

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.

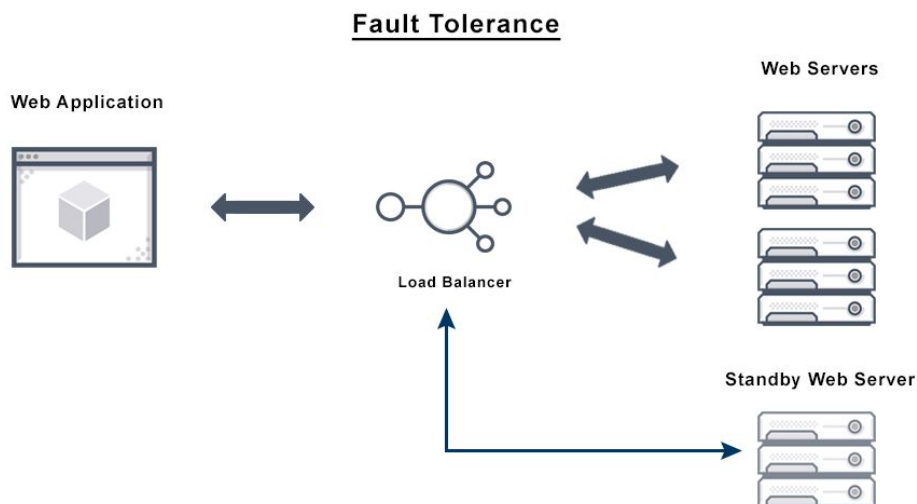
FAULT TOLERANCE

Fault tolerance refers to the ability of a system (computer, network, cloud cluster, etc.) to continue operating without interruption when one or more of its components fail.

The objective of creating a fault-tolerant system is to prevent disruptions arising from a single point of failure, ensuring the high availability and business continuity of mission-critical applications or systems.

Fault-tolerant systems use backup components that automatically take the place of failed components, ensuring no loss of service.

A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.[2] The term is most commonly used to describe computer systems designed to continue more or less fully operational with, perhaps, a reduction in throughput or an increase in response time in the event of some partial failure. That is, the system as a whole is not stopped due to problems either in the hardware or the software.



Code:

Implementation.java

```
balance.java LoadBalanceMain.java LoadBalance.java IpPool.java ImplExa
● ImplExample.java > Rmi > putCars(String, int)
261
262     }
263 }
264 public HashMap<String, Integer> getData(int serverId) {
265     HashMap<String, Integer> nmap = new HashMap<>();
266     if (serverId == 0) {
267         try {
268             Registry registry = LocateRegistry.getRegistry(null);
269             Rmi stub = (Rmi) registry.lookup("ServerB");
270             nmap = stub.returnData();
271         }
272     } catch (Exception e) {
273         try {
274             Registry registry = LocateRegistry.getRegistry(null);
275             Rmi stub = (Rmi) registry.lookup("ServerC");
276             nmap = stub.returnData();
277         }
278     }
279     catch (Exception er) {
280         System.out.println("No servers active");
281     }
282 }
283
284 }
285
286 else if (serverId == 1) {
287     try {
288         Registry registry = LocateRegistry.getRegistry(null);
289         Rmi stub = (Rmi) registry.lookup("ServerA");
290         nmap = stub.returnData();
291     }
292     catch (Exception e) {
293         try {
```

```
balance.java LoadBalanceMain.java LoadBalance.java IpPool.java ImplExa
● ImplExample.java > Rmi > putCars(String, int)
288         Registry registry = LocateRegistry.getRegistry(null);
289         Rmi stub = (Rmi) registry.lookup("ServerA");
290         nmap = stub.returnData();
291     }
292     catch (Exception e) {
293         try {
294             Registry registry = LocateRegistry.getRegistry(null);
295             Rmi stub = (Rmi) registry.lookup("ServerC");
296             nmap = stub.returnData();
297         }
298     }
299     catch (Exception er) {
300         System.out.println("No servers active");
301     }
302 }
303 }
304 } else if (serverId == 2) {
305     try {
306         Registry registry = LocateRegistry.getRegistry(null);
307         Rmi stub = (Rmi) registry.lookup("ServerA");
308         nmap = stub.returnData();
309     }
310     catch (Exception e) {
311         try {
312             Registry registry = LocateRegistry.getRegistry(null);
313             Rmi stub = (Rmi) registry.lookup("ServerB");
314             nmap = stub.returnData();
315         }
316     }
317     catch (Exception er) {
318         System.out.println("No servers active");
319     }
320 }
```

```

balance.java  LoadBalanceMain.java  LoadBalance.java  IpPool.java  Impl
ImplExample.java > Rmi > putCars(String, int)
306     Registry registry = LocateRegistry.getRegistry(null);
307     Rmi stub = (Rmi) registry.lookup("ServerA");
308     nmap = stub.returnData();
309
310 } catch (Exception e) {
311     try {
312         Registry registry = LocateRegistry.getRegistry(null);
313         Rmi stub = (Rmi) registry.lookup("ServerB");
314         nmap = stub.returnData();
315     }
316
317     catch (Exception er) {
318         System.out.println("No servers active");
319     }
320 }
321 }
322 }
323 return nmap;
324 }
325
326 public void setData(HashMap<String, Integer> a) {
327     map=a;
328 }
329 public HashMap<String,Integer> returnData() {
330     return map;
331 }
332
333
334 // map.put(car,i);
335 // this.i+=1;
336 // System.out.println(map);
337 // if (flag==0){

```

ServerC.java

```

LoadBalance.java  IpPool.java  ImplExample.java  ServerA.java  ImplExample.class
ServerC.java > ServerC > main(String[])
4  import java.rmi.server.UnicastRemoteObject;
5
6  public class ServerC extends ImplExample {
7      public ServerC() {}
8      Run | Debug
9      public static void main(String args[]) {
10         try {
11             // Clock synchronisation using Cristian's Algo
12             DefaultSystemTime obj1 = new DefaultSystemTime();
13             // Exporting the object of implementation class
14             SystemTime stub1 = (SystemTime) UnicastRemoteObject.exportObject(obj1, 0);
15             // Binding the remote object (stub) in the registry
16             //Naming.bind(objPath, obj1);
17             // Instantiating the implementation class
18             ImplExample obj = new ImplExample();
19
20             // Exporting the object of implementation class
21             // (here we are exporting the remote object to the stub)
22             Rmi stub = (Rmi) UnicastRemoteObject.exportObject(obj, 0);
23
24             // Binding the remote object (stub) in the registry
25             Registry registry = LocateRegistry.getRegistry();
26
27             registry.rebind("ServerC", stub);
28             map = stub.getData(2);
29             stub.setData(map);
30             System.out.println("ServerC ready");
31
32         } catch (Exception e) {
33             System.out.println("Server exception: " + e.toString());
34             e.printStackTrace();
35         }
36     }
37 }

```

Explanation of Implementation:

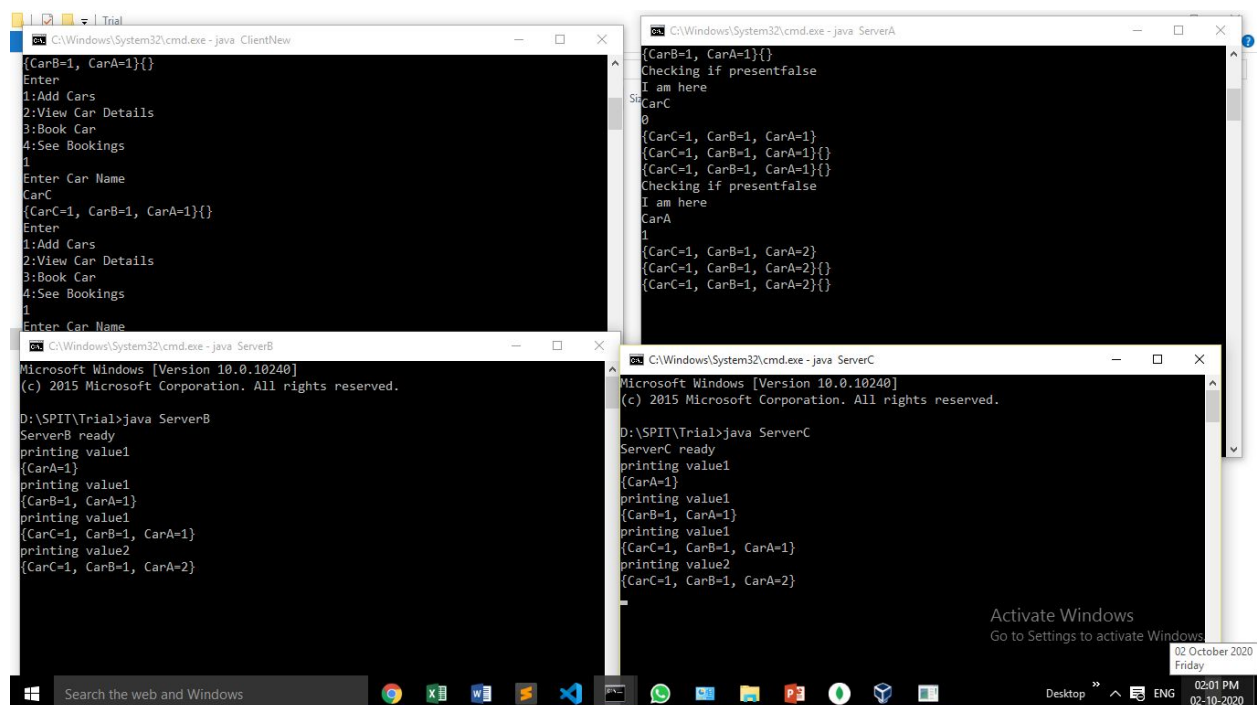
Each time a change or a new request is made to the server, it first checks if data already exists in the other server. If it does, it retrieves this data from the other server. If no other server is present or the data is the same, it adds the new data according to the new client request made. If one server crashes, then when it starts working again, it first checks the data store in the other replication servers and retrieves that data. In this way all the replication servers have the same data, no data is lost and data consistency is maintained.

When a server crashes, we lookup for the other servers, use the `getData()` function to get the data from the other replication servers and then use the `setData()` function to set the data in the server that has crashed. Every new request now will have the same data stored in all three replication servers and it will work as previously before one of the servers crashed.

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.
5. After adding a few cars, shut one server.
6. Now restart that server and see that the data will be retrieved.

Screenshots:



This is when all the servers are working normally and we have added a few cars which have been added to all the replication servers.

```
C:\Windows\System32\cmd.exe - java ClientNew
[CarC=1, CarB=1, CarA=2]{}
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
1
Enter Car Name
CarD
[CarC=1, CarB=1, CarA=2, CarD=1]{}
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings

C:\Windows\System32\cmd.exe - java ServerA
Impl.java:286)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java
:188)
    at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:172)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
    at java.net.Socket.connect(Socket.java:589)
    at java.net.Socket.connect(Socket.java:434)
    at java.net.Socket.<init>(Socket.java:211)
    at sun.rmi.transport.proxy.RMIDirectSocketFactory.createSocket(RMIDirect
SocketFactory.java:40)
    at sun.rmi.transport.proxy.RMIMasterSocketFactory.createSocket(RMIMaster
SocketFactory.java:148)
    at sun.rmi.transport.tcp.TCPEndpoint.newSocket(TCPEndpoint.java:613)
    ... 24 more
[CarC=1, CarB=1, CarA=2, CarD=1]{}

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

D:\SPIT\Trial>java ServerB
ServerB ready
printing value1
[CarA=1]
printing value1
[CarB=1, CarA=1]
printing value1
[CarC=1, CarB=1, CarA=1]
printing value2
[CarC=1, CarB=1, CarA=2]

D:\SPIT\Trial>

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

D:\SPIT\Trial>java ServerC
ServerC ready
printing value1
[CarA=1]
printing value1
[CarB=1, CarA=1]
printing value1
[CarC=1, CarB=1, CarA=1]
printing value2
[CarC=1, CarB=1, CarA=2]
printing value1
[CarC=1, CarB=1, CarA=2, CarD=1]
```

Now if serverB crashes, the data will be added to the other two servers. ServerA informs that replication serverB is not connected but it still updates its data stored according to the new request.

```
C:\Windows\System32\cmd.exe - java ClientNew
[CarC=1, CarB=1, CarA=2]{}
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
1
Enter Car Name
CarD
[CarC=1, CarB=1, CarA=2, CarD=1]{}
Enter
1:Add Cars
2:View Car Details
3:Book Car
4:See Bookings
1
Enter Car Name
CarE
[CarC=1, CarB=1, CarA=2, CarE=1, CarD=1]{}

C:\Windows\System32\cmd.exe - java ServerA
SocketFactory.java:40)
    at sun.rmi.transport.proxy.RMIMasterSocketFactory.createSocket(RMIMaster
SocketFactory.java:148)
    at sun.rmi.transport.tcp.TCPEndpoint.newSocket(TCPEndpoint.java:613)
    ... 24 more
[CarC=1, CarB=1, CarA=2, CarD=1]{}
Checking if presentfalse
I am here
CarE
0
[CarC=1, CarB=1, CarA=2, CarE=1, CarD=1]
[CarC=1, CarB=1, CarA=2, CarE=1, CarD=1]{}
[CarC=1, CarB=1, CarA=2, CarE=1, CarD=1]{}

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

D:\SPIT\Trial>java ServerB
ServerB ready
printing value1
[CarA=1]
printing value1
[CarB=1, CarA=1]
printing value1
[CarC=1, CarB=1, CarA=1]
printing value2
[CarC=1, CarB=1, CarA=2]

D:\SPIT\Trial>java ServerB
ServerB ready
printing value1
[CarC=1, CarB=1, CarA=2, CarE=1, CarD=1]

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

D:\SPIT\Trial>java ServerC
ServerC ready
printing value1
[CarA=1]
printing value1
[CarB=1, CarA=1]
printing value1
[CarC=1, CarB=1, CarA=1]
printing value2
[CarC=1, CarB=1, CarA=2]
printing value1
[CarC=1, CarB=1, CarA=2, CarD=1]
printing value1
[CarC=1, CarB=1, CarA=2, CarE=1, CarD=1]
```

Now we restart serverB. We can see that it retrieved the data from the other replication servers and now data consistency is maintained. Now every new request will work normally like before the server crashed.

Conclusion:

We implemented fault tolerance and saw how it is handled when a particular replication server crashes, how data consistency is maintained and how data loss is handled. Thus data will always be consistent in our system. In this manner, fault tolerance was implemented and integrated in our car booking system.