**Name:** Shivam Tiwari

**Roll No:** 5117060

**Experiment No. 6**

**AIM:** To implement a Load Balancing Algorithm.

**THEORY:**

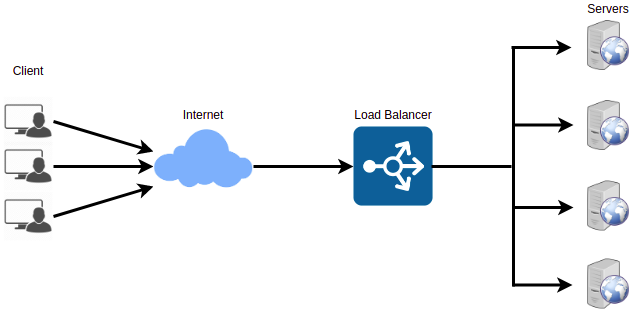
# **Load Balancing**

# 

# Load balancing is the process of distributing network traffic across multiple servers. This ensures no single server bears too much demand. By spreading the work evenly, load balancing improves application responsiveness. It also increases availability of applications and websites for users. Modern applications cannot run without load balancers. Over time, software load balancers have added additional capabilities including application security.

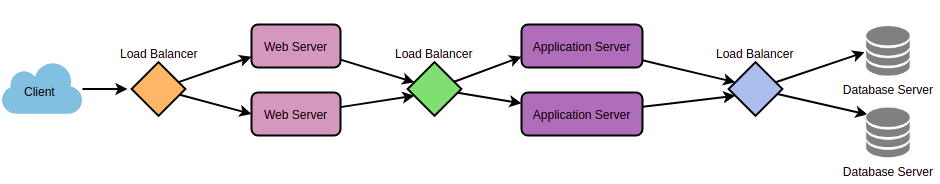
Load Balancer (LB) is another critical component of any distributed system. It helps to spread the traffic across a cluster of servers to improve responsiveness and availability of applications, websites or databases. LB also keeps track of the status of all the resources while distributing requests. If a server is not available to take new requests or is not responding or has elevated error rate, LB will stop sending traffic to such a server.

Typically a load balancer sits between the client and the server accepting incoming network and application traffic and distributing the traffic across multiple backend servers using various algorithms. By balancing application requests across multiple servers, a load balancer reduces individual server load and prevents any one application server from becoming a single point of failure, thus improving overall application availability and responsiveness.



To utilize full scalability and redundancy, we can try to balance the load at each layer of the system. We can add LBs at three places:

* Between the user and the web server
* Between web servers and an internal platform layer, like application servers or cache servers
* Between the internal platform layer and database.



## 

## 

## **Benefits of Load Balancing**

* Users experience faster, uninterrupted service. Users won’t have to wait for a single struggling server to finish its previous tasks. Instead, their requests are immediately passed on to a more readily available resource.
* Service providers experience less downtime and higher throughput. Even a full server failure won’t affect the end user experience as the load balancer will simply route around it to a healthy server.
* Load balancing makes it easier for system administrators to handle incoming requests while decreasing wait time for users.
* Smart load balancers provide benefits like predictive analytics that determine traffic bottlenecks before they happen. As a result, the smart load balancer gives an organization actionable insights. These are key to automation and can help drive business decisions.
* System administrators experience fewer failed or stressed components. Instead of a single device performing a lot of work, load balancing has several devices perform a little bit of work.

## 

## **Load Balancing Algorithms**

## **How does the load balancer choose the backend server?**

Load balancers consider two factors before forwarding a request to a backend server. They will first ensure that the server they choose is actually responding appropriately to requests and then use a pre-configured algorithm to select one from the set of healthy servers. We will discuss these algorithms shortly.

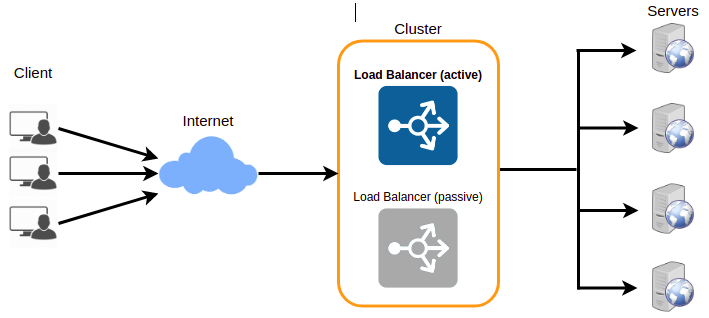
**Health Checks** - Load balancers should only forward traffic to “healthy” backend servers. To monitor the health of a backend server, “health checks” regularly attempt to connect to backend servers to ensure that servers are listening. If a server fails a health check, it is automatically removed from the pool, and traffic will not be forwarded to it until it responds to the health checks again.

There are a variety of load balancing methods, which use different algorithms for different needs:

* **Least Connection Method** — This method directs traffic to the server with the fewest active connections. This approach is quite useful when there are a large number of persistent client connections which are unevenly distributed between the servers.
* **Least Response Time Method** — This algorithm directs traffic to the server with the fewest active connections and the lowest average response time.
* **Least Bandwidth Method** - This method selects the server that is currently serving the least amount of traffic measured in megabits per second (Mbps).
* **Round Robin Method** — This method cycles through a list of servers and sends each new request to the next server. When it reaches the end of the list, it starts over at the beginning. It is most useful when the servers are of equal specification and there are not many persistent connections.
* **Weighted Round Robin Method** — The weighted round-robin scheduling is designed to better handle servers with different processing capacities. Each server is assigned a weight (an integer value that indicates the processing capacity). Servers with higher weights receive new connections before those with less weights and servers with higher weights get more connections than those with less weights.
* **IP Hash** — Under this method, a hash of the IP address of the client is calculated to redirect the request to a server.

## **Redundant Load Balancers**

The load balancer can be a single point of failure; to overcome this, a second load balancer can be connected to the first to form a cluster. Each LB monitors the health of the other and, since both of them are equally capable of serving traffic and failure detection, in the event the main load balancer fails, the second load balancer takes over.



**PROGRAM:**

import java.util.Scanner;

class LoadBalancer {

static void printLoad(int servers, int processes) {

System.out.print("\nCurrent status:\n");

System.out.print("Server\tProcesses\n");

int each = processes / servers;

int extra = processes % servers;

int total = 0;

for (int i = 0; i < servers; i++) {

if (extra-- > 0)

total = each + 1;

else total = each;

System.out.println((char)('A' + i) + "\t" + total);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of servers: ");

int servers = sc.nextInt();

System.out.print("Enter the number of processes: ");

int processes = sc.nextInt();

while (true) {

printLoad(servers, processes);

System.out.print("\n\*\*\* MENU \*\*\* \n1. Add Servers \n2. Remove Servers \n3. Add Processes \n4. Remove Processes \n5. Exit \nYour choice: ");

switch (sc.nextInt()) {

case 1:

System.out.print("How many servers to add? : ");

servers += sc.nextInt();

break;

case 2:

System.out.print("How many servers to remove? :");

servers -= sc.nextInt();

break;

case 3:

System.out.print("How many processes to add? :");

processes += sc.nextInt();

break;

case 4:

System.out.print("How many processes to remove? :");

processes -= sc.nextInt();

break;

case 5:

return;

}

}

}

}

**OUTPUT:**

C:\Desktop\College\College Exps\Sem 8\DC\Experiment 6> javac LoadBalancer.java

C:\Desktop\College\College Exps\Sem 8\DC\Experiment 6> java LoadBalancer

Enter the number of servers: 4

Enter the number of processes: 10

Current status:

Server Processes

A 3

B 3

C 2

D 2

\*\*\* MENU \*\*\*

1. Add Servers

2. Remove Servers

3. Add Processes

4. Remove Processes

5. Exit

Your choice: 1

How many servers to add? : 2

Current status:

Server Processes

A 2

B 2

C 2

D 2

E 1

F 1

\*\*\* MENU \*\*\*

1. Add Servers

2. Remove Servers

3. Add Processes

4. Remove Processes

5. Exit

Your choice: 2

How many servers to remove? :3

Current status:

Server Processes

A 4

B 3

C 3

\*\*\* MENU \*\*\*

1. Add Servers

2. Remove Servers

3. Add Processes

4. Remove Processes

5. Exit

Your choice: 3

How many processes to add? :4

Current status:

Server Processes

A 5

B 5

C 4

\*\*\* MENU \*\*\*

1. Add Servers

2. Remove Servers

3. Add Processes

4. Remove Processes

5. Exit

Your choice: 4

How many processes to remove? :1

Current status:

Server Processes

A 5

B 4

C 4

\*\*\* MENU \*\*\*

1. Add Servers

2. Remove Servers

3. Add Processes

4. Remove Processes

5. Exit

Your choice: 5

C:\Desktop\College\College Exps\Sem 8\DC\Experiment 6>

**CONCLUSION:** Thus we have successfully implemented a Load Balancing Algorithm.