Distributed Systems Lab
Car Hire Booking Distributed System

Team Members:
Tania Rajabally (2017130047)
Shruti Rampure(2017130048)
Rahul Ramteke(2017130049)

Batch: C

Problem Statement:
The purpose of this system is to monitor and control the bookings of cars in a distributed environment. All the features of a typical Car Hire Booking System are discussed here by considering a distributed system.

LOAD BALANCER

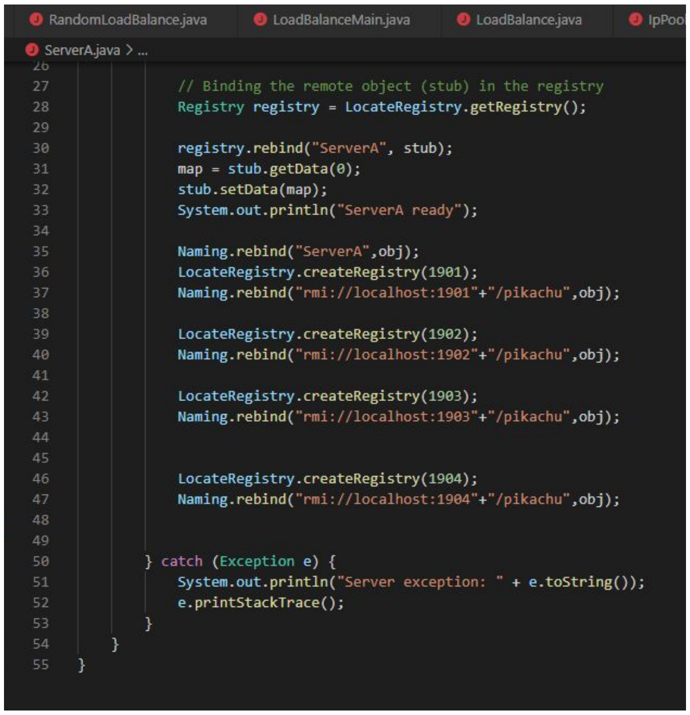
Load balancing is the process of distributing the load among various nodes of a distributed system to improve both job response time and resource utilization while also avoiding a situation where some of the nodes are heavily loaded while other nodes are idle or lightly loaded.
The users of a distributed system have different goals, objectives and strategies and their behavior is difficult to characterize. In such systems the management of resources and applications is a very complex task. A distributed system can be viewed as a collection of computing and communication resources shared by active users. When the demand for computing power increases, the load balancing problem becomes important. The purpose of load balancing is to improve the performance of a distributed system through an appropriate distribution of the application load.

Benefits of Load balancing:

- Load balancing improves the performance of each node and hence the overall system performance
- Load balancing reduces the job idle time
- Small jobs do not suffer from long starvation
- Maximum utilization of resources
- Response time becomes shorter
- Higher throughput
- Higher reliability
- Low cost but high gain
- Extensibility and incremental growth

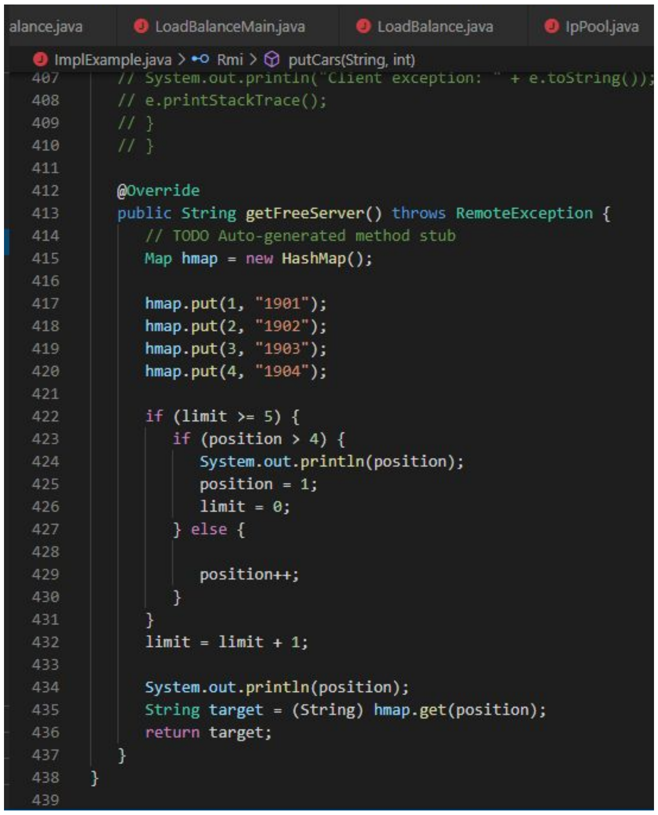
Code:

ServerA.java



```java
            // Binding the remote object (stub) in the registry
            Registry registry = LocateRegistry.getRegistry();

            registry.rebind("ServerA", stub);
            map = stub.getData(0);
            stub.setData(map);
            System.out.println("ServerA ready");

            Naming.rebind("ServerA",obj);
            LocateRegistry.createRegistry(1901);
            Naming.rebind("rmi://localhost:1901"+"/pikachu",obj);

            LocateRegistry.createRegistry(1902);
            Naming.rebind("rmi://localhost:1902"+"/pikachu",obj);

            LocateRegistry.createRegistry(1903);
            Naming.rebind("rmi://localhost:1903"+"/pikachu",obj);


            LocateRegistry.createRegistry(1904);
            Naming.rebind("rmi://localhost:1904"+"/pikachu",obj);


        } catch (Exception e) {
            System.out.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Implementation.java

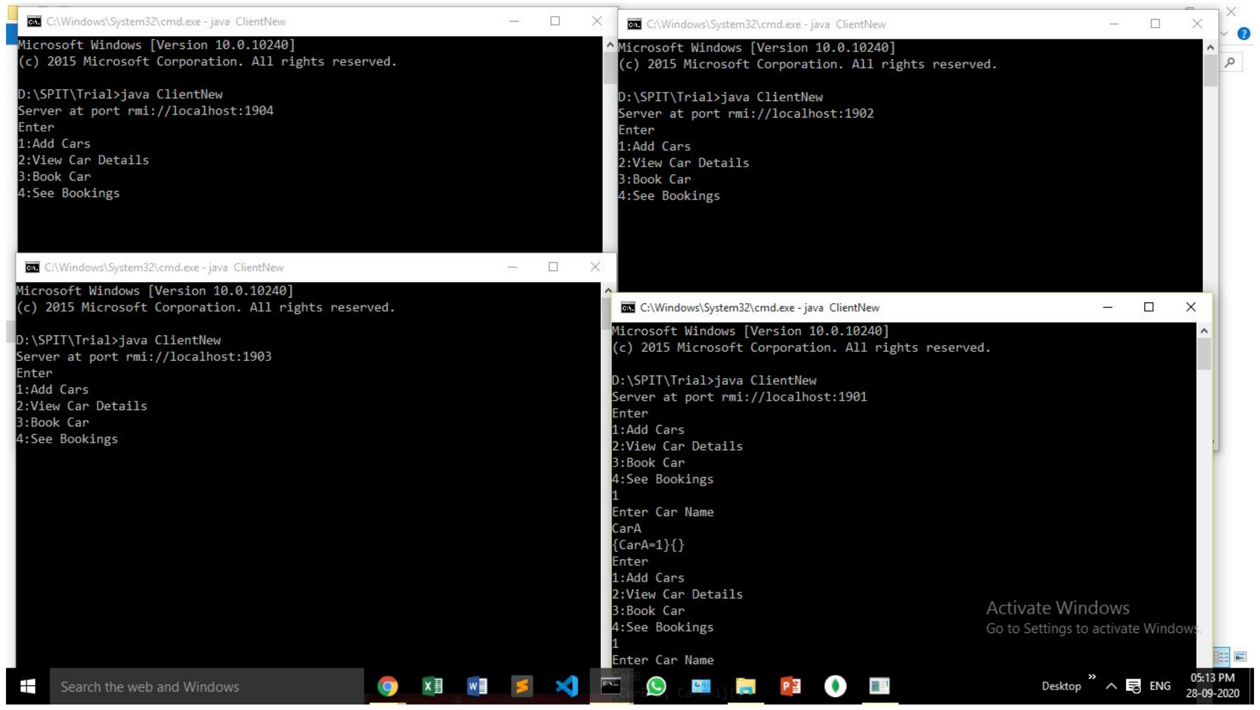


```java
    // System.out.println("Client exception: " + e.toString());
    // e.printStackTrace();
    // }
    // }

    @Override
    public String getFreeServer() throws RemoteException {
        // TODO Auto-generated method stub
        Map hmap = new HashMap();

        hmap.put(1, "1901");
        hmap.put(2, "1902");
        hmap.put(3, "1903");
        hmap.put(4, "1904");

        if (limit >= 5) {
            if (position > 4) {
                System.out.println(position);
                position = 1;
                limit = 0;
            } else {

                position++;
            }
        }
        limit = limit + 1;

        System.out.println(position);
        String target = (String) hmap.get(position);
        return target;
    }
}
```

Explanation of Implementation:
Server A is the main server. It has 4 child servers which have port numbers 1901,1902,1903 and 1904. ServerA creates a registry for all of them and binds them.
When more than one client is requesting, they are served as different ports as can be seen in the screenshot below. Before updating any data, the values are checked for data consistency. On updation, the values are changed in the other two replication servers too i.e serverB and serverC. We check that all the requests are served synchronously. When a new client requests, we first check for the free server and assign that to the client. In this way each client gets a server which is available at that time. We also checked for data consistency and that it is maintained at all times.
Here server A serves as the master server. In the screenshot below, you can see that we have opened four terminals and each is serving at a different port number thus handling multiple requests at the same time.
We further added that each child server can handle upto 5 clients. The next client i.e the sixth client will get the next free server.

Steps to run:
1. Start rmiregistry
2. Compile the files : javac *.java
3. Run the command:
   java ServerA so that the serverA will be ready.
   In a new window run :
   Java ServerB so that serverB is ready.
   In a new window run :
   Java ServerC so that serverC is ready.
4. In a new window, run the command: java ClientNew to run the client side program.
   Repeat this in 3 more windows. Each client will get a different port number.

Screenshots:



```
C:\Windows\System32\cmd.exe - java ClientNew
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1904
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

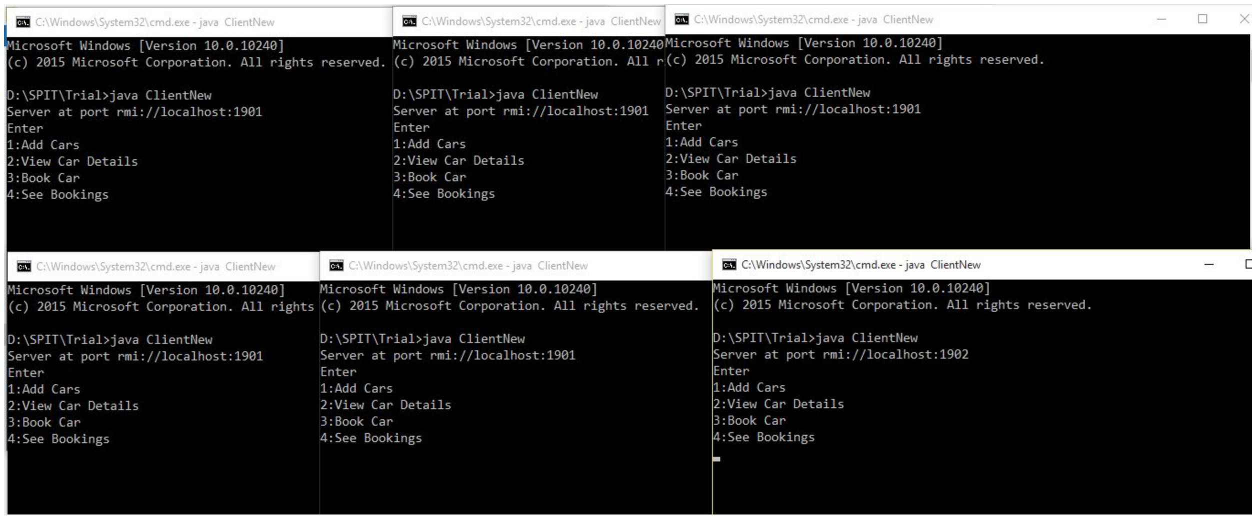
```
C:\Windows\System32\cmd.exe - java ClientNew
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1902
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

```
C:\Windows\System32\cmd.exe - java ClientNew
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1903
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

```
C:\Windows\System32\cmd.exe - java ClientNew
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1901
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
1
Enter Car Name
CarA
{CarA=1}{}
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
1
Enter Car Name
```

Activate Windows
Go to Settings to activate Windows.
Search the web and Windows
Desktop
ENG
05:13 PM
28-09-2020

Each client is served at a different port number. They are all child servers of the master server-serverA


C:\Windows\System32\cmd.exe - java ClientNew
C:\Windows\System32\cmd.exe - java ClientNew
C:\Windows\System32\cmd.exe - java ClientNew
C:\Windows\System32\cmd.exe - java ClientNew
C:\Windows\System32\cmd.exe - java ClientNew
C:\Windows\System32\cmd.exe - java ClientNew

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1901
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

```
Microsoft Windows [Version 10.0.10240
(c) 2015 Microsoft Corporation. All r

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1901
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1901
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1901
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1901
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

```
Microsoft Windows [Version 10.0.10240]
(c) 2015 Microsoft Corporation. All rights reserved.

D:\SPIT\Trial>java ClientNew
Server at port rmi://localhost:1902
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
```

We further added that each child server can handle upto 5 requests i.e it can connect to 5 child servers. The 6th client will get the next free child server.

Conclusion:
The load balancer ensures scalability and availability of services. In our system, it ensures availability when multiple users are requesting a service from the server simultaneously. In this manner, load balancer was implemented and integrated to our car booking system.