**Prepared By:**

Distributed Event Management System

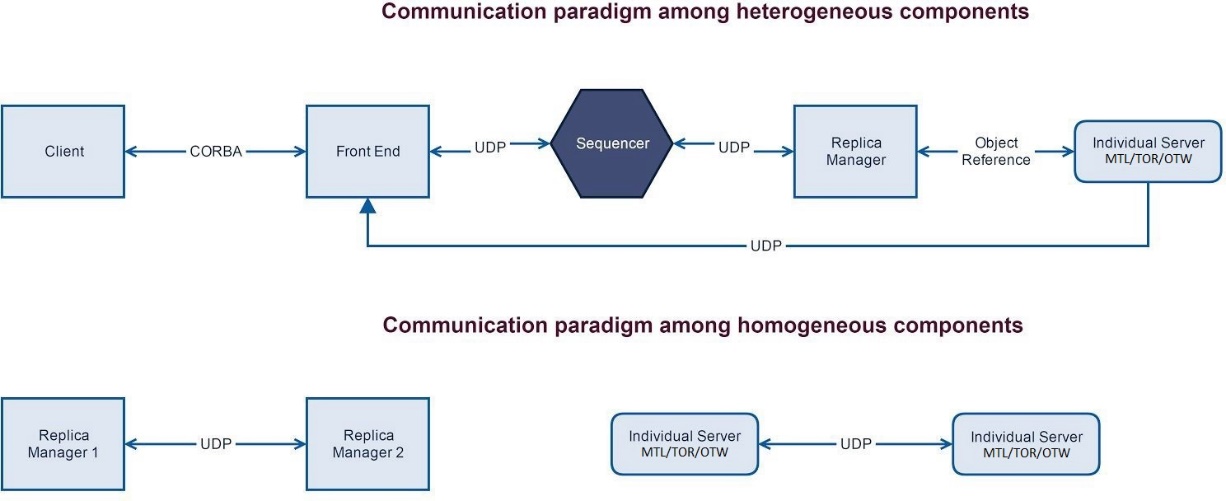
**Tanishq Bhatia (40075757) Xiyun Zhang (40059295) Bin Xue (40059809)**

# Objective:

Design and develop a distributed system which provides an infrastructure for Distributed Event Management System shared across three data centers. Expose CORBA APIs to the user which abstract the underlying distributed nature of the platform and provide network transparency. Implement multi-threading with proper synchronization to handle concurrent requests safely. Implement multiple replicas for each server to detect and recover from failures.

# Overall Design Architecture:

## Communication among components:



**Overall Workflow:**

* Front End will register CORBA remote reference with Naming Service
* Client will send request to Front End (FE) using CORBA remote invocation
* With the help of Sequencer FE will generate unique request id and multicast it to all three replicas
* Sequencer is responsible to maintain Request Ordering
* Replicas will buffer each request and make sure to execute in same order received
* Server implementations will process the request and send the result back to FE
* Replica will also buffer some last processed results along with Request Id to avoid any duplicate request processing
* On receiving response from all replicas, FE will do following:
  + In case of any inconsistency in result, FE will report Error to respective replica manager with request ID.
  + If FE doesn’t receive response from replica for a reasonable time then it reports crash to Replica Manager
  + FE will respond back client with most appropriate answer
* Replica Manager will declare a replica as corrupted and initializes new replica instance
  + if it receives consecutive three Error reports from FE or unresponsive for the same replica.

## Replica failure recovery:

1. Replica Manager(RM) will keep all new requests in the buffer for that replica and stop forwarding to the replica.
2. RM will parse through the journal file to generate the hashmap
3. Ask replica to complete all pending requests buffered at RM
4. On catching up all pending requests, new replica will start operating smoothly

**RM Description:**

* **CenterServer class**: This class is a thread which has the access to servers of three branches. It holds a task queue and reply queue assigned from the replica manager, and repeatedly takes a task from the task queue and picks the right server to process the task. And then it puts the response into the reply queue.
* **ReplicaManager**: This class holds every requests’ data from sequencer and frontend. And it holds the control over CenterServer thread. It has 4 queues for different purposes: deliver queue (priority queue) is used to temporarily store and sort the received requests by sequence number. After a request is inserted into the deliver queue, it checks if the sequence number inside the deliver queue is continuous and correct. If so it transfers a proper number of requests into the task queue (assigned to center server on initializing) and backup queue. If the center server is crashed, replica manager then rerun the center server thread and assigns a new task queue which is a deep copy of back up queue and continues listening to upcoming requests.

This class also holds a thread to repeatedly take a response out of reply queue and send the response to the frontend.

If replica manager is noticed a software failure from the front end. It checks if the replica number inside that notice is referring to itself. If so the replica manager increments the software failure counter. When the counter >= 3, it lets the center server to fix the bug.

* **RMID**: this is a enum class which holds all the address information and identification of each replica manager.

UDP Server Design:

## Concurrency:

* Location Server creates new thread to communicate to each UDP server
* UDP Server creates a new worker thread for each coming request and delegates the service request.

## Synchronization:

* Each request to access Data set is synchronized. Hence only one thread can get the hold on it which makes it thread safe.
* Additionally, each thread is a new object hence possesses its own memory space hence no shared resources among other threads.

# Logging for troubleshooting :

We have used the inbuilt logger provided by Java (java.util.logging) to perform logging in both server and client side.

## Log Format:

Generally each log entry should have the following data

* Timestamp
* Class and function name which performs the action

## Center Server:

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

* LOGS/MTL/logger.log
* LOGS/TOR/logger.log
* LOGS/OTW/logger.log

## Client:

Every Manager is a client. A log file will be created for each manager and actions performed by him/her will be logged.

* Exceptions

## Implementation:

* Create a Unique logger for each Server. (java.util.logging)
* Add a file handler to save the contents to the respective log file.
* Log using various levels like (INFO, WARNING, ERROR (java.util.logging.Level)) based on the severity.

# Challenges:

* Fine grained locking on shared objects
  + HashMap: Central data set
* Concurrency in UDP server

# Reliability:

We have implemented ACK mechanism for reliability measures.

# Test Scenarios:

# 1.No failure test case

**Scenario1**: One MTL event manager add a MTL event .

**Expected**: The action should be successful.

**Test result:**

Aug 06, 2019 8:17:04 PM Common.Log info

INFO:

MethodType: addEvent

Parameters:

EventID: MTLA080819

EventType: Conferences

bookingCapacity: 2

Status: Succeed, the event is added

**Scenario2**: One MTL event manager adds an TOR event

**Expected**: It can’t do that, MTL server only can add MTL event

**Test results:**

Aug 06, 2019 8:17:04 PM Common.Log info

INFO:

MethodType: addEvent

Parameters:

EventID: TORE110819

EventType: Seminars

bookingCapacity: 1

Status: Access denied

**Scenario3**: One OTW event manager removes an event

**Expected**: The action will remove event in the system and remove correspond customer’s event

**Test result**:

July 05, 2019 22:44:77 PM common.log info

INFO:

MethodType: removeEvent

Parameters:

EventID: OTWE100519

EventType: Seminars

Status: Success

**Scenario4**: One OTW event manager lists event availability

**Expected**: show all the event among the servers and their left seat.

**Test result:**

Aug 06, 2019 8:17:05 PM Common.Log info

INFO:

MethodType: listEventAvailability

Parameters:

EventType: Conferences

Status: MTLA080819, 2

**Scenario****5**: One OTW customer books an event

**Expected**: The action should be successful.

**Test result:**

Aug 06, 2019 8:17:05 PM Common.Log info

INFO:

MethodType: bookEvent

Parameters:

CustomerID: OTWC1234

EventID: MTLA080819

EventType: Conferences

Status: Succeed, client OTWC1234 is registered for the event

**Scenario6**: One TOR customer books an event he has already register

**Expected**: He can’t do that and system will mind him

**Test result:**

Aug 06, 2019 8:17:05 PM common.log info

INFO:

MethodType: bookEvent

Parameters:

CustomerID: TORC1111

EventID: TORA100519

EventType: Conferences

Status: The course is already registered

**Scenario7**: One TOR customer who has already booked 3 other city events in a month

Expected: He can’t do that and the system will mind him

**Test result:**

Aug 06, 2019 8:17:05 PM common.log info

INFO:

MethodType: bookEvent

Parameters:

CustomerID: TORC1111

EventID: OTWE100519

EventType: Seminars

Status: you cant registered 3 other city events in a month

**Scenario8**: One TOR customer lists his all booked event schedule

**Expected**: This action will return schedule of his events which he has already booked

**Test result**:

Aug 06, 2019 8:17:05 PM common.log info

INFO:

MethodType: getBookingSchedule

Parameters:

CustomerID: TORC1111

Status: Seminars : MTLM240519

Conferences : TORA100519,OTWE120519

TradeShows : MTLA110519

**Scenario9**: One TOR customer cancel the OTW event he has already registered in

**Expected**: This action will success

**Test result**:

Aug 06, 2019 8:17:05 PM common.log info

INFO:

MethodType: cancelEvent

Parameters:

CustomerID: TORC1111

EventID: OTWE100519

EventType: Conferences

Status: Success

**Scenario10**: One TOR customer swaps TOR event for OTW event in May, but he has already booked in 3 other city events in May.

**Expected**: He can’t do that and system will mind him

**Test result**:

Aug 06, 2019 8:17:05 PM common.Log info

INFO:

MethodType: swapEvent

Parameters:

CustomerID: TORC1111

newEventID: OTWE110519

newEventType: Conferences

oldEventID: TORA110519

oldEventType: Conferences

Status: If you swap you will register more than 3 other city event

**Scenario11**: One TOR customer swaps OTW event for OTW event in May, but he has already booked in 3 other city events in May.

**Expected**: This action will success. Since the total number for other city events in a month is still 3

**Test result**:

July 07, 2019 5:55:15 PM common.Log info

INFO:

MethodType: swapEvent

Parameters:

CustomerID: TORC1111

newEventID: OTWE110519

newEventType: Conferences

oldEventID: MTLA110519

oldEventType: TradeShows

Status: Success

# 2. Software failure test case:

\*\*\*\*\*Receiving java.net.DatagramPacket@5fea36ab from port: -1 null

receive Response:System bug!!!!;SeqNum:0;SourceIp:132.205.46.184;EventType:Conferences;Capacity:2;Request:AddEvent;Approved:False;ClientId:MTLM3456;EventId:MTLA080819;SourcePort:8001;

\*\*\*\*\*Receiving java.net.DatagramPacket@3fc5c9e4 from port: -1 null

receive Response:System bug!!!!;SeqNum:1;SourceIp:132.205.46.184;EventType:Seminars;Capacity:1;Request:AddEvent;Approved:False;ClientId:MTLM3456;EventId:MTLM110819;SourcePort:8001;

\*\*\*\*\*Receiving java.net.DatagramPacket@3c62e12e from port: -1 null

receive Response:System bug!!!!;SeqNum:2;SourceIp:132.205.46.184;EventType:Seminars;Capacity:1;Request:AddEvent;Approved:False;ClientId:MTLM3456;EventId:MTLM130819;SourcePort:8001;

\*\*\*\*\*Receiving java.net.DatagramPacket@44154432 from port: -1 null

receive Response:Succeed, the event is added;SeqNum:3;SourceIp:132.205.46.184;EventType:TradeShows;Capacity:1;Request:AddEvent;Approved:True;ClientId:OTWM9000;EventId:OTWE080819;SourcePort:8001;

In our test case, we set one server when it receives the add MTL event request, it always send false response. So you can see in the first 3 responses, it shows the “System bug!!!!”. But when we add an MTL event in 4th time, the response is Succeed which means the system recover automatically.

**3. Server crash failure test case:**

**One server crush, the result size = 2:**

\*\*\*\*\*Receiving java.net.DatagramPacket@677b6d0f from port: -1 null

receive SeqNum:20;SourceIp:132.205.46.184;Request:GetBookingSchedule;Approved:True;ClientId:OTWC9000;SourcePort:8001;

ResultSet size: 2

**The server recover, the resultsize = 3:**

;SeqNum:21;SourceIp:132.205.46.184;EventType:Conferences;Request:GetEventList;Approved:True;ClientId:OTWM9000;SourcePort:8001;

6097

ResultSet size: 3

132.205.46.183

result: MTLA080819, 1

You can see when we execute the request which the sequence number is 20, one sever crashed. So we just receive 2 responses. When frontend detect this, it judges which server crashed and send it to all replica managers. Then we execute next request, we receive 3 responses, which means the crashed server recovers itself.

*The above scenarios should also work when making multiple requests asynchronously using threads.*

# Learnings:

* Java
* Concurrency with multithreaded system
* Data modelling
* Fine grained locking and synchronization

# References:

* Java Serialization:

<https://stackoverflow.com/questions/2836646/java-serializable-object-to-byte-array> <http://www.javapractices.com/topic/TopicAction.do?Id=57>

* Java coding standards:<https://google.github.io/styleguide/javaguide.html>
* Java Set implementation:<http://tutorials.jenkov.com/java-collections/set.html>
* Get all values of enum:<https://docs.oracle.com/javase/8/docs/api/java/lang/Enum.html>
* Get current directory of Java application: [https://stackoverflow.com/questions/4871051/getting-the-current-working-directory-in-jav a](https://stackoverflow.com/questions/4871051/getting-the-current-working-directory-in-java)
* Java int datatype range:<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>
* Private port range:<https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers>