Banking System

(Experiment 3 - Load Balancing Algorithm)

Yash Brid 2019130008
Abhishek Chopra 2019130009
Sumeet Haldipur 2019130018

Aim:

To implement a load-balancing algorithm for a Banking System.

Objective:

- To learn about load balancing algorithms in a distributed system.
- To implement load balancing for a Banking System.

Theory

Load balancing is the process of redistributing the work load among nodes of the distributed system to improve both resource utilization and job response time while also avoiding a situation where some nodes are heavily loaded while others are idle or doing little work.

If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In this manner, a load balancer performs the following functions:

- Distributes client requests or network load efficiently across multiple servers
- Ensures high availability and reliability by sending requests only to servers that are online
- Provides the flexibility to add or subtract servers as demand dictates

Round robin algorithm is the simplest method of load balancing servers, or for providing simple fault tolerance. Multiple identical servers are configured to provide precisely the same services or applications. All are configured to use the same Internet domain name, but each has a unique IP address. The load balancer has a list of all the unique IP addresses that are associated with the Internet domain name.

When requests for sessions on the servers associated with the Internet domain name are received, they are allocated in a random, or rotating sequential manner. For example, the first request gets the IP address of server 1, the second request gets the IP address of server 2, and so forth, with requests starting again at server 1 when all servers have been assigned an access request during a cycle.

Program:

Server

```
import java.rmi.RemoteException;
import
java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import
java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class Server extends UnicastRemoteObject implements checkBal
    { public Server() throws RemoteException {
super();
}

static ArrayList<Account> a = new ArrayList<Account>();

    public double checkBalance(String acc_no, String password)
throws RemoteException {
        System.out.println("Request received for account number "
+ acc_no);
for (int i = 0; i < a.size(); i++) {
double bal = a.get(i).checkBalance(acc_no, password); if (bal
            != -1)
return bal;
}
return -1.0;
}


    public static void main(String[] args)
        { try {
int no = Integer.parseInt(args[0]); int
            portNumber = 8000 + no;
```

```
reg.rebind("bankServer"+args[0], new Server());
            System.out.println("Server no "+args[0]+" is
            running.."); a.add(new Account("123456", "password1",
            2000.0)); a.add(new Account("456789", "password2",
            3700.50)); a.add(new Account("234567", "password1",
            2000.0)); a.add(new Account("345678", "password2",
            3700.50));
        } catch (Exception e)
            {
            e.printStackTrace(
            );
}
}
}

class Account {
    String acc_no;
    String
    password;
    double
    balance;

    Account(String acc_no, String password, double balance)
        { this.acc_no = acc_no;
this.password = password;
        this.balance = balance;
}
```

```
public double checkBalance(String acc_no, String password) {
        if (this.acc_no.equals(acc_no) &&
            this.password.equals(password)) return this.balance;
```

loadBalancerInterface:

```
import java.rmi.*;

public interface loadBalancerInterface extends Remote {

public checkBal getServer() throws RemoteException;
    public int getServerName() throws RemoteException;

}
```

loadBalancer:

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class loadBalancer extends UnicastRemoteObject implements
loadBalancerInterface{

    int noOfServers;
    int noOfRequests;
    public loadBalancer() throws RemoteException{
        noOfServers = 4;
        noOfRequests = 0;
    }

    public int getServerName() throws RemoteException{

        int serverNo = noOfRequests % noOfServers;
        return serverNo;
    }

    public checkBal getServer() throws RemoteException{
        int serverNo = noOfRequests % noOfServers;
        noOfRequests++;
        checkBal server = null;
        // return "server"+serverNo;
        try{
            String path = "bankServer" + serverNo;
            System.out.println("Redirecting request to server "
+serverNo);
            Registry reg = LocateRegistry.getRegistry("localhost",
8000+serverNo);
            server = (checkBal) reg.lookup(path);
        }
        catch(NotBoundException e){
            System.out.println("Unable to connect to server, trying
nextone " + e);
            server = this.getServer();
```

```
}
return server;
}
    public static void main(String args[]) throws
        RemoteException{ loadBalancer token = new
        loadBalancer();
try{
String objPath = "loadBalancer"; //name of server location Registry
            reg = LocateRegistry.createRegistry(8081);
            reg.rebind(objPath, new loadBalancer());
            System.out.println("Load balancing server is running
            now.");
}
        catch(Exception e){
            System.out.println("Exception" +
            e);
```

Client

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Scanner;
import java.time.*;

public class Client {
    public static void main(String args[]) throws RemoteException {
        try {
            Scanner sc = new Scanner(System.in);
            // Registry reg = LocateRegistry.getRegistry("localhost",
8000);                                  // checkBal obj_bal = (checkBal)
                                        reg.lookup("bankServer");
                                        Registry reg =
                                        LocateRegistry.getRegistry("loca
                                        lhost", 8081);
                                        loadBalancerInterface lb =
                                        (loadBalancerInterface)
reg.lookup("loadBalancer");
            System.out.println("Connected to server "+
lb.getServerName());
```

1.
```
            checkBal obj_bal = lb.getServer();
            System.out.print("\nEnter account number:");
            String acc_no = sc.nextLine();
```

```
System.out.print("Enter password:");
```

```
String password = sc.nextLine();
Clock client_time = Clock.systemUTC(); Registry
            reg_time =
LocateRegistry.getRegistry("localhost",8080);
getTime obj = (getTime) reg_time.lookup("timeServer"); long
            start = Instant.now().toEpochMilli();
long serverTime = obj.getSystemTime();
            System.out.println("Server time "+
            serverTime); long end =
            Instant.now().toEpochMilli();
long rtt = (end-start)/2; System.out.println("Round
            Trip Time " + rtt); long updatedTime =
            serverTime + rtt; client_time =
            Clock.offset(client_time,
Duration.ofMillis(updatedTime -
            client_time.instant().toEpochMilli()));
            System.out.println("New Client time " +
client_time.instant().toEpochMilli());
double bal = obj_bal.checkBalance(acc_no, password); if (bal
            == -1) {
System.out.println("\nInvalid credentials"); return;
} else {
System.out.println("\nBalance: Rs." + bal+"\n");
}
        } catch (Exception e)
            {
            e.printStackTrace(
            );
}
}
```

Output

```
C-Lab\Exp3LoadBalancer> java loadBalancer
Load balancing server is running now.
Redirecting request to server 0
Redirecting request to server 1
Redirecting request to server 2
Redirecting request to server 3
```

```
C-Lab\Exp3LoadBalancer> java Server 3
Server no 3 is running..
Request received for account number 234567
```

```
C-Lab\Exp3LoadBalancer> java Client 3
Connected to server 3

Enter account number:234567
Enter password:password1
Server time 1636378655823
Round Trip Time 5
New Client time 1636378655829

Balance: Rs.2000.0
```

```
C-Lab\Exp3LoadBalancer> java Server 2
Server no 2 is running..
Request received for account number 345678
```

```
C-Lab\Exp3LoadBalancer> java Client 2
Connected to server 2

Enter account number:345678
Enter password:password2
Server time 1636378571326
Round Trip Time 3
New Client time 1636378571329

Balance: Rs.3700.5
```

```
C-Lab\Exp3LoadBalancer> java Server 1
Server no 1 is running..
Request received for account number 456789
```

```
C-Lab\Exp3LoadBalancer> java Client 1
Connected to server 1

Enter account number:456789
Enter password:password2
Server time 1636378555367
Round Trip Time 3
New Client time 1636378555370

Balance: Rs.3700.5
```

```
C-Lab\Exp3LoadBalancer> java Server 0
Server no 0 is running..
Request received for account number 123456
```

```
C-Lab\Exp3LoadBalancer> java Client 0
Connected to server 0

Enter account number:123456
Enter password:password1
Server time 1636378522896
Round Trip Time 5
New Client time 1636378522902

Balance: Rs.2000.0
```

Conclusion:
We implemented the round robin load balancing algorithm to implement a load balancer and distribute the incoming load on multiple servers. This ensures that the load is distributed equally among all the servers and there is no single point of failure.