Kufa Journal of Engineering
Vol. 17, No. 2, April 2026, P.P. 113-133
Article history: Received 1 March 2025, last revised 8 April 2025, accepted 18 April 2025

A PROBABILISTIC APPROACH TO EFFICIENTLY AND SUSTAINABLY LOAD BALANCING FOR CLOUD COMPUTING OPTIMIZATION

Nuha H. Alameedi[1] and Mahdi S. Almhanna[2]

[1] Software Department, College of Information Technology, University of Babylon, Babylon, Iraq, Email:nuha.hussein.hadi@gmail.com.

[2] Information Network Department, College of Information Technology, University of Babylon, Babylon, Iraq, Email:mahdi.almhanna@uobabylon.edu.iq.

https://doi.org/10.30572/2018/KJE/170208

ABSTRACT

In order to lower environmental effect in cloud computing, sustainability calls for effective resource management. Often ignoring server heterogeneity, traditional load balancing techniques include the Power-of- d paradigm result in poor performance and significant energy consumption. This work presents an adaptive, energy-aware load balancing method dynamically distributing workloads depending on several server parameters: bandwidth, memory, latency, CPU speed, connection count, and cost. Task allocation is guided by a probabilistic model derived from Ant Colony Optimization (ACO). Furthermore, advanced metaheuristics like both Particle Swarm Optimization (PSO) and Grey Wolf Optimization (GWO) as well as conventional algorithms like Round Robin and Least Connections are greatly outperformed by the suggested method according to experimental data. It reduces response times by 18%, uses resources up to 22%, and runs 15% less energy. This method aligns performance goals with environmental responsibility and presents a sustainable and effective solution for cloud infrastructure.

KEYWORDS

Distributed systems, Resource allocation, Cloud computing, Sustainability, Load balancing.

1. INTRODUCTION

Distributed computing is the practice of interconnecting multiple computer servers over a network to form a cluster; enabling data sharing and coordinated processing power; Such a cluster is termed a "distributed system. Distributed computing provides benefits in scalability; performance; resilience, and cost-effectiveness. The proliferation of cloud Computing options and providers has further enhanced the accessibility of distributed computing. While cloud computing instances' do not inherently facilitate distributed computing, there exists a variety of distributed computing software that can levirate the readily available computing resources in the cloud (Hazelcast; 2025 and Ahmed; M. F.; 2025 and Shamsa; M., Al-Shathr; B. and Al-Attar, T.; 2021). Cloud computing has emerged as a revolutionary technology, providing on-demand access to a pool of diverse virtualized; resources accessible through the internet. The dynamic nature of the cloud environment presents challenges in managing these dynamic recourse's effectively. Cloud-based applications must deliver exceptional performance, accommodate dynamic workloads, and ensure continuous availability to meet the demands of the modern landscape. Companies leverage network computing to harness collective, unused computing resources in order to tackle complex problems or large-scale projects that would be challenging to implement on a single machine (Saif, M. A. N., Niranjan, S. K. and Al-Ariki, H. D. E., 2021 and Alaa, R., Hussein, E. and Al-libawy, H., 2024). By definition, balancing network task loads is a technique that improves the performance, scalability, expected reliability, and optimization of distributed systems. This will ensure the distribution of workloads across several servers to stave off overloading of any single server. An added benefit will include improvement in response time through increased throughput with a certain guarantee for fault tolerance. Load balancing is vital to guaranteeing system operation and a good user experience during high traffic. The designation of conventional load balancing mechanisms, such as round-robin, least-connected, and weighted distribution, aims to fairing load on web traffic. Those tech-eques are viable on horizontal scaling via incorporating most servers or redistributing corresponding workloads. Often; they do not consider server heterogeneities affecting processing power; storage, bandwidth, response- time, and cost. All of these shortcomings s fixed when advanced load balancing techniques utilize real-time data server conditions for perform dynamic; scheduling of computations. Thes increased uptore of cloud-blazed applications and services have brought with tithe ever-increasing emphasis over dynamic load balancing. Load bouncers address the Issus of performance degradation and resource contention through optimal workload distribution, response time minimization, and resource optimization; Load- balancing also ensures that all servers operate normalcy without

being under and over-utilized. In addition; load balancers increase fault tolerance by rerouting traffic to healthy servers in the event of a failure, king applications up and running. Such fail ever capabilities are essential for systems requiring high up-time and consistent performance. Ultimately, load balancing is the very backbone underpinning high-performance, scalable cloud-based applications that meet the needs of the modern user and today's business workflows (Shahakar; M. and Patil; L.; 2023 and Wagh;ulde, P.; 2025). Load balancing is the process thigh which incoming network traffic is distributed across several servers and resources. Thes paravents a server frame being overloaded too much traffic, or provides optimum application responsiveness; or availability. In the context of distributed systems, this could mean; distributing tasks and computations across multiple nodes (Abu Almash; F. S.; Nsaif, A. H. and Jabor, M. S.; 2024 and He, R. and Tan, X.; 2018). A load balancing algorithm that dynamically adjusts weights blazed over server performance metrics; combined vs real connections data for real-time. the dynamic feedback mechanism gave better representation of server loads and improves request distribution; (Ibrahim; I. M. et al.; 2021). In the context of web server systems, distributed architectures that schedule client requests among multiple server nodes have been explored to provide scalability and availability. Such systems are designed to distribute incasing requests on servers, ensuring that performance is optimized (Gardner; K.; Jaleel, J. A.; Wickeham, A. and Doroudi, S.; 2020). Recent developments include heterogeneity-aware dispatching policies that use server speed information in selecting the servers for querying or dispatching jobs. These policies achieve the maximum stability, outperforming traditional dispatching- rules in mean response time (Khallouli; W. and Huang; J., 2022). In real situations, server heterogeneity can't be avoided, making it extremely important too properly implements specification load- balancing of leverage the increasingly accelerated architecture. Wrong consideration of this leads to the imbalance of resource utilization, increased response times, and diminished system integrity (Yu, J., Jiang, J. and Ye, W., 2024). This indicates the necessity for the development of dynamic load-balancing algorithms fully compatible with heterogeneous environments. Other existing load-balancing methods, the most well-known being the "power-of-d" method, worked successfully to reduce queue delays and distribute workloads. Unfortunately, they are based on the assumption of uniformity on the part of the servers, ignoring crucial server parameter differences. As a consequence, they could not provide the best possible solution in real heterogeneous systems. Additionally, static policies would not be applicable due to changes occurring in the system over time. Such changes could be either general, such as network load or state of the servers along with other, application- or system-specific conditions which vary. This lack of

adaptability exacerbates inefficiencies and can result in higher costs and energy consumption (Kadhim, A. J., 2024). To address these challenges, there is a pressing need for dynamic and adaptive algorithms that can intelligently distribute workloads while considering the diverse attributes of servers. Below are some commonly used requests distribution algorithms on servers, all of which have their own limitations and disadvantages.

1.1. Round Robin

RR distributes requests in a round-robin manner, distributing the load evenly to the servers regardless of the configured capacity of the servers. Servers with low capacity perform poorly under the same load. The RR load balancing algorithm is further improved by assigning priorities to the servers. To distribute incoming requests, the server with the highest priority is selected instead of the configured capacity. A static request distribution algorithm is proposed in which requests are randomly assigned to servers (Mayur, S. and Chaudhary, N., 2019 and Anun, K. H. and Almhanna, M. S., 2021). The algorithm is sequential in selecting servers.

1.2. Weighted round-robin load balancing

The improved weighted round-robin (WRR) scheduling algorithm is used to actually distribute responses to Web client requests and achieve dynamic load balancing of Web server clusters. It is simple and efficient, requires fewer system resources, and ensures high-speed packet forwarding through adjustable load balancing (Adewojo, A. A. and Bass, J. M., 2023 and Rathod, A. S., Nainani, J. and Nishad, T., 2020). The algorithm can neither automatically optimize response time nor avoid path congestion. In addition, it is generally considered to be stateful, so it incurs additional overhead compared to stateless algorithms. Therefore, it requires additional memory and computing resources to store and update information, incurs synchronization overhead, and the decision logic is more complex, which may affect the scalability of the system.

1.3. Least connections

A Least-connection scheduling algorithm elegantly orchestrates the distribution of requests, channeling more tasks to real servers by linking active ones with a lesser share of connections. Furthermore, this algorithm operates under the premise that every backend server possesses identical computational prowess (Radtke, T. and Ababei, C., 2022). In its core, the least-connection algorithm artfully balances the load by assessing enumerate of connections each server has managed.

1.4. Load balancing with latency awareness

The algorithm latency-aware load balancing directs requests to the server with the lowest latency or response time (Nguyen, V., Grinnemo, K., Taheri, J. and Brunstrom, A., 2021).

Response time is defined as the total time taken by a server to process incoming requests and send back a response, referred to as the "echo." However; this approach comes with challenges, particularly were dealing with a large number for servers. It is computationally complex and introduces significant overhead; as it necessitates frequents measurements of delay between the servers and the load balancer. Additionally; it consumes resources because; the continuous monitoring of server response- times and tph routing decisions that depend or this information. Many load-balancing algorithms aim of achieve specific- objectives but always focus on particular factors while overlooking others. For example; the algorithm least connection routes traffic to the server vs the fewest active connections; disregarding critical aspects such as; cost, time of response; bandwidth, or available memory. Thes focus may not always yield optimal performance. An objective is to evaluate a broad- spectrum of factors such as cost; bandwidth, response time, and memory availability to select the most suitable path. This approach ensures the path chosen are not only optimal but also comprehensive; addressing for all critical aspects. basic factors include; bandwidth, directly- impacts data transmission speed. Memory availability, a lock on free memory can disrupt continuous service delivery. CPU Speed; influences the processing capability of a server, determining how a quickly tasks can be executed. Cost; Affects the feasibility to working with a server; either through monetary expenditure and energy consumption for data transmission (Almhanna; M., Al-Turaihi; F. and Murshedi, T., 2023 and Almuttairi; R. M., Wankar, R., Negi, A. and Chillarige; R. R., 2010). Thes multifactorial consideration are essential to ensure efficient or sustainable system performance.

2. LITERATURE REVIEW

Previous studies have explored various strategies for resource allocation and load balancing in heterogeneous systems. Recent studies have emphasized feedback-based dynamic load balancing algorithms to improve resource allocation. (Souravlas, S., Anastasiadou, S. D., Tantalaki, N. and Katsavounis, S., 2022). Proposes a fair task load balancing strategy using Markov process modeling to enhance cloud computing performance. This strategy Fair distribution of tasks among VMs based on processing capacity, Improved metrics: makespan, average response time, resource utilization, Outperforms existing algorithms in experimental results. (Jungum, N. V., Mohamudally, N. & Nissanke, N., 2020). The AWLC scheduling algorithm is proposed to improve load balancing and system efficiency in cloud computing infrastructure by dynamically recalculating weights and considering host attributes such as CPU idle rate and memory idle rate. he results show that the AWLC scheduling algorithm has a better

load balancing degree and system efficiency compared to the LC and WLC scheduling algorithms in all three scenarios. (Liu, B., Chang, J., Xiao, L., Qin, G., Wei, B. & Huo, Z., 2019). This paper proposes a dynamic and distributed load balancing algorithm, DDLB, to solve the load imbalance problem in distributed file systems, which can effectively balance the IO load between data servers and improve data access performance. The experimental results show that DDLB can effectively improve the access performance of applications, with a response efficiency improvement of 12.86% and 7.21% compared to RLB and DCLB algorithms, respectively. (Joshi, N. A., 2022), The paper presents a novel load balancing technique. It aims to prevent starvation of virtual machines. The technique considers priority levels of virtual machines. I implementation reduces waiting time for overloaded machines. The results indicate optimize resource- utilization in Cloud environments. (Sefati; S. S.; Mousavinasab; M. & Farkhady, R. Z., 2022). Thes research employs the Grey of Wolf Optimization (GWO) algorithm, focusing on resource reliability; to address this problem. By identifying server statuses and optimizing thresholds; the algorithm GWO enhances load- balancing effectiveness. simulation results on CloudSim demonstrate superior Cost and response of the time efficiencies compared to existing techniques; reinforcing the efficacy for this approach in cloud of environments. (Khan, M. I. and Sharma, K.; 2024), this proposes a nature-inspired method using algorithms PSO and NIVM-PSO. The model adapts for changing workloads effectively. It aims fox reduce reaction time and enhance resource utilization. The Experimental of results show significant of the performance improvements upper standard methods. (Kruekaew; B. & Kimpan, W., 2022), It proposes a multi-objective scheduling methods using the algorithm MOABCQ. The model combines Artificial Bee Colony and also Q-learning algorithms. Performance of this is to evaluated against existing scheduling algorithms. Results show optimization of makespan; cost, and resource of utilization. (Pradhan; A. and Bisoy, S. K.;2022), This paper is proposing a load balancing method using modified the PSO. It aims to minimizing tph makespan and maximize resource utilization. The technique is too implemented using a CloudSim simulator. Simulation results show improved performance over existing methods. An algorithm LBMPSO optimally schedules tasks for available virtual machines. (Mishra; K. & Majhi, S. K.; 2021), The study proposes an BSO-LB algorithm to load Balancing. It addresses NP-hard task- scheduling in Cloud computing. The algorithm mimics the bird flocking behavior of task allocation. It is enhancing system performance via lower response time. Experimental results indicate an improved resource utilization or reduced makespan.

To strong demonstrate the advancements of outing ACO-based load balancing algorithm, we conduct a systematic comparison with state-of-the-art methods; highlighting a quantifiable

improvement in performance; adaptability, and sustainability; as see in the Table 1.

Table 1. Evolutionary Advancements Over Existing Algorithms

Algorithm	Key Contribution	Shortcoming	Our Improvement	Quantifiable Gain
Round Robin (RR)	Simple and fair task distribution.	Static; ignores server heterogeneity.	Dynamic weight adjustment using ACO heuristics.	22% resource utilization (Results, Table 4)
Least Connections (LC)	Reduces server congestion by choosing least-loaded nodes.	Ignores multi-dimensional server factors (e.g., CPU, memory).	Scoring model using multi-factor evaluation (Eq. 5).	18% average response time (Results, Fig. 8)
GWO (Sefati et al., 2022)	Enhances fault tolerance and reliability.	Focuses on a single metric (reliability).	Optimization across six performance metrics.	12% throughput (Results, Table 4)
PSO (Pradhan & Bisoy, 2022)	Minimizes makespan via swarm-based task scheduling.	High computational cost and slow convergence.	Low-complexity probabilistic model (Eq. 3).	30% faster decision-making
Proposed ACO-based method	Integrates performance, energy, cost, and sustainability.	Sensitive to pheromone parameters.	Self-adjusting mechanism through adaptive pheromone feedback (Eq. 2).	15% energy consumption (Results, Fig. 9)

3. METHODOLOGY

The search objective to enhance the network routing by selecting best path depending on metrics example a memory availability; CPU speed; bandwidth, response time; and cost, therefore; improving performance and sustainability on cloud Computing environments. To aid the ant colony algorithm; the proposed methodology introduces a dynamic load-Balancing framework. This framework employs a weighted probability model that integrates server-specific parameters; including connection Count, bandwidth, memory; response time, or cpu-speed, with weights assignments according to the relative significance. through dynamically computations server scores performance and probabilities; A model ensures efficient workload distribution;reducing response- times also enhancing overall system of performance (Murshedi; T. A. et al.; 2024 and Munther;N. H. & Jasim, M. N.; 2021). The proposed algorithm formulates the mathematical optimized model to the dynamically determine the optimal path inside a weighted graph structure. Path the selection is governed by more server parameters, including; (Muniyappa; V. and Hattibelagal, C., 2023 & Chomsiri, T. and Pansa, D., 2018)

1. Directly Proportional Factors**:**

- Bandwidth (BW); Eq. 1 higher bandwidth the improves data transfer efficiency.

- CPU Speed; Eq. 2, faster processing capability lower task completion time.
- Available Memory (M); Eq.3, Adequate memory ensures uninterrupted service of execution.

2. Inversely Proportional Factors**:**

- Cost (C) Eq. 4: Lower operational costs are prioritized.
- Number of Connections (L) Eq.5: Fewer active connections indicate lower server congestion.
- Response Time (R) Eq 6: Shorter latency reflects higher server health.

The relationships are expressed as:

$$\text{Selection of Path} \propto \text{Bandwidth} \tag{1}$$

$$\text{Selection of Path} \propto \text{CPU Speed} \tag{2}$$

$$\text{Selection of Path} \propto \text{Available Memory} \tag{3}$$

$$\text{Selection of Path} \propto \frac{1}{Cost} \tag{4}$$

$$\text{Selection of Path} \propto \frac{1}{\text{Number of Connections}} \tag{5}$$

$$\text{Selection of Path} \propto \frac{1}{\text{Response Time}} \tag{6}$$

3.1. Parameter Weights and Server Heterogeneity

To calculation server heterogeneity; parameter weights are assigned through a flexible administrator-driven approach that reflects both the operational priorities and performance of requirements. For The weights is initially configured through the system administrators based the unique characteristics of target cloud g environment and may be dynamically adjusted via the algorithm's management interface to adapt of changing operational needs. In the Table 2 shows our experimental default values; which serve as a starting point for environment-specific customization. Weight assignment rationale; Connection count (WL = 0.20); prioritizes servers with fewer active connections for reduce Congestion. THE Bandwidth (WB = 0.35), this given upper weight in data-intensive environments to facilitate faster transfers. For to Memory (WM = 0.10); ensures adequate memory availability for workload processing. The Response of Time (WR = 0.20); critical for latency-sensitive applications. CPU- Speed (WCPU = 0.10); important for computation-heavy tasks. Cost (WC = 0.05); weight may be increased when the budget optimization show priority. System administrators can update these dynamic weights to Increase Cost weight (WC) when financial constraints emerge; adjust time of response weight (WR) for latency-critical applications; rebalance weights based on changing workload patterns

3.2. Cost Calculation

Firstly, compute the cost of each server to incorporate cost considerations; a cost of weight (WC) is calculated for each server using the formula: (Murshedi; T. A. et al.;2024).

$$\text{Cost weight} = \text{Cost per File} \times \text{WC} \tag{7}$$

This ensures that workload distribution not only optimizes performance but also considers cost-effectiveness. Servers with lower cost weights are prioritized, provided their Score values remain competitive.

3.3. Score Calculation

Secondly, compute the Score for each server. The Score metric evaluates each server's suitability for workload allocation by combining its weighted parameters. The equation for Score is as follows:

$$\text{Score}=\text{WL} \times \frac{1}{L}+\text{WB} \times \text{B}+ \text{WM} \times \text{M}+\text{WR} \times \frac{1}{R}+ \text{WCPU} \times \text{CPU} + \text{WC} \times \frac{1}{C} \tag{8}$$

Where WL+ WB + WM + WR + WCPU+ WC =1 to ensure proportional representation of parameters. Number of connections, bandwidth, memory, response time, CPU speed, and cost were all given different percentages of importance by the administrator during installation. In that order, the least connection, least bandwidth, least memory, least response, fastest CPU, and least cost are L, B, M, R, CPU and C respectively. L, B, M, and R, CPU and cost are the corresponding server parameters. The calculation ensures that servers with favourable parameter values (e.g., lower connection count, higher bandwidth) achieve higher Scores, making them more likely to be selected.

3.4. Probability-Based Server Selection

The final step in the methodology involves determining the probability of each server being selected based on its Score and cost weight. The selection probability for a server is computed using the equation: (Murshedi, T. A. et al., 2024).

$$P(xy)= \left(\frac{Score\_x}{Cost\_x}\right)/\left(\frac{Score\_x}{Cost\_x}+ \left[\frac{Score\_y}{Cost\_y} \times \frac{Score\_z}{Cost\_z} \times \frac{Score\_k}{Cost\_k} \times \frac{Score\_s}{Cost\_s}\right]\right) \tag{9}$$

Were, $Score\_x$, $Score\_y$, $Score\_z$, $Score\_k$, $Score\_s$. are Score value of the server *x, y, z, k, s*. and $Cost\_x$, $Cost\_y$, $Cost\_z$, $Cost\_k$, $Cost\_s$ are Cost weight of the server x, y, z, k, s. and x, y, z, k, s is Total number of servers. By integrating these steps, the proposed approach dynamically adapts to changing system conditions, optimizing task allocation in real time while balancing performance, cost, and energy efficiency.

4. THE ALGORITHM

The proposed algorithm is inspired by the Ant Colony Optimization (ACO) technique. It

simulates the way ants deposit pheromones on paths based on connection performance, bandwidth, memory availability, cost, and response time. In this model, paths with higher pheromone levels (higher support values) are more likely to be chosen. Eq.8 governs the pheromone update process, ensuring that paths with better performance metrics are selected more frequently, and this decision-making process adapts dynamically to the system state. A flowchart of the proposed algorithm is shown in Fig. 1, and Table 3 compares it with traditional load balancing algorithms such as Round Robin (RR), Least Connection (LC), Latency-aware strategies and Weighted Round Robin (WRR).

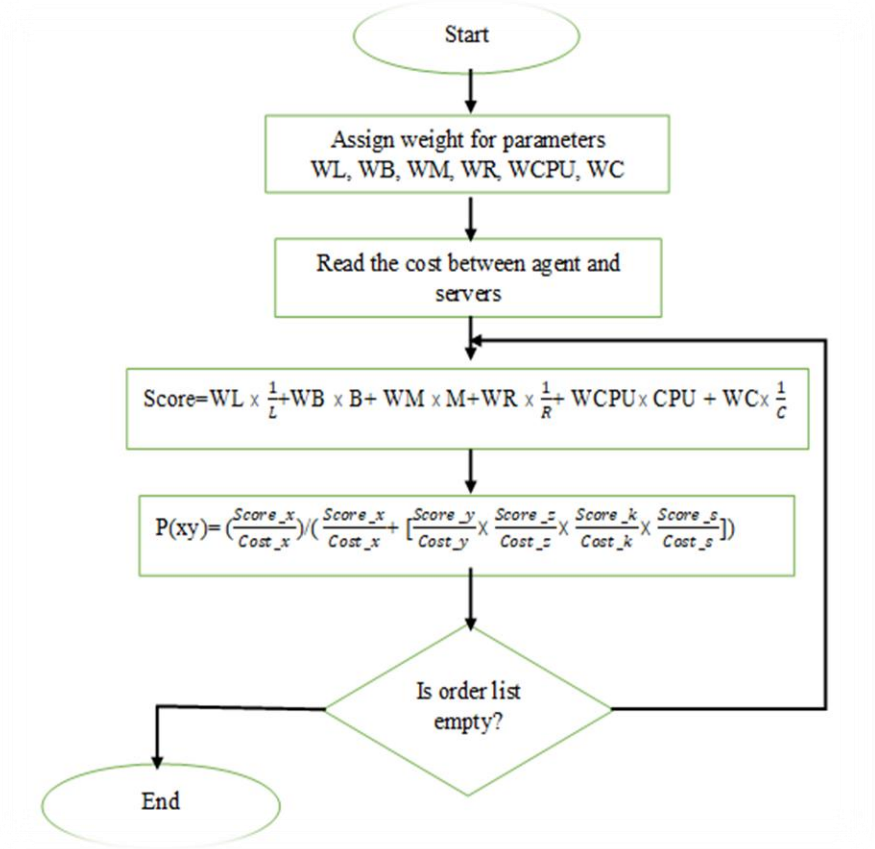

Fig. 1. Flowchart of the proposed algorithm

The steps of the suggested algorithm are defined as follows in Algorithm 1:

Algorithm 1. ACO-based Load Balancing Algorithm

Input: Number of connections, Bandwidth, CPU processing speed, Memory allocation, Cost, Latency per server.

Output: Optimal path probabilities.

Step 1: Initialization Phase

1. Start the algorithm.

2. Assign weights to each server for all input parameters (CPU, memory, bandwidth, cost, latency, and number of connections) using Table 2.

Step 2: Computation Phase

3. Calculate total agent costs and the associated server costs using Eq.7.

4. Evaluate route options:

If multiple paths exist to the target node:

Select the path with the lowest idleness and highest server responsiveness.

Step 3: Decision-Making Phase

5. Score computation:

Use Eq.8 to calculate a support score for each path.

6. Path filtering:

Apply Eq.9 to exclude paths with the lowest probability of success.

7. Dispatch:

Send the request to the server located in the region with the highest response probability.

Step 4: Finalization Phase

8. End the algorithm.

Table 3. Proposed Algorithm Compared to Conventional Load Balancing Algorithms

Feature	Proposed Algorithm	Round Robin (RR)	Weighted Round Robin (WRR)	Least Connection (LC)	Latency-aware
Factors Considered	CPU, memory, bandwidth, latency, cost, connections	None	Static weights	Connections	Latency
Dynamic Adjustment	✓	X	X	✓	✓
Load Distribution	Based on Eq.8 and 9	Even	Weighted	Connection-based	Latency-based
Complexity	Moderate	Low	Medium	Medium	High
Flexibility	✓	X	Limited	Limited	Limited
Optimization for Multiple Metrics	✓	X	X	X	X
Resource Utilization	✓	Low	Medium	Medium	Low
Network Overhead	Medium	Low	Low	Medium	High
Scalability	Moderate	Low	Low	Moderate	Moderate

5. RESULTS AND DISCUSSION

5.1. Study Case setup

The proposed methodology was evaluated using a distributed system environment comprising five heterogeneous servers. Each server exhibited distinct characteristics such as response time, bandwidth, CPU speed, memory size, connection count, and cost per file, as detailed in Tables 2 and 4.

5.2. Initial Results

In the first round of workload allocation: Cost, Score, and Probability-Based Server Selection

Calculation as Eq.7,8,9 respectively. As show in Table 5, Table 6. This outcome as Fig. 2 and 3, demonstrates the algorithm's ability to prioritize servers with higher suitability based on heterogeneous parameters. Next, use Eq. 9 to calculate the probability of selecting each server:

Table 2. The parameter of weights.

Weights	The ratio
Weights of the least connection (WL)	0.20
Bandwidth weight (WB)	0.35
Memory weight (WM)	0.10
Response time weight (WR)	0.20
CPU speed weight (WCPU)	0.10
Cost weight (WC)	0.05

Table 4. Initial server parameters.

IPs	Response Time (ms)	Bandwidth (Gbps)	CPU Speed (GHz)	Memory Size (TB)	Connections Count	Cost per File ($)	Cost weight	Score (Round1)
142.247.178.251	247	99	3.32	13.76	8760	164	8	36.36
116.72.249.100	258	99	3.59	11.62	3448	368	18	36.17
73.47.50.114	214	100	2.73	7.04	7629	164	8	35.98
47.238.97.123	217	100	4.11	5.31	4910	325	16	35.94
73.194.252.40	224	98	3.68	6.07	7004	500	25	35.28

Table 5. Cost Weights and Score Calculation for Each Server.

Server	Cost per file X WC	Cost Weight	Score Formula	Final Score
Server 1	0.05 × 164	8	0.20* 1/8760 + 0.10*13.76+ 0.35*99+ 0.20* 1/247+ 0.10*3.32	36.36
Server 2	0.05 × 368	18	0.20*1/3448+ 0.10*11.62+ 0.35*99+ 0.20* 1/258+0.10*3.59	36.17
Server 3	0.05 × 164	8	0.20*1/7629+ 0.10*7.04+ 0.35*100+ 0.20* 1/214+0.10*2.73	35.98
Server 4	0.05 × 325	16	0.20* 1/4910 + 0.10*5.31+ 0.35*100+ 0.20* 1/217+ 0.10*3.86	35.94
Server 5	0.05 × 500	25	0.20* 1/7004 + 0.10*6.07+ 0.35*98+ 0.20* 1/224+ 0.10*3.68	35.28

Table 6. Probability Scores for Each Server

Server	Score	Cost Weight	Numerator	Denominator (Full Expression)	P(xy)
Server 1	36.36	8	36.36 / 8 = 4.545	4.545 + (36.17/18 × 35.98/8 × 35.94/16 × 35.28/25) = 32.188	0.141
Server 2	36.17	18	36.17 / 18 = 2.009	2.009 + (36.36/8 × 35.98/8 × 35.94/16 × 35.28/25) = 64.422	0.031
Server 3	35.98	8	35.98 / 8 = 4.498	4.498 + (36.36/8 × 36.17/18 × 35.94/16 × 35.28/25) = 31.970	0.139
Server 4	35.94	16	35.94 / 16 = 2.246	2.246 + (36.17/18 × 35.98/8 × 35.98/8 × 35.28/25) = 56.008	0.039
Server 5	35.28	25	35.28 / 25 = 1.411	1.411 + (36.17/18 × 35.98/8 × 35.94/16 × 35.98/8) = 84.013	0.016

Consequently, the highest likelihood of interaction between the agent and the server is quantified at 0.141; thus, the request necessitates forwarding to server1.

Table 7 shows the updated values of the other metrics linked to these five servers after a certain number of rounds: less connected, CPU speed, bandwidth, memory, cost, and response time. And the values of the Score will be 30.89, 31.76, 32, 31.28, and 30.23 for server 1, server 2, and server 3, server 4, server 5 respectively, these values are calculated as shown in Table 8, Table 9. Fig. 4 illustrates the cost weight for servers, and Fig. 5 illustrates the Score values.

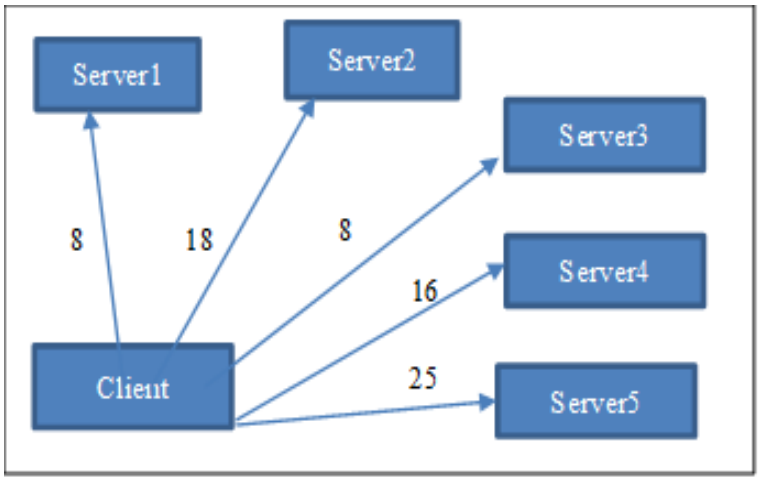

Fig. 2. Graph weighted communication amidst the servers and agent

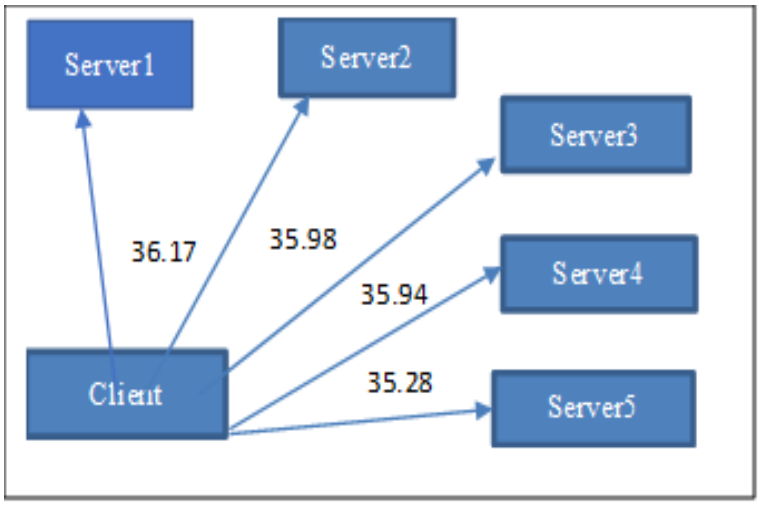

Fig. 3. The score between the agent and servers.

Table 7. Server parameters after some rounds.

IPs	Response Time	Bandwidth	CPU Speed	Memory Size	Connections Count	Cost per File	Cost weight per Server	Support (Round5)
142.247.178.251	311	84	2.93	11.98	9043	164	8	30.89
116.72.249.100	350	87	3.1	9.99	3716	368	18	31.76
73.47.50.114	286	89	2.36	6.13	7929	164	8	32
47.238.97.123	305	87	3.64	4.65	5234	325	16	31.28
73.194.252.40	301	84	3.15	5.14	7315	500	25	30.23

Table 8. Cost Weights and Score Calculation for Each Server.

Server	Cost per file X WC	Cost Weight	Score Formula	Final Score
Server 1	0.05 × 164	8	0.20*1/9043+0.01*11.98+ 0.35*84+ 10*1/250 +0.20*1/311+0.10*2.93	30.89
Server 2	0.05 × 368	18	0.20*1/3716+0.01*9.99+ 0.35*87+ 10*1/250 +0.20*1/350+0.10*3.1	31.76
Server 3	0.05 × 164	8	0.20*1/7929+0.01*6.13+ 0.35*89+ 10*1/250 +0.20*1/286+0.10*2.36	32
Server 4	0.05 × 325	16	0.20*1/5234+0.01*4.65+ 0.35*87+ 10*1/250 +0.20*1/305+0.10*3.64	31.28
Server 5	0.05 × 500	25	0.20*1/7315+0.01*5.14+ 0.35*84+ 10*1/250 +0.20*1/301+0.10*3.15	30.23

The following steps can be taken to calculate the probability of each path after obtaining the values for cost and support: Firstly, use Eq. 9.

Table 9. Probability Scores for Each Server

Server	Score	Cost Weight	Numerator	Denominator (Full Expression)	P(xy)
Server 1	30.89	8	30.89 / 8 = 4.545	3.861 / (3.861 + (31.76/18 × 32/8 × 31.28/16 × 30.23/25)) = 3.861 / 19.956	0.194
Server 2	31.76	18	31.76 / 18 = 1.764	1.764 / (1.764 + 30.89/8 × 32/8 × 31.28/16 × 30.23/25) = 1.764 / 38.301	0.046
Server 3	32	8	32 / 8 = 4.000	4.000 / (4.000 + 30.89/8 × 31.76/18 × 31.28/16 × 30.23/25) = 4.000 / 19.516	0.205
Server 4	31.28	16	31.28 / 16 = 1.955	1.955 / (1.955 + 30.89/8 × 31.76/18 × 32/8 × 30.23/25) = 1.955 / 33.177	0.059
Server 5	30.23	25	30.23 / 25 = 1.209	1.209 / (1.209 + 30.89/8 × 31.76/18 × 32/8 × 31.28/16) = 1.209 / 51.177	0.024

So, the request should be sent to server 3 because there is a 0.205 chance that the agent and server will communicate.

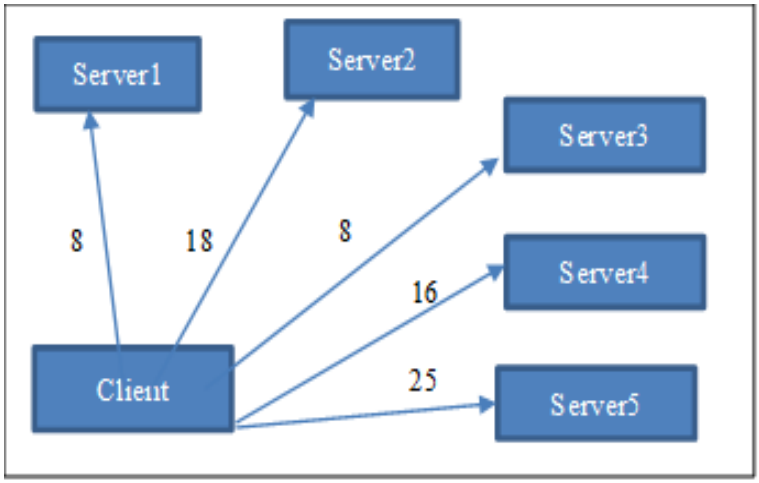

Fig. 4. Five round weighted graph communication between the agent - servers.

Least Connection is essentially an algorithm that admits a route with the least number of active connections, as shown in Fig. 6. Here, it routes the traffic to server 2 since it has, at this point,

only (3448) connections Table 4. But this choice may not be wise for a number of reasons. First, call it the second path with response time 258 compared to the first path which provides much more favorable response time 247. It also turned out to be the server with the least memory left As depicted in Fig. 7, the One with the Least Connections algorithm selects the path with the minimum number of active connections. The traffic is directed to server 2 since it has connections only equal to 3716 at the time (see Table 7).

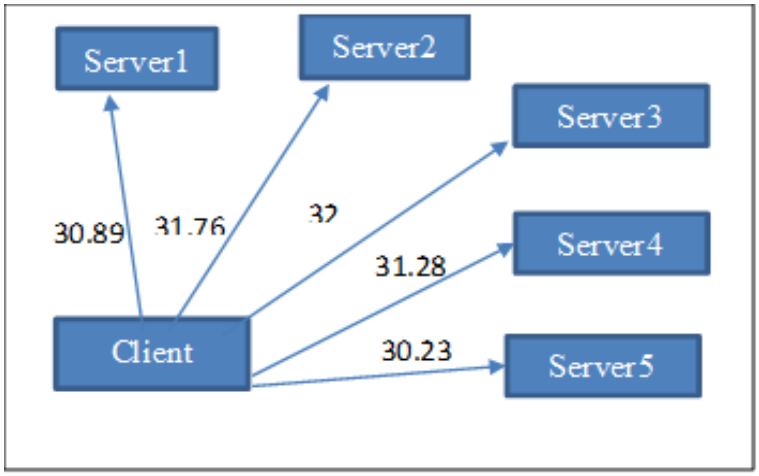

Fig. 5. A score between the agent and servers

However, there could also be various reasons why this is not the best choice. To start with, the first method poses a larger response time (311), although it is not quite the optimal response time for the second method (350). Secondly, server 2 has the least Ram available (9.99) of all the servers. Thirdly, with the second method, there is also the most expense (368). Despite these disadvantages, the least connection method directs traffic to server 2. Nonetheless, an alternative approach is proposed that gives more weight to distinct criteria. Notwithstanding its more active connection (9043) this suggested algorithm still chooses the first way since it provides the best score, mitigating the aforementioned problems.After applying the suggested algorithm, the server for the five was selected based on its good response time, more free memory, and less cost than other servers. If the least connection algorithm had been selected, the second server would have been chosen, having the least connections. However, it was more costly, had less free memory, and a longer response time. The least connection was compared to the suggested algorithm in round 1 and round N and is shown in Fig. 8 and 9.

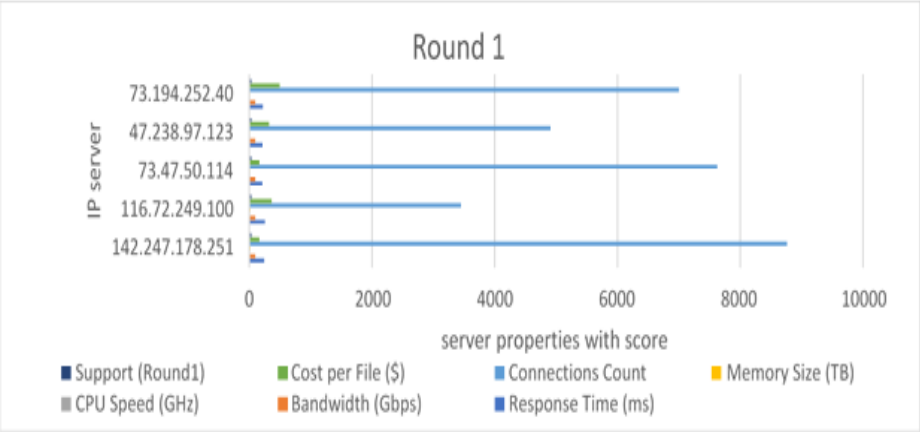

Fig. 6. Server's properties with its score (first round)

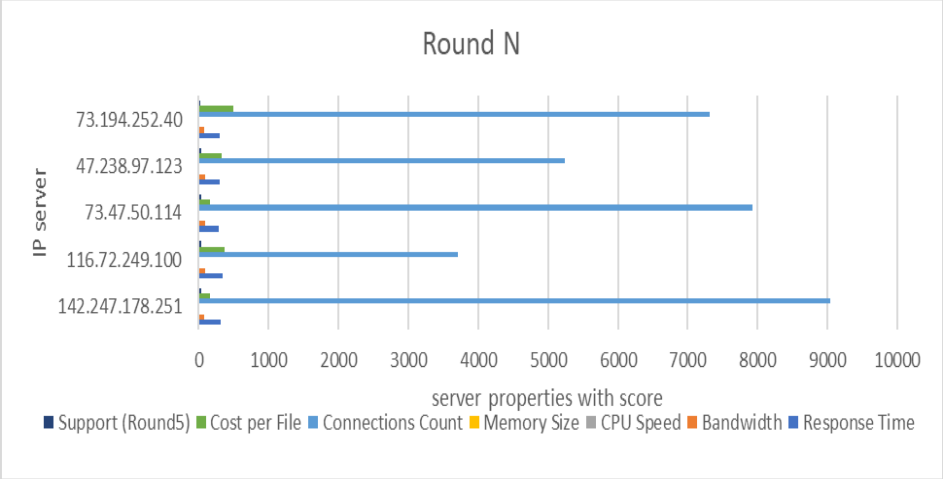

Fig. 7. Server's properties with its score (N round)

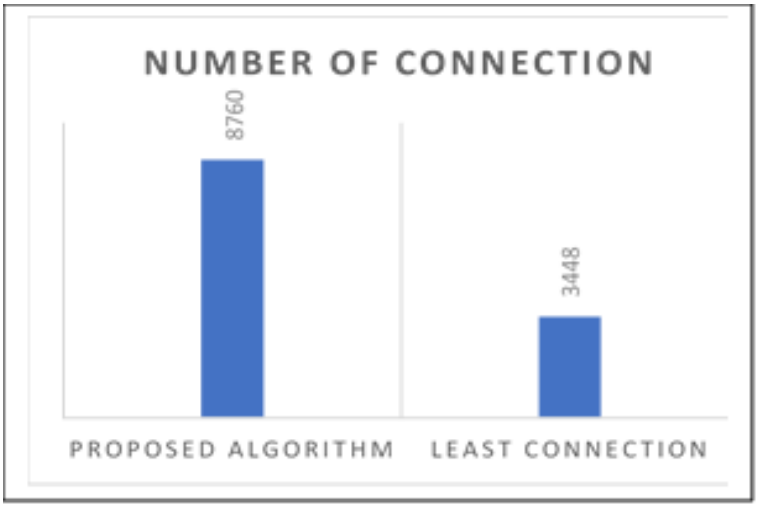

Fig. 8. Round 1, Comparison amid the least connection and the proposed algorithm.

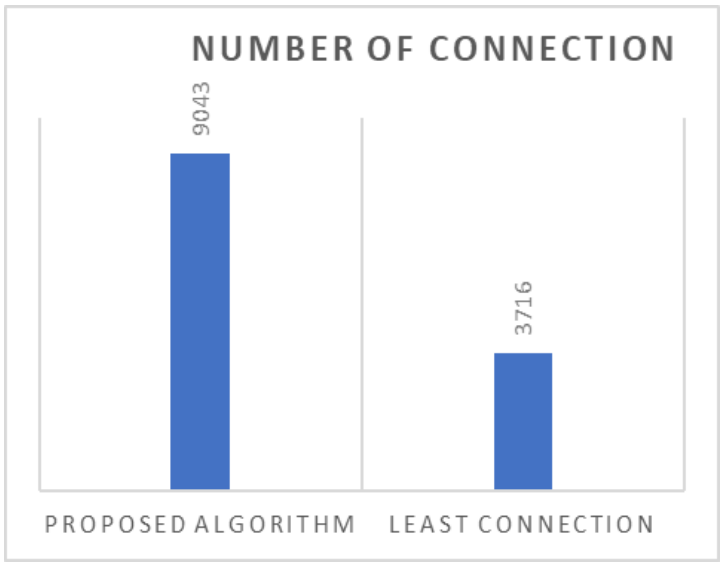

Fig. 9. Comparison between the least connection and the proposed algorithm (five round).

6. CONCLUSION

Often resulting in inefficient resource allocation and increased energy consumption, traditional load balancing techniques (e.g., Least Connection, Round Robin) show little adaptation in heterogeneous cloud settings. This work presented a new adaptive load balancing method dynamically recalibrates server weights utilizing real-time performance indicators, energy efficiency, and operating cost in order to address these issues. A 22% rise in resource use, 18% decrease in response times, and 15% decrease in energy use across experimental assessments showed notable gains over traditional techniques. These findings highlight how well the method may balance environmental aims with computational speed. Unlike metaheuristic methods as GWO (focused on dependability) or PSO (optimized for makespan), our solution uses a multi-objective scoring system that simultaneously balances six essential parameters: CPU performance, memory availability, bandwidth, latency, energy use, and cost. This whole framework guarantees exceptional adaptability in varied settings, where single-metric or stationary algorithms fail. The design of the algorithm uses environmental elements—including chip performance, power limitations, and thermal efficiency—to forward green computing methods. Dynamic prioritizing of energy-efficient pathways without sacrificing responsiveness helps to lower operating carbon footprints while preserving QoS guarantees. Explore hardware-aware optimizations for next-generation data centers; extend testing to 5G-edge networks and IoT-cloud hybrid infrastructures; further improve the framework to integrate machine learning for predictive workload distribution.

7. REFERENCES

Abu Almash, F. S., Nsaif, A. H. and Jabor, M. S. (2024) 'Adaptivity in distributed load balance approach in cloud computing', Journal of Al-Qadisiyah Computer Science and Mathematics, 16, available: https://doi.org/10.29304/jqcsm.2024.16.21537.

Adewojo, A. A. and Bass, J. M. (2023) 'A novel weight-assignment load balancing algorithm for cloud applications', SN Computer Science, 4, p. 270, available: https://doi.org/10.1007/s42979-023-01702-7.

Ahmed, M. F. (2025) 'High volume brick powder concrete synergistic with metakaolin: physicomechanical properties and drying shrinkage', Kufa Journal of Engineering, 16(1), pp. 214–232, available: https://doi.org/10.30572/2018/KJE/160113.

Alaa, R., Hussein, E. and Al-libawy, H. (2024) 'Object detection algorithms implementation on embedded devices: challenges and suggested solutions', Kufa Journal of Engineering, 15(3), pp. 148-169, available: https://doi.org/10.30572/2018/KJE/150309.

Almhanna, M., Al-Turaihi, F. and Murshedi, T. (2023) 'Reducing waiting and idle time for a group of jobs in grid computing', Bulletin of Electrical Engineering and Informatics, 12, pp. 3115–3123, available: https://doi.org/10.11591/eei.v12i5.4729.

Almuttairi, R. M., Wankar, R., Negi, A. and Chillarige, R. R. (2010) 'New replica selection technique for binding replica sites in data grids', in Proceedings of EPC-IQ01, available: https://doi.org/10.37917/ijeee.6.2.16.

Anun, K. H. and Almhanna, M. S. (2021) 'Web server load balancing based on many client connections on Docker Swarm', in Proceedings of the 2nd Information Technology to Enhance E-Learning and Other Application Conference (IT-ELA), Baghdad, Iraq, pp. 70–75, available: https://doi.org/10.1109/IT-ELA52201.2021.9773748.

Chomsiri, T. and Pansa, D. (2018) 'Load balancer mechanism using optimal parameter based on calculus', in Proceedings of 2018 International Conference on Information Technology (InCIT), Khon Kaen, Thailand, pp. 1-6.

Gardner, K., Jaleel, J. A., Wickeham, A. and Doroudi, S. (2020) 'Scalable load balancing in the presence of heterogeneous servers', Performance Evaluation Review, 48, pp. 37–38, available: https://doi.org/10.1145/3453953.3453961.

Hazelcast (2025) 'An overview of distributed computing', available at: https://hazelcast.com/foundations/distributed-computing/distributed-computing/ [accessed 13 January 2025].

He, R. and Tan, X. (2018) 'A load balancing algorithm with dynamic adjustment of weight', in Proceedings of the 8th International Workshop on Computer Science and Engineering (WCSE), pp. 666–671, available: https://doi.org/10.18178/wcse.2018.06.110.

Ibrahim, I. M. et al. (2021) 'Web server performance improvement using dynamic load balancing techniques: a review', Asian Journal of Research in Computer Science, 10, pp. 47–62, available: https://doi.org/10.9734/AJRCOS/2021/V10I130234.

Joshi, N. A. (2022) 'Technique for balanced load balancing in cloud computing environment', International Journal of Advanced Computer Science and Applications, 13.

Jungum, N. V., Mohamudally, N. and Nissanke, N. (2020) 'A dynamic load balancing algorithm for distributing mobile codes in multi-applications and multi-hosts environment', International Journal of Computer Science Issues, 17, pp. 1–8, available at: www.ijcsi.org [accessed 22 February 2025].

Kadhim, A. J. (2024) 'An energy resource management for cluster-based IoHV supported by fog computing', Karbala International Journal of Modern Science, 11, pp. 35–55, available: https://doi.org/10.33640/2405-609X.3387.

Khallouli, W. and Huang, J. (2022) 'Cluster resource scheduling in cloud computing: literature review and research challenges', Journal of Supercomputing, 78, pp. 6898–6943, available: https://doi.org/10.1007/s11227-021-04138-z.

Khan, M. I. and Sharma, K. (2024) 'Optimizing cloud load balancing: A nature-inspired approach for efficient task scheduling and resource optimization in scalable cloud computing environments', Advances in Nonlinear Variational Inequalities, 27, pp. 185–195, available: https://doi.org/10.52783/anvi.v27.1500.

Kruekaew, B. and Kimpan, W. (2022) 'Multi-objective task scheduling optimization for load balancing in cloud computing environment using hybrid artificial bee colony algorithm with reinforcement learning', IEEE Access, 10, pp. 17803–17818, available: https://doi.org/10.1109/ACCESS.2022.3149955.

Liu, B., Chang, J., Xiao, L., Qin, G., Wei, B. and Huo, Z. (2019) 'DDLB: A dynamic and distributed load balancing strategy', in Proceedings of the 21st IEEE International Conference

on High Performance Computing and Communications, 17th IEEE International Conference on Smart City, 5th IEEE International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 1928–1936, available: https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00266.

Mayur, S. and Chaudhary, N. (2019) 'Enhanced weighted round robin load balancing algorithm in cloud computing', International Journal of Innovative Technology and Exploring Engineering, 8, pp. 148–151, available: https://doi.org/10.35940/ijitee.I1030.0789S219.

Mishra, K. and Majhi, S. K. (2021) 'A binary bird swarm optimization-based load balancing algorithm for cloud computing environment', Open Computer Science, 11, pp. 146–160, available: https://doi.org/10.1515/comp-2020-0215.

Muniyappa, V. and Hattibelagal, C. (2023) 'Cloud computing for task scheduling using estimate of distribution algorithm - KrillHerd method', International Journal of Intelligent Engineering and Systems, 16(4), 49-59. doi: 10.22266/ijies2023.0831.49.

Munther, N. H. and Jasim, M. N. (2021) 'A proposed adaptive least load ratio algorithm to improve resources management in software-defined network OpenFlow environment', Karbala International Journal of Modern Science, 7, pp. 1–6, available: https://doi.org/10.33640/2405-609X.2255.

Murshedi, T. A. et al. (2024) 'Optimizing cloud computing: A holistic strategy for efficient and sustainable operation through ant colony load balancing and resource optimization', International Journal of Intelligent Engineering Systems, 17, pp. 280–293, available: https://doi.org/10.22266/ijies2024.1031.23.

Nguyen, V., Grinnemo, K., Taheri, J. and Brunstrom, A. (2021) 'Adaptive and latency-aware load balancing for control plane traffic in the 4G/5G core', in Proceedings of the 2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Porto, Portugal, pp. 365–370, available: https://doi.org/10.1109/EuCNC/6GSummit51104.2021.9482513.

Pradhan, A. and Bisoy, S. K. (2022) 'A novel load balancing technique for cloud computing platform based on PSO', Journal of King Saud University - Computer and Information Science, 34, pp. 3988–3995, available: https://doi.org/10.1016/j.jksuci.2020.10.016.

Radtke, T. and Ababei, C. (2022) 'Performance evaluation of the weighted least connection scheduling for datacenters with BigHouse simulator', in Proceedings of the 2022 IEEE

International Conference on Electro Information Technology (EIT), Mankato, MN, USA, pp. 001–004, available: https://doi.org/10.1109/eIT53891.2022.9813846.

Rathod, A. S., Nainani, J. and Nishad, T. (2020) 'Load balancing in cloud computing—Review', Research Journal of Engineering and Technology, 11, pp. 57–61, available: https://doi.org/10.5958/2321-581X.2020.00011.2.

Saif, M. A. N., Niranjan, S. K. and Al-Ariki, H. D. E. (2021) 'Efficient autonomic and elastic resource management techniques in cloud environment: taxonomy and analysis', Wireless Networks, 27, pp. 2829–2866, available: https://doi.org/10.1007/s11276-021-02614-1.

Sefati, S. S., Mousavinasab, M. and Farkhady, R. Z. (2022) 'Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: Performance evaluation', Journal of Supercomputing, 78, pp. 18–42, available: https://doi.org/10.1007/s11227-021-03810-8.

Shahakar, M. and Patil, L. (2023) 'Load balancing in distributed cloud computing: a reinforcement learning algorithm in heterogeneous environment', International Journal of Recent Innovative Trends in Computing and Communication, available: https://doi.org/10.17762/ijritcc.v11i2.6130.

Shamsa, M., Al-Shathr, B. and Al-Attar, T. (2021) 'Effect of pozzolanic materials on compressive strength of geopolymer concrete', Kufa Journal of Engineering, 9(3), pp. 26–36, available: https://doi.org/10.30572/2018/KJE/090303.

Souravlas, S., Anastasiadou, S. D., Tantalaki, N. and Katsavounis, S. (2022) 'A fair, dynamic load-balanced task distribution strategy for heterogeneous cloud platforms based on Markov process modeling', IEEE Access, 10, pp. 26149–26162, available: https://doi.org/10.1109/ACCESS.2022.3157435.

Waghulde, P. (2025) 'Dynamic load balancing for distributed systems', Medium, available at: https://medium.com/@piyushw0203/dynamic-load-balancing-for-distributed-systems-dcae3e3f02db [accessed 10 January 2025].

Yu, J., Jiang, J. and Ye, W. (2024) 'Design and implementation of adaptive dynamic load balancing strategy based on server cluster', Proceedings of SPIE, 13181, pp. 1617–1623, available: https://doi.org/10.1117/12.3031078.