Project Documentation

Approach ➖

Load Balancing Algorithm which I used is Weighted Round Robin.

Weighted Round-Robin is to allocate weight according to the performance of the server on the basis of round-robin. The more requests the server can support, the higher the weight will be, and the more requests will be allocated.

For the same 10 requests, the request distribution using the weighted round-robin algorithm will be as shown in the following figure:

requestId	weight	receive request server
5	1	192.168.1.1
3,8	2	192.168.1.2
2,6,9	3	192.168.1.3
1,4,7,10	4	192.168.1.4

You can see that 192.168.1.4 has the largest weight and the largest number of requests are allocated.

Java Implementation for rough idea :

```
public class RoundRobinTest {

    public class Node{
        private String ip;
```

```
    private Integer weight;

    private Integer currentWeight;

    public Node(String ip,Integer weight) {
        this.ip = ip;
        this.weight = weight;
        this.currentWeight = weight;
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
        this.ip = ip;
    }

    public Integer getWeight() {
        return weight;
    }

    public void setWeight(Integer weight) {
        this.weight = weight;
    }

    public Integer getCurrentWeight() {
        return currentWeight;
    }

    public void setCurrentWeight(Integer currentWeight) {
        this.currentWeight = currentWeight;
    }
}

List<Node> servers = Arrays.asList(
        new Node("192.168.1.1",1),
        new Node("192.168.1.2",2),
        new Node("192.168.1.3",3),
        new Node("192.168.1.4",4));
private Integer totalWeight;

public RoundRobinTest() {
    this.totalWeight = servers.stream()
```

```
            .mapToInt(Node::getWeight)
            .reduce((a,b)->a+b).getAsInt();
    }


    public  String getServer() {
        Node node =
servers.stream().max(Comparator.comparingInt(Node::getCurrentWeight)).get();
        node.setCurrentWeight(node.getCurrentWeight()-totalWeight);

servers.forEach(server->server.setCurrentWeight(server.getCurrentWeight()+server.getWeight()
));
        return node.getIp();
    }


    public static void main(String[] args) {
        RoundRobinTest roundRobinTest = new RoundRobinTest();
        for (int i = 0; i < 10; i++) {
            String server = roundRobinTest.getServer();
            System.out.println("select server: "+server);
        }
    }
```

The core of this algorithm is the dynamic calculation of currentWeight. After each server is selected, currentWeightneeds to subtract the sum of the weights of all servers, so that the servers with high weight can be avoided from being selected all the time. The servers with high weight have more requests for allocation, and the requests can be evenly distributed among all servers.

##

Design Patterns and Data Structures used are:-

- MVC(Model View And Controller)
- Load Balancer Implementation (observer pattern)
- REST APIs and Database(MySQL).
- ArrayLists, Classes, etc.

##

Approach to test the application simulation :

- I have created three distinct services to test load balancing among them namely Airtel, Jio, VI using SpringBoot, MySQL, POSTMAN(rest api testing) and STS(Spring Tool Suite).

- The load balancer testing class is different and not any api service. It can be tested by opening Weighted Round Robin class in Eclipse editor.

- For testing the application, Spin up all three servers individually in the local environment using STS.

- Now go to Eclipse and run the Weighted Round Robin Class in which local ip address is already hardcoded to test message pings.

- The mentioned three edge cases have been tested by myself.

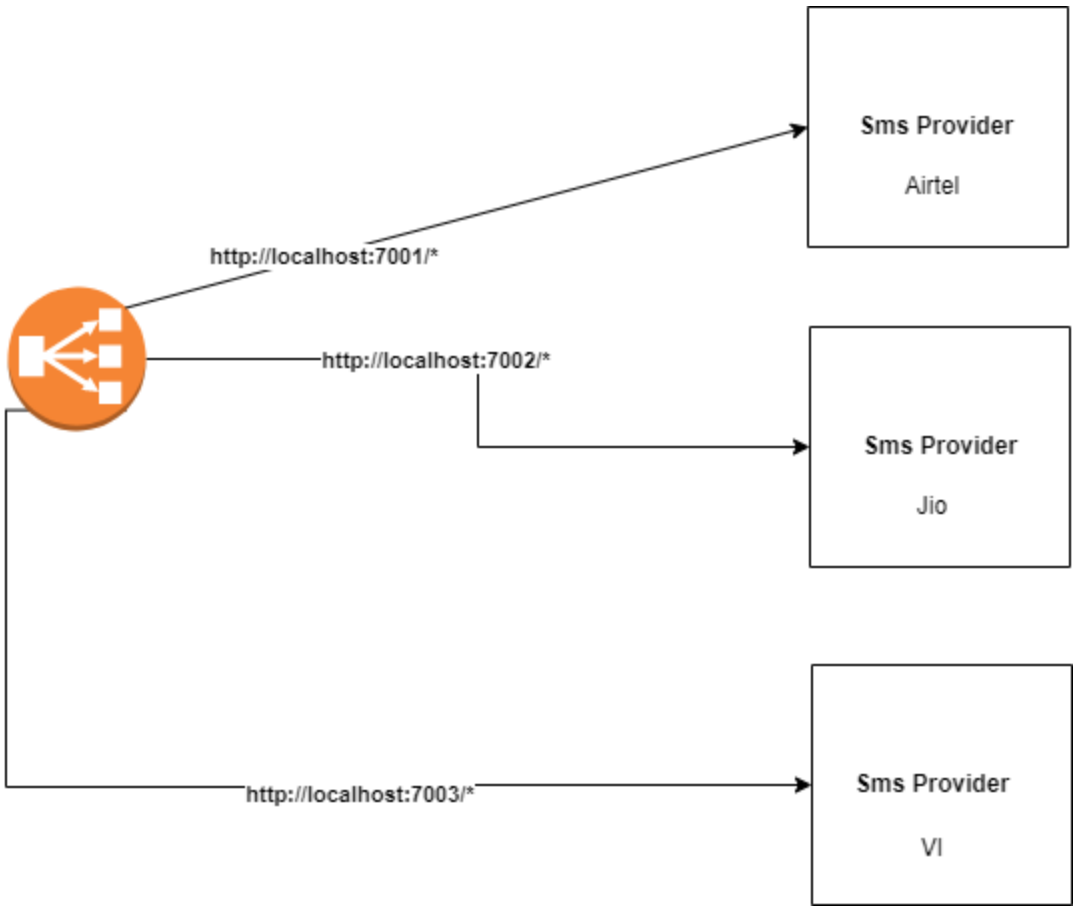
- In order to simulate server down behaviour and check if its working or not, you will have to put its throughput as 0 SMS/min only.

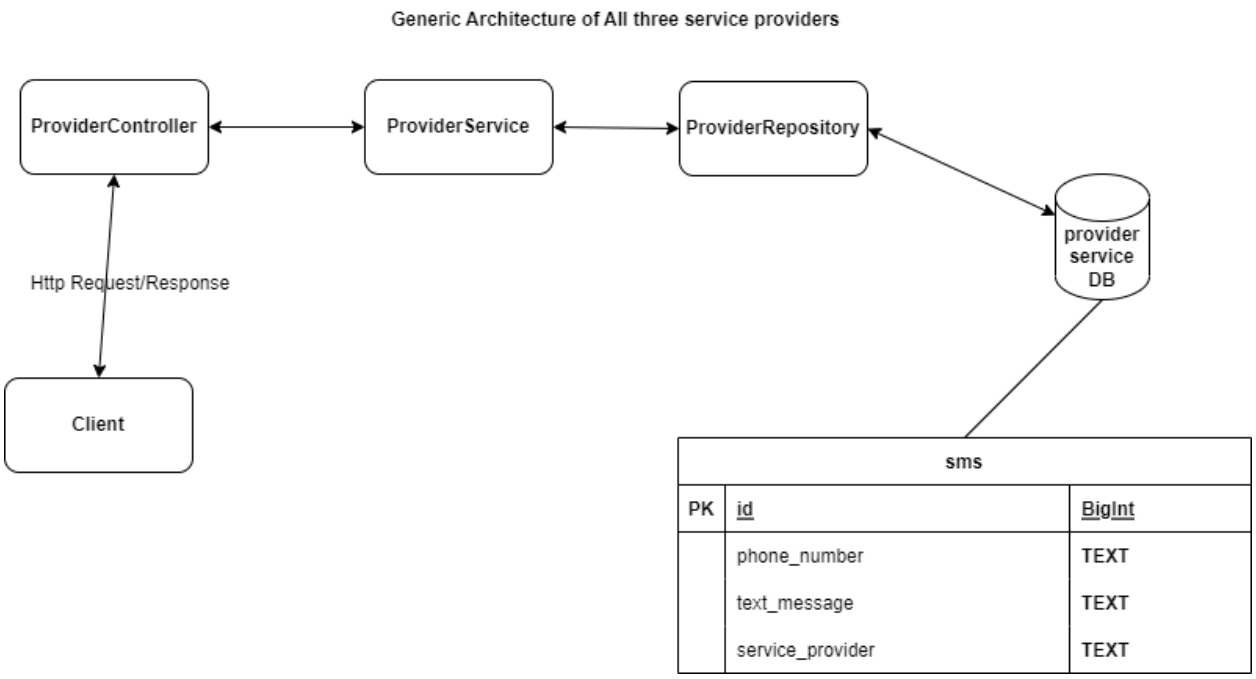
- I have taken serving interval as 1 minute only.

- Asynchronous calls to each of the spinned up server is made and load balancing stats are observed as console logs.

- These logs can further be taken on the file or database for further assessment as per team requirements.

High Level Design :


Sms Provider
Airtel
http://localhost:7001/*
http://localhost:7002/*
Sms Provider
Jio
Sms Provider
VI
http://localhost:7003/*



##

Note :-

All the required files and database dump files are provided in the zip file itself to be run in the local environment.