Load Balancers

Load Balancing is the process of distributing incoming network traffic across multiple servers to ensure no single server becomes overwhelmed, improving performance, reliability, and availability.

What a load balancer does

- Distributes requests across multiple instances
- Improves **availability & fault tolerance**
- Detects unhealthy servers and stops sending traffic to them
- Enables **horizontal scaling**

Types of Load Balancers

By Network Layer (OSI Model)

Layer 4 (Transport Layer) - Network Load Balancer

- Operates at TCP/UDP level
- Routes **based on IP address and port**
- Very fast (doesn't inspect packet content)
- Can't make decisions based on HTTP headers or URLs
- Lower latency, higher throughput

Example: Routes all traffic on port 443 to web servers without looking at the actual HTTP request.

Layer 7 (Application Layer) - Application Load Balancer

- Operates at HTTP/HTTPS level
- Routes **based on content: URLs, headers, cookies, query parameters**
- Can terminate SSL/TLS connections
- More intelligent routing but slightly slower
- Enables advanced features like path-based routing

Example: Routes /api/* to API servers and /images/* to image servers.

By Implementation

Hardware Load Balancers

- Physical appliances (F5, Citrix NetScaler, Barracuda)
- Very high performance and reliability
- Expensive ($10,000-$100,000+)
- Complex to configure
- Common in large enterprises and data centers

Software Load Balancers

- Run on standard servers (HAProxy, Nginx, Apache)
- More flexible and customizable
- Much cheaper (often free/open source)
- Easier to scale and integrate with cloud infrastructure
- Modern choice for most organizations

Cloud Load Balancers

- Managed services (AWS ELB/ALB/NLB, Google Cloud Load Balancing, Azure Load Balancer)
- Fully managed—no servers to maintain
- Auto-scaling, high availability built-in
- Pay-per-use pricing
- Easiest to implement and operate

DNS Load Balancing

- Uses DNS to return different IP addresses for the same domain
- Distributes traffic at DNS resolution level
- Simple but crude (no health checks, slow to update)
- Good for geographic distribution

Load Balancing Strategies

1. **Round Robin** Distributes requests sequentially. Server 1 → Server 2 → Server 3 → Server 1... Simple but doesn't account for server load differences.

   Pros: Simple, ensures even distribution
   Cons: Doesn't account for server capacity or current load
   Best for: Servers with identical specs and similar request complexity

2. **Least Connections** Sends requests to the server with fewest active connections. Better for long-lived connections like websockets.

   Pros: Better for long-lived connections
   Cons: Requires tracking connection state
   Best for: Applications with variable request durations (chat, streaming)

3. **Least Response Time** Routes to the server responding fastest. Good for ensuring consistent user experience.

   Pros: Optimizes for user experience
   Cons: Requires active monitoring and calculation
   Best for: Ensuring consistent performance

4. **IP Hash** Routes based on client IP address, ensuring the same user always hits the same server. Useful when you need session affinity (though generally you want to avoid this).

   Pros: Session affinity without shared state
   Cons: Uneven distribution if client IPs aren't diverse, problems when servers are added/removed
   Best for: Applications requiring sticky sessions (though better to avoid this need)

5. **Weighted Distribution** Assigns different capacities to servers. Powerful servers get more traffic than weaker ones. Useful in heterogeneous environments.

   Best for: Heterogeneous server pools with different capacities

6. **Geographic/Location-Based** Routes users to the nearest server geographically. Reduces latency for global applications.

7. **Random** Randomly selects a server for each use

Pros: Very simple **Cons:** May not distribute evenly in short time periods **Best for:** Stateless applications where distribution precision doesn't matter

Health Checks

Load balancers continuously monitor server health to avoid routing traffic to failed servers.

Active Health Checks Load balancer actively probes servers at regular intervals.

```
Every 10 seconds:
- Send HTTP GET to /health endpoint
- Expect 200 OK response within 5 seconds
- If fails 3 consecutive times, mark server unhealthy
- Remove from rotation
```

Passive Health Checks Monitor actual traffic responses.

```
If server returns 5xx errors for 5 consecutive requests:
- Mark as unhealthy
- Remove from rotation
```

Health check criteria:

- HTTP status codes (200 OK = healthy)
- Response time (timeout = unhealthy)
- Custom application logic (check database connectivity, etc.)

Recovery: Once unhealthy servers start passing health checks again, they're automatically added back to the pool.

Session Persistence (Sticky Sessions)

Sometimes you need the same client to always hit the same server.

Why it's needed:

- Server stores session data in local memory (not recommended)
- Stateful applications that haven't been refactored
- WebSocket connections

Implementation methods:

Cookie-based Load balancer injects a cookie identifying which server to use.

```
Set-Cookie: SERVERID=server2
```

IP-based Use source IP hashing (same IP → same server)

Application-controlled Application generates session token that includes server identifier

Why to avoid it:

- Reduces scalability (can't freely add/remove servers)
- Uneven load distribution (one user's heavy usage impacts one server)
- Complexity when servers fail

Better alternative: Use shared session storage (Redis, Memcached, database) so any server can handle any request.

Key Features

SSL/TLS Termination Load balancer handles SSL decryption, forwards unencrypted traffic to backend servers.

Advantages:

- Offloads CPU-intensive encryption from application servers
- Centralized certificate management
- Backend servers can use plain HTTP

Connection Multiplexing Reuses connections between load balancer and backend servers for multiple client requests.

Benefit: Reduces overhead of establishing new connections

Compression Load balancer can compress responses before sending to clients.

Benefit: Reduces bandwidth and improves load times

Caching Some load balancers cache static content.

Benefit: Reduces backend load for frequently requested resources

Rate Limiting Throttle requests from specific IPs or overall traffic.

Benefit: Protection against DDoS and abuse

Web Application Firewall (WAF) Filter malicious requests (SQL injection, XSS attacks).

Benefit: Security layer before traffic reaches application