ASSIGNMENT 4

PROBLEM STATEMENT: -

Write code to simulate requests coming from clients and distribute them among the servers using the load balancing algorithms.

OBJECTIVE:

1. Students should be able to Implement various load balancing algorithms to distribute requests among servers efficiently.
2. Students should be able to Simulate the processing of requests by servers.

THEORY:

What are Load Balancers?

In case multiple servers are present the incoming request coming to the system needs to be directed to one of the multiple servers. We should ensure that every server gets an equal number of requests. The requests must be distributed in a uniform manner across all the servers. The component which is responsible for distributing these incoming requests uniformly across the servers is known as Load Balancer. A Load Balancer acts as a layer between the incoming requests coming from the user and multiple servers present in the system.

We should avoid the scenarios where a single server is getting most of the requests while the rest of them are sitting idle. There are various Load Balancing Algorithms that ensure even distribution of requests across the servers.

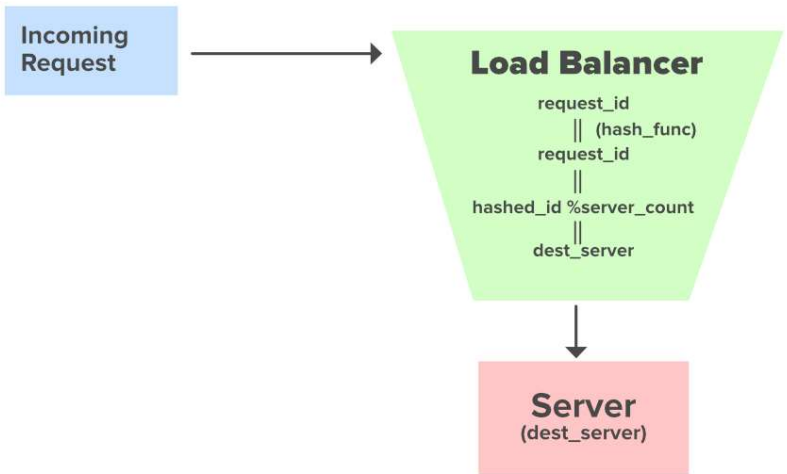


Fig. Hashing Approach to direct requests from the Load Balancer

We will be discussing the Hashing Approach to direct the requests to multiple servers uniformly. Suppose we have server_count as the Total number of servers present in the System and a load_balancer to distribute the requests among those servers. A request with an id request_id enters the system. Before reaching the destination server it is directed to the load_balancer from where it is further directed to its destination server. When the request reaches the load balancer the hashing approach will provide us with the destination server where the request is to be directed.

- **request_id** : Request ID coming to get served
- **hash_func** : Evenly distributed Hash Function
- **hashed_id** : Hashed Request ID
- **server_count** : Number of Servers

Java Code:

```
class GFG {
  public static int hash_func(int request_id)
  {
    // Computing the hash request id
    int hashed_id = 112;
    return hashed_id;
  }

  public static void route_request_to_server(int dest_server)
  {
    System.out.println("Routing request to the Server ID : " + dest_server);
  }

  public static int request_id = 23; // Incoming Request ID
  public static int server_count = 10; // Total Number of Servers

  public static void main(String args[])
  {
    int hashed_id = hash_func(request_id); // Hashing the incoming request id
    int dest_server = hashed_id % server_count; // Computing the destination server id

    route_request_to_server(dest_server);
  }
}
```

Python:

The Load Balancer class has two methods: round robin for round-robin load balancing and random selection for random load balancing. The simulate_client_requests function simulates client requests and prints the server selected by each algorithm for each request.

```
import random


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

```
        self.server_index_rr = 0

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

    def random_selection(self):
        return random.choice(self.servers)

def simulate_client_requests(load_balancer, num_requests):
    for i in range(num_requests):
        # Simulating client request
        print(f"Request {i+1}: ", end="")

        # Using Round Robin algorithm for load balancing
        server_rr = load_balancer.round_robin()
        print(f"Round Robin - Server {server_rr}")

        # Using Random algorithm for load balancing
        server_random = load_balancer.random_selection()
        print(f"Random - Server {server_random}")

        print()

if __name__ == "__main__":
    # List of servers
    servers = ["Server A", "Server B", "Server C"]

    # Create a LoadBalancer instance
    load_balancer = LoadBalancer(servers)

    # Simulate 10 client requests
    simulate_client_requests(load_balancer, 10)
```

CONCLUSION:

In this way we have design a code that simulates client requests and distributes them among servers using the Round Robin load balancing algorithm. This demonstrates a basic implementation of load balancing in a simulated environment.

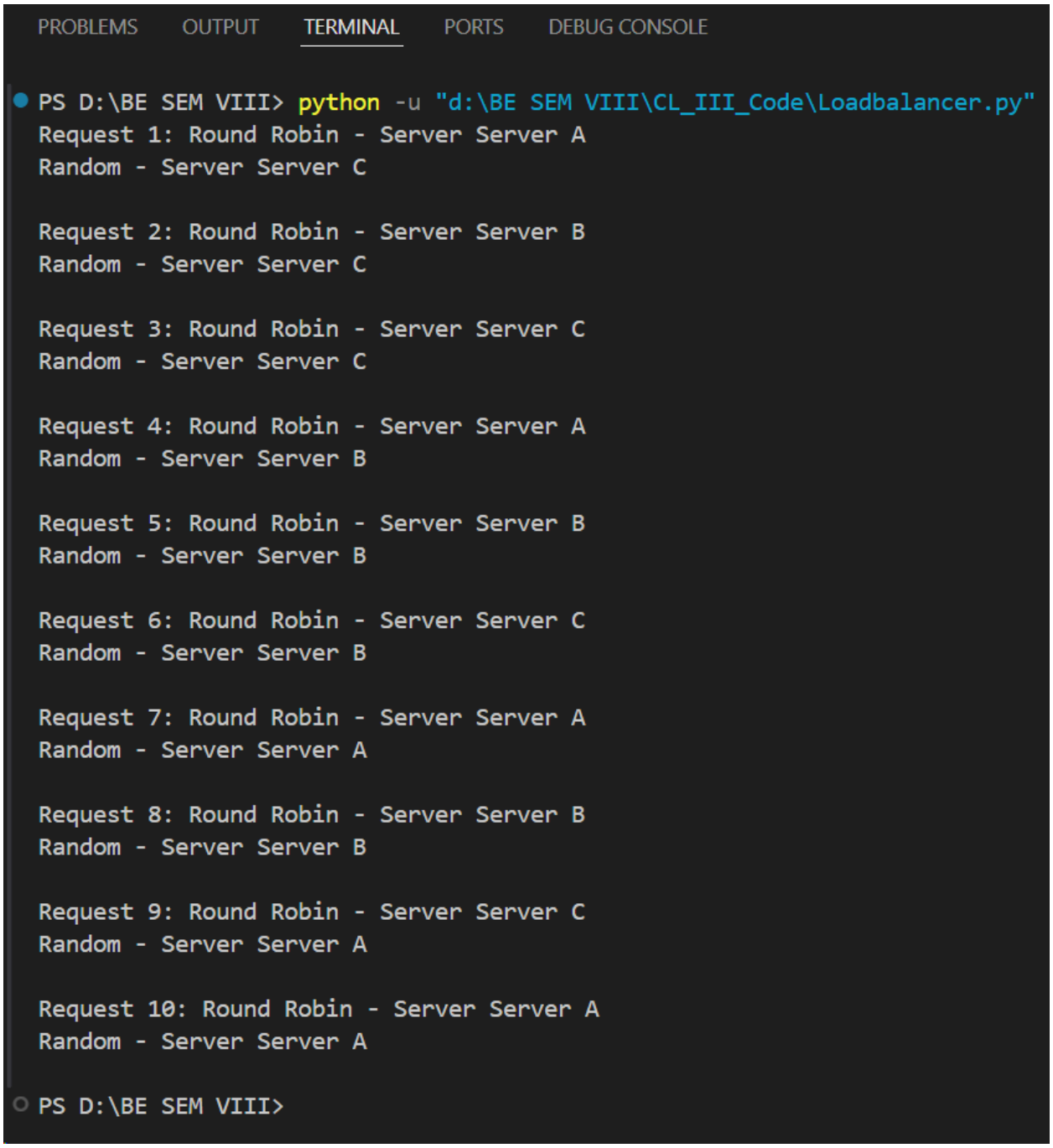
ORAL QUESTION:

1. What are the common load balancing algorithms?
2. How does the round-robin algorithm distribute requests among servers?
3. How does load balancing contribute to the scalability and efficiency of a distributed system?

Code:

```
import random

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

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

    def random_selection(self):
        return random.choice(self.servers)

def simulate_client_requests(load_balancer, num_requests):
    for i in range(num_requests):
        # Simulating client request
        print(f"Request {i+1}: ", end="")

        # Using Round Robin algorithm for load balancing
        server_rr = load_balancer.round_robin()
        print(f"Round Robin - Server {server_rr}")

        # Using Random algorithm for load balancing
        server_random = load_balancer.random_selection()
        print(f"Random - Server {server_random}")
        print()

if __name__ == "__main__":
    # List of servers
    servers = ["Server A", "Server B", "Server C"]

    # Create a LoadBalancer instance
    load_balancer = LoadBalancer(servers)

    # Simulate 10 client requests
    simulate_client_requests(load_balancer, 10)
```

Output:


PROBLEMS
OUTPUT
TERMINAL
PORTS
DEBUG CONSOLE
PS D:\BE SEM VIII> python -u "d:\BE SEM VIII\CL_III_Code\Loadbalancer.py"
Request 1: Round Robin - Server Server A
Random - Server Server C
Request 2: Round Robin - Server Server B
Random - Server Server C
Request 3: Round Robin - Server Server C
Random - Server Server C
Request 4: Round Robin - Server Server A
Random - Server Server B
Request 5: Round Robin - Server Server B
Random - Server Server B
Request 6: Round Robin - Server Server C
Random - Server Server B
Request 7: Round Robin - Server Server A
Random - Server Server A
Request 8: Round Robin - Server Server B
Random - Server Server B
Request 9: Round Robin - Server Server C
Random - Server Server A
Request 10: Round Robin - Server Server A
Random - Server Server A
PS D:\BE SEM VIII>