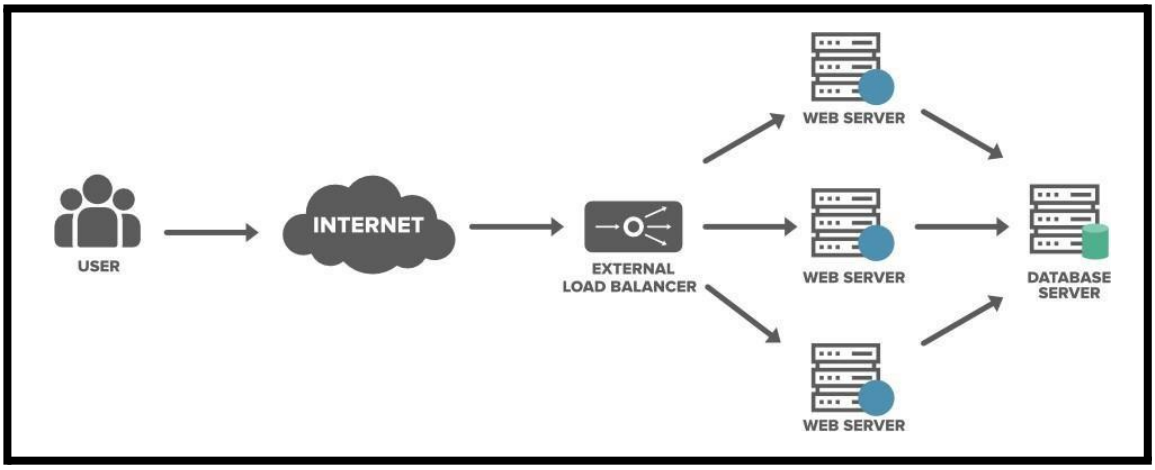
Aim: Implementation of a Load Balancing Algorithm.

Theory:

Load balancing is the method of distributing network traffic equally across a pool of resources that support an application. Modern applications must process millions of users simultaneously and return the correct text, videos, images, and other data to each user in a fast and reliable manner. To handle such high volumes of traffic, most applications have many resource servers with duplicate data between them. A load balancer is a device that sits between the user and the server group and acts as an invisible facilitator, ensuring that all resource servers are used equally.



Benefits :

- **Application availability**: Server failure or maintenance can increase application downtime, making your application unavailable to visitors. Load balancers increase the fault tolerance of your systems by automatically detecting server problems and redirecting client traffic to available servers.
- **Application scalability:** You can use load balancers to direct network traffic intelligently among multiple servers. Prevents traffic bottlenecks at any one server
- **Application security:** Load balancers come with built-in security features to add another layer of security to your internet applications. They are a useful tool to deal with distributed denial of service attacks, in which attackers flood an application server with millions of concurrent requests that cause server failure.

Algorithms in Load balancing:

1. **Round Robin**: This algorithm distributes workload evenly across a set of servers in a cyclic manner. Each new request is forwarded to the next server in the rotation.
2. **Least Connections**: This algorithm selects the server with the fewest active connections to distribute the workload to. This ensures that heavily loaded servers are not further burdened.

3. **Weighted Round Robin**: This algorithm assigns a weight to each server based on its capacity and distributes workload in proportion to the assigned weights. Servers with higher weights receive more requests.
4. **IP Hash**: This algorithm uses the client's IP address to determine which server should handle the request. Requests from the same client are consistently routed to the same server.
5. **Least Response Time:** This algorithm measures the response time of each server and directs new requests to the server with the fastest response time.
6. **Random**: This algorithm selects a server at random to handle each new request.

Code:

```
class LoadBalancer:
    def __init__(self, servers):
        self.servers = servers
        self.current_index = 0

    def balance(self):
        server = self.servers[self.current_index]
        self.current_index = (self.current_index + 1) % len(self.servers)
        return server

# Example usage
if __name__ == "__main__":
    servers = ["Server1", "Server2", "Server3", "Server4"]
    load_balancer = LoadBalancer(servers)

    # Simulate incoming requests
    for i in range(10):
        server = load_balancer.balance()
        print(f"Request {i+1} routed to {server}")
```

Output:


IDLE Shell 3.10.1
File Edit Shell Debug Options Window Help
Python 3.10.1 (tags/v3.10.1:2cd268a, Dec 6 2021, 19:10:37) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: C:/Users/VISHAL BHAGAT/OneDrive/Desktop/VISHAL/DC4.py ========
Request 1 routed to Server1
Request 2 routed to Server2
Request 3 routed to Server3
Request 4 routed to Server4
Request 5 routed to Server1
Request 6 routed to Server2
Request 7 routed to Server3
Request 8 routed to Server4
Request 9 routed to Server1
Request 10 routed to Server2
>>>