COMP6231 Project Report

Tianlin Yang	**40010303**
Yongxuan Zhang	**40084728**
Gaoshuo Cui	**40085020**
Haitun Liao	**40080732**

A Report

In the Department

of

Computer Science & Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Project of

COMP6231 (Distributed System) at

Concordia University

Montreal, Quebec, Canada

August 2019

TABLE OF CONTENTS

1.Overview

The Software Failure Tolerant and Highly Available CORBA Distributed Event Management System is a distributed system for a leading corporate event management company: a distributed system used by an event manager who manages the information about the events and customers who can book ,cancel or swap events across the company’s different branches. There have three branches in different cities: Toronto (TOR), Montreal (MTL) and Ottawa (OTW) for our implementation.We using active replication scheme where each server will act as a server replica to implement this system.

The users of the system are event managers and event booking customers. Event Managers and customers are identified by a unique manager ID and customer ID respectively. (e.g.: MTLC1234, OTWM1234). A customer can book, drop, swap events and view his schedule whereas a Manager can perform all the customer operations and even can add/drop event and List Event Availability.

Each server maintains its internal in-memory database, basically composed of a HashMap, which it uses to store various information and a user interacts with the server using the Java RMI connection. The inter server communication is done using UDP sockets. Each server maintains a log file for all the operation performed on it. A log file per user login is also maintained.

To the make the system more robust, after successful login, the user interaction is performed on a separate thread. Each logged in user communicates with its corresponding department server and performs necessary operations. Since multiple users access a server concurrently, so the proper synchronization of data is implemented in the code for thread safe communication. Moreover, the user input is case insensitive.

1.1 Develop tools

All code writing in Java IDE Eclipse, Java IDK version is Java 8. In order to generate Diagram for classes in the project, Object Aid Plugin has been used.

2.System Architecture

2.1 Basic Architecture

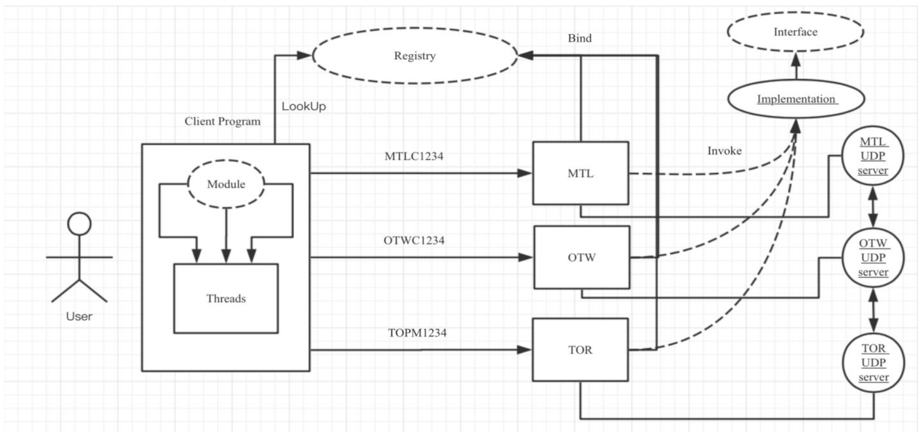
Three different servers (MTL, OTW, TOR) are started, which then start their own UDP servers for socket communication. Each of these servers bind their object reference in the registry using a unique name.

Apart from these servers, there’s a client program which handles the user interaction on the client program, a user log-in using its unique ID. This unique ID is validated to identify the server locations, role and ID of the user.

After successful login, a separate thread is started to handle the requests from this user. Since a thread is a light weight process, so this make the system more robust and responsive.

In the thread, we consult the registry to get reference of its corresponding server. Then after, based on the user type (manager or customer), a list of available operations is displayed.

In particular, a user only communicates with its corresponding server. But if the operation requires data from other departmental servers, then the user server makes a UDP request to the target server and gets the required data using a socket communication.



2.2 Advanced Structure Description

Software Failure Tolerant and Highly Available CORBA Distributed Event Management System implemented over the CORBA project implemented as part of assignment2. We implement a Front-End module which would accept the Client Request through CORBA. And Front End will send the request to the sequencer and the sequencer send the messages with a sequence id replica manager.

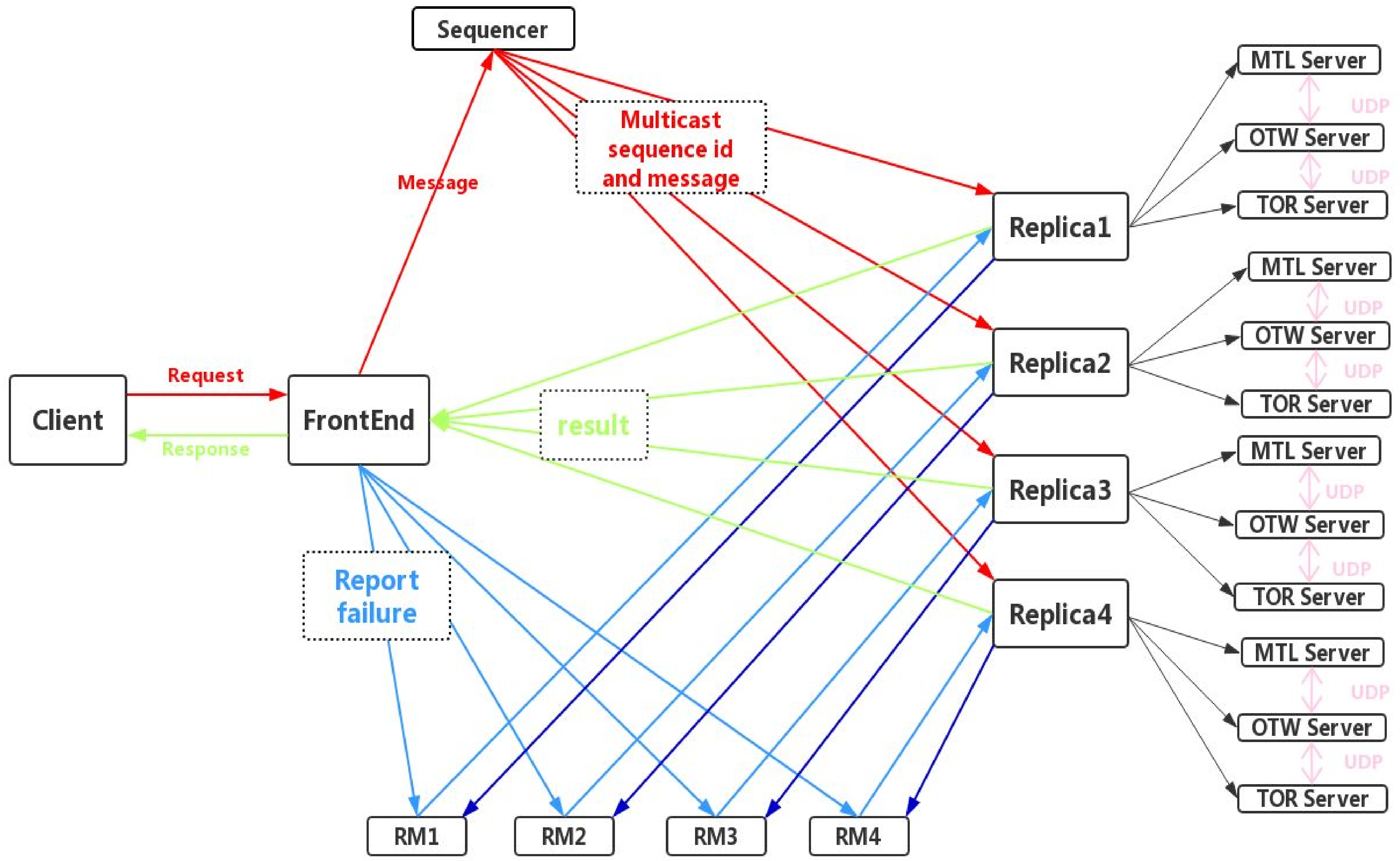
This is multicasting of each request from sequencer to the group of RMs. It requires totally ordered reliable multicast so that all RMs perform the same operations in the same order. And then RMs will process each request identically and send it to the corresponding servers.

Specific servers are going to execute that request and going to reply to the frontend. The communication between Front End and the sequencer established through UDP IP connection.

From the Sequencer to RMs multicast UDP IP connection. Communication between RMs and replica established by the UDP IP connection. Replica and frontend is going to communicate with each other by reliable UDP connection.

Here, the CORBA implementation is between client and front end instead of client and server. CORBA object and the implementation will be in the front end. But instead of logic it will call the UDP connection of sequencer and add one sequence id and send it to each RMs through Multicast UDP/IP from the sequencer. Then all the RMs are going to call the procedure from the server and will execute the function and will reply. And front end will receive the replies and correct result will be sent to the client.

- This system using active replication scheme where each server will act as a server replica . And will handle all the client requests using group replication and reliable communication using the front end, sequencer, replica manager and the replica's.
- Here each replica will receive the client's request with a unique sequence number from the sequencer and there by executes it in a total ordering scheme. Then the result is sent back to the front end of the server.
- Front end is responsible for receiving the client request and it is the one which forwards it to the sequencer for processing. It also receives the processed information from the replica's and sends it back to the client.
- Replica manager is used to maintain the replica's by creating and removing them when needed. It is also responsible for failure detection and recovery of replica's during failure crash or byzantine crash.
- Sequencer is used to receive the client request from the FE and assigns a unique sequencer no to it before multicasting it to all the replicas for processing.



Sequencer
Message
Multicast sequence id and message
Client
Request
Response
FrontEnd
result
Report failure
Replica1
Replica2
Replica3
Replica4
MTL Server
OTW Server
TOR Server
UDP
RM1
RM2
RM3
RM4

2.3Data Flow

1)Cient to replicas

- Client sends requests to FrontEnd by RPC, FrontEnd send back responses.
- FrontEnd send the message to sequencer through UDP (must be guaranteed to arrive to sequencer).
- Sequencer multicasts the message from FrontEnd plus a unique sequence id to all the replicas.
- Once results are delivered from replicas to FrontEnd , it will take the major result and responds to client.
- If some failures happen, the FrontEnd will report them the replica managers.
- The replica manager communicates with each other by heartbeat to make decision when replica is not valid and to relaunch the replica.
- The replica managers also receive failure reports from the FrontEnd.

2)Replicas to server

- Replicas send requests to server by RPC, server do operations and send back response.
- And three servers use UDP to communicate with each other.

2.4 Directory structure

Package	Function	Classes
Client	User interface.	Client.java Customers.java Managers.java User.java
FECorba	Generated by compiler.	_FrontEndStub.java FrontEnd.java FrontEndHelper.java FrontEndHolder.java FrontEndOperations.java FrontEndPOA.java listHelper.java listHolder.java
FrontEnd	Timer,Front end and emplementation of it.	FrontEnd.java FrontEndImpl.java Timer.java
functions	Check whether the information of city,role and eventtype is correct.	City.java Constants.java EventType.java FunctionMembers.java Role.java
logTool	Record the information.	allLogger.java logFormat.java
MultiThreadTest	Initial the date and do operations at the same time.	DataInitialize.java MultiClient.java MultiCustomers.java MultiManagers.java
PortInformation	Information about ports.	AdderssInfo.java AlivePort.java FEPort.java Replica.java Replication.java SequencerPort.java
ReplicaHost1	One replica, contain three servers.	BackupThread.java CheckAlive.java FailureEnum.java LoggerFormatter.java

		MainThread.java MessageForReplica.java Replica1.java Replica1Manager.java TimerForSleep.java
ReplicaHost1Remote Object	Implementation of operations.	EventSystemImplementation.ja va EventSystemSTHWrong.java
Sequencer	receive the client request,assigns a unique sequencer and multicasting it to all the replicas.	sequencer.java

3.Key Features

- Software failure tolerant and highly available: We designed 4 replicas to implement active replication, each time front end will compare the messages send back from each replica and return the correct message to client. We let replica 3 return wrong messages to implement software (non-malicious Byzantine) failure and let replica 4 sleeping at the beginning to implement crash . After 3 times failure replica 3 will recovery , and replica 4 will recovery after the sleep time. Then 4 replicas succesfully run at the same time.
- Sequencer: The Sequencer is the central part of the DEMS. It integrates the actions of the Front end with those of the Replicas so as to guarantee the total order at the part of the replica in order to maintain the consistency in the order in which the replicas executes the requests given by the client.It accepts the

request from the FRONT END and multicast the request to the replicas running.

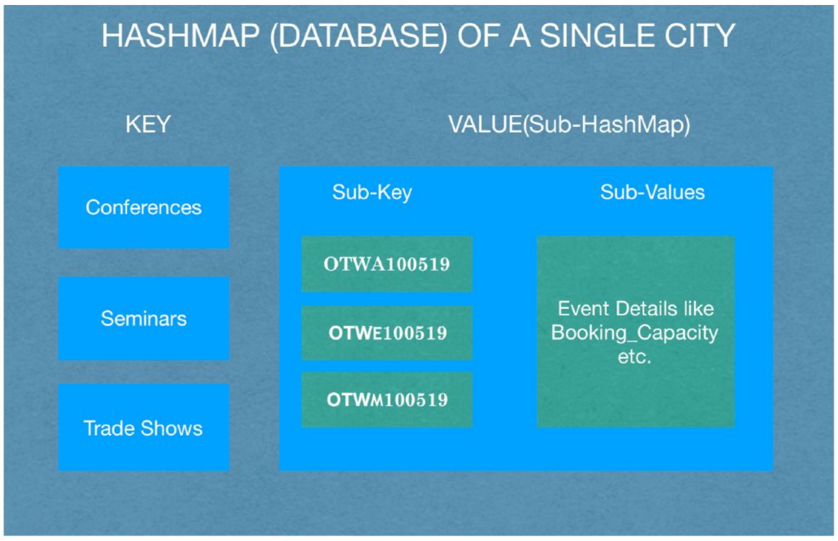
Mode of Operation of Sequencer: The sequencer accepts the Request from the Front End in The Form of UDP message and that UDP message contains the data in the form of the serialization and the Sequencer DE serializes the data and after the de serialization is performed the amendments are made by the sequencer in the Serial Number field of the de serialized object and after putting in the Sequence Number to guarantee the total order it multicast the message to the Replicas.

The Order of Operation:

1. Accepting the UDP packet from the Front End: The Replica Manager has the particular service running that is waiting continuously for the UDP packet from the Front end.
2. De Serialization of the Data: The Sequencer De serializes the Data received from the From the Front End.
3. Guarantee The Total Order: The sequencer Guarantee the total order By appending the sequence number in the object that it de serialized from the UDP packet sent by the sequencer.
4. Serialization of the Data: The sequencer then Serializes the Data that it after appending sequence number into it.
5. Multicasting the Request to the Replicas: After the Request has been Serialized again then it is multicasted to the Replicas to Process the request.

- Reliable UDP:UDP protocol is been used between all the front end - sequencer, sequencer-replica and replica-replica managers. And the protocol is made reliable using ack message and time out specifications.
- Replica: This system using active replication scheme where each server will act as a server replica . And will handle all the client requests using group replication and reliable communication using the front end, sequencer, replica manager and the replica's.
- Replica Manager: Replica manager is used to maintain the replica's by creating and removing them when needed. It is also responsible for failure detection and recovery of replica's during failure crash or byzantine crash.
- Data type for parameters: All the parameters passed are of type "String" and thus comply with the interface contract.
- In-Memory database: Each server maintains its in-memory database. This database is implemented as:

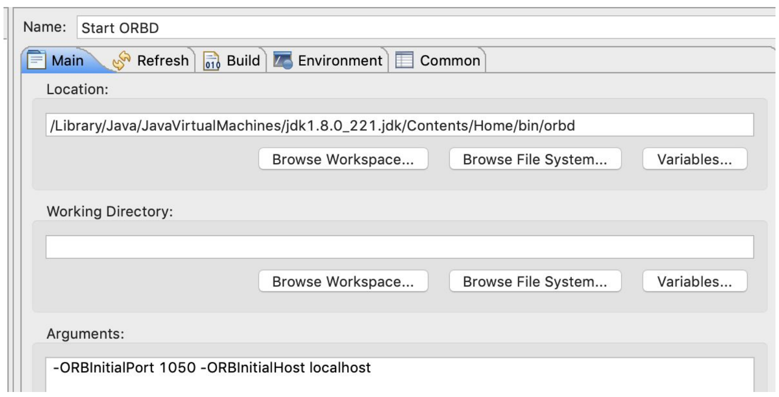
  HashMap<String,HashMap<String,HashMap<String,Object>>>cityDatabase



- Case insensitive: The user input is case insensitive, all the letters entered are case-insensitive. For users, this setting will make some operations easier.
- Tread safe: Since each user runs on a thread. So, the system needs to be able to handle concurrent requests and provide thread safety to its data. This is achieved by using **"*synchronized*"** blocks whenever there is a need to update the in-memory database.
- Logging: Custom logger is used to log all the messages. Each user operation is logged into the user specific log file. The same is true for departmental servers too.

4.TestCases

Firstly ,configure eclipse to ensure we can use CORBA without the plugin.



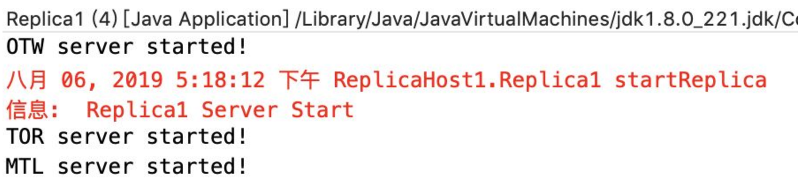
Name: Start ORBD

Location: /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/orbd

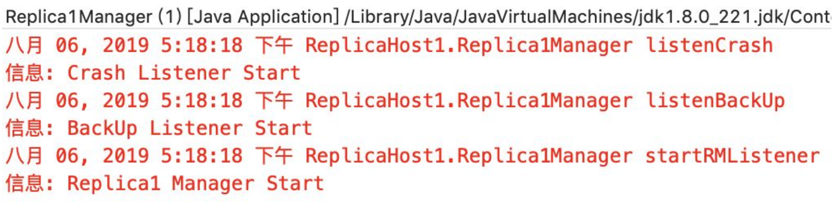
Working Directory:

Arguments: -ORBInitialPort 1050 -ORBInitialHost localhost

Start Replica and ReplicaManager(from 1 to 4).

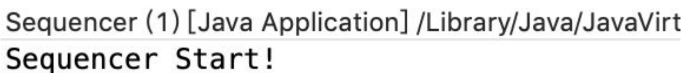


```
Replica1 (4) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/C
OTW server started!
八月 06, 2019 5:18:12 下午 ReplicaHost1.Replica1 startReplica
信息:  Replica1 Server Start
TOR server started!
MTL server started!
```



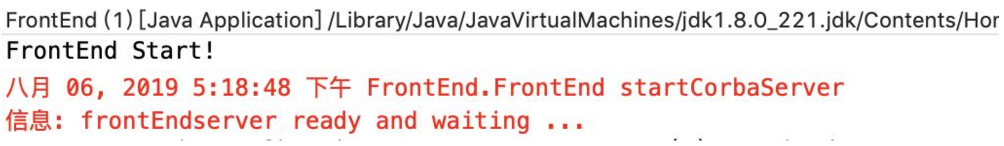
```
Replica1Manager (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Cont
八月 06, 2019 5:18:18 下午 ReplicaHost1.Replica1Manager listenCrash
信息: Crash Listener Start
八月 06, 2019 5:18:18 下午 ReplicaHost1.Replica1Manager listenBackUp
信息: BackUp Listener Start
八月 06, 2019 5:18:18 下午 ReplicaHost1.Replica1Manager startRMListener
信息: Replica1 Manager Start
```

Start Sequencer.



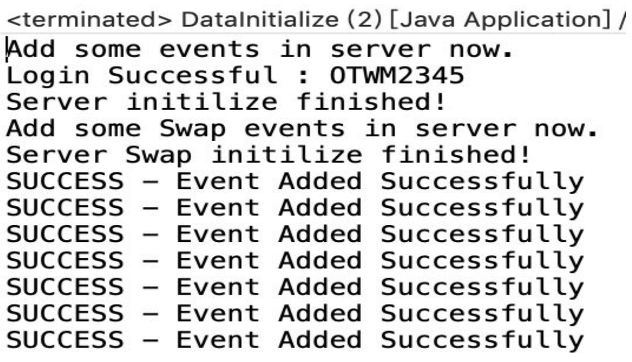
```
Sequencer (1) [Java Application] /Library/Java/JavaVirt
Sequencer Start!
```

Start FrontEnd.



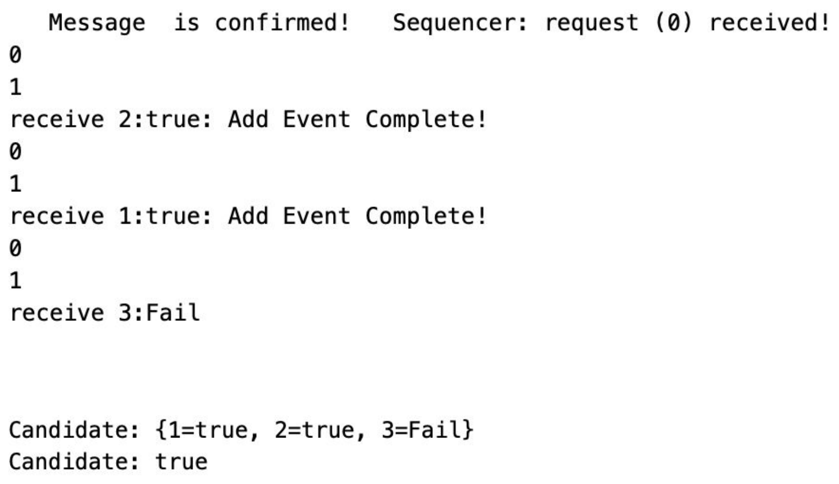
```
FrontEnd (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Hor
FrontEnd Start!
八月 06, 2019 5:18:48 下午 FrontEnd.FrontEnd startCorbaServer
信息: frontEndserver ready and waiting ...
```

Run DataInitialize to initialize the data, then runMultiClient to do operations.



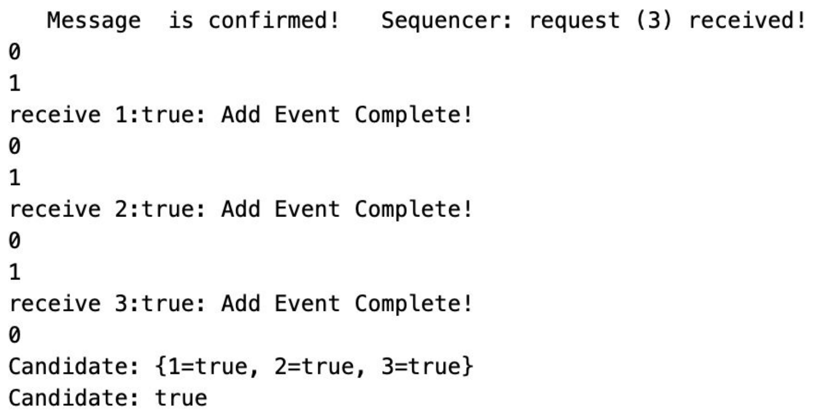
```
<terminated> DataInitialize (2) [Java Application] /
Add some events in server now.
Login Successful : OTWM2345
Server initilize finished!
Add some Swap events in server now.
Server Swap initilize finished!
SUCCESS - Event Added Successfully
SUCCESS - Event Added Successfully
SUCCESS - Event Added Successfully
SUCCESS - Event Added Successfully
SUCCESS - Event Added Successfully
SUCCESS - Event Added Successfully
SUCCESS - Event Added Successfully
```

As our setting we can see information in FE shows first time replica 1 and 2 can successfully run , replica 3 fail and replica 4 crashed. After comparation Front End will return the correct massage to client.(System can tolerate software failure and crash at the same time.)



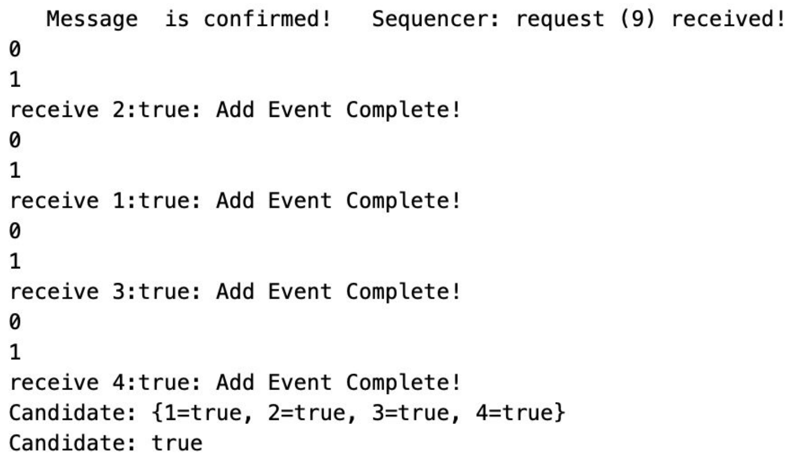
```
   Message  is confirmed!   Sequencer: request (0) received!
0
1
receive 2:true: Add Event Complete!
0
1
receive 1:true: Add Event Complete!
0
1
receive 3:Fail


Candidate: {1=true, 2=true, 3=Fail}
Candidate: true
```

After 3 times of failure, replica 3 will recover and successfully run.



```
   Message  is confirmed!   Sequencer: request (3) received!
0
1
receive 1:true: Add Event Complete!
0
1
receive 2:true: Add Event Complete!
0
1
receive 3:true: Add Event Complete!
0
Candidate: {1=true, 2=true, 3=true}
Candidate: true
```

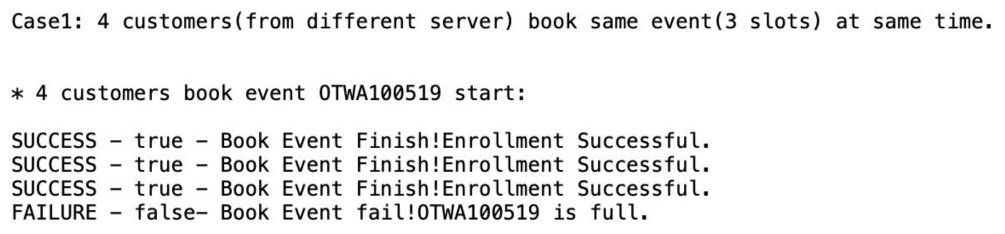
After sleep time, replica 4 will recover from Crash and succesfully run.



```
   Message  is confirmed!   Sequencer: request (9) received!
0
1
receive 2:true: Add Event Complete!
0
1
receive 1:true: Add Event Complete!
0
1
receive 3:true: Add Event Complete!
0
1
receive 4:true: Add Event Complete!
Candidate: {1=true, 2=true, 3=true, 4=true}
Candidate: true
```

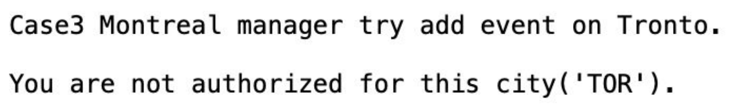
Here are some senarios of test case.

- Only 3 customers can successfully book event when 4 customers book same event(3 slots) at same time .



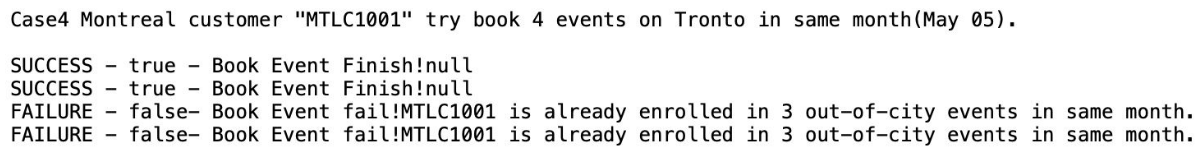
```
Case1: 4 customers(from different server) book same event(3 slots) at same time.


* 4 customers book event OTWA100519 start:

SUCCESS - true - Book Event Finish!Enrollment Successful.
SUCCESS - true - Book Event Finish!Enrollment Successful.
SUCCESS - true - Book Event Finish!Enrollment Successful.
FAILURE - false- Book Event fail!OTWA100519 is full.
```

- Manager cannot add event for other cities.



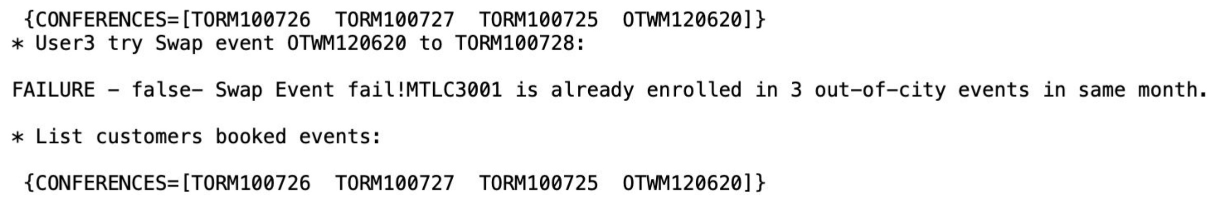
```
Case3 Montreal manager try add event on Tronto.

You are not authorized for this city('TOR').
```

- Customer cannot book events over 3 times in other city in the same month.



```
Case4 Montreal customer "MTLC1001" try book 4 events on Tronto in same month(May 05).

SUCCESS - true - Book Event Finish!null
SUCCESS - true - Book Event Finish!null
FAILURE - false- Book Event fail!MTLC1001 is already enrolled in 3 out-of-city events in same month.
FAILURE - false- Book Event fail!MTLC1001 is already enrolled in 3 out-of-city events in same month.
```

- Swap event for a customer who already booked 3 events in other cities in the same month will fail.



```
 {CONFERENCES=[TORM100726  TORM100727  TORM100725  OTWM120620]}
* User3 try Swap event OTWM120620 to TORM100728:

FAILURE - false- Swap Event fail!MTLC3001 is already enrolled in 3 out-of-city events in same month.

* List customers booked events:

 {CONFERENCES=[TORM100726  TORM100727  TORM100725  OTWM120620]}
```

5.Task of each team member:

Tianlin Yang :

Design and implement the replica manager (RM) which creates and initializes the actively replicated server subsystem. The RM also implements the failure detection and recovery for both types of failures

Yongxuan Zhang:

Design and implement the front end (FE) which receives a client request as a CORBA invocation, forwards the request to the sequencer, receives the results from the replicas and sends a single correct result back to the client as soon as possible. The FE also informs all the RMs of a possibly failed replica that produced incorrect result.

Gaoshuo Cui:

Design and implement a failure-free sequencer which receives a client request from the FE, assigns a unique sequence number to the request and reliably multicast the request with the sequence number and FE information to all the four server replicas.

Haitun Liao:

Design proper test cases for all possible error situations and implement a client program to run these test cases.

Each Student:

Modify the server implementation from Assignment 2 to ensure it can work as a (non-CORBA) server replica. A server replica receives the client requests with sequence numbers and FE information from the sequencer, executes the client requests in total order according to the sequence number and sends the result back to the FE.

Integrate all the modules properly, deploy the application on a local area network, and test the correct operation of the application.

6.Important part/Difficulty:

Setting IDL and ORB properties, Concurrency, MultiThreading, UDP, Sequencer, Replica.

7.References:

- https://www.tutorialspoint.com/java_rmi/java_rmi_application.htm
- https://docs.oracle.com/javase/tutorial/rmi/index.html
- https://www.mkyong.com/java/java-rmi-distributed-objects-example/
- https://www.oreilly.com/library/view/learning-java-4th/9781449372477/ch13s04.html
- https://stackoverflow.com/questions/36739915/sending-rmi-stub-over-udp
- https://cs.nyu.edu/courses/fall02/G22.3033-009/Lectures/lecture%20006.pdf