## What is Load Balancing?

**Load balancing** is the technique of distributing incoming network or application traffic across multiple servers (called **backend servers** or **nodes**) to ensure **no single server** is overloaded.

It’s like having several counters at a bank — customers (requests) are directed to whichever counter is available, so service stays fast and efficient.

**Main Objectives:**

* Improved performance: Requests are shared among servers for faster response.
* High availability: If one server fails, others continue serving traffic.
* Scalability: Easy to add or remove servers without downtime.
* Efficient resource use: Keeps workloads balanced across all available resources.

**Where it's used:** Web servers, cloud applications, microservices, databases, and distributed systems.

## Load Balancing Strategies

### **1. Round Robin**

**Concept:** Each incoming request is sent to servers one by one in order, and then the cycle repeats.  
 If you have 3 servers (A, B, C), the pattern goes like:  
 → A → B → C → A → B → C → …

**Example:** If 6 users send requests, each server gets two requests, keeping things even.

**Advantages:**

* Simple to configure and understand.
* Works well when all servers have **equal processing power** and **similar request sizes**.

**Disadvantages:**

* Doesn’t consider **server load or response time**.
* If one server is slower, it still receives the same number of requests.

**When to Use:**

* For **stateless applications** (no user session dependency).
* When all servers have **similar performance capacity**.

### **2. Least Connection**

**Concept:** The load balancer sends each new request to the server with the **fewest active connections**.  
 This ensures that lightly loaded servers get more new requests.

**Example:** If:

* Server A → 5 active connections
* Server B → 2 active connections  
   Then the next request goes to **Server B**.

**Advantages:**

* Distributes load more accurately, especially when connection times vary.
* Useful for applications with **long-lived connections** (like chat apps or video streaming).

**Disadvantages:**

* Requires real-time tracking of active connections.
* Slightly more computational overhead for maintaining connection counts.

**When to Use:**

* For **dynamic workloads** where request durations vary significantly.
* Best for **stateful** or **long-duration** services.

### **3. Random**

**Concept:** Each request is assigned to a **randomly chosen server** in the pool.  
 Every server has an equal probability of receiving the next request.

**Example:** If there are 4 servers (A, B, C, D), request distribution might look like:  
 → B → A → D → A → C → B → D → C (no fixed order).

**Advantages:**

* Extremely **fast and simple** to implement.
* Works well when there are **many servers** — randomness averages out load.

**Disadvantages:**

* Can cause temporary imbalance if randomness isn’t perfect.
* Doesn’t adapt to server performance or load changes.

**When to Use:**

* When simplicity and **speed of routing** are more important than perfect balance.
* For **large clusters** where load tends to even out statistically.

## Key Benefits of Load Balancing

1. **High Availability:** Automatic rerouting if a server fails.
2. **Scalability:** New servers can be added easily.
3. **Flexibility:** Multiple balancing methods for different workloads.
4. **Optimized Resource Use:** Maximizes performance from all servers.
5. **Reduced Downtime:** Ensures continuous service during maintenance or failure.