Smart Internet of Things

www.siot.reapress.com

Smart. Internet. Things. Vol. 1, No. 1 (2024) 1–16.

Paper Type: Original Article

Optimizing Cloud Performance: A Comprehensive Study of Load Balancing Strategies and Algorithms

Ayush Singh[1], Aman Kumar Sahu[1], Nabeel Anwar Siddiqui[1], Siddharth Singh[1,*]

[1] School of Computer Science Engineering, KIIT University, Bhubaneswar, India; 21053209@kiit.ac.in; 2105601@kiit.ac.in; 2105975@kiit.ac.in; 21052029@kiit.ac.in.

Citation:

Received: 3 February 2024 *Revised: 11 April 2024* *Accepted: 19 June 2024*	Singh, A., Sahu, A. K., Siddiqui, N. A., & Singh, S. (2024). Optimizing cloud performance: a comprehensive study of load balancing strategies and algorithms. *Smart internet of things, 1* (1), 1-16.

Abstract

Cloud computing is a big system of interconnected servers that store data and run programs over the internet. As this technology grows, ensuring it runs smoothly and efficiently is important. One way to do this is through load balancing, where tasks and data are distributed evenly across the servers to avoid overloading any of them. However, achieving effective load balancing can be challenging due to factors like servers' geographical spread and differences in capabilities. Our study delved into various load balancing strategies used in cloud computing, including Min-Min, Max-Min, Least Connection, Source Hash, Least Bandwidth, and Round Robin. While these strategies help optimize performance, they also come with their own set of limitations and challenges. By examining the pros and cons of different methods, our study gives us a better understanding of how load balancing works in cloud computing right now. It also helps us see how we can improve things in the future. We want to keep improving cloud computing so it can handle all the tasks it needs to in today's digital world. Our research helps us learn more about cloud computing and how we can make it stronger and more efficient in the years to come.

Keywords: Cloud Computing, Load Balancing, Scheduling Algorithms

1 | Introduction

Cloud computing technology is growing rapidly. Cloud computing is a technology that utilizes the internet and central isolated servers to sustain data and applications. Cloud computing provides a flexible way to retain data and files. Cloud computing involves virtualization, distributed computing and web services. Cloud computing aims to provide maximum services at minimum cost, enhance response time, and provide better performance. With advancements in technologies like the internet, today, millions of computing devices connect to the cloud and access data at any given time, and these devices get a response from a cloud in a matter of seconds [1]. *Fig. 1* shows cloud computing architecture.

Corresponding Author: 21052029@kiit.ac.in

https://doi.org/10.22105/siot.v1i1.34

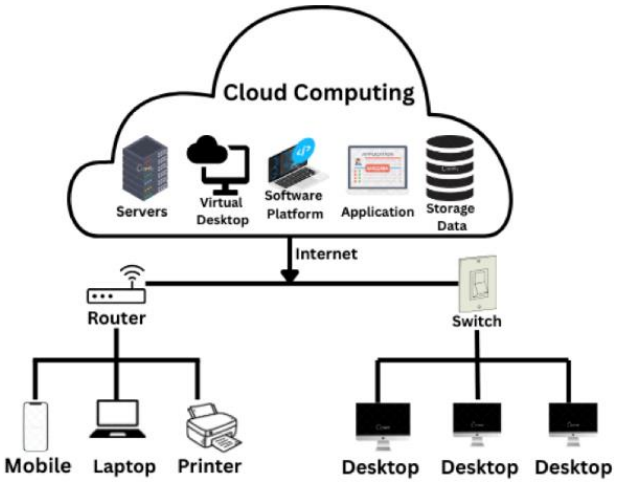

Fig. 1 Cloud computing architecture.

Cloud computing is becoming quite popular; hence, there has been a significant increase in the amount of processing carried out in the cloud, and the load on the cloud is increasing [2]. In a cloud computing environment, there are multiple types of loads: CPU load, Network load, Memory load, Storage load, etc., which may affect the efficiency and availability of resources in that environment. To overcome this problem, various load balancing techniques are devised, which are used to distribute incoming network traffic and computational tasks across multiple servers or resources in a cloud environment with the primary goals of optimizing resource utilization by distributing loads across different servers, improving performance by preventing single servers from becoming overloaded, enhance scalability and availability by allowing the cloud to scale by adding or removing servers depending on workload demand and also mitigate downtime and failures [3]. *Fig. 2* shows load balancing in cloud computing.

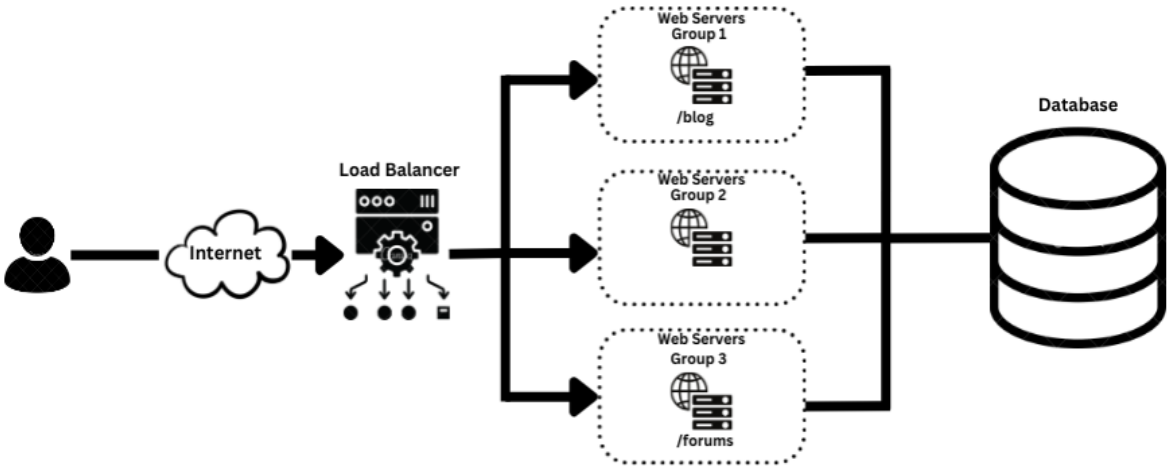

Fig. 2 Load balancing in cloud computing.

Load-balancing concepts encounter multiple challenges due to various physical and logical issues that can impact their effectiveness [4]. Challenges associated with load balancing are outlined in *Table 1*.

Table 1. Issues associated with load balancing.

Challenges	Description
Geographical distribution [5]	Cloud data centres are geographically distributed, but load balancing may overlook communication delays and resource allocation.
Virtual machine migration [6]	Multiple Virtual Machines (VMs) on the same physical machine can overload due to the distinct structures of these VMs.
Algorithm complexity [7]	Load balancing algorithms should be straightforward and succinct to uphold cloud efficiency.
Heterogeneous nodes [8]	Different user needs require diverse nodes, impacting load-balancing choices.
Single point of failure [9]	Load balancing algorithms run on a central node, risking complete computing failure if the central node malfunctions.
Load balancer scalability [10]	The response time of load balancing is influenced by factors such as computing power, topology, and storage, highlighting the importance of scalability.

The paper is structured as follows. Section 2 provides a literature review of load-balancing algorithms. Section 3 discusses the challenges associated with load balancing, while Section 4 highlights the limitations of various load-balancing algorithms. Section 5 explores potential improvements in these algorithms to enhance efficiency and fault tolerance. Finally, Section 6 summarizes the key findings, future research directions, and references.

2 | Literature Review

There are two load-balancing strategies: 1) static load-balancing algorithms, and 2) dynamic load-balancing algorithms [2]. A static load balancing algorithm does not consider a node's previous state or behaviour while distributing the load. On the other hand, a dynamic load balancing algorithm checks the prior state of a node while distributing the load, such as CPU load, amount of memory used, delay or network load, and so on. We can go with static algorithms if the systems have low load variations; otherwise, dynamic algorithms are good [11]. Load balancing techniques classification are outlined in *Table 2*.

Table 2. Load balancing techniques classification.

Static load balancing [3] fixed rules do not consider the current state or previous knowledge of resources such as storage, preprocessing, etc.	Optimal load balancing Collect information for a load balancer to allocate tasks to the resources at the optimum time Sub-optimal load balancing Load Balancer cannot make optional decisions, so it comes up with a suboptimal solution.	Examples of Static techniques: min-min, max-min, round robin, shortest job first, opportunists load balancing, appropriate load balancing, heuristic load balancing, IP Hash
Dynamic load balancing [3] Making decisions based on the system's current status. In addition to task transfer from overloaded machine to underloaded machine.	Distributed load balancing All nodes participate in task scheduling, load distribution, resource allocation, and distributing and redistributing tasks effectively. Centralized load balancing Only a Single node is responsible for load distribution and decision-making.	Examples of dynamic techniques cooperative load balancing, non-cooperative load balancing, centralized load balancing, sesmi-distributed load balancing, least connection

2.1 | Load Balancing Algorithms

2.1.1 | Min-Min scheduling algorithm

The Min-Min algorithm is a simple cloud scheduling algorithm and is the basis of contemporary cloud scheduling algorithms. The Min-Min algorithm finds the resource which can perform the given tasks in the least time and execute them in that resource [5]. The algorithm Min-Min is adapted from [5] is presented below:

Algorithm 1. Min-Min.

Step 1. For all submitted tasks in the task set Ti.

Step 2. For all resources Ri.

Step 3. Compute Ctij = Etij + rtij.

Step 4. Do while set is not empty.

Step 5. Find task Tk that gives minimum execution time.

Step 6. Assign task Tk to resource Rj that gives minimum expected completion time.

Step 7. Delete task Tk from the set.

Step 8. Update ready time rtj for the selected Resources j.

Step 9. Update Ctij for all Ti .

Step 10. End do.

Fig. 3 shows the flow chart of the Min-Min algorithm.

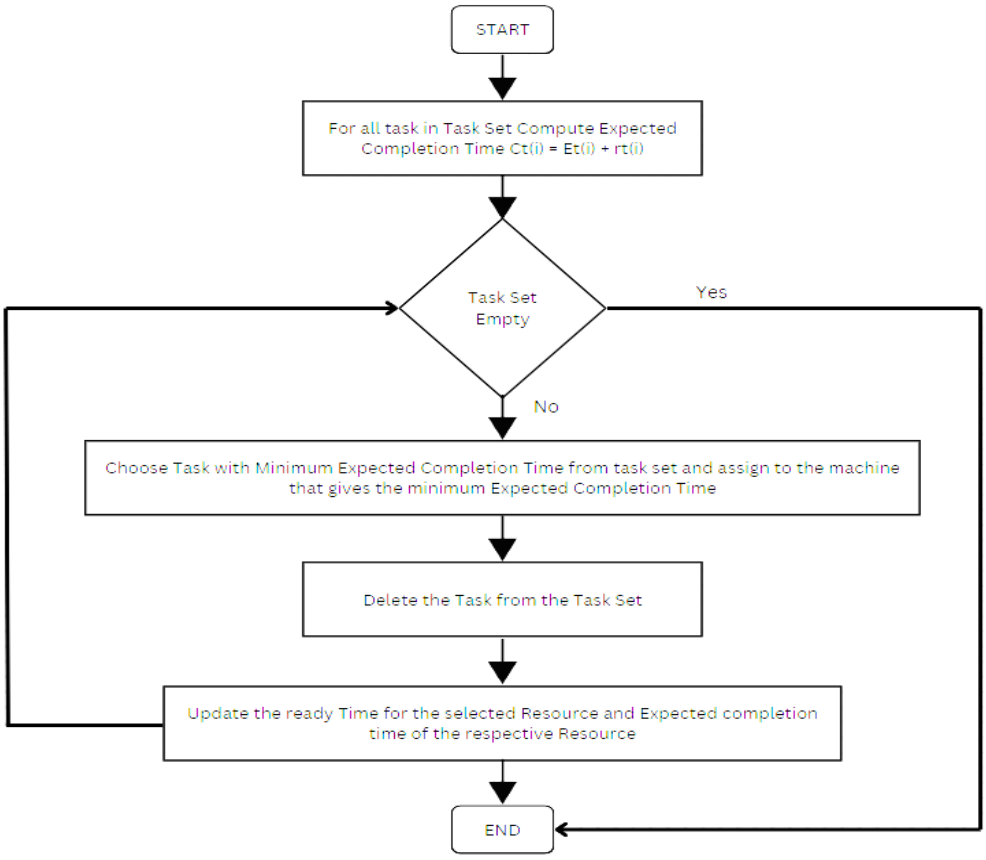

Fig. 3. Min-Min algorithm flow-chart.

2.1.2 | Max-Min scheduling algorithm

This algorithm is the same as the Min-Min algorithm. First, we select the machine that can perform various tasks in minimum time. From the selected tasks, we choose the task that will execute in maximum time and execute that. Then, we remove that task from the set [12]. The algorithm Max-Min is adapted from [6] is presented below:

Algorithm 2. Max-Min.

For all task ti in meta-task Mv (in an arbitrary order) do

for all machines mj (in a fixed arbitrary order) do

ctij = etij + rj

end for

carry out some processing

while all task in Mv are mapped do

for each task ti in Mv find its earliest completion time and the machine that obtain it do

find the task ti with maximum earliest completion time

end for

assign the task ti to the machine mi that gives the earliest completion time

. delete the task tk from Mv

update rj

update ctij for all ti belong Mv

end while

end for.

Fig. 4 shows the flow chart of the Max-Min algorithm.

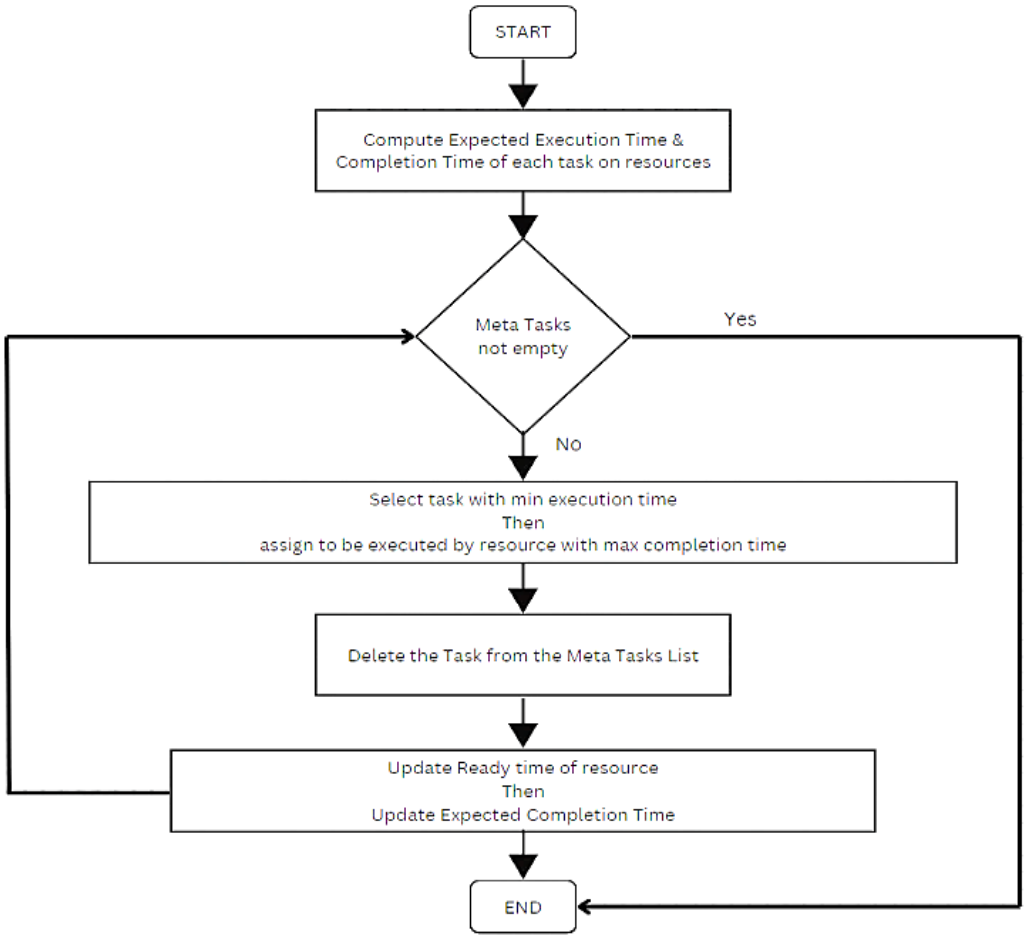

Fig. 4. Max-Min algorithm flow chart.

2.1.3 | Least connection algorithm

As proposed by [13], the Least Connection algorithm effectively facilitates dynamic scheduling and routing of incoming visitors to the server with the fewest active connections, as demonstrated in *Fig. 5*. This approach ensures fair load distribution by continuously monitoring and selecting the least-loaded server based on active connections. It is particularly beneficial in high-traffic scenarios, as emphasized in [8], as it optimizes resource utilization and system performance. The algorithm performs exceptionally well in environments with longer session requirements, such as databases (e.g., MariaDB or SQL) managing high transaction rates. However, alternative load balancing strategies, as recommended in [14], may be more suitable for short-lived connections like HTTP.

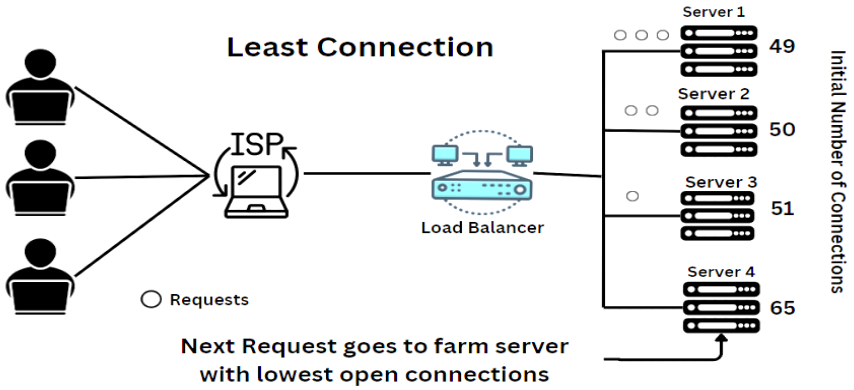

Fig .5. Least connection algorithm.

Algorithm 3 below, known as the Least Connection Algorithm adapted from [9], dynamically selects the server with the fewest active connections to efficiently manage incoming requests in distributed systems.

Algorithm 3. Least Connection algorithm.

```
function least_conn_algorithm(servers):
    min_conn = infinity
    selected_server = null
    // Loop through each server in the server list
    for server in servers:
      // If the server's connections are less than min_conn
        if server.connections < min_conn:
            // Update min_conn and selected_server
            min_conn = server.connections
            selected_server = server
    // Return the server with the least connections
  return selected_serve.
```

Explanation

The least connection algorithm examines a list of servers and identifies the server with the fewest active connections. It initializes the minimum connections variable to a very large value and the selected server variable to null. Then, it iterates through the servers, comparing the number of connections on each server with the current minimum [15]. If a server has fewer connections, it updates the minimum connections variable and selects that server. Ultimately, it returns the server with the least connections. This approach ensures that incoming requests are distributed to the server with the lightest load, balancing the overall load across the servers. The key variables used in the algorithm are the server list, the minimum connections, and the selected server, while the primary function is responsible for iterating through the servers and identifying the server with the least connections [16].

2.1.4 | Source hash algorithm

As proposed in [17], the source hashing algorithm determines server selection in this load-balancing approach. Utilizing the source IP address as the hash key in the hash table, as explained in [10], ensures consistent routing of a user's request to the server that previously handled their request. Illustrated in *Fig. 6*, this sincere load-balancing algorithm provides continuity in routing for the same user based on their source IP address. This algorithm is advantageous for applications requiring consistent routing for the same user. However, challenges, as noted in [17], arise due to dynamic IP addresses provided by ISPs, impacting the maintenance of this server-specific routing approach.

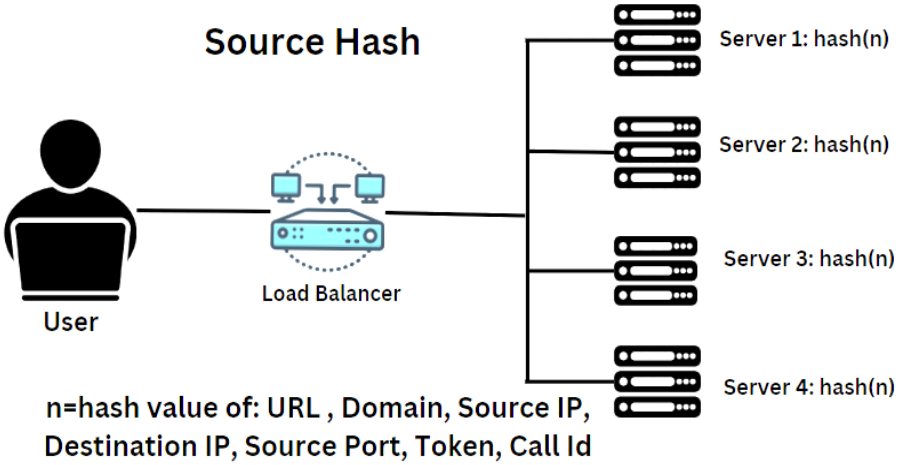

Fig. 6. Source hash algorithm.

Algorithm 4 given below, referred to as the Source Hash Load Balancing algorithm adapted from [11], dynamically selects the server based on the hash value generated from the source address to manage incoming requests efficiently.

Algorithm 4. IP Hash algorithm.

```
Function source_hash_load_balancing(servers, source_address):
        Hash_value = hash_function(source_address)  // Use a hash function to generate a hash value
        Hash_int = convert_hex_to_int(hash_value)   // Convert hexadecimal hash value to integer
    Selected_server = null
    // Determine the index of the selected server based on the hash value
    Selected_server_index = hash_int % length(servers)
    // Return the selected server
Return servers[selected_server_index].
```

Explanation

The Source Hash Load Balancing algorithm is designed to distribute incoming requests among a group of servers based on the hash value generated from the source address. It uses a hash function to convert the source address into a hash value, which is then transformed into an integer. This integer is used to determine the index of the selected server by taking the modulo of the server list length. The algorithm returns the server associated with the calculated index, ensuring that requests from the same source address consistently go to the same server [18]. The key variables include the server list, source address, hash value, hash integer, and the selected server. The algorithm contributes to load balancing by providing a stable mapping of source addresses to servers, promoting efficient resource utilization.

2.1.5 | Least bandwidth

The Least Bandwidth algorithm, referenced in [8], dynamically prioritizes servers based on Mbps capacity, directing requests to the server with the least network traffic (*Fig. 7*). The details in [12] explain how it adapts to varying network bandwidths and changing conditions. By monitoring network traffic and considering weight-based variations, it intelligently selects the least-loaded server, optimizing resource utilization and system performance for diverse scenarios [19]. This approach provides an efficient and adaptable solution for load balancing in environments with fluctuating network conditions, aligning with modern strategies, as outlined in [8].

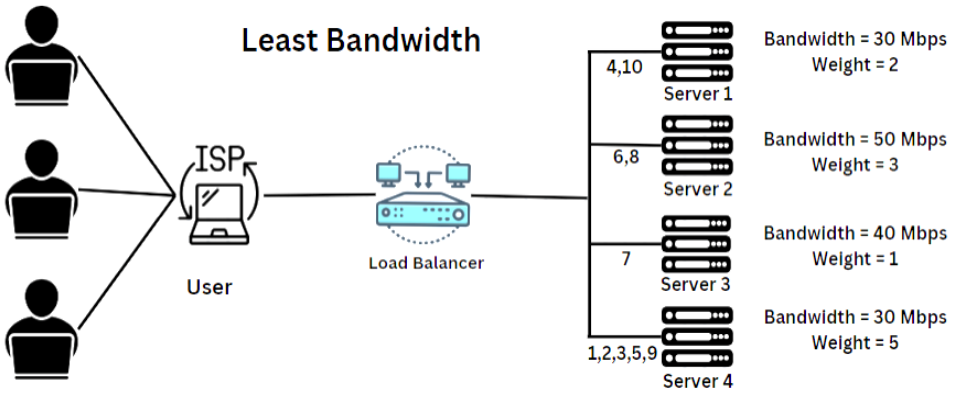

Fig. 7. Least Bandwidth algorithm.

Algorithm 5, known as the Least Bandwidth Algorithm and adapted from [12], dynamically prioritizes servers based on their Mbps capacity to manage incoming requests in distributed systems efficiently.

Algorithm 5. Least Bandwidth algorithm.

```
Function least_bandwidth_algorithm(servers):
    min_bandwidth = infinity
    selected_server = null
    // Loop through each server in the server list
    for server in servers:
      // If the server's Mbps capacity is less than min_bandwidth
      if server.mbps_capacity < min_bandwidth:
          // Update min_bandwidth and selected_server
          min_bandwidth = server.mbps_capacity
          selected_server = server
     // Return the server with the least Mbps capacity
return selected_server.
```

Explanation

The Least Bandwidth algorithm prioritizes servers by evaluating their Mbps capacity. It initializes the minimum bandwidth variable to a very large value and the selected server variable to null. Then, it iterates through the servers, comparing the Mbps capacity of each server with the current minimum. If a server has a lower Mbps capacity, it updates the minimum bandwidth variable and selects that server. Ultimately, it returns the server with the least Mbps capacity. This strategy efficiently distributes incoming requests to servers with lower network traffic, optimizing resource utilization and system performance. The key variables are the server list, minimum bandwidth, and the selected server, while the primary function iterates through servers to identify the one with the least Mbps capacity [20].

2.1.6 | Round Robin scheduling algorithm

The round-robin algorithm is one of the oldest, simplest, fairest and most widely used scheduling algorithms, designed especially for time-sharing systems. In the RR algorithm, the jobs share the CPU time by allocating a slice of time; usually between 10 and 100 ms for each job, called Quantum Time (QT) [13]. All runnable processes are kept in a circular queue. The CPU scheduler goes around this queue, allocating the CPU to each process for a time interval of one quantum. New processes are added to the tail of the queue [13], [14].

Algorithm 6 shows the pseudocode of the RR algorithm as described in [13]. *Table 3* shows an analysis of various load-balancing algorithms [21].

Algorithm 6. The Pseudocode of the RR algorithm in CPU scheduling.

Step 1. Keep the ready queue as a FIFO queue of tasks.

Step 2. New tasks added to the tail of the queue will be selected, set a timer to interrupt after one time slot, and dispatch the tasks.

Step 3. The task may have executed less than one time quantum. In this case:

I. The task itself will release the resources voluntarily.

II. The scheduler will then proceed to the next task in the ready queue.

Step 4. Otherwise, if the running task is longer than one time quantum, the timer will go off and will cause an interruption to the OS.

Table 3. Load Balancing algorithms analysis.

Algorithm	Overhead	Advantages	Disadvantages	Use Cases
Min-Min [3]	Medium overhead	Efficiently assigns tasks to resources with the least execution time Simple and easy to implement	May lead to suboptimal makespan due to resource imbalance Limited dynamic workload handling	Applications with predictable and stable workloads
Max-Min [3]	Medium overhead	Prioritizes tasks on the most loaded resources Fair distribution of workload	Similar limitations as Min-Min May lead to suboptimal makespan due to resource imbalance	Similar to Min-Min, suitable for static workloads
Least connection [3]		Dynamically selects the server with the fewest active connections Effective in high-traffic scenarios	May lead to server overload in certain scenarios Performance may degrade with short-lived connections	High-traffic applications, databases with high transaction rates
Source Hash [3]	Low to moderate	Consistent routing for the same user based on source IP address Efficient for long-lived connections	Vulnerable to DDoS attacks and IP spoofing Challenges with dynamic IP addresses	Applications requiring consistent routing for the same users
Least Bandwidth [3]	Low to moderate	Prioritizes servers with the least network traffic Adaptive to changing network conditions	Risk of bandwidth constraints leading to service interruptions Dependency on network speed	Environments with fluctuating network conditions
Round Robin [3]	No overhead	Simple and fair distribution of tasks Suitable for time-sharing systems	Assumes servers have equivalent loads, leading to resource imbalance Limited scalability	Basic load balancing in simple environments

3|Challenges Associated with Load Balancing

Cloud computing depends on the proper utilization of resources to fulfil customer requirements. Load balancing techniques are just responsible for this, but there are various challenges associated with load balancing, which are mentioned below [22]:

3.1 | Geographical Distribution of Nodes

Data centres are strategically positioned according to the geographical characteristics of an area or place to facilitate computational tasks. In this configuration, dispersed nodes are highly valued as an integrated system for efficiently executing user-requested operations.

3.2 | Single Point of Failure

Load-balancing decisions are centralized and managed by the master node using various dynamic load-balancing algorithms, resulting in a non-distributed setup. If the master node experiences a failure, it disrupts the entire computing domain.

3.3 | Virtual Machine Migration

Virtualization involves consolidating multiple VMs onto a single physical system. Each deployed VM exhibits unique behaviour with diverse configurations. In cases where the physical system becomes overwhelmed, certain VMs may need to be relocated to a remote location using Cloudlet migration techniques.

3.4 | Algorithm Complexity

Algorithm design should prioritize simplicity and ease of implementation. Increased algorithm complexity corresponds to decreased performance and efficiency within the cloud environment.

3.5 | Load Balancer Scalability

Cloud services offer users the flexibility to access services at any time or location by swiftly scaling resources up or down based on demand. An effective load-balancing algorithm should dynamically adjust to rapid changes in demand related to network topology, power, etc., to optimize system performance.

4 | Limitations of Scheduling Algorithms

4.1 | Limitations of Min-Min Scheduling Algorithm

While the Min-Min algorithm offers simplicity and ease of implementation, it comes with some limitations that can hinder its effectiveness in cloud computing environments [23]. Here are some key problems associated with Min-Min:

4.1.1 | Suboptimal makespan

Min-Min focuses on assigning tasks to resources with the minimum execution time for each individual task. This might lead to some resources overloading while others remain idle. This can result in a suboptimal overall completion time (makespan) for all tasks [16].

4.1.2 | Limited dynamic workload handling

Min-Min is designed for static workloads, where the number of tasks and resource capabilities remain constant. However, cloud environments are dynamic, with workloads fluctuating. This can lead to inefficient load balancing when workloads change [17].

4.1.3 | Resource heterogeneity assumption

Min-Min typically assumes homogeneous resources (all resources have the same capabilities). In reality, cloud environments consist of heterogeneous resources (e.g., CPU, memory, storage) with varying capacities. This can lead to suboptimal task allocation if resource heterogeneity is not considered [17].

4.2 | Limitations of Max-Min Scheduling Algorithm

4.2.1 | Suboptimal makespan (like Min-Min)

Like Min-Min, Max-Min prioritizes minimizing the workload on the most loaded resource. While this might seem beneficial, it can lead to situations where other resources remain underutilized. This can result in a suboptimal overall completion time (makespan) for all tasks [24].

4.2.2 | Limited dynamic workload handling (like Min-Min)

Max-Min, like Min-Min, is designed for static workloads. When workloads fluctuate in dynamic cloud environments, Max-Min's efficiency can decrease, potentially leading to imbalanced load distribution [17].

4.2.3 | Resource heterogeneity assumption (like Min-Min)

Max-Min typically assumes homogeneous resources (all resources have the same capabilities). In reality, cloud environments consist of heterogeneous resources (e.g., CPU, memory, storage) with varying capacities. This can lead to suboptimal task allocation if resource heterogeneity is not considered [17].

4.3 | Limitations of Source Hash Algorithm

4.3.1 | Distributed denial of service (DDoS) attacks (privacy concerns)

Attackers could potentially manipulate source addresses to target specific servers, leading to uneven loads or overloading a particular server [18].

4.3.2 | Hash Polarization

Uneven distribution of hash values can lead to certain servers consistently handling a disproportionate share of the load, like the Traffic Polarization Effect [19] in networking, in which certain traffic or congestion gets collected in certain routers or servers.

4.3.3 | Dynamic IP addresses causing IP spoofing

Dynamic IP addresses, especially those assigned by Internet Service Providers (ISPs), introduce the risk of IP spoofing in systems that rely on IP-based authentication or verification. Attackers may exploit the dynamic nature of IP assignments to impersonate legitimate users or devices by forging their source IP addresses [18].

4.4 | Limitations of Least Bandwidth Algorithm

4.4.1 | Bandwidth limit exceeded (509 Error)

Choosing routes with the least traffic may lead to bandwidth constraints, suspending the site due to the Bandwidth Limit Exceeded 509 error [21]. Users experience service disruption, and the site becomes temporarily unavailable.

4.4.2 | Dependency on Network Speed

Internet or connection speed heavily depends on the data transfer rate, affecting the user experience. Slower network performance in low-bandwidth systems negatively impacts data transfer rates [20].

4.4.3 | Bandwidth consumption management

The inability to manage bandwidth consumption may lead to unpredictable spikes, potentially causing service interruptions. Difficulty in maintaining a consistent and reliable level of service [20].

4.5 | Limitations of Round Robin

The biggest limitation of using the round-robin algorithm in load balancing is that the algorithm assumes that servers are in such a manner that they can handle equivalent loads. If certain servers have more CPU, RAM, or other specifications, the algorithm has no way to distribute more requests to these servers. As a result,

servers with less capacity may overload easily and fail; meanwhile, servers with higher capacity may remain idle and not be utilized completely.

To overcome this, the weighted round-robin load balancing algorithm can be introduced to allow site administrators to assign weights to each server based on criteria like traffic-handling capacity. Servers with higher weights receive a higher proportion of client requests.

For a simplified example, assume that a company has a cluster of three servers:

I. Server A can handle 30 requests per second, on average.

II. Server B can handle 20 requests per second, on average.

III. Server C can handle 10 requests per second, on average.

Next, assume that the load balancer receives 6 requests.

I. 3 requests are sent to Server A.

II. 2 requests are sent to Server B.

III. 1 request is sent to Server C.

In this manner, the weighted round-robin algorithm distributes the load according to each server's capacity and solves the issue our simple round-robin had [21].

5 | Proposed Work

Considering the advantages and disadvantages of each load-balancing algorithm, the proposed work aims to address the limitations while leveraging the strengths to enhance overall performance and efficiency in diverse cloud computing environments.

5.1 | Optimization of Min-Min and Max-Min Algorithms [23]

I. Develop enhanced versions of Min-Min and Max-Min algorithms that incorporate dynamic workload handling mechanisms to address the limitations related to suboptimal makespan and limited scalability [22].

II. Introduce adaptive task allocation strategies to balance resource utilization effectively, particularly in environments with fluctuating workloads [22].

5.2 | Improvement of Least Connection Algorithm

I. Implement algorithms or mechanisms to mitigate the risk of server overload in high-traffic scenarios while maintaining performance consistency [23].

II. Investigate methods to improve the performance of the Least Connection algorithm in handling short-lived connections, potentially through optimization techniques or adaptive adjustments [23].

5.3 | Enhancement of Source Hash Algorithm

I. Enhance security measures to address DDoS attacks and IP spoofing vulnerabilities, such as integrating robust authentication and access control mechanisms [24].

II. Explore strategies to mitigate challenges associated with dynamic IP addresses, potentially through adaptive routing mechanisms or dynamic IP management solutions [24].

5.4 | Refinement of Least Bandwidth Algorithm

I. Develop algorithms or approaches to proactively manage and prevent bandwidth constraints, including dynamic bandwidth allocation strategies and network traffic optimization techniques.

II. Investigate methods to reduce dependency on network speed and enhance adaptability to varying network conditions through intelligent routing and protocol optimizations.

5.5 | Scalability and Adaptability of Round Robin Algorithm

I. Design enhancements to improve the scalability of the Round Robin algorithm, allowing it to handle larger and more complex environments effectively.

II. Introduce dynamic load balancing policies or mechanisms to address resource imbalances and optimize task distribution across servers with varying capacities.

5.6 | Performance Evaluation and Comparative Analysis

I. Conduct comprehensive performance evaluations and comparative analyses of the proposed enhancements against traditional algorithms.

II. Utilize real-world workload scenarios and benchmarks to assess the effectiveness and efficiency of the proposed improvements in diverse cloud computing environments.

These proposed enhancements and evaluations aim to advance the state-of-the-art load balancing algorithms, address existing limitations, and pave the way for more efficient and adaptive load management in cloud computing infrastructures.

6. Conclusion

In conclusion, this paper has comprehensively studied various load-balancing strategies and algorithms in cloud computing environments. Through analyzing the strengths and weaknesses of prominent algorithms such as Min-Min, Max-Min, Least Connection, Source Hash, Least Bandwidth, and Round Robin, we have gained insights into their applicability, limitations, and potential for optimization.

We have identified key challenges and opportunities to improve load-balancing techniques by scrutinizing these algorithms and their performance characteristics. Our exploration has highlighted the importance of addressing suboptimal makespan, limited scalability, vulnerability to attacks, and dependency on network conditions.

Through the proposed work, which aims to optimize existing algorithms and enhance their adaptability, scalability, and security, we seek to contribute to the evolution of load-balancing methodologies in cloud computing. By developing more robust and efficient load-balancing strategies, we aim to improve resource utilization, enhance system performance, and effectively meet the demands of modern digital environments.

Overall, this study serves as a foundation for further research and development in the field of load balancing in cloud computing. By addressing the identified challenges and implementing proposed enhancements, we can strive towards realizing the full potential of cloud computing technologies, ultimately leading to more resilient, scalable, and efficient cloud infrastructures for future applications and services.

Author Contributions

Ayush Singh: Conceptualization of the study, methodology development, and writing the original draft.

Aman Kumar Sahu: Data analysis, implementation of load balancing algorithms, and manuscript review.

Nabeel Anwar Siddiqui: Assisted with research, validation of the results, and contributed to discussions on the limitations of the strategies.

Siddharth Singh: Supervision, overall project administration, and final editing of the manuscript.

Funding

This research received no external funding.

Data Availability

The data used and analyzed during the current study are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

These sections should be tailored to reflect the specific details and contributions if necessary.

References

[1] Pradeep, K., & Jacob, T. P. (2016). Comparative analysis of scheduling and load balancing algorithms in cloud environment. *2016 international conference on control, instrumentation, communication and computational technologies (ICCICCT)* (pp. 526–531). IEEE.

[2] Aslam, S., & Shah, M. A. (2015). Load balancing algorithms in cloud computing: a survey of modern techniques. *2015 national software engineering conference (NSEC)* (pp. 30–35). IEEE.

[3] Alkhatib, A. A. A., Alsabbagh, A., Maraqa, R., & Alzubi, S. (2021). Load balancing techniques in cloud computing: Extensive review. *Advances in science, technology and engineering systems journal*, *6*(2), 860–870. https://dx.doi.org/10.25046/aj060299

[4] Shah, N., & Farik, M. (2015). Static load balancing algorithms in cloud computing: challenges & solutions. *International journal of scientific & technology research*, *4*(10), 365–367.

[5] Chen, H., Wang, F., Helian, N., & Akanmu, G. (2013). User-priority guided min-min scheduling algorithm for load balancing in cloud computing. *2013 national conference on parallel computing technologies (PARCOMPTECH)* (pp. 1–8). IEEE.

[6] Ghosh, T. K., Goswami, R., Bera, S., & Barman, S. (2012). Load balanced static grid scheduling using max-min heuristic. *2012 2nd IEEE international conference on parallel, distributed and grid computing* (pp. 419–423). IEEE.

[7] Mohapatra, H., & Rath, A. K. (2019). Fault tolerance in WSN through PE-LEACH protocol. *IET wireless sensor systems*, *9*(6), 358–365. https://doi.org/10.1049/iet-wss.2018.5229

[8] Alankar, B., Sharma, G., Kaur, H., Valverde, R., & Chang, V. (2020). Experimental setup for investigating the efficient load balancing algorithms on virtual cloud. *Sensors*, *20*(24), 7342. https://doi.org/10.3390/s20247342

[9] Rahmika, A. R., Tahir, Z., Paundu, A. W., & Zainuddin, Z. (2023). Web server load balancing mechanism with least connection algorithm and multi-agent system. *CommIT (communication and information technology) journal*, *17*(2), 245–258. https://doi.org/10.21512/commit.v17i2.8872

[10] Yang, M., Wang, H., & Zhao, J. (2015). Research on load balancing algorithm based on the unused rate of the cpu and memory. *2015 fifth international conference on instrumentation and measurement, computer, communication and control (IMCCC)* (pp. 542–545). IEEE.

[11] Ma, C., & Chi, Y. (2022). Evaluation test and improvement of load balancing algorithms of nginx. *IEEE access*, *10*, 14311–14324.

[12] Mohapatra, H., & Rath, A. K. (2019). Detection and avoidance of water loss through municipality taps in India by using smart taps and ICT. *IET wireless sensor systems*, *9*(6), 447–457. https://doi.org/10.1049/iet-wss.2019.0081

[13] Alhaidari, F., & Balharith, T. Z. (2021). Enhanced round-robin algorithm in the cloud computing environment for optimal task scheduling. *Computers*, *10*(5), 63. https://doi.org/10.3390/computers10050063

[14] Pradhan, P., Behera, P. K., & Ray, B. N. B. (2016). Modified round robin algorithm for resource allocation in cloud computing. *Procedia computer science*, *85*, 878–890. https://doi.org/10.1016/j.procs.2016.05.278

[15] Mohapatra, H., & Rath, A. K. (2020). Fault-tolerant mechanism for wireless sensor network. *IET wireless sensor systems*, *10*(1), 23–30. https://doi.org/10.1049/iet-wss.2019.0106

[16] Kaur, R., & Luthra, P. (2014). Load balancing in cloud system using max min and min min algorithm. *International journal of computer applications*, *975*, 8887.

[17] Kumar, P., & Kumar, R. (2019). Issues and challenges of load balancing techniques in cloud computing: A survey. *ACM computing surveys (CSUR)*, *51*(6), 1–35. https://dl.acm.org/doi/abs/10.1145/3281010

[18] Mohapatra, H., & Rath, A. K. (2020). Survey on fault tolerance-based clustering evolution in WSN. *IET networks*, *9*(4), 145–155. https://doi.org/10.1049/iet-net.2019.0155

[19] Martin, R., Menth, M., & Hemmkeppler, M. (2007). Accuracy and dynamics of multi-stage load balancing for multipath internet routing. *2007 IEEE international conference on communications* (pp. 6311–6318). IEEE.

[20] Mohapatra, H., & Rath, A. K. (2019). Fault tolerance through energy balanced cluster formation (ebcf) in WSN. *Smart innovations in communication and computational sciences: proceedings of icsiccs-2018* (pp. 313–321). Springer.

[21] Mohapatra, H., & Rath, A. K. (2022). IoE based framework for smart agriculture: Networking among all agricultural attributes. *Journal of ambient intelligence and humanized computing*, *13*(1), 407–424. DOI:10.1007/s12652-021-02908-4

[22] Derakhshan, M., & Bateni, Z. (2018). Optimization of tasks in cloud computing based on max-min, min-min and priority. *2018 4th international conference on web research (ICWR)* (pp. 45–50). IEEE.

[23] Mohapatra, H., & Rath, A. K. (2021). An IoT based efficient multi-objective real-time smart parking system. *International journal of sensor networks*, *37*(4), 219–232. https://doi.org/10.1504/IJSNET.2021.119483

[24] Khan, B. U. I., Olanrewaju, R. F., Morshidi, M. A., Mir, R. N., Kiah, M. L. B. M., & Khan, A. M. (2022). Evolution and analysis of secure hash algorithm (Sha) family. *Malaysian journal of computer science*, *35*(3), 179–200. https://doi.org/10.22452/mjcs.vol35no3.1