Distributed Event Management System

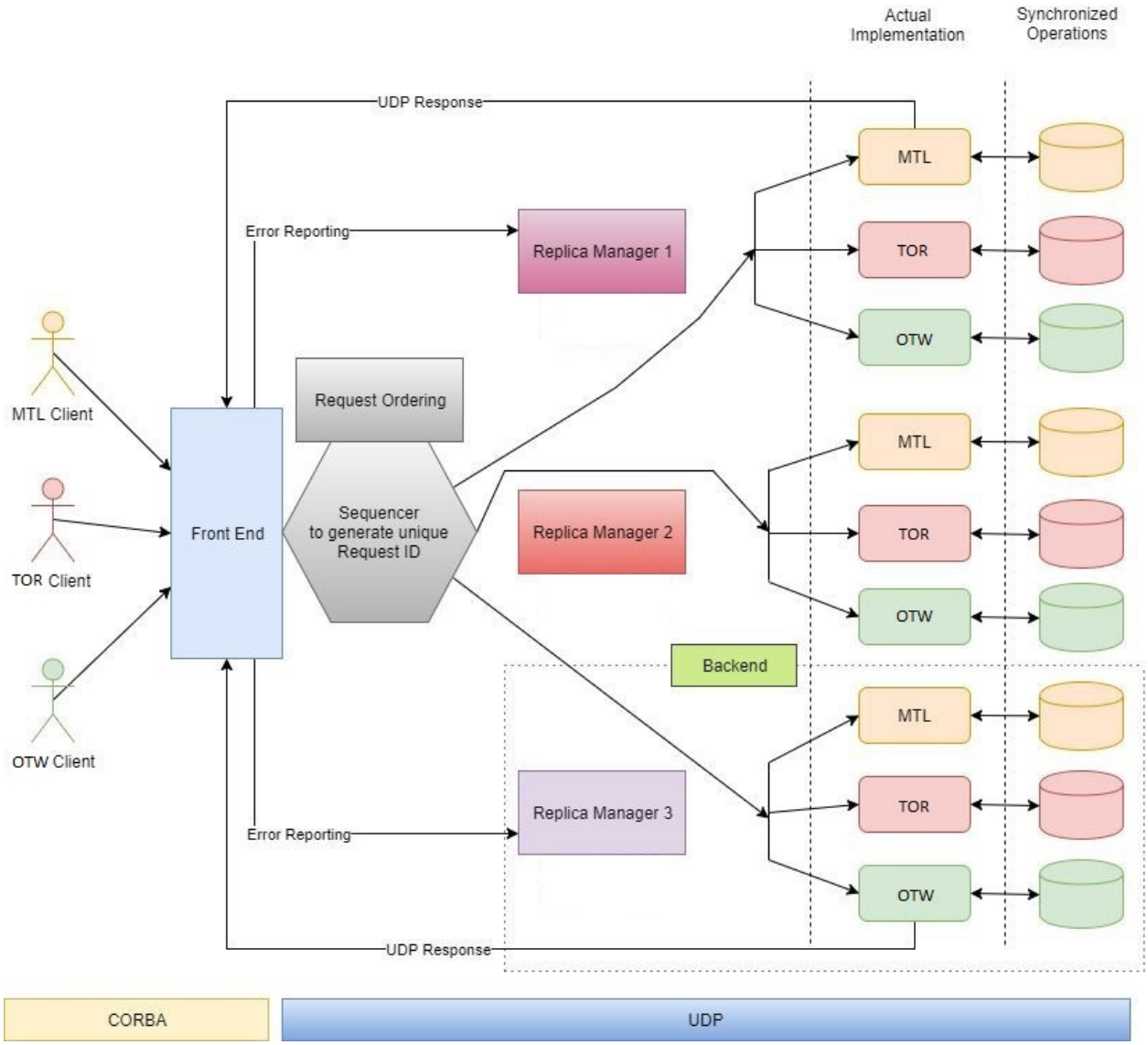
Prepared By:

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

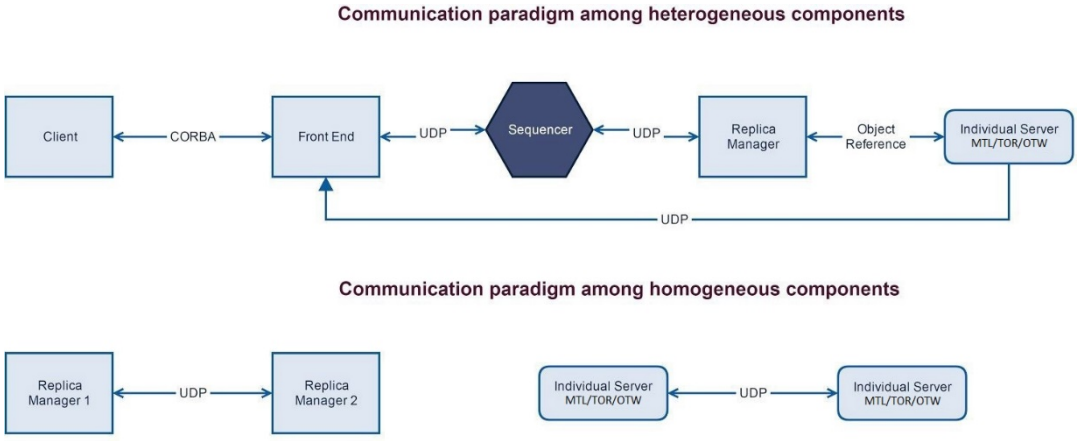
Objective:

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. Add Events with different inputs
2. Book Events with different inputs

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-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