

# **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.

### **CLOCK SYNCHRONIZATION**

Distributed System is a collection of computers connected via the high speed communication network. In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share their resources with other nodes. So, there is a need for proper allocation of resources to preserve the state of resources and help coordinate between the several processes. To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks. The physical clocks are used to adjust the time of nodes. Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system. The clock synchronization can be achieved by 2 ways: External and Internal Clock Synchronization.

- External clock synchronization is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
- Internal clock synchronization is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

Cristian's Algorithm is a clock synchronization algorithm that is used to synchronize time with a time server by client processes. This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy while redundancy prone distributed systems/applications do not go hand in hand with this algorithm. Here Round Trip Time refers to the time duration between start of a Request and end of corresponding Response.

## Code:

### ClientNew.java

```
ClientNew.java X RoundRobin.java RandomLoadBalance.java LoadBalanceMain.java LoadBalance.java Ip
ClientNew.java > ClientNew > main(String[])
52 // Stub = (Stub) registry.lookup("Stub");
53 Clock clientClock = Clock.systemUTC();
54 // Looking up the registry for the remote object
55 SystemTime stubTime = (SystemTime) registry.lookup("SystemTime");
56
57 // Get current time before calling the server to calculate RTT
58 long start = Instant.now().toEpochMilli();
59
60 // Calling the remote method using the obtained object
61 long serverTime = stubTime.getSystemTime();
62 System.out.println("Server time " + serverTime);
63
64 long end = Instant.now().toEpochMilli();
65
66 // Calculate RTT
67 long rtt = (end-start)/2;
68 System.out.println("RTT " + rtt);
69
70 // Calculate updatedTime to set the client clock with RTT delay
71 long updatedTime = serverTime+rtt;
72
73 // Calculate offset
74 Duration diff = Duration.ofMillis(updatedTime - clientClock.instant().toEpochMilli());
75
76 // Set Client clock based on offset to server time
77 clientClock = clientClock.offset(clientClock, diff);
78 System.out.println("\nNew Client Time " + clientClock.instant().toEpochMilli());
79 // -->
80 Scanner myObj = new Scanner(System.in);
81 int choice =0;
```

### DefaultSystemTime.java

```
ClientNew.java DefaultSystemTime.java X RoundRobin.java RandomLoadBalance.java
DefaultSystemTime.java > ...
1 import java.time.Instant;
2
3 // Implementing the remote interface
4 public class DefaultSystemTime implements SystemTime {
5     // Implementing the interface method
6     public long getSystemTime() {
7         // Calculating Epoch time on server
8         long time = Instant.now().toEpochMilli();
9         System.out.println("This is the current time on the server, " + time);
10        return time;
11    }
12 }
```

## ServerA.java

```
RandomLoadBalance.java LoadBalanceMain.java LoadBalance.java IpPool.java ImplExam
ServerA.java > ...
6  import java.rmi.registry.*;
7
8  public class ServerA extends ImplExample {
9      public ServerA() {}
10
11      Run|Debug
12      public static void main(String args[]) {
13          String objPath = "///localhost:1099/SystemTime";
14          try {
15              // Instantiating the implementation class
16              ImplExample obj = new ImplExample();
17
18              DefaultSystemTime obj1 = new DefaultSystemTime();
19              // Exporting the object of implementation class
20              SystemTime stub1 = (SystemTime) UnicastRemoteObject.exportObject(obj1, 0);
21              // Binding the remote object (stub) in the registry
22              Naming.bind(objPath, obj1);
23
24              // Exporting the object of implementation class
25              // (here we are exporting the remote object to the stub)
26              Rmi stub = (Rmi) UnicastRemoteObject.exportObject(obj, 0);
27
28              // Binding the remote object (stub) in the registry
29              Registry registry = LocateRegistry.getRegistry();
30
31              registry.rebind("ServerA", stub);
32              map = stub.getData(0);
33              stub.setData(map);
34              System.out.println("ServerA ready");
35
36              Naming.rebind("ServerA",obj);
37              LocateRegistry.createRegistry(1901);
38              Naming.rebind("rmi://localhost:1901"+"pikachu",obj);
39          }
40      }
41  }
```

## SystemTime.java

```
LoadBalanceMain.java LoadBalance.java IpPool.java
SystemTime.java > ...
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3
4
5  // Creating Remote interface for our application
6  public interface SystemTime extends Remote {
7      long getSystemTime() throws RemoteException;
8  }
```

### Explanation of Implementation:

When we run ClientNew.java, the process on the client machine sends the request for fetching clock time(time at server) to Clock Server i.e ServerA.java in our system. The Clock Server listens to the request made by the client process and returns the response in form of clock server time. Now the client process fetches the response from the Clock Server and calculates the synchronised client clock time. Basically, the round trip time is calculated. When a second client is started, the same process is carried out for that client. This is how Cristian algorithm is implemented in our system.

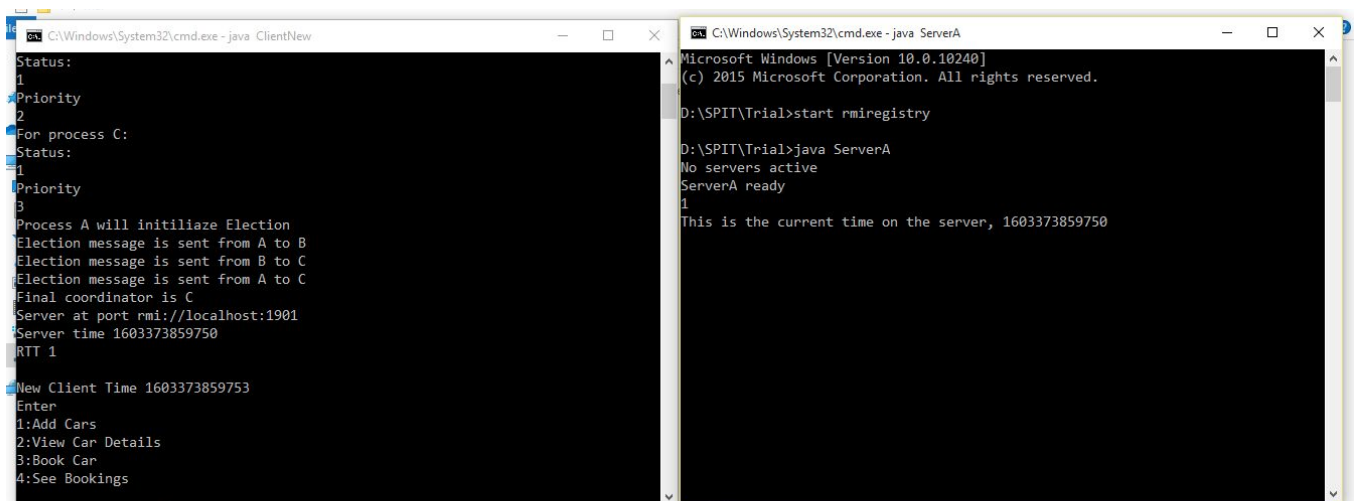
First the time of the client is calculated. Then we call the server to calculate the round trip time.  
$$rtt = (end - start) / 2$$

The round trip time is calculated with the above formula. Start is the time before calling the server. End is the time after response is received. The client clock is then updated by adding the roundtrip time to the server time. The client clock is then based on the offset to the server time.

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.

### Screenshots:



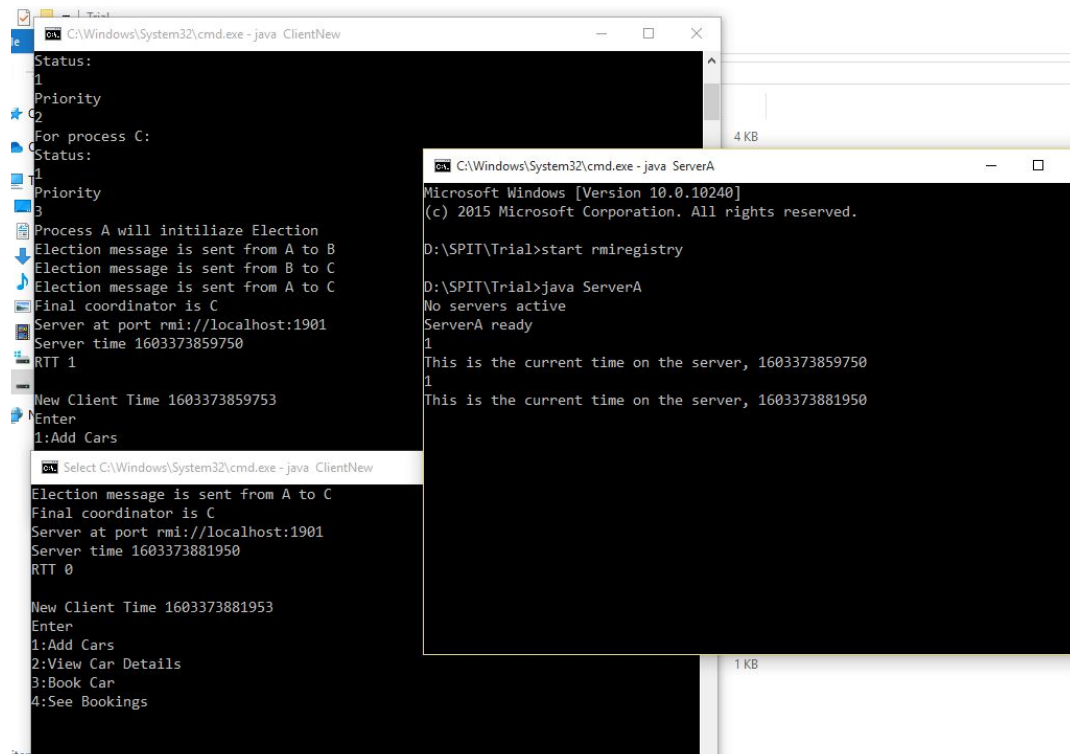
```
C:\Windows\System32\cmd.exe - java ClientNew
Status:
1
Priority
2
For process C:
Status:
1
Priority
3
Process A will initialize Election
Election message is sent from A to B
Election message is sent from B to C
Election message is sent from A to C
Final coordinator is C
Server at port rmi://localhost:1901
Server time 1603373859750
RTT 1
New Client Time 1603373859753
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings

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

D:\SPIT\Trial>start rmiregistry

D:\SPIT\Trial>java ServerA
No servers active
ServerA ready
1
This is the current time on the server, 1603373859750
```

A single client and server is there. The client time and server time is printed respectively.



The image shows two overlapping Windows command prompt windows. The background window is titled 'C:\Windows\System32\cmd.exe - java ClientNew' and displays the output of a Java client program. It shows a status of 1, priority 3, and a process A that initiates an election. It details election messages sent from A to B and B to C, identifies C as the final coordinator, and shows the server at port rmi://localhost:1901 with a server time of 1603373859750 and an RTT of 1. It also shows a new client time of 1603373859753 and a menu with options: 1:Add Cars, 2:View Car Details, 3:Book Car, 4:See Bookings. The foreground window is titled 'C:\Windows\System32\cmd.exe - java ServerA' and shows the output of a Java server program. It displays the Microsoft Windows version (10.0.10240), the copyright notice (c) 2015 Microsoft Corporation, and the command 'D:\SPIT\Trial>start rmiregistry'. It then shows 'D:\SPIT\Trial>java ServerA' with the output 'No servers active' and 'ServerA ready'. It also shows the current time on the server as 1603373859750 and 1603373881950.

```
C:\Windows\System32\cmd.exe - java ClientNew
Status:
1
Priority
3
For process C:
Status:
1
Priority
3
Process A will initilize Election
Election message is sent from A to B
Election message is sent from B to C
Election message is sent from A to C
Final coordinator is C
Server at port rmi://localhost:1901
Server time 1603373859750
RTT 1

New Client Time 1603373859753
Enter
1:Add Cars

Select C:\Windows\System32\cmd.exe - java ClientNew
Election message is sent from A to C
Final coordinator is C
Server at port rmi://localhost:1901
Server time 1603373881950
RTT 0

New Client Time 1603373881953
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings

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

D:\SPIT\Trial>start rmiregistry

D:\SPIT\Trial>java ServerA
No servers active
ServerA ready
1
This is the current time on the server, 1603373859750
1
This is the current time on the server, 1603373881950
```

There are 2 clients and a single server. Both the clients have different times. Each client prints its respective time. The server prints its time.

### Conclusion:

We have implemented clock synchronization. The server clock time as well as the client time is printed. Each time a new client connects, the time for that is printed. In this manner, clock synchronization was implemented and integrated in our car booking system.