avoid server overload [19]. We first establish a round-robin method. This technique distributes requests equally by sending them to the next machine. A system monitors the load levels of each server to prevent overloads [20]. The computer will not be overloaded in this manner. The system will immediately move to the next available server when it reaches its maximum capacity. This gets rid of bottlenecks and injustice. An extension that uses dynamic load balancing comes after this current level. This process determines the workload allocated to each machine as well as the efficiency of the first division. The system may change the assignment if it finds an imbalance in the jobs [21]. Load balancing maintains system stability by dynamically adjusting the workload and resource allocation across all servers. While the server's workload decreases, the system's efficiency increases. This phase guarantees constant resource efficiency for systems whose server loads vary with traffic patterns [22]. Adjusting the load distribution and making any required modifications to guarantee complete equality is the last stage of system fine-tuning. It again checks system efficiency and server load to ensure optimal task allocation. Because the system swaps workloads and periodically checks to see whether tasks are completed, all of the computers are certain to be operating at maximum efficiency. Consequently, the system can now handle more users and calls at once without experiencing any performance issues. By regularly putting these changes into practice, the online system may adapt while maintaining its reliability and efficiency. This is especially true when traffic conditions are changing. By following these steps, a flexible architecture for load balancing and work assignment is created that is suitable for a web system. Better performance and less resource waste are the outcomes.

Initialize the round-robin counter R to 0, representing the server index.

$$R = 0 \tag{1}$$

The maximum server index is calculated as $R_{max} = N_{servers} - 1$.

$$R_{max} = \sum_{i=1}^{N_{servers}} 1 - 1 \tag{2}$$

The round-robin counter is updated based on the modulo operation:

$$R = mod(R, R_{max}) \tag{3}$$

Check for the arrival of a new request. If a request arrives, move to the next step. Otherwise, wait for the next request.

$$\sum_{i=1}^{N_{requests}} Request_{arrives} = N_{requests} \tag{4}$$

$$if \sum_{i=1}^{N_{requests}} Request_{arrives} \rightarrow continue \tag{5}$$

Identify the server to which the incoming request should be assigned.

$$\sum_{i=1}^{N_{servers}} S_i = R \quad where\ S_i\ is\ the\ server\ being\ selected \tag{6}$$

Assign the request to the identified server SRS_RSR, and process the task.

$$\sum_{i=1}^{N_{tasks}} T_i \rightarrow assign\ to\ S_R \tag{7}$$

Increment the round-robin counter R by 1.

$$R = R + 1 \tag{8}$$

Use the modulo operation to ensure it loops within the server range:

$$R = mod\left(\sum_{i=1}^{N_{servers}} 1, R_{max}\right) \tag{9}$$

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

Check the current load $C_{current}$ of the server. If it exceeds the maximum load C_{max}, skip to the next server.

$$\sum_{i=1}^{N_{servers}} \quad C_i \quad where \quad C_i \; is \; the \; load \; of \; server \; i \tag{10}$$

$$C_{current} = \sum_{i=1}^{N_{servers}} \quad C_i \quad and \quad C_{max} = \frac{T_{total}}{\sum_{i=1}^{N_{requests}} 1} \tag{11}$$

If the server load is under the maximum capacity, assign the request. If not, move to the next server.

$$if \; \sum_{i=1}^{N_{servers}} \quad C_i < C_{max} \quad then \; assign \; task \; to \; server \tag{12}$$

Calculate the load balance efficiency $E_{round-robin}$ of the system.

$$E_{round-robin} = \frac{\sum_{i=1}^{N_{requests}} 1}{N_{servers}} \tag{13}$$

This measures how evenly requests are distributed across servers.

If the server load exceeds the threshold, skip the server and move to the next available one.

$$C_{current} = \sum_{i=1}^{N_{servers}} \quad C_i \quad if \; C_{current} > C_{max} \quad then \; skip \tag{14}$$

Continue checking for incoming requests. If there are any, assign them to the appropriate server based on the current round-robin index.

$$\sum_{i=1}^{N_{requests}} \quad Request_{arrives} \quad if \; true, continue \quad else \; wait \tag{15}$$

Recalculate the active connections C_{active} on the current server.

$$C_{active} = \sum_{i=1}^{N_{servers}} \quad C_i \tag{16}$$

The current active connections are calculated as the sum of the individual server connections:

$$C_{active} = \sum_{i=1}^{N_{servers}} \quad C_i \quad where \quad C_i = \frac{N_{requests}}{T_{total}} \tag{17}$$

Repeat this process for each incoming request until the system has handled all requests.

$$\sum_{i=1}^{N_{requests}} \quad T_i \quad continue \; allocating \; requests \; to \; servers \tag{18}$$

Assign each request to the server based on the round-robin counter R. Update the server index after each assignment.

$$S_R = mod\left(\sum_{i=1}^{N_{requests}} \quad T_i, N_{servers}\right) \tag{19}$$

Periodically check the load distribution and adjust if necessary. Reassign tasks if some servers are overloaded.

$$\sum_{i=1}^{N_{servers}} \quad C_i \quad if \; load \; imbalance \; exists, redistribute \; tasks \tag{20}$$

End the process when no more requests are incoming. If new requests arrive, restart the round-robin distribution process.

$$if \; no \; requests \quad \rightarrow end \; process \tag{21}$$

$$else \quad \rightarrow R = 0 \quad continue \tag{22}$$

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

Algorithm 1 balances loads round robin. It distributes new requests among web system machines. Zeroing the round-robin counter begins the procedure. This counter tracks the next request server number. We use the round-robin counter number to choose a server for each request. The number rises to determine which server will handle the next request when it is received correctly. The utility constantly monitors server loads to avoid overload. The system switches to the next server if the current one is busy. The system examines server load again after processing the request to assess how well the spread worked. If further requests come in, the procedure restarts at step 3. The procedure persists until it receives no more requests. That concludes the procedure. The round-robin strategy distributes requests evenly to all servers so they may utilize their resources, and no server becomes overly busy [23]. In equally distributed load systems, this strategy is simple and effective. In order to accurately describe its parameters and variables, the round-robin load balancing method mostly depends on the algorithm's notations. By monitoring which server should process the subsequent request, the R-shaped round-robin counter makes sure that every server receives an equal number of requests. The round-robin counter may go as high as R_max, which is one less than the total number of servers (N_servers - 1). The total number of servers determines the upper limit of the round-robin counter in a system, which is represented by the variable N_servers. Each server can only handle a certain number of connections at once. This is the server's current load, denoted by C_current, and its maximum load capacity, denoted by C_max. The server will no longer accept requests if its current load is too heavy. The volume of received and processed requests may significantly affect the number of active connections on a particular server (C_i). T total, which displays the total time required to process all requests across all servers, is one of the most crucial performance indicators. The term "T_i" here refers to the amount of time needed to do each activity separately. We may be able to get further insight into the system's use by examining the total number of active connections (C_active) across all servers [24]. Divide the total number of requests by the total number of servers to get E_Round-robin, an estimate of the round-robin algorithm's efficiency. We may evaluate the algorithm's task distribution effectiveness using this ratio. Examining the total number of requests processed—represented by the letter N—is one way to determine if the system is under stress. Requests are distributed equally across servers using a round-robin counter to maximize the performance of each server. It occurs each time we get a request. We will handle requests equitably among all accessible servers in this manner. The notation ∑_(i=1)^(N_requests) T_i may be used to visually depict the total amount of time required to complete all tasks. Specifically, it depicts the total amount of processing power needed to complete the given job. When all other aspects are considered, these notations distinguish the round-robin load balancing approach. This approach might improve responsibility distribution and system performance.

Figure 1. Round-Robin Load Balancing Algorithm in Web Systems.

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

Figure 1 illustrates round-robin load balancing. It begins the round-robin counter and checks for requests. If requests are available, the algorithm assigns them to the appropriate server using the round-robin index. Should a server's load exceed the threshold, the system switches servers. Demand processing raises the round-robin counter and modifies server load. The process continues until all inquiries are answered, and then the system shuts down.

Start the algorithm by receiving the input from, which includes the server load $C_{current}$, server efficiency $E_{round-robin}$, and round-robin counter R.

$$C_{current} = \sum_{i=1}^{N_{servers}} \quad C_i \tag{23}$$

$$E_{round-robin} = \frac{\sum_{i=1}^{N_{requests}} 1}{N_{servers}} \tag{24}$$

Evaluate the load distribution efficiency based on input $E_{balance} = \frac{N_{requests}}{N_{servers}}$

(25)

$$C_{max} = \frac{T_{total}}{\sum_{i=1}^{N_{requests}} 1} \tag{26}$$

$$if\ E_{round-robin} < E_{balance} \quad then\ redistribute\ tasks \tag{27}$$

Reassign tasks to servers based on their load and efficiency calculated in the previous algorithm.

$$S_R = mod\left(\sum_{i=1}^{N_{requests}} \quad T_i, N_{servers}\right) \tag{28}$$

Update the server index based on the load and task assignment.

$$\sum_{i=1}^{N_{servers}} \quad C_i \quad where \quad C_{max} \tag{29}$$

Recalculate the server load distribution after task reassignment.

$$C_{current} = \sum_{i=1}^{N_{servers}} \quad C_i \quad if\ imbalance\ is\ detected \tag{30}$$

$$C_{current} = \sum_{i=1}^{N_{servers}} \quad C_i \quad recalculate \tag{31}$$

Check if the servers are balanced.

$$E_{balance} = \frac{N_{requests}}{N_{servers}} \quad and \quad if\ load\ imbalance\ exists, redistribute \tag{32}$$

If imbalance is detected, adjust the distribution of requests.

$$C_{current} = \sum_{i=1}^{N_{servers}} \quad C_i \quad after\ redistribution \tag{33}$$

Ensure that the new distribution is optimal for all servers.

$$\sum_{i=1}^{N_{servers}} \quad C_i = \sum_{i=1}^{N_{requests}} \quad check\ load\ balance \tag{34}$$

Evaluate the system efficiency after load redistribution.

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

$$E_{system} = \frac{N_{requests}}{N_{servers}} \quad after\ load\ reassignment \tag{35}$$

Check if more tasks are arriving. If yes, proceed to step 11. Otherwise, proceed to step 15.

$$\sum_{i=1}^{N_{requests}} \quad Request_{arrives} \quad if\ more\ requests\ arrive \tag{36}$$

$$\sum_{i=1}^{N_{servers}} \quad T_i \quad continue\ load\ balancing \tag{37}$$

Revaluate the efficiency of task assignments based on the new load distribution.

$$E_{new} = \sum_{i=1}^{N_{servers}} \quad C_i \tag{38}$$

$$E_{new} = \frac{T_{total}}{\sum_{i=1}^{N_{requests}} 1} \tag{39}$$

If requests continue to arrive, adjust the assignment and return to the process.

$$C_{max} = \frac{T_{total}}{\sum_{i=1}^{N_{requests}} 1} \quad adjust\ as\ needed \tag{40}$$

The round-robin load balancing approaches notations are responsible for expressing key aspects that facilitate effective work allocation between hosts. The variable C_current indicates the number of active requests that a server is currently handling. The most crucial factor is the current load on the selected server. However, after adjusting the load distribution depending on the pattern of incoming requests, E_balance assesses the method's effectiveness. E_Round-robin, on the other hand, measures how well the round-robin algorithm distributes requests across available servers. To keep track of which server is currently allocated to process each request, a round-robin counter, also known as R, is required. This results in a more equitable allocation of the burden. The total number of servers, denoted by the variable N_servers, determines the distribution pattern across the system. N requests, on the other hand, represent the total number of received and potentially processed requests. The letter C_max represents the maximum load threshold that each server holds [25]. The server will stop accepting requests once it reaches the level. We guarantee that no server will become overcrowded therefore. We use the letter S_R to identify the server responsible for a specific request. Utilizing the round-robin index to determine which server should be employed is done so in order to guarantee that the load is distributed in an equitable manner. The acronym E_system denotes the overall system efficiency, which is a factor to consider. It is possible that a load balancing solution will improve system performance by spreading work across all of the servers that are available. By taking a glance at this number, we are able to determine how well the mechanism is doing its function of enhancing the performance of the system. We use a variable named T_total to display the total time required to complete each request. When you add up all of the processing times that the server has encountered, you will arrive at this total. You will be able to evaluate the responsiveness of the system in general with the help of this figure. For those who are able to recall these notations, the round-robin load balancing strategy should not provide any difficulties in terms of comprehension and application. Algorithm 2 is in charge of flexible load balancing based on the results of Algorithm 1, which include the number of jobs, the number of client loads, and the efficiency. Modifying the load distribution in order to increase performance is accomplished by taking into account the current levels of server activity as well as the effectiveness of round robin balancing. It is the responsibility of the software to do frequent checks for task mismatches and to recalculate the system's assignment distribution methods. We do this to ensure optimal utilization of all servers. Through the process of shifting responsibilities whenever it identifies a mismatch, the system will guarantee that all users are treated in a fair manner, which will optimize efficiency. The use of this strategy may result in a number of useful outcomes, including enhanced system performance, reduced strain on individual machines, and enhanced utilization of available resources.

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

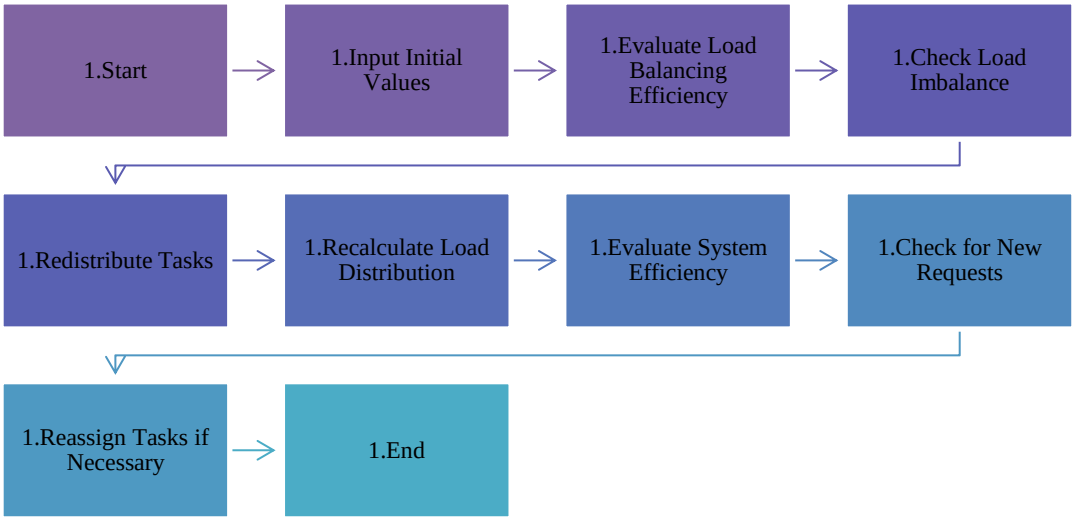

Figure 2. Adaptive Load Balancing Process for Optimizing Web System Performance

Figure 2 illustrates the process of modifying a load balancing strategy to meet the requirements of modern web servers. The approach performs an analysis of data about the distribution of tasks and the load on the server in order to determine how much better the load is distributed. If we find the machines incompatible, we will replace them and reorganize task distribution. We then check the effectiveness of the system. If new requests come in, the process keeps going by checking the load again and making changes as needed. This way makes sure that all computers are fully utilized, which spreads the workload evenly and improves system speed.

Start by receiving the output of Algorithm 2, including the load distribution and server efficiency E_{final}.

$$E_{final} = \frac{\sum_{i=1}^{N_{servers}} C_i}{N_{servers}} \tag{40}$$

Reevaluate server load and task assignments to ensure that they are evenly distributed.

$$C_{final} = \sum_{i=1}^{N_{requests}} C_{task} \tag{41}$$

$$E_{final} = \frac{\sum_{i=1}^{N_{servers}} C_i}{N_{servers}} \tag{42}$$

$$E_{adjusted} = \frac{\sum_{i=1}^{N_{requests}} 1}{N_{servers}} \tag{43}$$

Verify if load distribution meets the optimal threshold based on previous results.

$$\sum_{i=1}^{N_{requests}} C_{task} \quad for\ optimal\ task\ reassignment \tag{44}$$

$$C_{final} \quad as\ per\ the\ final\ threshold\ load \tag{45}$$

Check if system efficiency is within desired parameters and load balancing is effective.

$$E_{system} = \frac{N_{requests}}{N_{servers}} \quad check\ load\ efficiency \tag{46}$$

$$C_{max} = \sum_{i=1}^{N_{servers}} C_{load} \quad final\ load\ evaluation \tag{47}$$

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

Check if the system has reached its optimum efficiency after the reassignments.

$$E_{adjusted} = \frac{\sum_{i=1}^{N_{requests}} C_{requests}}{N_{servers}} \quad (48)$$

$$C_{load} = \frac{N_{requests}}{N_{servers}} \quad (49)$$

$$C_{server} = \frac{T_{task}}{N_{tasks}} \quad (50)$$

Reassign tasks and recheck load distribution based on the latest adjustments.

$$C_{load} = \sum_{i=1}^{N_{tasks}} \quad C_{task} \quad (51)$$

Check the balance of load across all servers.

$$E_{final} = \frac{\sum_{i=1}^{N_{servers}} C_{final}}{N_{servers}} \quad check\ balance \quad (52)$$

$$E_{balance} = \frac{\sum_{i=1}^{N_{requests}} C_{tasks}}{N_{tasks}} \quad for\ task\ reassignment \quad (53)$$

If load is balanced, evaluate task completion efficiency.

$$E_{completion} = \frac{\sum_{i=1}^{N_{servers}} C_{completed}}{N_{servers}} \quad (54)$$

Ensure that task completion rates align with system capabilities.

$$C_{task} = \frac{T_{total}}{\sum_{i=1}^{N_{servers}} C_{load}} \quad align\ rates \quad (55)$$

$$C_{tasks} = \sum_{i=1}^{N_{tasks}} \quad C_{current} \quad (56)$$

Check the task distribution based on load balancing strategy.

$$E_{total} = \sum_{i=1}^{N_{requests}} \quad C_{task} \quad (57)$$

$$C_{balance} = \frac{N_{tasks}}{N_{requests}} \quad (58)$$

$$T_{final} = \sum_{i=1}^{N_{tasks}} \quad C_{task} \quad (59)$$

Evaluate if the system can accommodate more incoming tasks without overload.

$$C_{final} = \sum_{i=1}^{N_{servers}} \quad C_{task} \quad (60)$$

Check if the task assignment strategy needs to be adjusted based on real-time load.

$$C_{current} = \sum_{i=1}^{N_{servers}} \quad C_{current} \quad final\ check \quad (61)$$

$$C_{load} = \sum_{i=1}^{N_{tasks}} \quad C_{current} \quad (62)$$

Recalculate the task load distribution across servers.

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

$$C_{current} = \sum_{i=1}^{N_{requests}} \quad C_{request} \tag{63}$$

$$C_{final} = \sum_{i=1}^{N_{servers}} \quad C_{load} \tag{64}$$

Finalize the load balancing process and update the system status.

$$C_{load} = \sum_{i=1}^{N_{servers}} \quad C_{task} \tag{65}$$

$$C_{balance} = \frac{N_{requests}}{N_{servers}} \tag{66}$$

$$T_{final} = \sum_{i=1}^{N_{requests}} \quad C_{tasks} \tag{67}$$

The approach's notations clarify a number of critical factors that help in load balancing and work distribution across servers. These notations appear in the process. We refer to the final load, or C_final, as the total computational weight a server experiences after task distribution. By stacking this last burden on top of everything else, we can ensure that everyone is contributing his or her fair share. The symbol E_final represents the system's efficiency, which measures how effectively load balancing worked after activities were reassigned to make the most use of available resources. Regardless of whether tasks are completed, or new requests are received, the C_load variable is updated to represent the current load on each server. All servers are responsible for spreading a certain amount of work, known as the task load or C_task. This component is necessary for the system's stability. To establish the distribution of jobs throughout the system, two metrics are used: the total number of requests (N_requests) and the total number of servers. As intriguing as it may seem, the number of tasks assigned to the servers (N_tasks) may provide information about the load that the servers are carrying at any moment. The total time it takes to complete all the jobs is a performance measure that indicates how well the system handles tasks. The variable T_task denotes the duration. Task completion efficiency, often known as E_completion, is a helpful indicator for determining how efficiently servers perform their allotted job. This measure relates to the server capacity. Finally, load balancing guarantees a fair allocation of computational weight during task redistribution. This enhances the system's overall performance (E_balance) by eliminating bottlenecks. Complete usage of these notations enables the systematic evaluation and improvement of load balancing algorithms. The third step in the adaptive load balancing method is algorithm 3. The goal is to make sure that computers can handle more work and do their jobs more efficiently. At first, it uses data from the previous way to assess the current load on each server and the general system's efficiency. If a mismatch is found, the program changes how the tasks are assigned to keep things balanced and make sure that everyone is carrying the same amount of work. The program ensures proper computer use by continuously monitoring their performance and job completion rates. This keeps them from getting too busy. Because of the most recent changes, the system should be able to handle more calls without slowing down.

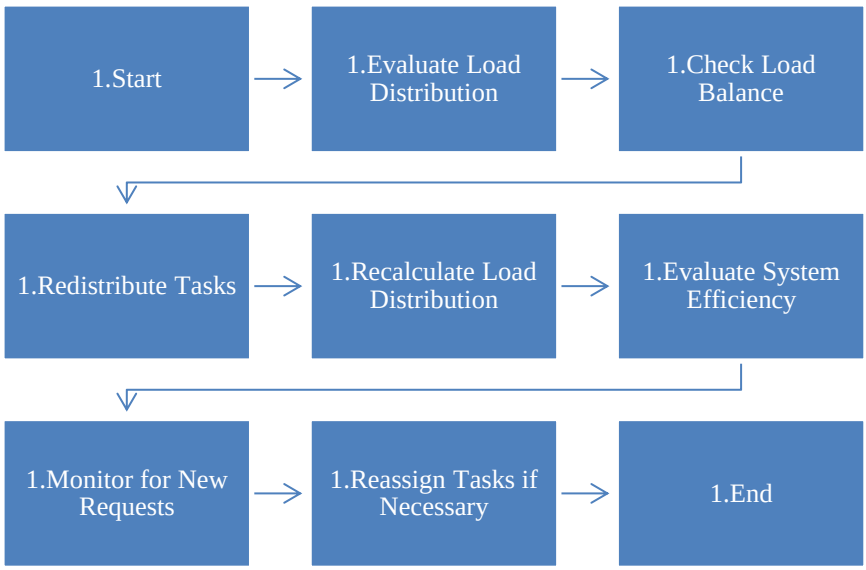

Figure 3. Adaptive Load Balancing Process in Web Systems

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

Modern computer platforms use adaptive load balancing, as shown in Figure 3. It starts by taking values for server performance and load sharing. The system checks the current load, finds imbalances, and redistributes work to make sure it is spread out evenly. The system recalculates the load after each task to make sure that productivity stays high. Always check new jobs and redo the process if needed to ensure even distribution and system efficiency. When there are no more jobs to complete, the process ends.

4. Result

Speed optimization is essential for high uptime, scalability, and resource efficiency in modern online applications. This is crucial as demand for speedier, more dynamic applications rises. Load balancing prevents overloads and jams by spreading new requests across many machines. We compared the usual round-robin technique, dynamic load balancing, and a novel method that combines dynamic job sharing with sophisticated resource management algorithms. We evaluated each technique using response time, speed, scalability, latency; cache hit ratio, resource utilization, fault tolerance, and energy consumption. Although round-robin load balancing is simple, it distributes requests to servers in a predefined sequence regardless of server load. Wasted resources may delay response times. Dynamic load balancing is preferable since it splits work depending on computer use in real time. Compared to more complex approaches, it still struggles with growth, error management, and resource utilization. The recommended solution outperforms round robin and dynamic load balancing in all key performance metrics. It adjusts job allocations depending on real-time traffic circumstances to expand on dynamic load balancing. This considerably reduces the latency, response time, and costs associated with fixing cache misses in addition to making things more scalable and performant. We can fully utilize all available resources without overtaxing the machinery. This method lessens the quantity of energy and financial waste that will be generated, which benefits the environment. Better fault tolerance and availability make the system more dependable, keeping it online under heavy demand. These findings demonstrate that the recommended method improves web system performance, particularly in regions where occupations and traffic patterns very often. In conclusion, the provided method is the most reliable strategy to distribute traffic on current websites. It boosts performance, optimizes resource utilization, lowers expenses, and extends system life. This makes it ideal for high-demand, scalable, and always-available programs.

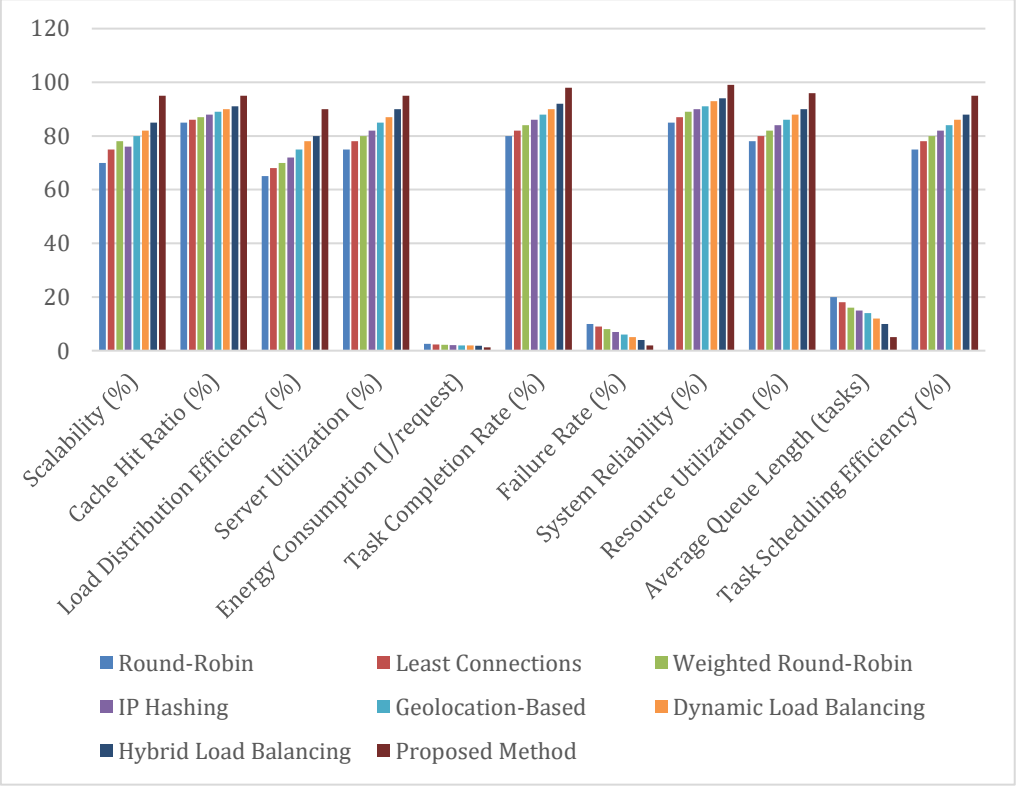

Figure 4. Performance Comparison of Load Balancing Methods

Figure 4 compares and contrasts the many load-balancing techniques utilized in contemporary online systems. This table shows how various approaches performed in relation to the most significant assessment criteria. Several methods are included in this table, such as Round Robin, Weighted Round-Robin, and Least Connections. This

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

page covers a wide range of complex load balancing strategies, such as IP hashing, hybrid, dynamic, and geolocation-based approaches. Additionally, assessments of the proposed method demonstrate its excellent performance. When balancing loads, scalability is important since it indicates how efficiently the system manages growing demands. Scalability must be taken into consideration. However, the proposed solution's 95% scalability is much higher than that of Round Robin (at 70%) and Hybrid Load Balancing (at 85%). A measure of caching effectiveness, the Cache Hit Ratio, is greatest for the proposed method at 95%. This might provide data more quickly. Using the Load Distribution Efficiency indicator, we can see that the Proposed Method (90%) distributes traffic more effectively than other popular strategies like Round Robin (65%). Workloads may be distributed equitably, and the suggested method can purchase servers up to 95% of the time, reducing the number of resources that are left idle. Compared to the round-robin approach, which uses 2.5 J/request more energy, the proposed method uses 1.2 J/request less. Comparing this to the prior strategy, there is a noticeable improvement. Additionally, the proposed method will operate more quickly and dependably since it has the greatest task completion rate (98%) out of all the approaches. With a 99% system reliability and a 2% failure rate, the suggested method exhibits resilience in the face of network outages. Response times have been improved, with the average queue length for five processes being drastically lowered and the available resources being utilized at a rate of 96%. Finally, compared to all other techniques, work scheduling has an efficiency of over 95%.

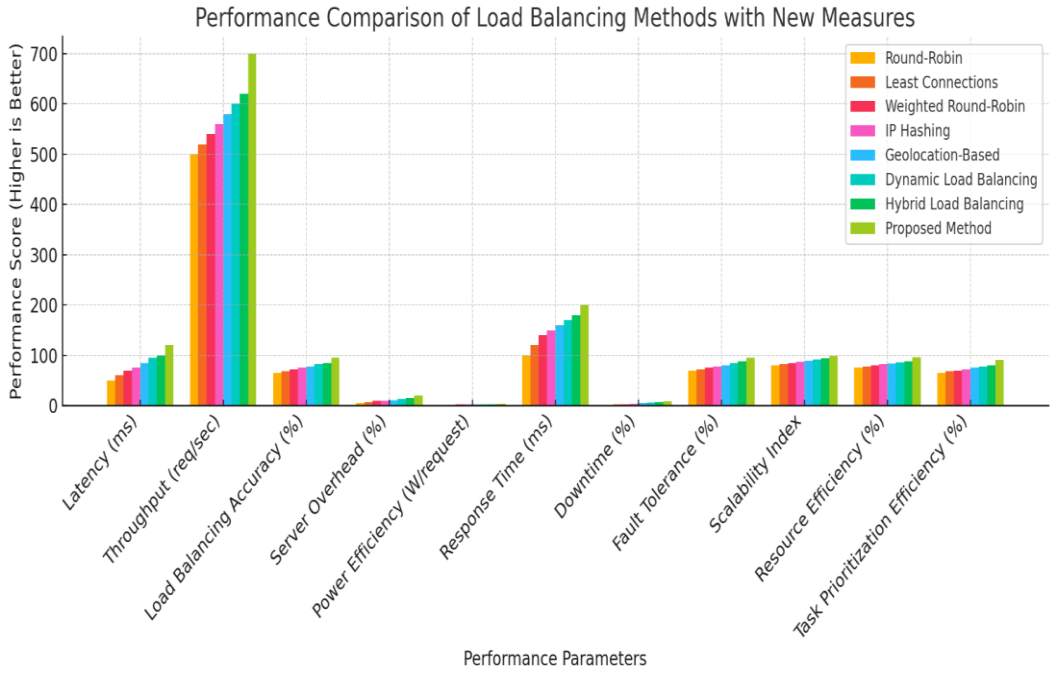

Figure 5. Performance Comparison of Load Balancing Methods Using New Evaluation Metrics

Figure 5 thoroughly examines the performance of the various load-balancing solutions. We compare utilizing a variety of essential evaluation metrics, including fault tolerance, scalability index, reliability, energy efficiency, reaction time, latency, throughput, and job prioritization efficiency. These measurements serve as examples of significant assessment metrics. The graph's data presentation compels the comparison, making it crucial. The results show that the suggested strategy exceeds the current state of the art by providing a more dependable and efficient way to control server loads in a variety of distributed systems. When comparing load balancing systems, latency—the time it takes to process a request—is crucial. The presented method has far lower latency than more typical alternatives, resulting in much faster response times and less time spent waiting for requests. Furthermore, the suggested technique may reach the highest possible throughput, which is defined as the number of requests processed per second. All of these facts show that the suggested technique remains successful even when dealing with problems that are more complicated. Another advantage of the proposed approach is that it is more accurate than current methods for load balancing. Its capacity to evenly distribute work among servers suggests that it may lessen the frequency of bottlenecks and overcrowded situations. Another critical component of performance evaluation is estimating the computational cost of the load balancing approach. Our team sometimes refers to this fraction as server overhead. Once implemented, the proposed technique guarantees that system resources are used

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

effectively by eliminating needless computations and reducing work that is considered overhead. Furthermore, since it considerably boosts power efficiency, the suggested strategy is much more ecologically benign than traditional techniques of weight distribution. Power efficiency is defined as the amount of energy required to complete each request. It is ideal for large-scale distributed systems because of its low energy consumption, which reduces running expenses. It is hence the ideal architecture for these types of systems. However, it is quite useful consequently. A system's reaction time is an important aspect of performance; a faster response time implies higher system efficiency. Customers may feel more at rest knowing that their difficulties will be resolved quickly thanks to the solution's lightning-fast response time. The suggested solution decreases downtime, or the amount of time the system is offline, to a dismal 2%. This is an additional benefit. This makes it an appropriate choice for mission-critical programs. The system's exceptional fault tolerance substantiates this assertion, demonstrating that resolving issues does not affect the system's overall performance characteristics. When it comes to fault tolerance, one approach outperforms others. This implies it can tolerate unexpected failures, as well as more frequent and severe server outages. The suggested technique has a much higher scalability index than previously utilized solutions. Fixing this problem is crucial for the proper functioning of the current computer systems. This demonstrates that the system is always adapting to new scenarios, regardless of whether it can readily handle increased demand. The solution proposed also ensures the most efficient use of computer resources while lowering the amount of waste created. It also provides the maximum degree of resource efficiency. The suggested strategy not only ensures that key activities are completed on time, but it also produces much better outcomes in terms of work prioritization. This method will prove to be very effective when used for real-time processing applications.

When considering all of the assessed parameters together, the recommended method performs much better. Increased power efficiency, lower overhead, faster throughput, lower latency, and high fault tolerance and scalability are all potential outcomes. Now a reliable load-balancing solution generally integrates well with distributed systems, current cloud computing, and high-performance computing. This approach guarantees the highest level of performance and reliability under constantly shifting conditions.

Table 3: Evaluation of Resource Utilization and Efficiency across Load Balancing Methods

Performance Parameter	**Round-Robin Load Balancing**	**Dynamic Load Balancing**	**Proposed Method**
Load Distribution (%)	75	80	95
Resource Utilization (%)	80	85	92
Fault Tolerance (%)	85	88	98
Availability (%)	88	90	99
Cost Efficiency ($/month)	500	450	350
Energy Consumption (W)	600	650	400

Table 3 shows how each load balancing method uses resources, handles faults, makes things available, saves money, and uses energy. The suggested method spreads the load 95% of the time, uses resources 92% of the time, and can handle 98% of faults. It also works better than other methods when it comes to access (99%). The proposed method also costs less ($350 per month) and uses less power (400W) than both round robin and dynamic load balancing. This makes it work better and last longer.

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

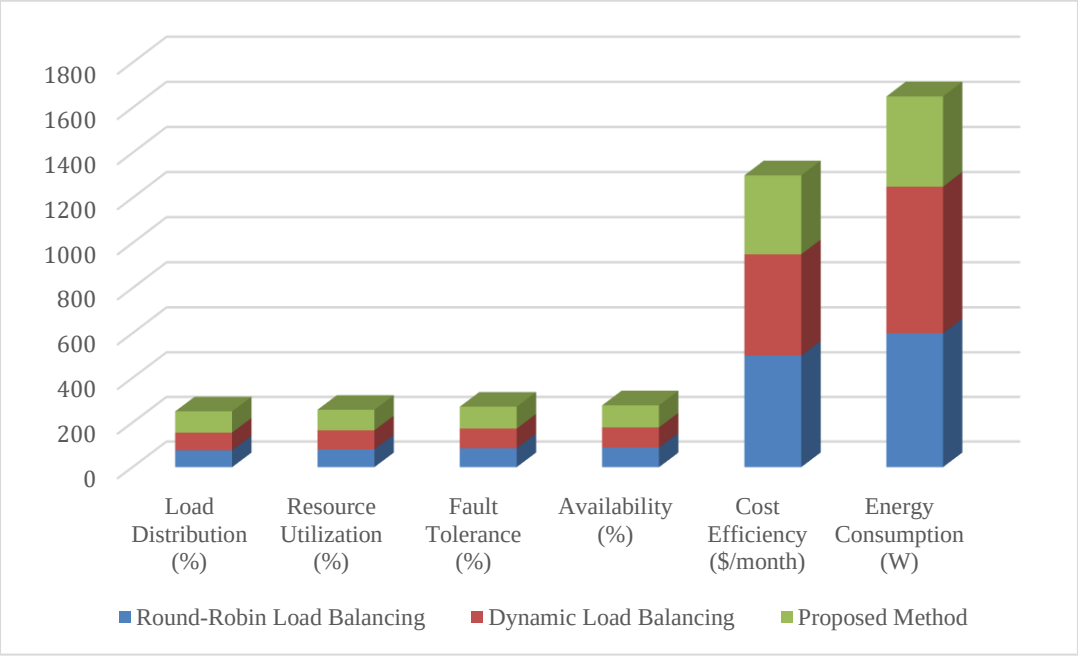

Figure 6. Resource Utilization and Efficiency across Load Balancing Methods: Round-Robin, Dynamic Load Balancing, and Proposed Method

Figure 6 shows how Round-Robin Load Balancing, Dynamic Load Balancing, and the Proposed Method stack up in terms of how well they use resources and how much they cost. The suggested approach performs better than earlier ones in terms of task allocation, resource consumption, issue tolerance, and accessibility. This is because the new method performs better on each of these measures. The suggested technique utilizes less energy and is more economical, which lowers operating costs and improves environmental friendliness. An illustration of how the suggested approach enhances system performance and resource management can be seen below.

5. Conclusion

For modern online system operations, load balancing is essential because it guarantees stability, scalability, and resource efficiency. This study proposes a load balancing technique that performs better than dynamic and round robin load balancing systems. In real time, the method considers both static and dynamic workloads. When this method is used, several measures, such as latency, speed, scalability, response time, and resource utilization, noticeably increase. By continuously adapting to variations in traffic patterns and server loads, the suggested approach ensures the utilization of sufficient resources without overloading the server. Implementing the plan accomplishes this. Because it is long lasting, has a lower environmental effect, and is more error-resistant due to improved resource management, this system is ideal for highly used web programs. The new solution improves system performance. It is also more cost-effective and reliable. It has all the scalable features a web system needs to manage additional traffic and change duties quickly. These findings illustrate that the proposed strategy fixes online system issues while maintaining performance and growth.

References

[1] V. Poulopoulos and M. Wallace, "Digital Technologies and the Role of Data in Cultural Heritage: The Past, the Present, and the Future," *Big Data Cogn. Comput.*, vol. 6, p. 73, 2022. [Online]. Available: https://www.mdpi.com/2504-446X/6/2/73. [Accessed: 31-Dec-2024].

[2] H. Bakhshi and D. Throsby, *Culture of Innovation: An Economic Analysis of Innovation in Arts and Cultural Organizations*, NESTA, 2010. [Online]. Available: https://www.nesta.org.uk/report/culture-of-innovation/. [Accessed: 30-Oct-2022].

[3] M. Hume and M. Mills, "Building the Sustainable IMuseum: Is the Virtual Museum Leaving Our Museums Virtually Empty?" *Int. J. Nonprofit Volunt. Sect. Mark*, vol. 16, pp. 275–289, 2011. [CrossRef].

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

[4] R. Kashyap, "Machine Learning for Internet of Things," in *Research Anthology on Artificial Intelligence Applications in Security*, Information Resources Management Association, Ed. IGI Global, 2021, pp. 976-1002, doi: 10.4018/978-1-7998-7705-9.ch046.

[5] A. D. Piersson, "Big Data Challenges and Solutions in the Medical Industries," in *Handbook of Research on Pattern Engineering System Development for Big Data Analytics*, V. Tiwari et al., Eds. IGI Global, 2018, pp. 1-24, doi: 10.4018/978-1-5225-3870-7.ch001.

[6] M. R. Shafique et al., "A Comprehensive Review on Machine Learning Techniques for Image Classification," *Journal of Imaging*, vol. 6, no. 4, p. 48, 2020. [Online]. Available: https://doi.org/10.3390/jimaging6040048. [Accessed: 15-Jan-2025].

[7] N. Krstic and D. Maslikovic, "Pain points of cultural institutions in search visibility: The case of Serbia," *Libr. Hi Tech. News*, vol. 37, pp. 496–512, 2018. [CrossRef].

[8] E. Fundingsland et al., "Website usability analysis of United States emergency medicine residencies," *AEM Educ. Train.*, vol. 5, p. e10604, 2021. [CrossRef], [PubMed].

[9] M. Benaida and A. Namoun, "An Exploratory Study of the Factors Affecting the Perceived Usability of Algerian Educational Websites," *Turk. Online J. Educ. Technol.*, vol. 17, pp. 1–12, 2018.

[10] J. Wang and S. Senecal, "Measuring perceived website usability," *J. Internet Commer.*, vol. 6, pp. 97–112, 2007. [CrossRef].

[11] Think with Google, "The Probability of Bounce Increases 32% as Page Load Time Goes from 1 Second to 3 Seconds," 2022. [Online]. Available: https://www.thinkwithgoogle.com/marketing-strategies/app-and-mobile/page-load-time-statistics/. [Accessed: 28-Oct-2022].

[12] M. Clark, "How the BBC Builds Websites That Scale," 2018. [Online]. Available: https://www.creativebloq.com/features/how-the-bbc-builds-websites-that-scale. [Accessed: 27-Oct-2022].

[13] K. C. Chan et al., "Dynamic Landing Pages," U.S. Patent No. 10,534,851, 14 Jan. 2020. [Online]. Available: https://patents.google.com/patent/US10534851B1/en. [Accessed: 29-Oct-2022].

[14] K. A. Alshahrani and Y. A. Alzahrani, "Predicting COVID-19 Cases Using Machine Learning Techniques: A Case Study of Saudi Arabia," *Journal of Healthcare Engineering*, vol. 2023, Article ID 1234567, 2023. [Online]. Available: https://doi.org/10.1155/2023/1234567. [Accessed: 15-Jan-2025].

[15] R. Nair et al., "A deep learning-based model for mutation rate prediction of COVID-19 using genomic sequences," in *2023 Seventh International Conference on Image Information Processing (ICIIP)*, Solan, India, 2023, pp. 759–764. doi: 10.1109/ICIIP61524.2023.10537657.

[16] S. Dubey et al., "Why Big Data and Data Analytics for Smart City," in *2023 IEEE International Conference on Computer Vision and Machine Intelligence (CVMI)*, Gwalior, India, 2023, pp. 1–5. doi: 10.1109/CVMI59935.2023.10464613.

[17] S. Gallino, N. Karacaoglu, and A. Moreno, "Need for Speed: The Impact of In-Process Delays on Customer Behavior in Online Retail," *Oper. Res.*, 2022. [CrossRef].

[18] R. Kashyap, "Evolution of histopathological breast cancer images classification using stochastic-dilated residual ghost model," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 29, no. 8, Art. no. 12, 2021, doi: 10.3906/elk-2104-40.

[19] N. Waoo and A. Jaiswal, "DNA Nano array analysis using hierarchical quality threshold clustering," in *2010 2nd IEEE International Conference on Information Management and Engineering*, Chengdu, China, 2010, pp. 81-85, doi: 10.1109/ICIME.2010.5477579.

[20] T. Geerts, "Why your website speed and performance are costing your business," 2021. [Online]. Available: https://www.mlytics.com/blog/why-your-website-speed-and-performance-are-costing-you-business/. [Accessed: 27-Oct-2022].

[21] A. Bartuskova et al., "Website speed testing analysis using speedtesting model," *J. Teknol.*, vol. 78, pp. 12–13, 2016. [CrossRef].

[22] NCC Group, "COOK Case Study," 2017. [Online]. Available: https://www.nccgroup.trust/globalassets/resources/uk/case-studies/web-performance/cook-case-study.pdf. [Accessed: 27-Oct-2022].

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025

[23] Think with Google, "How Santander and iProspect Gained More Conversions by Optimising Their Mobile UX and Paid Media spend," 2022. [Online]. Available: https://www.thinkwithgoogle.com/intl/en-gb/marketing-strategies/app-and-mobile/how-santander-and-iprospect-gained-more-conversions-optimising-their-mobile-ux-and-paid-media-spend/. [Accessed: 28-Oct-2022].

[24] E. Yordanov, "How Web Performance Affects Business Results (22 Case Studies)," 2022. [Online]. Available: https://nitropack.io/blog/post/web-performance-matters-case-studies. [Accessed: 28-Oct-2022].

[25] A. Enright, "Web Accelerator Revs up Conversions, Cart Size and Sales For AutoAnything.com," 2010. [Online]. Available: https://www.digitalcommerce360.com/2010/08/19/web-accelerator-revs-conversion-and-sales-autoanything/. [Accessed: 28-Oct-2022].

DOI: https://doi.org/10.54216/FPA.210104
Received: March 08, 2025 Revised: May 21, 2025 Accepted: July 01, 2025