PCET's Pimpri Chinchwad University, Pune

Learn | Grow | Achieve

Checked By	Dr. Avinash Golande
Sign	
Date	
Grade	

EXPERIMENT NO . 05

- **Title:-** Write a Python program to simulate load balancing using **Round Robin** or **Least Connection** algorithm across multiple service instances.
- **Objective :-**
    - To understand the concept of load balancing in distributed systems.
    - To implement load balancing strategies in Python.
    - To compare **Round Robin** and **Least Connection** methods.
- **Resources used :-** PC / Laptop with Linux OS (Ubuntu/Debian recommended) , Random module (for simulation)
- **Theory :-**

Concept of Load Balancing
Load balancing is the process of distributing incoming client requests across multiple servers (or service instances) to:

- Improve performance
- Avoid overloading a single server
- Ensure fault tolerance and scalability

Without load balancing, one server may be overloaded while others remain idle. A load balancer ensures requests are distributed fairly.

Algorithms Used

1. Round Robin – Assign requests to servers one after another in cyclic order.
    - Example (3 servers):
        - Request 1 → Server 1
        - Request 2 → Server 2
        - Request 3 → Server 3
        - Request 4 → Server 1 (cycle repeats)

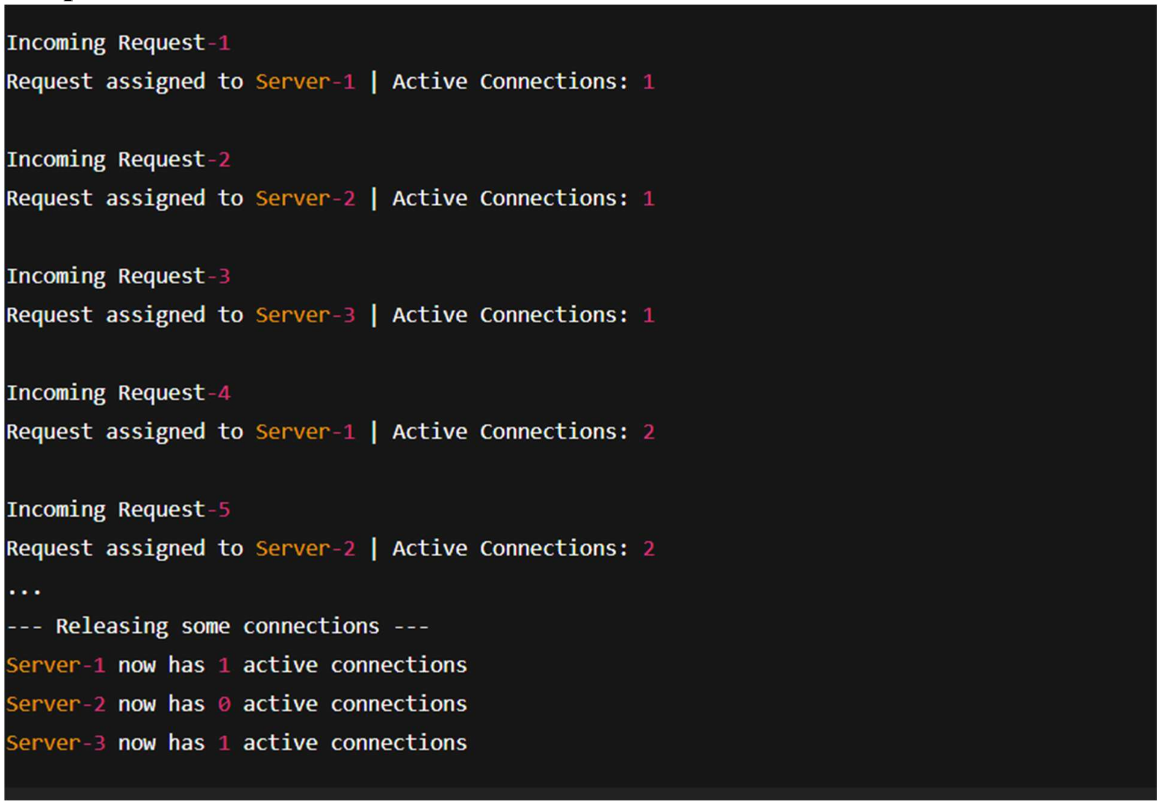
2. Least Connection – Assign requests to the server with the fewest active connections.
    - Example:
        - Server 1: 5 connections
        - Server 2: 2 connections
        - Server 3: 1 connection → next request goes to Server 3.

- **Code**

```
import random


class Server:
    def __init__(self, server_id):
        self.server_id = server_id
        self.active_connections = 0

    def handle_request(self):
        self.active_connections += 1
        print(f"Request assigned to Server-{self.server_id} | Active Connections:
{self.active_connections}")

    def release_request(self):
        if self.active_connections > 0:
            self.active_connections -= 1



class LoadBalancer:
    def __init__(self, servers, strategy="round_robin"):
        self.servers = servers
        self.strategy = strategy
        self.index = 0  # for round robin

    def get_server(self):
        if self.strategy == "round_robin":
            # Pick server in a cyclic order
            server = self.servers[self.index]
            self.index = (self.index + 1) % len(self.servers)
            return server
        elif self.strategy == "least_connection":
            # Pick server with least active connections
            return min(self.servers, key=lambda s: s.active_connections)
        else:
            raise ValueError("Invalid load balancing strategy")

    def handle_request(self):
        server = self.get_server()
        server.handle_request()



# ---------------- Simulation ----------------
if __name__ == "__main__":
    # Create 3 servers
    servers = [Server(i) for i in range(1, 4)]

    # Choose strategy: "round_robin" or "least_connection"
    lb = LoadBalancer(servers, strategy="least_connection")

    # Simulate 10 incoming requests
    for i in range(10):
        print(f"\nIncoming Request-{i+1}")
        lb.handle_request()

    # Randomly release some requests (simulate completion)
```

```
print("\n--- Releasing some connections ---")
for s in servers:
    release_count = random.randint(0, s.active_connections)
    for _ in range(release_count):
        s.release_request()
    print(f"Server-{s.server_id} now has {s.active_connections} active connections")
```

- **Output**



```
Incoming Request-1
Request assigned to Server-1 | Active Connections: 1

Incoming Request-2
Request assigned to Server-2 | Active Connections: 1

Incoming Request-3
Request assigned to Server-3 | Active Connections: 1

Incoming Request-4
Request assigned to Server-1 | Active Connections: 2

Incoming Request-5
Request assigned to Server-2 | Active Connections: 2
...
--- Releasing some connections ---
Server-1 now has 1 active connections
Server-2 now has 0 active connections
Server-3 now has 1 active connections
```

- **Conclusion:**

Successfully studied and implemented **libvirt** as a virtualization management framework.

- Understood its architecture (library, daemon, CLI tools).
- Learned real VM management with libvirt Python bindings.
- Explored VM scheduling simulator to test placement strategies.