SVKM's INSTITUTE OF TECHNOLOGY, DHULE

Department of Information Technology

Academic Year 2025-26

Experiment No. 1

Name: Bhupesh Anil Suryawanshi

Roll No.: 63

Batch: B3

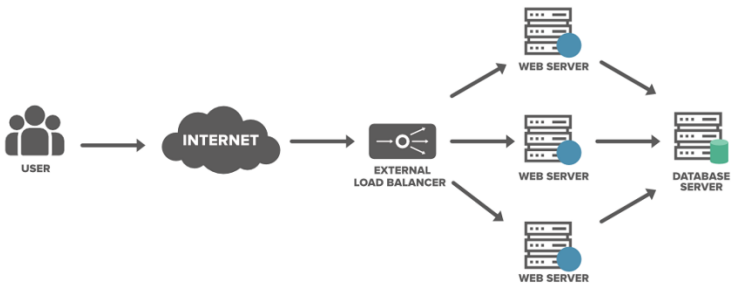
Aim: To implement the python program for demonstrating a load-balancing approach.

Objective:

To implement a simple Round Robin Load Balancer in Python that distributes incoming requests evenly across a list of servers.

Theory:

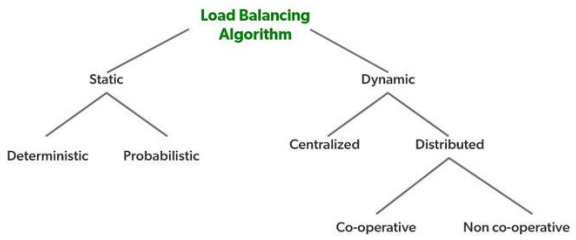
A load balancer is a device that acts as a reverse proxy and distributes network or application traffic across a number of servers. Load adjusting is the approach to conveying load units (i.e., occupations/assignments) across the organization which is associated with the distributed system. Load adjusting should be possible by the load balancer. The load balancer is a framework that can deal with the load and is utilized to disperse the assignments to the servers. The load balancers allocates the primary undertaking to the main server and the second assignment to the second server.



Purpose of Load Balancing in Distributed Systems:

- Security: A load balancer provide safety to your site with practically no progressions to your application.
- Protect applications from emerging threats: The Web Application Firewall (WAF) in the load balancer shields your site.
- Authenticate User Access: The load balancer can demand a username and secret key prior to conceding admittance to your site to safeguard against unapproved access.
- Protect against DDoS attacks: The load balancer can distinguish and drop conveyed refusal of administration (DDoS) traffic before it gets to your site.
- Performance: Load balancers can decrease the load on your web servers and advance traffic for a superior client experience.
- SSL Offload: Protecting traffic with SSL (Secure Sockets Layer) on the load balancer eliminates the upward from web servers bringing about additional assets being accessible for your web application.
- Traffic Compression: A load balancer can pack site traffic giving your clients a vastly improved encounter with your site.

Types of Load Balancing:



- Static Load Balancing: In static load balancing, the distribution of tasks is predetermined. The processors are assigned jobs at compile time based on the performance capabilities of each node. Once assigned, the load does not change dynamically during execution. This approach is effective in environments where the workload is predictable.
- Dynamic Load Balancing: Dynamic load balancing, on the other hand, makes decisions at runtime. The system monitors the state of each node and adjusts the distribution of tasks accordingly. This method is more flexible and responsive to changes in the workload or system state, but it can introduce additional overhead due to the continuous monitoring and adjustment processes.

Load Balancing Approaches:

- Round Robin
- Least Connections
- Least Time
- Hash
- IP Hash

Round Robin Load Balancer: Round Robin is a simple, widely-used load balancing algorithm. It assigns requests to each server in a cyclic order, ensuring that all servers are utilized equally. This method does not consider the current load on each server, making it straightforward but less efficient in scenarios where servers have varying capacities or when tasks have different resource requirements.

Benefits of Load Balancing:

1. **Improved Performance:** Distributes workload evenly, enhancing the overall performance of the system.
2. **Reduced Idle Time:** Minimizes the time during which servers are idle, maximizing resource utilization.
3. **Prevention of Starvation:** Ensures that short jobs do not suffer from long delays or starvation.
4. **High Throughput:** Increases the number of tasks processed within a given time period.
5. **Reliability:** Enhances the system's reliability by preventing any single point of failure.
6. **Extensibility:** Facilitates incremental growth by allowing the addition of more servers to the system.
7. **Cost-Effective:** Offers high gains in performance and reliability with minimal additional cost.

Procedure:

1. **Initialize the Load Balancer:**
- Define a class RoundRobinLoadBalancer.
- In the __init__ method, initialize the load balancer with a list of servers (self.servers) and set the current_server_index to 0.
2. **Implement the get_next_server Method:**
   - Create a method get_next_server(self) that returns the server at the current index (self.current_server_index).
   - Update self.current_server_index to point to the next server using the modulo operation to loop back to the first server after reaching the end of the list.
3. **Implement the add_server Method:**
   - Create a method add_server(self, server) to append a new server to the self.servers list.
4. **Implement the remove_server Method:**
   - Create a method remove_server(self, server) to remove a server from the self.servers list.
   - Ensure the current_server_index is reset to 0 if it exceeds the length of the updated server list.
5. **Test the Load Balancer:**
- In the __main__ block, create an instance of RoundRobinLoadBalancer with an initial list of servers.
- Simulate incoming requests by looping through n requests and calling get_next_server() for each request.
- Print the server handling each request to verify the round-robin distribution.

Conclusion: In this practical, we learned how to implement a Round Robin Load Balancer that efficiently distributes requests evenly across multiple servers to ensure balanced resource utilization.

Program:

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

    def get_next_server(self):
        if not self.servers:
            return "No servers available"
        server = self.servers[self.current_index]
        self.current_index = (self.current_index + 1) % len(self.servers)
        return server
```

```
    def add_server(self, server):
        self.servers.append(server)
        print(f"Server '{server}' added.")

    def remove_server(self, server):
        if server in self.servers:
            self.servers.remove(server)
            print(f"Server '{server}' removed.")
            if self.current_index >= len(self.servers):
                self.current_index = 0
        else:
            print(f"Server '{server}' not found.")


if __name__ == "__main__":
    lb = RoundRobinLoadBalancer(["Server1", "Server2", "Server3"])

    for i in range(1, 11):
        server = lb.get_next_server()
        print(f"Request {i} handled by {server}")

    lb.add_server("Server4")

    for i in range(11, 16):
        server = lb.get_next_server()
        print(f"Request {i} handled by {server}")

    lb.remove_server("Server2")

    for i in range(16, 21):
        server = lb.get_next_server()
        print(f"Request {i} handled by {server}")
```

Output:

```
Request 1 handled by Server1
Request 2 handled by Server2
Request 3 handled by Server3
Request 4 handled by Server1
Request 5 handled by Server2
Request 6 handled by Server3
Request 7 handled by Server1
Request 8 handled by Server2
Request 9 handled by Server3
Request 10 handled by Server1
Server 'Server4' added.
Request 11 handled by Server2
Request 12 handled by Server3
Request 13 handled by Server4
Request 14 handled by Server1
Request 15 handled by Server2
```

Server 'Server2' removed.
Request 16 handled by Server4
Request 17 handled by Server1
Request 18 handled by Server3
Request 19 handled by Server4
Request 20 handled by Server1