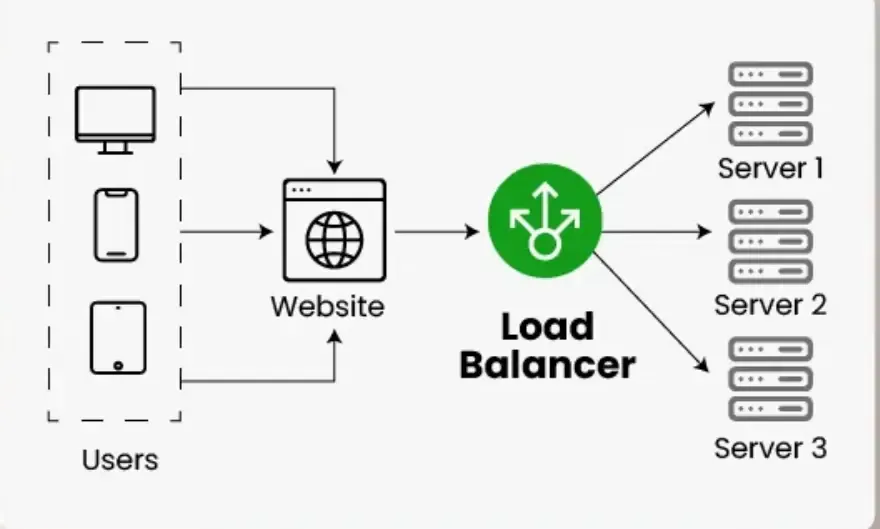
LOAD BALANCING

**What is Load Balancer & How Load Balancing works?**

A load balancer is a networking device or software application that distributes and balances the incoming traffic among the servers to provide high availability, efficient utilization of servers and high performance.



* Works as a "traffic cop" routing client requests across all servers
* Ensures that no single server bears too many requests, which helps improve the performance,[reliability](https://www.geeksforgeeks.org/system-design/reliability-in-system-design/) and [availability](https://www.geeksforgeeks.org/system-design/availability-in-system-design/) of applications.
* Highly used in cloud computing domains, data centers and large-scale web applications where traffic flow needs to be managed.

**Issues without Load Balancer?**

Several problem will occur without the load balancer, these are:

A diagram of a load balancing system

AI-generated content may be incorrect.

* **Single Point of Failure: I**f the server goes down or something happens to the server the whole application will be interrupted and it will become unavailable for the users for a certain period. It will create a bad experience for users which is unacceptable for service providers.
* **Overloaded Servers:** There will be a limitation on the number of requests that a web server can handle. If the business grows and the number of requests increases the server will be overloaded.
* **Limited Scalability**: Without a load balancer, adding more servers to share the traffic is complicated. All requests are stuck with one server and adding new servers won’t automatically solve the load issue.

A diagram of a diagram

AI-generated content may be incorrect.

**Key characteristics of Load Balancers**

* **Traffic Distribution:** To keep any one server from becoming overburdened, load balancers divide incoming requests evenly among several servers.
* **High Availability:** Applications' reliability and availability are improved by load balancers, which divide traffic among several servers. The load balancer reroutes traffic to servers that are in good condition in the event that one fails.
* **Scalability:** By making it simple to add servers or resources to meet growing traffic demands, load balancers enable horizontal scaling.
* **Optimization:** Load balancers optimize resource utilization, ensuring efficient use of server capacity and preventing bottlenecks.
* **Health Monitoring:** Load balancers often monitor the health of servers, directing traffic away from servers experiencing issues or downtime.
* **SSL Termination:** Some load balancers can handle SSL/TLS encryption and decryption, offloading this resource-intensive task from servers.

**How Load Balancer Works?**

A diagram of a computer server

AI-generated content may be incorrect.

* **Receives Incoming Requests**: When users try to access a website or application, their requests first go to the load balancer instead of directly to a server.
* **Checks Server Health**: The load balancer continuously monitors the status of all servers. It checks which servers are healthy and ready to handle requests.
* **Distributes Traffic**: Based on factors like server load, response time or proximity, the load balancer forwards each request to the most appropriate server. This helps avoid any server getting overloaded.
* **Handles Server Failures**: If a server goes down or becomes unresponsive, the load balancer automatically stops sending traffic to that server and redirects it to others that are still functioning properly.
* **Optimizes Performance**: By spreading traffic efficiently and using healthy servers, load balancers improve overall performance and reduce delays.

**Benefits of Load Balancer**

* **Better Performance** - Distributes traffic across servers so no single server gets overloaded, reducing downtime and improving speed.
* **Scalability** - Works with auto-scaling to add more servers during high traffic and remove them when traffic is low.
* **Failure Handling** - Detects unhealthy servers and redirects traffic to healthy ones, keeping the system available.
* **Prevents Bottlenecks** - Handles sudden spikes in traffic smoothly by spreading requests evenly.
* **Efficient Resource Use** - Ensures all servers share the workload fairly.
* **Session Persistence** - Can maintain user sessions so apps that need continuous sessions (like shopping carts) work properly

**Challenges of Load Balancer**

* **Single Point of Failure** - If the load balancer itself goes down, it can disrupt traffic flow unless a backup exists.
* **Cost and Complexity** - Good load balancing solutions can be expensive and require proper setup and management.
* **Configuration Issues** - Setting up correctly can be tricky, especially for complex applications.
* **Extra Overhead** - Adds a small delay since every request passes through the load balancer.
* **SSL Management** - Handling encryption (SSL termination) at the balancer can make end-to-end security more complicated.

**Different strategies for Load Balancing**

**Round Robin load balancing method**

Round-robin load balancing is the simplest and most commonly-used load balancing algorithm. Client requests are distributed to application servers in simple rotation. For example, if you have three application servers: the first client request is sent to the first application server in the list, the second client request to the second application server, the third client request to the third application server, the fourth to the first application server, and so on.

Round robin load balancing is most appropriate for predictable client request streams that are being spread across a server farm whose members have relatively equal processing capabilities and available resources (such as network bandwidth and storage).

Imagine dealing a deck of cards to players around a table. The dealer (load balancer) gives one card (client request) to the first player (server), then one to the second player and continues around the table until all players have a dealt card. Once the dealer reaches the last player, they start again with the first.

This is essentially how round robin load balancing operates. If there are three servers, the load balancer will send the fourth request to server 1. If there are 10 servers in the backend server pool, then the eleventh request starts over with server 1.

Organizations typically use the basic round robin method in environments where:

* Servers have similar specifications and processing capabilities.
* The workload consists of relatively uniform requests.
* Simplicity of setup and maintenance is a priority.
* Predictable distribution patterns are desired.

The simplicity of round robin makes it easy to implement and manage, which is why it remains one of the commonly used load balancing algorithms despite the availability of more sophisticated methods.

A diagram of a round robin load balancing

AI-generated content may be incorrect.

**How It Works:**

Imagine you have 3 servers:

* Server A
* Server B
* Server C

Incoming requests are assigned like this:

1. Request 1 → Server A
2. Request 2 → Server B
3. Request 3 → Server C
4. Request 4 → Server A  
   ... and so on.

**🧠 Key Features:**

* Simple to implement
* No need for server health or load checks (unless combined with other strategies)
* Works well when all servers have similar capacity and performance

**⚠️ Limitations:**

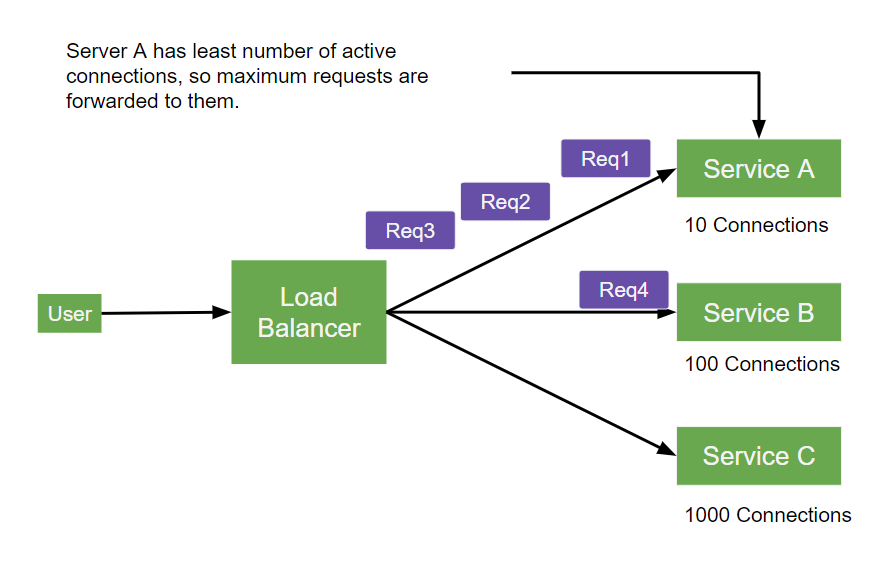
* Doesn’t account for server load — a busy server might still get requests.
* Not ideal for stateful applications unless session persistence is handled.

**🔧 Variants:**

* Weighted Round Robin: Assigns more requests to servers with higher capacity.
* Dynamic Round Robin: Adjusts based on real-time server performance.

## **Least Connections**

Least Connections is a type of dynamic [load balancing approach](https://www.geeksforgeeks.org/computer-networks/load-balancing-approach-in-distributed-system/). In this network load balancing approach the request gets distributed across the servers who have the least amount of active connections. This approach offers unique features of distributing the incoming requests or workloads in such a way that it minimizes the load present on the server. It helps to get rid of congestion and optimizes the performance. This load balancing approach is useful in scenarios where the load of the server changes over the time.



**Key Features**

* **Dynamic Distribution**: Unlike static methods like Round Robin, Least Connections adapts to real-time server loads by considering the number of active connections on each server.
* **Efficient Resource Utilization**: By directing traffic to the least-burdened server, it ensures better load distribution and minimizes congestion.
* **Scalability**: This method is particularly effective in environments with fluctuating traffic patterns or varying server capacities.

The Least Connections algorithm distributes incoming requests to the server currently serving the fewest connections. This approach ensures that busy servers are not overwhelmed further while idle servers are utilized efficiently.  
 **Advantages of Least Connections:**  
1. Dynamic Adaptation: By focusing on the number of active connections, this algorithm can adapt to varying loads across servers, making it more efficient in environments with varying request loads.  
2. Resource Optimization: It helps in balancing the load more evenly across servers, optimizing resource utilization.  
  
**Ideal Use Cases:**  
Least Connections is particularly useful in situations where there is a significant disparity in server capabilities or when the volume and nature of requests are unpredictable. It is often used in environments where server performance can fluctuate, such as cloud-based setups.’

**Random allocation**

**A diagram of a server

AI-generated content may be incorrect.**

Random allocation is a **static load balancing algorithm** that assigns incoming tasks or requests to servers randomly without considering their current load or state. This approach is simple, lightweight, and effective in many scenarios, especially when the servers have similar configurations.

**Key Principles of Random Allocation**

Random allocation can be modeled as a **balls and bins problem**, where tasks (balls) are distributed into servers (bins) uniformly at random. Each task is assigned to a server with equal probability, ensuring fairness in the long run. This method avoids the need for maintaining state information about server loads, making it computationally efficient.

The **average load** on each server is calculated as the total load divided by the number of servers. While the average load is evenly distributed, there is a small probability of deviation where some servers may receive more tasks than others. However, this probability is low and can be analyzed using mathematical tools like the **Chernoff bound**.

**Advantages of Random Allocation**

* **Simplicity**: It does not require tracking server states or loads, reducing overhead.
* **Scalability**: Works well with a large number of servers and tasks.
* **Fairness**: On average, tasks are evenly distributed across servers.

**Limitations**

* **Load Imbalance**: Random allocation may occasionally lead to uneven distribution, where some servers handle significantly more tasks than others.
* **No Adaptability**: It does not account for server capacity or current load, which can lead to inefficiencies in heterogeneous environments.

**Enhancements**

The **power of two choices** is an improvement over pure random allocation. Instead of assigning a task to a single random server, two servers are chosen randomly, and the task is assigned to the one with fewer tasks. This significantly reduces the maximum load on any server, achieving better balance while maintaining simplicity[**2**](https://www.bing.com/ck/a?!&&p=dff6e1970c1ccbe58fa6e863b822fb47551a0da795be0a58f417c53addddde80JmltdHM9MTc2MDA1NDQwMA&ptn=3&ver=2&hsh=4&fclid=329d357b-8a28-62a8-364a-23128b36631d&u=a1aHR0cHM6Ly9wYWdlcy5jcy53aXNjLmVkdS9-c2h1Y2hpL2NvdXJzZXMvNzg3LUYwOS9zY3JpYmUtbm90ZXMvbGVjNy5wZGY&ntb=1).

**Use Cases**

Random allocation is suitable for scenarios where:

* Servers have similar configurations and capacities.
* The system requires a lightweight and fast load balancing mechanism.
* The probability of load imbalance is acceptable.

In conclusion, random allocation is a practical and efficient load balancing strategy for many distributed systems. Its simplicity and effectiveness make it a popular choice, especially when combined with enhancements like the power of two choices.