DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6231, Summer 2019

Instructor: Sukhjinder Narula

Distributed Event Management System

Prepared By:

Himen Sidhpura (40091993)

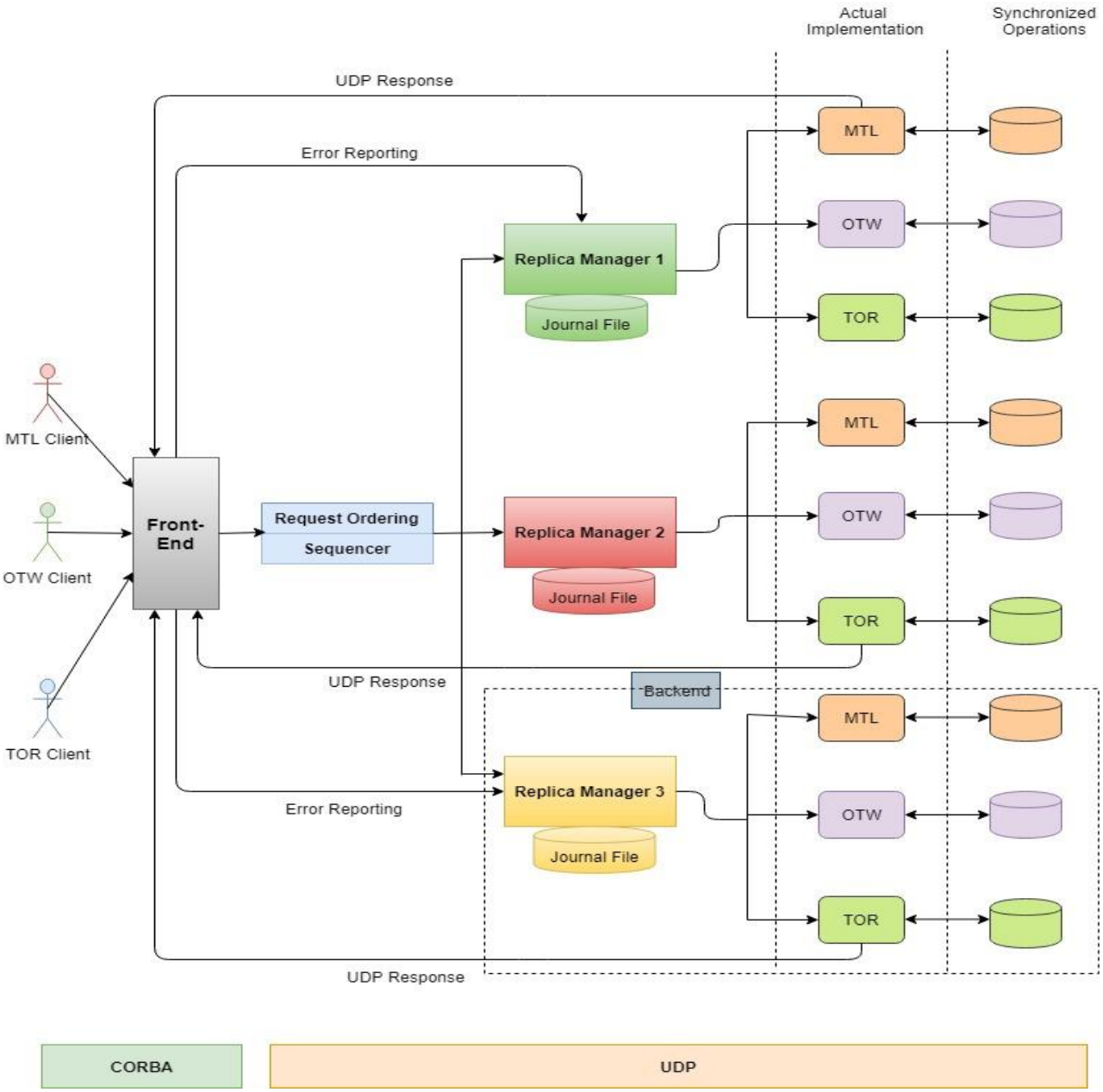
Jenny Mistry (40092281)

Ayush Dave (40080515)

Overall Description:

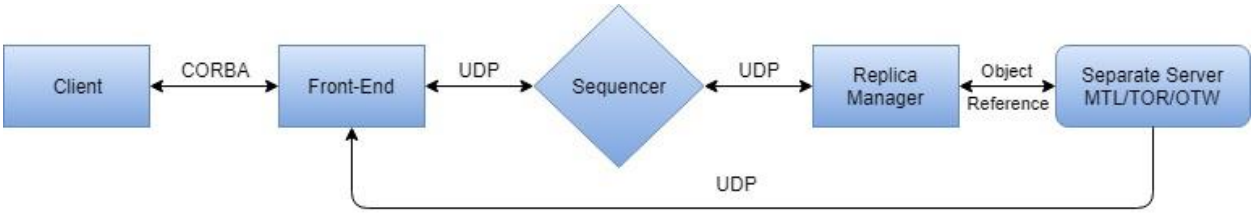
Event management is implemented as a distributed system to book and manage events across different branches of a corporate event management company. The system exposes CORBA Features using active replication and the users can see a single system handling user requests providing location and language transparency. It also manages simultaneous requests with adequate synchronization with the help of multithreading.

Design Architecture:

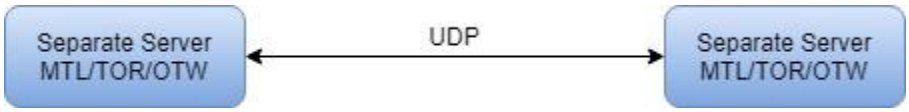


Communication about Components:

Communication paradigm among heterogeneous components



Communication paradigm among heterogeneous components



Overall Workflow:

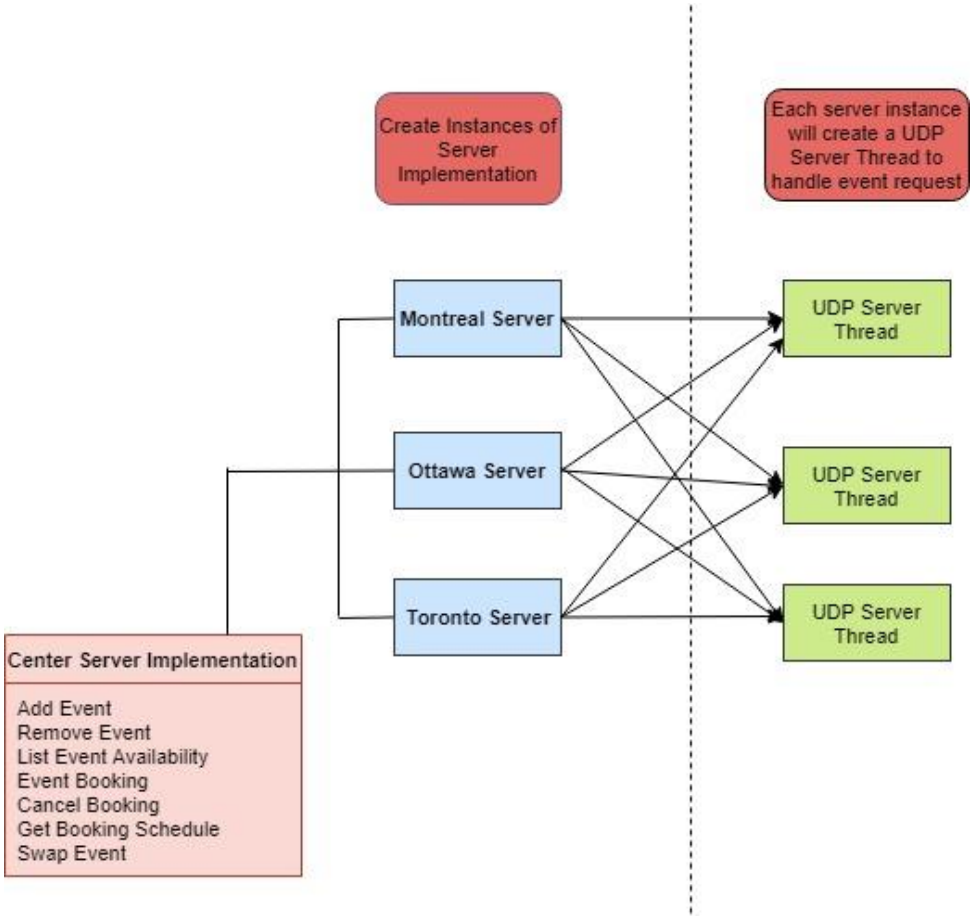
- The client sends request to Front End using CORBA implementation.
- Front End register with CORBA Naming Service.
- After the FE forwards the request to sequencer, a unique sequence number is appended to the request and multicast to all the replica managers by the sequencer.
- Thus a sequencer keeps a track of order of all incoming requests.
- The replicas will store the request in a buffer and make sure that request is executed using total ordering. Replicas will buffer last processed results along with request id to avoid any duplicate request processing.
- The individual servers will process the request and send the result to FE.
- When the FE receives the response from the replicas, it perform the following:
    - The FE will inform about the error to the corresponding replica manager with the request id if any inconsistent result is found.
    - The FE will report a process crash to a replica manager if there is no response for a predefined amount of time from the replica.
    - The FE provides the response to the client after taking majority from the results of all the replicas.
- If a Front End receives different response from any Replica Manager for three consecutive times, the Front End declares the corresponding RM to be faulty. On detection of software bug, the respective RM will replace the faulty replica.
- On receiving no response from a Replica Manager, the Front End will announce server crash from respective replica and initiates the process of fault tolerance by restarting the

server and resume the current state by synchronizing the data with the remaining server to maintain data consistency.

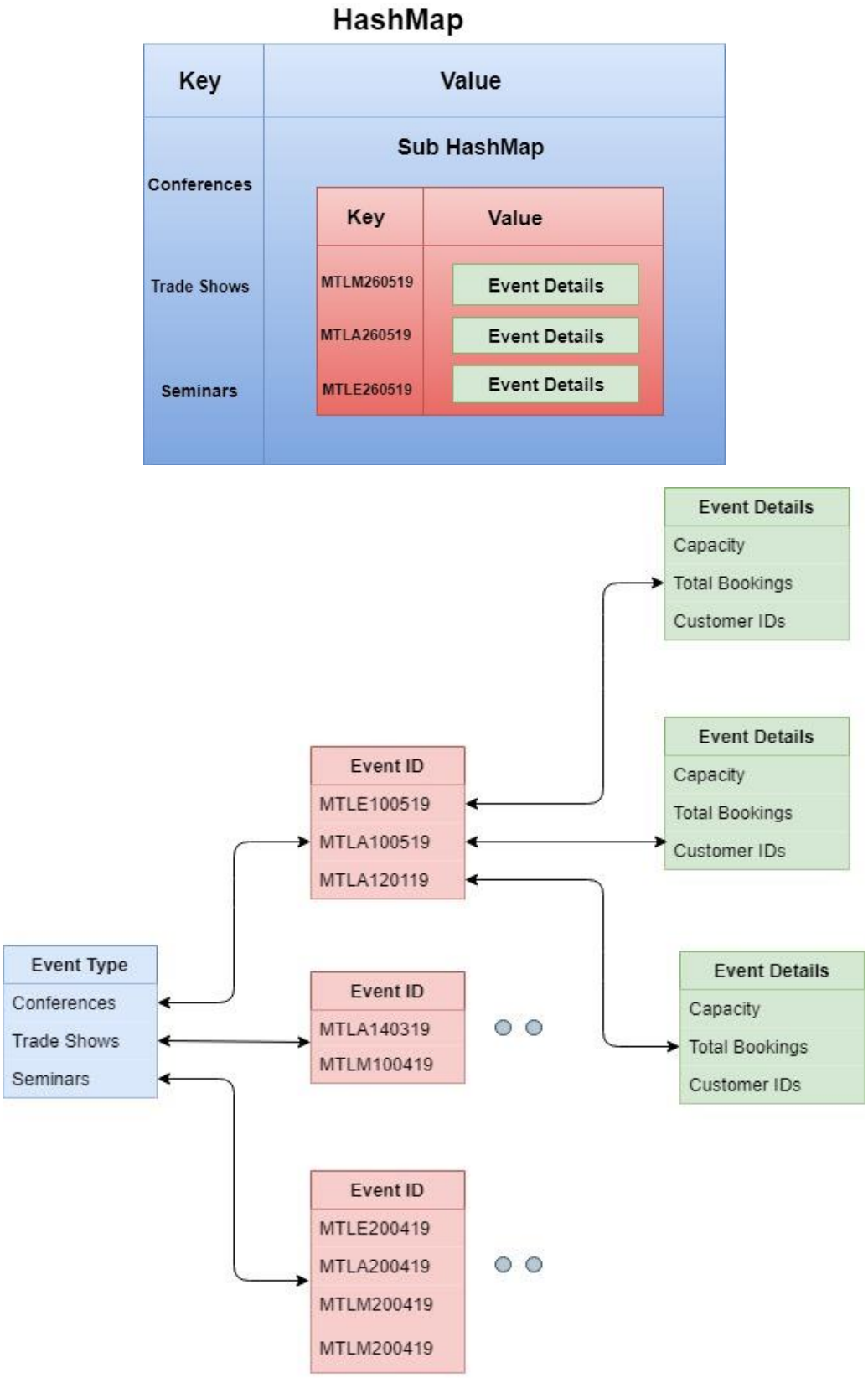
Intermediate Failure:

- The kind of failure implemented in this project is: **Non-Malicious Byzantine failure**.
- On the event of third party attack, we have attempted to demonstrate the buggy behavior or corruption of messages in the system until three consecutive error reports from Front End.
- After the attacked replica is identified, failure recovery mechanism is initiated by the corresponding replica.
- The remaining replica will send their respective data to the affected replica to bring it back to the consistent state and make the data synchronized.

Server Architecture:



Data Models:


HashMap
Key
Value
Conferences
Trade Shows
Seminars
Sub HashMap
Key
Value
MTLM260519
Event Details
MTLA260519
Event Details
MTLE260519
Event Details
Event Details
Capacity
Total Bookings
Customer IDs
Event Details
Capacity
Total Bookings
Customer IDs
Event ID
MTLE100519
MTLA100519
MTLA120119
Event Type
Conferences
Trade Shows
Seminars
Event ID
MTLA140319
MTLM100419
Event Details
Capacity
Total Bookings
Customer IDs
Event ID
MTLE200419
MTLA200419
MTLM200419
MTLM200419

Algorithm:

Active Replication:

In this method, the Sequencer will multicast the client's request to all the Resource Managers (RMs). The following events happen as a part of this replication method:

1. Request: The sequencer after receiving request from the FE (Front End), will multicast to each of the RMs using a totally ordered (FIFO), reliable multicast primitive. Next request will only be sent after the response for the first request has been received.

2. Coordination: The group communication system ensures that the request is received by each of the RMs in the same (total) order.

3. Execution: Each RM would process the request in the same total order as all the RMs are state machines and in an ideal scenario should respond identically. Each response would contain a unique request identifier.

4. Agreement: As multicast delivery semantics is used, this phase is not needed.

5. Response: Every RM sends its response to the FE. Depending on the type of failure, the system handles, for e.g. for crash failure where FE would send the first response to the client and inform the RMs if such a failure is detected or for software failure, FE would select the result with consensus, send that response to the client and inform the RMs in case of such failure.

The goal of our system is to handle either crash or software failure. So, to handle 1(f) crash failure, 2(f+1) replicas will be needed. In case of software failure, to handle 1(f) failure, 3(2f+1) replicas are needed. So, in general we have decided to maintain 3 replicas in order to handle both the failures.

Logs:

To perform logging for troubleshooting on both server and client end, we have utilized the logger functionality of Java (java.util.logging).

Log Format:

Each log data comprises of the below mentioned details:

- Date and time the request was sent.
- Request type (book an event, cancel an event, etc.).

- Request parameters (clientID, eventID, etc.).
- Request successfully completed/failed.
- Server response for the particular request.

Center Server:

Each server log (Montreal, Ottawa, Toronto) will be saved in their respective folder

- logs/MTL.txt
- logs/OTW.txt
- logs/TOR.txt

These logs include:

- Event added
- Event cancelled
- Availability of events
- Events swapped

Client:

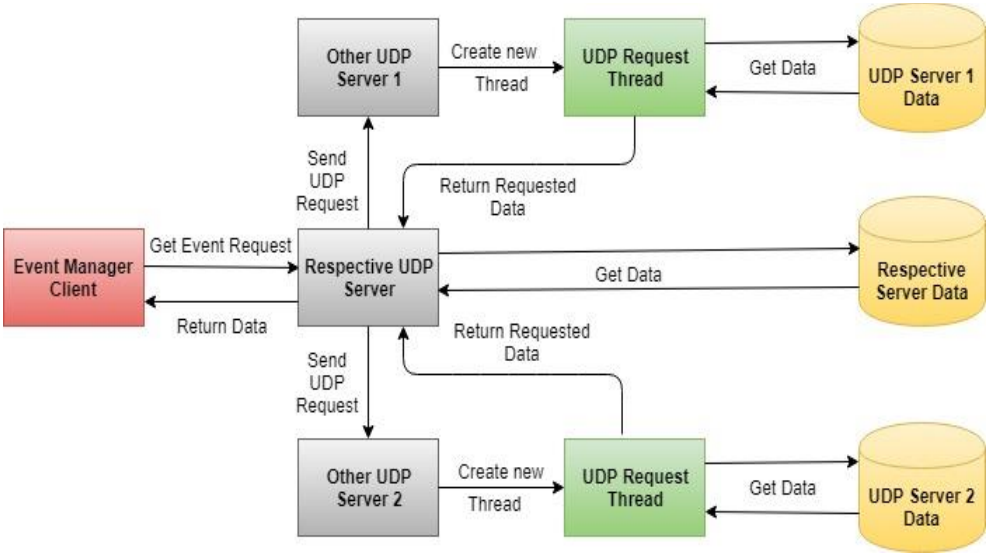
For every action performed by the client, a log file with clientID is created such as:

- Booking an event
- Canceling an event
- Retrieving booking schedule

Implementation:

- A separate logger file is created for each of the three servers.
- To save contents of the corresponding log file, a file handler is used.
- Various server responses are recorded using levels like WARNING, ERROR etc.

UDP Server Design:



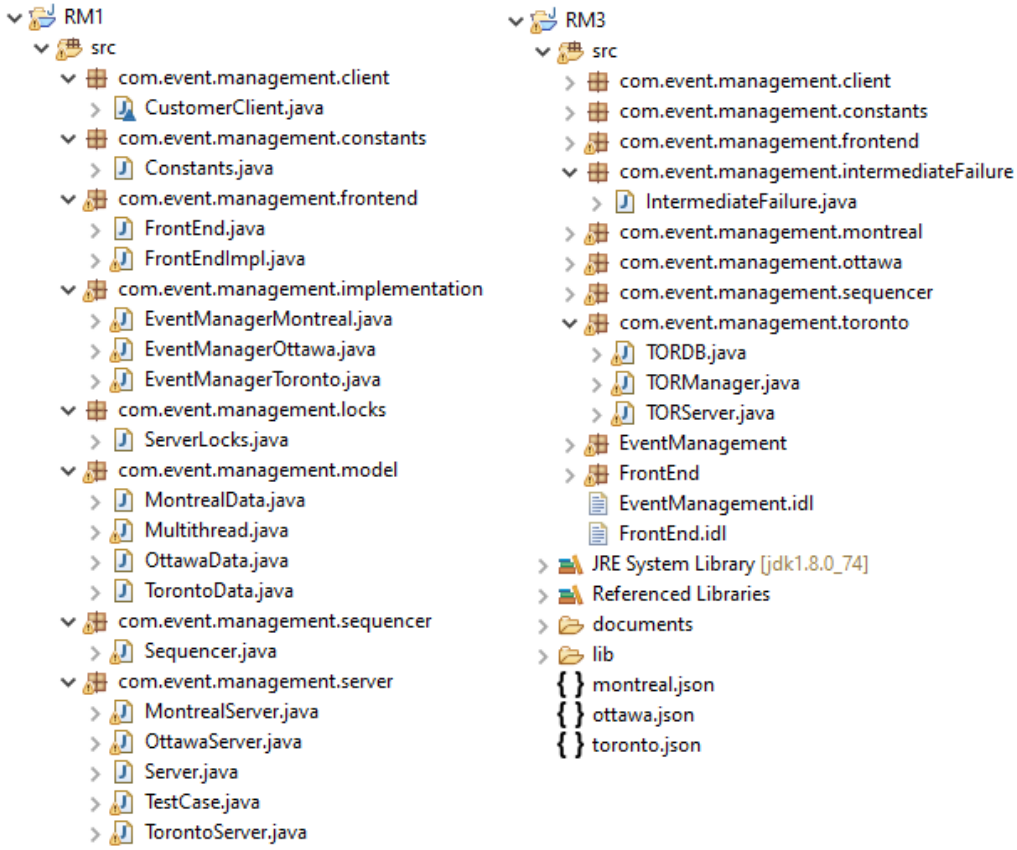
Flow:

- The event manager client sends event request to the respective server.
- The server fetches the requested data.
- It forks new requests to send the event request to the other servers located at various locations.
- The UDP servers at these locations receives the request and creates new threads to process the request.
- The newly created threads fetches the respective data and responds to the request.
- The server which received the request responds to the manager client with appropriate data.

Concurrency:

- The manager client creates new thread to communicate to each of servers to handle requests for same or different events at the same time.
- RM1 and RM2 uses Reentrant locks for concurrent transactions whereas RM3 has utilized synchronized identifier. Hence, the system has a fine grain locking managing mutual exclusions.

Code Structure:



Challenges:

- Implementation of synchronization while managing multiple event requests at the same time has been challenging.
- The response of each RM are different. Hence, we faced few challenges to change the return types of some replicas to make the systems to coordinate with each other.
    - For the transmission of data amongst the servers over the network, we used JSON file and objects for the data communication. This transmission flow of data was efficient, but the main challenge was to change all the return type of methods of replicas so that the data format in the JSON file is the same.
- Creation of unique request id by sequencer for identifying each request uniquely.
- Implementing algorithms for total ordering of incoming requests was a time-consuming task.
- It was a tough task to maintain consistent states of all the replicas after recovering from intermediate failures, server crash and software bug. Additional code had to be written to replicate data from all the non-faulty replicas.
- Since one RM was using synchronized identifier and the other RMs were handling multiple concurrent requests with the help of reentrant locks, we had to modify various methods and functionalities at code level to make it work together flawlessly.

Test Scenarios:

- If the availability of an event is full, more customers cannot book the event.
- A customer can book as many events in his/her own city, but only at most 3 events from other cities overall in a month.
- A customer can perform only customer operation and cannot perform any event manager operation but an event manager can perform all operations for its own branches.
- If the user tries to add an event with an event id already added, then event details get updated.
- The user gets an error message “No events available”, if he/she tries to add an event which is not created by manager.
- All the user and manager event requests have been synchronized to handle multiple concurrent event requests for the same/different branches.
- The swap event is successful only if old event is removed and new event add operation are successful.
- If old/new event does not exist for swap event, an error message is shown.
- The swap event throws an error if new event add operation exceeds the month’s max limit.
- After three consecutive error reports from a replica manager, the software bug is detected from the corresponding replica manager and is declared faulty.

- The system is made fault tolerant. Hence, when the fault is identified, the bug is resolved and the replica is up with the current server state.

Software Bug Test Case:

Even though, there are no events added into the system initially, RM3 shows some data depicting buggy behavior. For three consecutive list Event requests, Replica Manager 3 will provide buggy data after which the bug is identified from the respective RM and reconciled to consistent state.

```
Aug 07, 2019 4:45:32 PM com.event.management.frontend.FrontEndImpl listEventAvailability
INFO: waiting for response...
Aug 07, 2019 4:45:33 PM com.event.management.frontend.FrontEndImpl ReplicaOneReply
INFO: RM 1 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:45:33 PM com.event.management.frontend.FrontEndImpl ReplicaThreeReply
INFO: RM 3 : TORM1324  :  Seminars = MTLM020819 2,
Aug 07, 2019 4:45:33 PM com.event.management.frontend.FrontEndImpl ReplicaTwoReply
INFO: RM 2 : No Data Found or Might be data issue. Please try again
 RM3_FAIL_COUNTER 1
Aug 07, 2019 4:45:51 PM com.event.management.frontend.FrontEndImpl listEventAvailability
INFO: waiting for response...
Aug 07, 2019 4:45:51 PM com.event.management.frontend.FrontEndImpl ReplicaOneReply
INFO: RM 1 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:45:51 PM com.event.management.frontend.FrontEndImpl ReplicaTwoReply
INFO: RM 2 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:45:51 PM com.event.management.frontend.FrontEndImpl ReplicaThreeReply
INFO: RM 3 : TORM1324  :  Seminars = MTLM020819 2,
 RM3_FAIL_COUNTER 2
Aug 07, 2019 4:46:09 PM com.event.management.frontend.FrontEndImpl listEventAvailability
INFO: waiting for response...
Aug 07, 2019 4:46:09 PM com.event.management.frontend.FrontEndImpl ReplicaOneReply
INFO: RM 1 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:46:09 PM com.event.management.frontend.FrontEndImpl ReplicaThreeReply
INFO: RM 3 : TORM1324  :  Seminars = MTLM020819 2,
Aug 07, 2019 4:46:09 PM com.event.management.frontend.FrontEndImpl ReplicaTwoReply
INFO: RM 2 : No Data Found or Might be data issue. Please try again
 RM3_FAIL_COUNTER 3
Aug 07, 2019 4:46:12 PM com.event.management.frontend.FrontEndImpl udpReply
INFO: FRONTEND : RM1 sending to RM3
Aug 07, 2019 4:47:26 PM com.event.management.frontend.FrontEndImpl listEventAvailability
INFO: waiting for response...
Aug 07, 2019 4:47:26 PM com.event.management.frontend.FrontEndImpl ReplicaOneReply
INFO: RM 1 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:47:26 PM com.event.management.frontend.FrontEndImpl ReplicaThreeReply
INFO: RM 3 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:47:26 PM com.event.management.frontend.FrontEndImpl ReplicaTwoReply
INFO: RM 2 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:47:48 PM com.event.management.frontend.FrontEndImpl listEventAvailability
INFO: waiting for response...
```

Server Crash Test Case:

When there is no response from the Replica Manager 3, the Front End declares server crash from the respective RM. The system starts recovering from the crash by sending the data from the most active replica.

```
INFO: waiting for response...
Aug 07, 2019 4:48:14 PM com.event.management.frontend.FrontEndImpl ReplicaOneReply
INFO: RM 1 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:48:14 PM com.event.management.frontend.FrontEndImpl ReplicaThreeReply
INFO: RM 3 : Server Crash
Aug 07, 2019 4:48:14 PM com.event.management.frontend.FrontEndImpl ReplicaTwoReply
INFO: RM 2 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:50:29 PM com.event.management.frontend.FrontEndImpl listEventAvailability
INFO: waiting for response...
Aug 07, 2019 4:50:29 PM com.event.management.frontend.FrontEndImpl ReplicaOneReply
INFO: RM 1 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:50:29 PM com.event.management.frontend.FrontEndImpl ReplicaThreeReply
INFO: RM 3 : No Data Found or Might be data issue. Please try again
Aug 07, 2019 4:50:29 PM com.event.management.frontend.FrontEndImpl ReplicaTwoReply
INFO: RM 2 : No Data Found or Might be data issue. Please try again
```

Module Responsibility:

1. **Himen Sidhpura (40091993):** 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.
2. **Jenny Mistry (40092281):** Design and implement the replica manager (RM) which creates and initializes the actively replicated server system. The RM also implements the failure detection and recovery for the required type of failure.
3. **Ayush Dave (40080515):** Design and implement a failure-free sequencer which receives a client request from a FE, assigns a unique sequence number to the request and reliably multicast the request with the sequence number and FE information to all the three server replicas.

Inspite of the above mentioned responsibilities, each team member has worked on overall aspects of the project to make it work perfectly fine.

References:

- https://www.geeksforgeeks.org/multithreading-in-java/
- https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html
- https://www.geeksforgeeks.org/client-server-software-development-introduction-to-common-object-request-broker-architecture-corba/
- https://www.math.uni-hamburg.de/doc/java/tutorial/idl/hello/idltojava.html
- https://xennis.org/wiki/CORBA_-_Advanced_example_with_server-client_in_Java_and_C%2B%2B
- https://docs.oracle.com/javase/7/docs/technotes/guides/idl/POA.html
- https://www.geeksforgeeks.org/synchronized-in-java/
- https://www.baeldung.com/java-broadcast-multicast
- https://www.geeksforgeeks.org/reentrant-lock-java/
- Distributed Systems- Concepts and Design, 4th Edition, by George Coulouris, Jean Dollimore, Tim Kindberg.