Load Balancing and Its Algorithms: Round Robin, Weighted Round Robin, Least Connections, and IP Hashing.

Sahiti Vankayalapati
Department of Electronics and Communication engineering
Koneru lakshmaiah educational foundation
Hyderabad, India
Dr.sahiti@klh.edu.in
A.VAISHNAVI
Department of Electronics and Communication engineering
Koneru lakshmaiah educational foundation
Hyderabad, India
M.HUSSAIN BASHA
Department of Electronics and Communication engineering
Koneru lakshmaiah educational foundation
Hyderabad, India
T.THARUN TEJ
Department of Electronics and Communication engineering
Koneru lakshmaiah educational foundation
Hyderabad, India
G.NIKITHA
Department of Electronics and Communication engineering
Koneru lakshmaiah educational foundation
Hyderabad, India

Abstract—. With the advent of high-availability computing and the explosion in digital traffic, load balancing has become essential for optimal scalability, performance, and reliability in distributed systems. It is a simple yet efficient technique employed to optimize resource usage, reduce response times, and enhance application availability and fault tolerance. In this study, major load-balancing algorithms such as Round Robin, Weighted Round Robin, IP Hashing, and Least Connections were studied. Each of them routes client requests to servers in order to prevent overload and improve response times but differs in strategy, effectiveness, and responsiveness to fluctuating workloads. Round Robin distributes requests uniformly, but Weighted Round Robin is fair by considering server capacity. IP Hashing preserves session persistence by routing users consistently based on their IP addresses. Least Connections, the most responsive of the algorithms, adapts to actual time server loads and is thus best suited for variable traffic systems. Through a comparative assessment and graphical performance metrics, this study demonstrates the strengths and weaknesses of each method, providing insight into their use in different infrastructure conditions. The paper ends with recommendations for choosing the right load-balancing method based on certain system needs, particularly for cloud systems and dynamic network conditions.

Keywords: Load Balancing, Session Persistence, Distributed System, Dynamic Load distributions

1. Introduction

The rapid development of distributed systems, cloud computing, and web applications with high traffic has significantly increased the need for intelligent and effective load balancing mechanisms. With digital platforms growing to serve millions of users simultaneously, it is imperative to ensure that no server is loaded while optimizing resource utilization, reducing response times, and increasing system reliability. Load balancing[1][2] solves these issues by sending incoming client requests to multiple servers, improving the performance, fault tolerance, and scalability aspects that are vital to modern cloud-native and large-scale applications. Where there is quick-changing traffic volume and pattern, load balancing is clearly required. Effective load-balancing methods offer seamless service in spite of unexpected traffic bursts or server failure by reacting to real-time situations. Recent load-balancing algorithms[3] not only distribute requests but also take into account server loading, traffic type, session stickiness, and resource availability, and thus play an important role in allowing smooth user experiences and robust system operations. This article offers a comprehensive discussion of four prominent load-balancing algorithms[4][5] widely used in contemporary systems: Round Robin, Least Connections, Weighted Round Robin, and IP Hashing. These algorithms operate on different principles and address various operational needs. Round Robin is a simple, circular selection algorithm, which is optimal for homogeneous server environments. Weighted Round Robin is one step ahead of it in the sense that it considers server capacity while making the selection so that heterogeneous server clusters are made equitable. Least Connections dynamically routes traffic to the server with the minimum active sessions and thus is optimal for workloads with variable session durations. Conversely, IP Hashing maintains session persistence by routing client requests to a specific server based on the client's IP address, a critical process in stateful applications like multiplayer games or banking over the internet. The purpose of this study is to provide a comparative assessment of these algorithms and examine their performance metrics, such as scalability, flexibility, fault tolerance, session affinity, and simplicity of implementation. In this manner, we attempt to compare the

pros and cons of both approaches and make concrete recommendations on applying them to different kinds of network infrastructure. Finally, this paper offers recommendations for system architects and engineers in selecting the most appropriate load-balancing algorithm based on some requirements of the system, particularly for cloud infrastructures and dynamic networks.

2. Background and Motivation

As distributed computer systems grow in size, complexity, and diversity, the effective management and distribution of network traffic is totally crucial. In earlier computing configurations, load balancing was quite an easy task, typically comprising the distribution of work across a small set of servers. With growths in system size and demands, more advanced techniques are now necessary to address scalability, fault tolerance, and diverse server capacities. This study is motivated by the imperative demand for getting a well-informed view of the fundamental concepts, benefits, and constraints of commonly used load-balancing methods. In the rapidly changing age of web applications, cloud computing, and shifting traffic directions, it is imperative for engineers, system designers, and developers to know how these algorithms function and where they are optimally applied. Building efficient systems requires not only knowledge about load-balancing options[6][7] but also the ability to map algorithmic behavior against application needs and infrastructure limitations. Next-generation load-balancing software must be very versatile, being able to service a wide range of workloads, from stateless web applications and long client sessions to varied server configurations. How good a load-balancing approach is can largely be credited to its ability to adapt to varying network conditions, provide session persistence, and optimize utilization of server resources. With increasing distribution and interconnectivity, the need for load balancing in facilitating smooth operation and uninterrupted services is increasingly speeding up. By careful examination of several of the best load-balancing techniques[8], this study aims to improve system design decision-making through increased understanding of how different techniques work, what their strengths are, and what compromises they make.

.

3. Algorithms

3.1 *Round-Robin Algorithm*

The round-robin algorithm[9] is an efficient and elegant most common load balancing technique used in distributed computing and network. It's a basic concept of allocating the incoming requests of the clients sequentially and cyclically to an available set of servers so that a server would service roughly an equal number of requests in the long term. The ease of this method makes it particularly well-suited to those applications wherein ease of installation, ease of operation, and low overhead are an absolute necessity. For operational purposes, the round-robin algorithm[10] takes its input on a list of back-end servers.

When new requests arrive, the load balancer sends them to the next server of the list. When it comes to the end of the list, the mapping becomes the first server and the loop repeats. If one had servers A, B, C, and D, for example, the first request would be sent to server A, the second to B, the third to C, the fourth to D, and the fifth to A. This will evenly and regularly distribute requests, of most help when traffic is high, as it will not allow any single server to be a bottleneck. The strength of the Round Robin[11] is evident in homogeneous systems, where the processing power, memory, and network bandwidth of every server are comparable. The algorithm can readily distribute the load in such cases, thereby making the resource utilization and throughput of the system as high as possible. As a stateless algorithm, Round Robin does not monitor the load or state of specific servers, thereby lowering computational overhead and latency. This statelessness also means that it is extremely scalable; adding or removing servers from the pool involves very little reconfiguration, as new servers are merely added to the rotation.

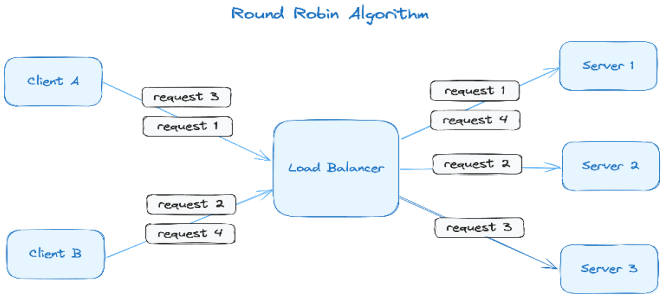


Fig. 1.1

Fig. 1.1 illustrates the function of a load balancer using the round-robin algorithm. This shows how incoming client requests are evenly distributed across multiple servers. Clients A and B send several requests to the load balancer, which then forwards each request to the servers in circular order. For example, Request 1 goes to Server 1, Request 2 to Server 2, and Request 3 to Server 3, and the cycle repeats with Request 4 going back to Server 1. This ensures that all servers handle an equal number of requests over time, thereby promoting fairness and optimal resource utilization. The diagram effectively represents how round-robin scheduling enables an efficient load distribution in distributed systems.

Advantages and Operational Benefits

The main benefits of the round-robin algorithm are its simplicity and implementation ease. The algorithm needs

fewer computational resources, thus well-suited for low to moderately trafficked networks and also for systems with balanced servers. The algorithm is most ideal in scenarios where session persistence is not a concern and incoming requests are usually of similar size and processing rates.

Limitations and Challenges

The round-robin algorithm makes the assumption that all the servers are of equal processing capacity, which is often not feasible in the modern computing setup. The assumption results in inefficient resource usage, particularly if the servers are of different processing power or incoming requests are of different complexity. Also, because the algorithm is not considering the load or health of the servers that are already available, the algorithm could be overloading certain servers if the requests are unevenly distributed. Moreover, Round Robin is not session persistence-supportive and hence is not applicable to stateful applications that involve long-lasting conversations of the user with the server.

Application Scenarios

Round Robin is used in most stateless applications whose each request is independent and not reliant on the previous interactions. It is best suited for use in scenarios such as handling static assets, handling API calls, or load distribution where all the calls have roughly the same size and complexity. Environments in such cases optimize with the simplicity of this algorithm and close to equal distribution of loads, provided server variation and session persistence are not major concerns.

3.2 Least Connections Algorithm

The Least Connections method[12] is dynamic and flexible load balancing mechanism that aims to optimize the allocation of client requests across an array of servers based on their current workload. As opposed to static scheduling algorithms such as Round Robin that route the requests in a pre-defined order regardless of the servers' states, Least Connections continues to track the current active connections on each server and routes new requests to the server with the fewest active connections when the request arrives. This solution is especially useful in those environments where client session duration and resource utilization are significantly varied. For instance, in web applications or web services where requests are short and simple at one end and long and resource-hungry at the other, the least-connection algorithm avoids any server from being a bottleneck. It offers better resource balancing through routing of new connections to less busy servers, thus keeping the system extremely responsive and available. The least-connection algorithm[12] works with the load balancer maintaining up-to-date counts of present connections on every backend server. When the load balancer receives a new client request, it looks at such counts and sends the request to the server having the smallest count. In case there are multiple servers that have the smallest count, the algorithm has a second policy, e.g., Round Robin, to decide between them. This dynamic process enables the system to respond instantly when sessions start and end, thereby producing a more equitable and effective load balancing, particularly as traffic patterns change.

Empirical and real-world tests have proven that the least-connection algorithm performs better with an increasing number of concurrent connections. For example, when subjected to high-load testing with thousands of concurrent requests, servers handled by the least-connection algorithm recorded lower average response times than those handled by the round-robin approach. This is due to the fact that servers that finish their work more rapidly become available for new connections earlier, thus enhancing overall throughput and lowering latency.

.

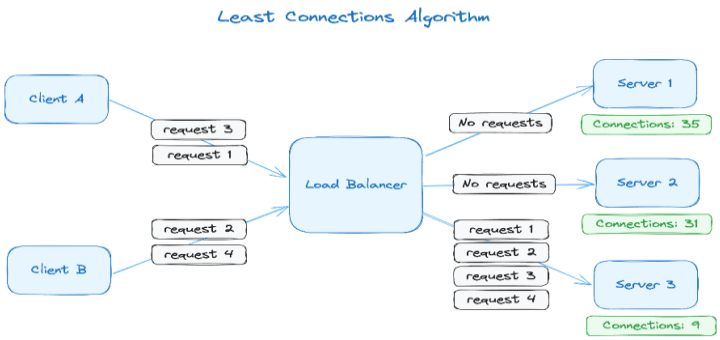


Fig. 1.2

Figure 1.2 illustrates the working principle of the least-connection load-balancing algorithm. In this setup, two clients, referred to as Client A and Client B, send numerous requests to a central Load Balancer. Then The Load Balancer checks the number of active connections already being handled by each server before distributing the incoming requests. In our example, Servers 1 and 2 already have 35 and 31 active connections, respectively, whereas Server 3 is serving a mere nine active connections. Therefore, all new requests (Requests 1 to 4) are serviced by Server 3, which has the lightest load. This method provides a dynamic and effective distribution of requests by directing traffic to the server with the least number of active connections, hence optimizing system performance overall.

.

Advantages and Operational Benefits

The Least Connections load-balancing technique is characterized by its dynamism in adjusting according to real server load. It sends new requests to the server with the least number of active connections, so no individual server gets overloaded. This leads to better performance, particularly in systems where request latencies are different, like database access or streaming media programs.

Drawbacks and Challenges

This algorithm has the disadvantage of the overhead needed to keep track of active connections on every server at all times. This is extra processing, which, in systems with high traffic, can be demanding. Also, like the round-robin algorithm, it does not have inherent session persistence, so it is less appropriate for applications that rely on the maintenance of user-session states.

Application Scenarios

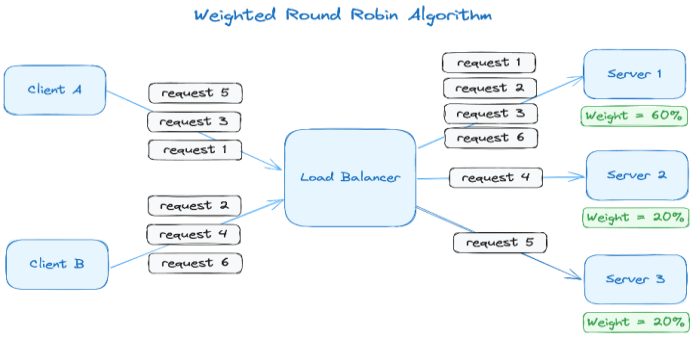
This algorithm is especially useful where request lengths are random, e.g., database systems, online video sites, and those applications with random user interaction time.

Practical Deployment Example

A very good example is YouTube, where users stream video of varying sizes in terms of length. The Least Connections approach ensures that most recently accessed videos are delivered by the least busy servers at the time, so the system will operate as smoothly and efficiently as possible.

3.3 Weighted Round-Robin Algorithm

Weighted Round Robin (WRR)[13] is a complex load-balancing scheme that improves upon the basic round-robin technique by including weights for servers. With this policy, every server in the backend pool has a numeric weight that reflects its capacity to handle or process requests. The weights typically are designated by the network administrator based on CPU power, memory, bandwidth, or past performance statistics. For instance, an excellent server may be assigned a weight of 100, and a poor server may be assigned a weight of 50, so that the former will process twice the requests than the latter. The primary behaviour of WRR[14] is round-robin in nature as that of the basic Round Robin. But rather than allocating requests in proportion to each server, WRR allocates a sequence of requests to a server based on its weight allocated per pass. An example would be with three servers allocated weights of 5, 3, and 2, the algorithm would allocate ten requests per pass as follows: five to Server 1, three to Server 2, and two to Server 3, and again. Such a method allows for wider use of high-capacity servers, optimizing the load to servers' maximum capacities while limiting possible overloading of low-capacity nodes. WRR is highly useful in homogeneous environments when servers vary significantly in hardware setup or network bandwidth. Optimizing request to server capacity, WRR optimizes the resources utilization, improves the system throughput, and minimizes response time. It can be applied to uses such as content delivery networks (CDNs) with differently sized edge servers or database clusters with differently sized replicas. While useful, WRR[15] is not perfect. The algorithm must assign precise and timely weights to all the servers. As server capacities fluctuate due to hardware upgrades, failures, or resource contention, weights need to be manually updated to ensure that they can continue to work with optimum efficiency. The static properties of WRR make it non-self-tuning with respect to server health and load changes at run time, and this could be a limiting factor in high-variability environments. Second, WRR itself does not natively handle session persistence; as a consequence, requests issued from the same client may or may not receive responses from the same server, unless special means are taken to support it. For some mitigation of these drawbacks, dynamic algorithms have been considered for WRR. These improved algorithms recompute server weights at regular intervals based on real-time performance statistics like CPU utilization, memory usage, and response times for requests. Dynamically adjusting weights, these algorithms are able to react more effectively to changing circumstances, enhancing load distribution and system stability.



.Fig. 1.3

Fig. 1.3 illustrates the Weighted Round Robin (WRR) algorithm employed for load balancing. In this, clients A and B make a large volume of requests for services. These are passed on to a load balancer, who sends these to three different servers: Servers 1, 2, and 3. WRR contrasts with the normal round robin in that in WRR each one of the servers is given a weight based on its capability. Here, Server 1 gets 60% weightage and Servers 2 and 3 receive 20% weightage each. This makes Server 1 handle more requests and utilize its larger capacity to full levels. Requests 1, 2, 3, and 6 are handled by Server 1 and request 4 by Server 2 and request 5 by Server 3. WRR also achieves maximum performance under varying capacities of the servers since WRR never allows the higher-capacity servers to get less of the load.

Benefits and Working Advantages

Weighted Round-Robin is an enhanced version of the original round-robin algorithm and utilizes weights assigned to each server in relation to its capacity. It offers chances to more powerful servers to serve more requests than less powerful servers. Thus, it maximizes the utilization of resources in multi-hardware or virtual machines setups by assigning loads according to the capacity of each server

Limitations and Challenges

A significant limitation of this method is that it's hardcoded on weights. The weights will never be dynamic to real-time server performance, health, and state of the load. If the server

bursts or crashes suddenly in traffic, the system will not respond unless updated manually with altered weights. Also, session persistence is not provided by nature, and hence it is not so suitable for scenarios where user sessions need to be maintained corresponding to a particular server.

Application Scenarios

This method is best suited to cloud or hybrid environments, where server capacities are different, including• Web hosting providers, Content Delivery Networks (CDNs), Cloud platforms with blended virtual machines or physical machines

Practical Deployment Example

In systems such as Amazon Web Services (AWS) and Microsoft Azure, instances can have different computing capacities. Weighted Round-Robin guarantees that high-capacity instances (e.g., 8-core VMs) receive more traffic than less capacity instances (e.g., 2-core VMs), thus achieving maximum utilization of resources. For instance, AWS Elastic Load Balancing (ELB) can dynamically direct traffic according to the capability of a server to ensure system efficiency

3.4 IP Hash Algorithm

The IP Hash algorithm[16] maps incoming client requests to backend servers using the hash value of the client's IP address. This approach provides a consistent routing of requests from the same IP address to the same server . Thus, it provides natural session persistence, and hence, it is best suited for stateful applications that require user-specific data.

Operation Principle

When a client makes a request, the load balancer takes the client's IP address (occasionally, both source and destination IPs are utilized) and hashes that value using a hash function. The hash output is then routed to one of the servers in the backend pool. For example, if there are three servers and the hash of a client's IP address yields a value of two, then the request is routed to the third server (using zero-based indexing). This mapping remains consistent for all subsequent requests from the same client IP, thus achieving session stickiness without the need for a load balancer to maintain the explicit session state information.

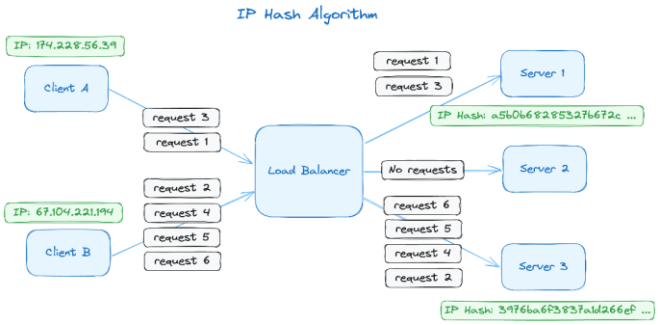


Fig. 1.4

Fig.1.4 illustrates how the IP Hash algorithm distributes incoming client requests to servers. In this method, the IP address of the client is processed using a hash function to generate a hash key. This key was then used to assign requests from the client to a specific server. For example, client A with IP 174.238.56.39 is consistently routed to Server 1 based on its hash value, whereas client B with IP 67.104.221.194 is directed to Server 3. This ensures that requests from the same client always reach the same backend server, thereby providing session persistence and stability in connection handling.

Advantages and Operational Benefits

The IP Hash load-balancing algorithm is a simple and effective technique to implement session persistence. This guarantees that every client request, depending on the IP address of the client, will always be forwarded to the same backend server. This is particularly useful in cases where a consistent client-server session needs to be maintained, particularly when the client's IP address does not shift frequently.

Limitations and Challenges

Although simple, an IP Hash can result in uneven server loads. If a small number of client IPs— for example, those for large organizations or networks—send high traffic, they can bog down one server. In addition, the modification of the server pool (such as adding or removing servers) interferes with the current mapping of the IP to the server. This destroys session persistence and potentially degrades user experience in session-sensitive applications.

Application Scenarios

The IP Hash is well-suited for applications that require sustained user sessions in the long term. Some of these include: Shopping cart tracking and user activity on e-commerce websites, Banking and financial institutions where session continuity is important, Gaming apps where connection states need to be maintained consistently, Content Delivery Networks (CDNs) deliver cached content from the same edge server for multiple client requests.

Practical Deployment Example

On e-commerce websites, such as Amazon or Flipkart, maintaining a user session is essential for functionality, such as shopping carts, wish lists, login sessions, and payment flows. An IP Hash ensures that a customer's actions are directed to the same backend server, resulting in a smooth and consistent shopping experience throughout a session.

4. *Observation*

The Fig.1.5 presents a comparative analysis of load balancing algorithms, highlighting their performance across three key parameters [2][3][4]: Load Awareness, Session Persistence, Complexity.

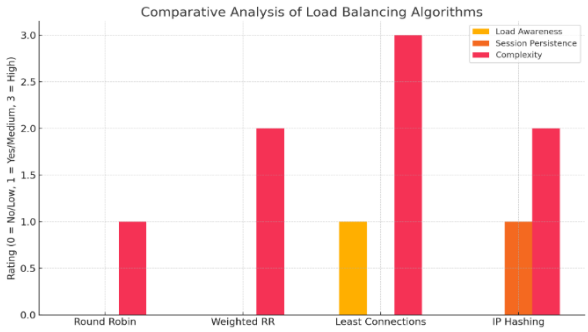


Fig.1.5

The analysis included four load-balancing algorithms: Round Robin, Weighted Round Robin (WRR), Least Connections, and IP Hashing.

Focus on the round-robin algorithm, The Load Awareness in Round Robin scored low in load awareness, as shown by a rating of 1. This means that it does not consider the current server load when distributing traffic. Each incoming request is simply assigned to the next server in the list regardless of how busy the server is. Session Persistence: It receives a rating of 0, indicating that Round Robin does not maintain session persistence. In other words, it does not ensure that the user requests are directed to the same server across multiple sessions. Complexity: Round Robin has a low complexity (rated 1), which makes it simple and easy to implement. This is one of the reasons why it is widely used in basic load balancing scenarios.

Overall Interpretation

From the chart, it is clear that although Round Robin is simple to implement, it lacks sophistication in handling real-time loads and maintaining user sessions. This makes it less suitable for high-demand or stateful applications, where session persistence and load-based decision making are critical.

In contrast, algorithms such as Least Connections and IP Hashing offer better session persistence and load awareness, but come with increased complexity.

5. *Comparative Analysis*

The table. 1.1 below summaries the all the parameters for the load balancing algorithms, such as Round Robin, Weighted Round Robin, Least Connections and IP Hashing.

Table. 1.1

Parameter	*Round Robin[9]*	*Weighted Round Robin[13]*	*Least Connections[12]*	*IP Hashing[16]*
Algorithm Type	Static	Static	Dynamic	Static
Load Distribution	Cyclical, uniform	Cyclical, weight-proportional	Connection-count-based	Hash-based
Session Persistence	No	No	No	Yes
Server Capacity Consideration	No	Yes (predefined weights)	Optional (via weighted variants)	No
Use Cases	Homogeneous server farms	Heterogeneous infrastructures	Variable session durations	Stateful applications

For the parameter of Algorithm Type, the Static Algorithms (Round Robin, Weighted Round Robin, IP Hashing) work on predefined rules without responding to actual system conditions in real-time. Round Robin[9][10][11] sends traffic sequentially to servers, whereas Weighted Round Robin[13] does the same but with requests being distributed according to pre-defined server weights[13]. IP Hashing[16] distributes traffic using client IP addresses to consistently map traffic to particular servers consistently. These are appreciated for being simple and requiring low computational overhead but lack the ability to respond to sudden traffic spikes or server failures. Whereas The least connections dynamically track server loads and distribute new requests to the server with the fewest active connections [12]. It differs from static approaches in that it adapts to changing traffic patterns in real time, making it well-suited for unpredictable environments. However, it introduces complexity by necessitating continuous monitoring of server states.

The Load Distribution in Round Robin, serves the requests in a constant cyclical pattern. If there are three servers, request 1 is to Server A request 2 to Server B, and request 3 to Server C, with the cycle repeating. This method assumes that all servers are equally capable and have the same processing level. Whereas The Weighted Round Robin is an extension of the round-robin method with the addition of the server weights. For example, a server with a weight of three processes three requests per request processed by a server with a weight of 1. It works well in heterogeneous capacity environments but must be updated manually if the server capacities are altered. In the Least Connections[12], new

requests are routed to a server with the lowest number of active connections. If Server A is active with five sessions and Server B has only two sessions, then new requests are routed to Server B. This is best for services with mixed session lengths, such as video streaming sites. Lastly, in the IP Hashing, The IP address of the client is passed through a hash function (such as CRC32) to determine the server that should service the traffic. For instance, IP address 192.168.1.1 can always be resolved to Server C. This guarantees that a client session always goes to the same server, but can lead to imbalances if many users share the same IP range.

When we compare the session persistence for each algorithm, Round Robin, Weighted Round Robin, and Least Connections do not inherently support session persistence. This indicates that a client's requests may be sent to servers varying between sessions, which can break stateful applications. IP Hashing, however, provides session persistence by mapping each client to a particular server using an IP-based hash. This is essential in applications such as e-commerce shopping carts or user authentication servers, where user sessions must be preserved. However, this complicates traffic rebalancing if one of the servers fails.

When it comes to server capacity, Round Robin/IP Hashing assume that all servers have equal performance capacity. Consequently, high- and low-powered servers would be given the same traffic, which can cause bottlenecks. While the Weighted Round Robin considers server capacity by having fixed weights. For instance, if a server has a very high CPU, it receives a weight of 100 and an in-between server receives a weight of 50. These weights must be manually adjusted if the hardware setup is to be altered. The Least Connection algorithm only considers the number of active connections, without considering the power of the servers. A powerful server with ten connections may still be bypassed by a less powerful server with five connections. Alternatives, such as Weighted Least Connections, solve this problem, but at the expense of increased complexity.

Each algorithm is suitable for specific types, Round Robin is best suited for environments where all servers have comparable capacities and perform uniform workloads, such as hosting static website content. While Weighted Round Robin is best suited for mixed-server setups such as hybrid cloud deployments, particularly when traffic patterns are predictable and stable. Least Connections is effective for dynamic environments where the session duration is highly variable, such as database services or real-time analytics platforms. IP Hashing is Critical to applications that require stable, continuous sessions, such as online banking, multiplayer games, or any other stateful service.

6. Conclusion

Load balancing is an useful element in delivering the performance, reliability, and scalability of modern Web applications. This article compared four o load balancing algorithms—Round Robin, Weighted Round Robin, Least Connections, and IP Hashing—and their specific advantages in various network environments. Round Robin and Weighted Round Robin are easy to implement and efficient in distributing traffic with minimal complexity, and therefore suitable for systems wherein ease of implementation and low overhead are taken into account. Least Connections offer a more dynamic system by considering the load each server is experiencing now, and therefore allow resources to be used more efficiently with fluctuating traffic patterns. Whereas, IP Hashing is extremely helpful for when session persistence is required as it assigns users to the same server persistently based on their IP address. Having a good knowledge of these algorithms is important in designing and managing high-performance systems capable of processing fluctuating traffic loads optimally. Choosing the right load-balancing strategy depending on some application needs, such as deployment simplicity, reaction to immediate loads, or session persistence, can significantly enhance both server performance and user satisfaction.

REFERENCES

[1] K. A. Nuaimi, N. Mohamed, M. A., Nuaimi, J., Al-Jaroodi, "A Survey of Load Balancing in Cloud Computing: Challenges and Algorithms," 2012 Second Symposium on Network Cloud Computing and Applications, London, UK, 2012, pp. 137-142, doi: 10.1109/NCCA.2012.29.

[2] Shah, Nadeem, and Mohammed Farik. "Static load balancing algorithms in cloud computing: challenges & solutions." International Journal of Scientific & Technology Research.

[3] Sharma, Sandeep, Sarabjit Singh, and Meenakshi Sharma. "Performance analysis of load balancing algorithms." World academy of science, engineering and technology.

[4] H. Rahmawan and Y. S. Gondokaryono, "The simulation of static load balancing algorithms," 2009 International Conference on Electrical Engineering and Informatics, Bangi, Malaysia, 2009, pp. 640-645, doi: 10.1109/ICEEI.2009.5254739.

[5] Beniwal, Payal, and Atug Garg. "A comparative study of static and dynamic load balancing algorithms." International

journal of advance research in computer science and management studies.

[6] Sidhu, Amandeep Kaur, and Supriya Kinger. "Analysis of load balancing techniques in cloud computing." International Journal of computers & technology 4.2 (2013).

[7] Tripathi, Aanjey Mani, and Sarvpal Singh. "A literature review on algorithms for the load balancing in cloud computing environments and their future trends." Computer Modelling & New Technologies 21.1 (2017).

[8] Deepa, T., and Dhanaraj Cheelu. "A comparative study of static and dynamic load balancing algorithms in cloud computing." 2017 international conference on energy, communication, data analytics and soft computing (ICECDS). IEEE, 2017.

[9] Hidayat, Taufik, Yasep Azzery, and Rahutomo Mahardiko. "Load balancing network by using round Robin algorithm: a systematic literature review." Jurnal Online Informatika 4.2 (2019).

[10] Youm, Dong Hyun, and Ravi Yadav. "Load balancing strategy using round robin algorithm." Asia-pacific Journal of Convergent Research Interchange 2.3 (2016).

[11] Matarneh, Rami J. "Self-adjustment time quantum in round robin algorithm depending on burst time of the now running processes." American Journal of Applied Sciences 6.10 (2009).

[12] Mustafa, Mustafa ElGili. "LOAD BALANCING ALGORITHMS ROUND-ROBIN (RR), LEASTCONNECTION, AND LEAST LOADED EFFICIENCY." Computer Science & Telecommunications 51.1 (2017).

[13] Saidu, Ibrahim, et al. "A load-aware weighted round-robin algorithm for IEEE 802.16 networks." EURASIP Journal on Wireless Communications and Networking 2014 (2014).

[14] Katangur, Ajay, Somasheker Akkaladevi, and Sadiskumar Vivekanandhan. "Priority weighted round robin algorithm for load balancing in the cloud." 2022 IEEE 7th international conference on smart cloud (SmartCloud). IEEE, 2022.

[15] Vyakaranal, Shashidhara B., and Jayalaxmi G. Naragund. "Weighted round-robin load balancing algorithm for software-defined network." Emerging Research in Electronics, Computer Science and Technology: Proceedings of International Conference, ICERECT 2018. Singapore: Springer Singapore, 2019.

[16] Cao, Zhiruo, Zheng Wang, and Ellen Zegura. "Performance of hashing-based schemes for internet load balancing." Proceedings IEEE INFOCOM 2000. Conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064). Vol. 1. IEEE, 2000.