Software Failure Tolerant and/or Highly Available Distributed Appointment Management System (DAMS)

Shreyas Kishorkumar Patel 40192955
Rahul Ashwinkumar Patel 40202836
Yash Chiragkumar Patel 40202674
Anurag Shekhar 40219195

07th April 2022

—

Distributed System Design

COMP 6231 – Winter 2022

Concordia University

Department of Computer Science and Software Engineering

—

Instructor: R. Jayakumar

Table of Contents

Overview

Distributed Appointment Management System (DAMS)

DAMS is a distributed system used by hospitals to facilitate booking, modifying and maintenance of appointments. Admins can manage the information about appointments and patients can book or cancel an appointment across different hospitals.

Our architecture consists of Front-end, failure free sequencer, Replica Manager and Replicas which contains all the functionalities described below.

DAMS consists of three hospitals (servers) in different cities:

- Montreal (MTL)
- Quebec (QUE)
- Sherbrooke (SHE)

The system is being used by two types of users:

- Admin - Identified by 'A' at 4^{th} position of user ID.
- Patient - Identified by 'P' at 4^{th} position of user ID.

DAMS identifies the server to which the user belongs and performs the operation on/through that server.

Appointment Types:

- Surgeon
- Dentist
- Physician

Appointment ID consists of server ID(MTL/QUE/SHE) + Slot(M/A/E) + Date (DDMMYY).

Servers and Users have their own logs for the operation being performed on/by them.

Patient Functions:

- bookAppointment: Patients can book appointments on their same server city as we as in other cities, given the condition that they should not be able to book more tha 3 appointments in other cities in a week.
- getAppointmentSchedule: Patient can get the details of all the appointments from the hospitals booked under his ID.
- cancelAppointment: Patients can cancel the appointment booked under their id.
- swapAppointment: Patients can swap one booked appointment with another appointment.

Admin Functions:

- Admins can perform all the operations listed under “Patient Functions”.
- addAppointment: Admin can add new appointments in their hospitals.
- removeAppointment: Admin can remove an appointment. If the appointment has been booked by patients, alternate bookings need to be made after removal of appointment.
- listAppointmentAvailability: Admin can view all the appointments and their available capacity for a specified appointment type. The result must have details from all the servers.

Design Architecture:

The Distributed System of DAMS will have 4 replica managers (RM). Each RM will have different server implementations of four servers including their own implementation of data structures. The individual server implementations may undergo some changes so that the return types of each operation become comparable. Web Services implementations of the servers will be used as part of the design of the fully fledged distributed system.

Our highly available and fault tolerant distributed design will lead to implementation of the following features in DAMS.

1. Fault Tolerance

   Fault tolerance relates to the ability of a system to continue operating without intervention when one or more of its components fail.

   The goal of designing a fault-tolerant system is to avoid interruption caused by a failure while also providing the high availability and business continuity of mission-critical systems.

2. Byzantine Failure (non - malicious) / Software Failure

   To address a software failure, the four replicas perform client requests in total order and send the results back to the FE, which then returns a single correct response to the client once the replicas have returned three identical (correct) results. If any of the replicas provide consecutively three incorrect results, the FE notifies the respective RM about that replica, then the RM substitutes that replica with another correct one.

3. Total Order

   A sequence number is generated for each request to maintain total ordering for every replica. We use hashmap to store coming requests in Replica Manager. Using sequence number, each coming request is forwarded to replica in total order.

4. Data Consistency

   In order to keep the data uniform between every replica:

   - Total Ordering is used to maintain the execution sequence in each and every replica
   - Each request must be executed by all or none of the replicas.

5. Timeout / Crash Failure

   Each FE process starts with a constant timeout (e.g. 4 seconds). The timeout will then be set to the longest response time multiplied by two once the initial request is completed.

   If FE does not receive the response within a reasonable amount of time (the longest response time multiplied by two) then It determines that replica might have crashed and notifies the respective RM about the possibility of a crash. If the RM concurs that the replica that did not generate the response has crashed, then RM replaces that with another working replica.

6. UDP Communication (Unicast and MultiCast)

   Unicast :

   The connection between Front-End and Sequencer as well with the Replica Managers uses Unicast protocol to maintain point-to-point communication

   Multicast :

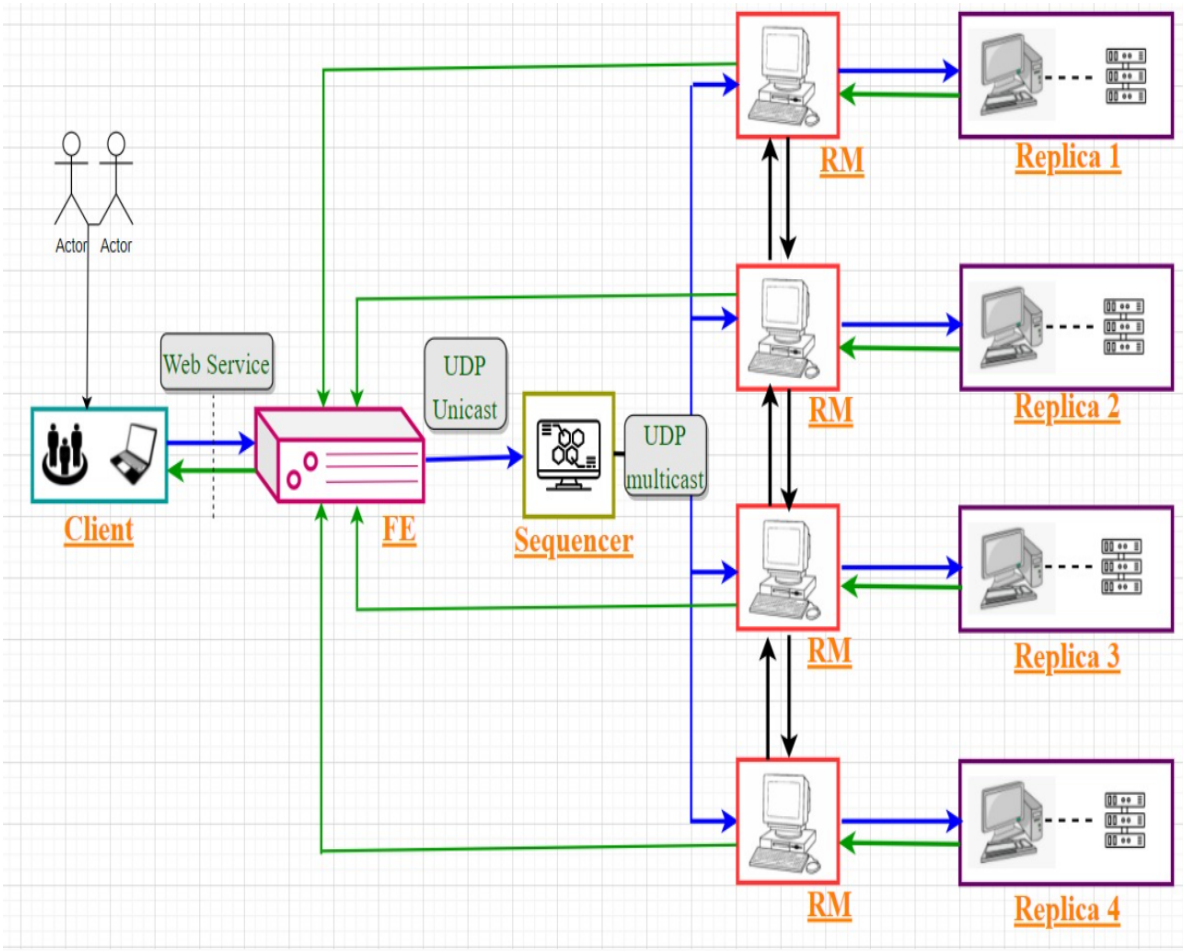
   In this system,the connection between the Replica Manager and Sequencer uses the concept ofMulticasting.

7. Replica Crash and Recovery

   In order for the system to be highly-available, it must be ensured that a server crash is tolerated and the replica must be recovered as soon as possible.

   The Front-End will check for the responses from the RM. If a particular RM does not send the response to the FE for 3 consecutive times. It will detect a replica crash and start the Replica Recovery process for the crashed replica.

   In recovery mode, the RM will execute all the previous as well as backlog requests after the point of crash which will be subsequently executed one by one. After that, the replica data will be consistent with other replicas and replica recovery is successfully executed.



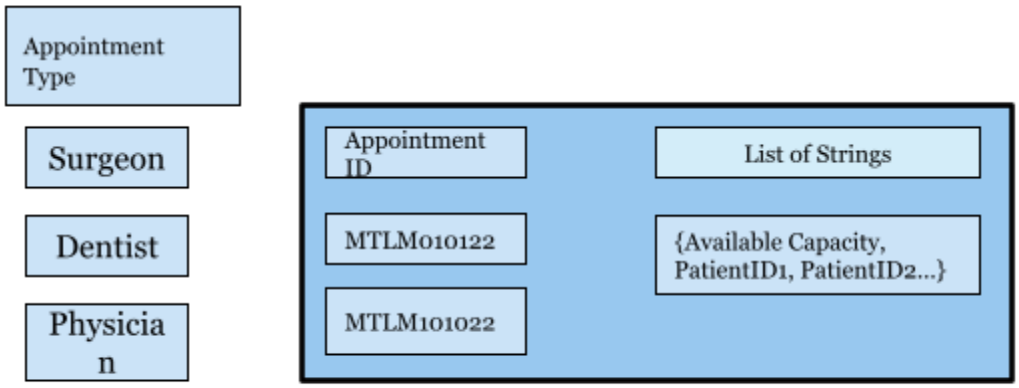
Data Flow:

- The front-end (FE) will receive the requests from clients as a web service invocation. The FE will forward the request to a failure-free sequencer.
- The failure-free Sequencer will assign a sequencer number to each client request. It will use reliable multicast to send the requests to all the RMs.
- The Replica Managers (RMs) will be responsible for maintaining the replicated distributed server system. It will boot up the different replicas and implement the crash detection, failure detection and the recovery of the crashed replica. The RMs will pick up the requests from the sequencer. The hashmap is used to store coming requests in Replica Manager. Using sequence number, each coming request is forwarded to the replica in total order.
- Each replica will process the request in their own sub-system and return the output to their RM.
- The RMs will unicast the results to the Front-End.

- The FE will compare the results and display the correct result to clients. It also informs the RM of the replica generating the incorrect output.

- Concurrent hashmap Architecture:

```
map = new ConcurrentHashMap<String,HashMap<String,List<String>>>();
```



Implementation:

Client:

- ➢ Implementation of Client Program is to validate all the inputs and process the Admin and Patient requests.
- ➢ To access the Server service, we are using Web Service.
- ➢ Client is used to access the End-Point which is subsequently published by Front-End.

Front End:

- ➢ The placement of Front-End between the Client and Sequencer is useful to send and receive messages which can be further used by the RM's.
- ➢ The FE uses Web Service Interface to communicate with the client.
- ➢ For the system to be Highly Available and Fault Tolerant, the FE detects errors and bugs from the responses received by the different Replica Manager.
- ➢ The FE receives the request from Client, which converts it into uniform format which can be understood by all the replicas.
- ➢ Packet (SequenceId; Front-EndIpAddress; Port-Number; Function-Name; userID; appointmentID; appointmentType; oldAppointmentID; oldAppointmentType; bookingCapacity)
- ➢ Also, it uses Boyer–Moore majority vote algorithm to find the majority of the correct responses from the Replicas.
- ➢ A Dynamic Timeout is used to check if a particular replica didn't send a response to the Front-End which is useful to detect the possibility of Crash Failure of a Replica.
- ➢ Compare the response received from all the replicas to check the correct output which can be shown to the client which indicates high availability in case any replica fails or sends an incorrect response.
- ➢ In case of a packet loss due to unreliability of UDP , FE implements exceptions to retry again to send the packet after the timeout.
- ➢ Front-End uses UDP Unicast to send messages to Sequencer instead of Multicast to avoid unnecessary packets in the network.
- ➢ Due to standardization of packet requests it is easier to compare the correct output and send it to the client.

Sequencer:

- The sequencer in DAMS is responsible for assigning a continuously incrementing sequence number to each request coming from the front end .
- The sequencer receives the client requests through the front end as a UDP message and forwards it to all RMs.
- The sequencer sends the requests as a message using UDP unicast to each RM simultaneously.
- Simultaneous UDP unicast has been used instead of UDP multicast due to the restriction of multicast that the IP address should be in range 200.xxx.xx.xx . But the LAN only allows the addresses with 192.xxx.xx.xx as it has the subnet mask of 192.168.xx.xx, as set by ISP.
- The sequencer ensures the total order execution by assigning the sequence number to the incoming requests.
- The algorithm used in implementation of sequencer is Kaashoek's algorithm, which also stores the incoming request messages as a history buffer.

Replica Manager

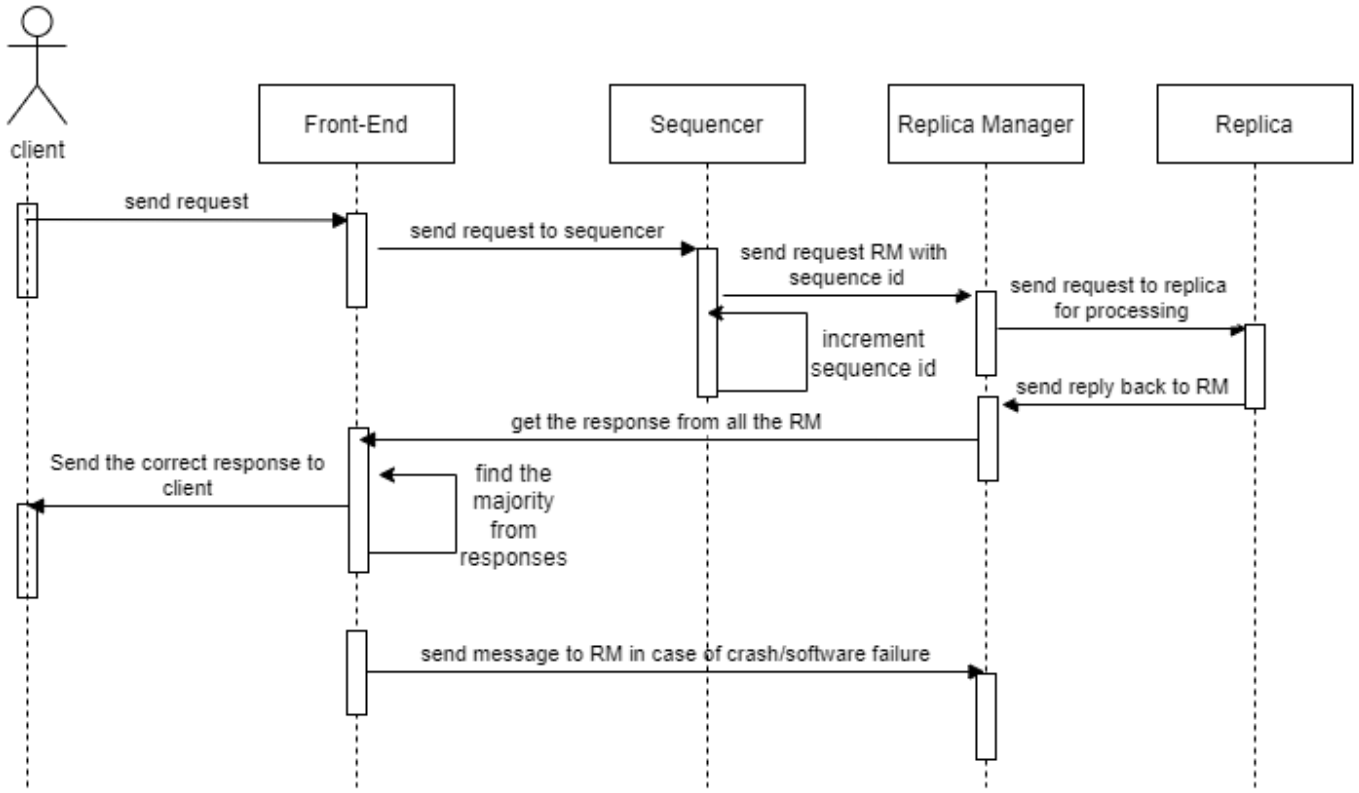
- The distributed system DAMS consists of 4 replicas which are capable of handling both crash failure and software failure simultaneously, making the whole system highly available and fault tolerant.
- The RMs receive the requests from sequencer and facilitates the processing of the requests by sending it to replicas.
- The RM receives the requests and stores it in a HashMap, using sequence number as key and actual request message as value.
- The requests are processed based on the sequence number in increasing order, which ensures the total order execution.
- The RMs are also responsible for sending the outputs of processed requests to Front End along with their sequence number as UDP unicast messages.
- In addition to above functionalities, the RMs are also responsible for booting up the replicas and to shut down a failed replica.
- The RMs perform recovery of crash failure. Upon receiving the crash failure notification from Front end, the RM sends a ping to check liveness of the replica. If it is unable to send a ping to the replica or it does not receive any reply, it confirms that the replica has failed or is inactive. It then performs recovery by rebooting the replica server.

- The RMs are also responsible for recovery from software failure. The front end notifies the RM about the failed replica. The RMs then replace the replica producing incorrect results with a new replica.
- In both kinds of failures, during recovery it ensures that the replicas process all the backlog requests making the replica data consistent across all replicas in the distributed system.

Why was Web-Service used?

- Due to the components Reusability it is easier for developers to re-implement different functionalities.
- Web-Service uses HTTP Protocol which is more user friendly and accessible than CORBA's which uses GIOP and IIOP.
- As functionality of Web-Service is done using Message-Passing , Application Independence is improved due to it as two applications implemented in different languages can easily communicate over the web.
- Configuration of CORBA requires more expertise which is a major setback for clients as well as complex firewall rules have to be used for CORBA to be used on a network(IOR).
- Web-Services on the other hand don't require such complex rules as HTTP is used.

Sequence Diagram



Screenshots of working project

Software Failure:

Front end informs RM about software failure:

```
Receiving Request from RM
Receiving Request from RM
Message from Replica Manager: 2;Success: Appointment MTLA020622 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 2;1Success: Appointment MTLA020622 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.170
Message from Replica Manager: 2;Success: Appointment MTLA020622 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.1
Message from Replica Manager: 2;Success: Appointment MTLA020622 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.176
Software Failure Frequency: 2 in 192.168.54.170
In FrontEndImpl:
192.168.54.67;00;addAppointment;null;MTLE030622;Dental;null;null;23
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Message from Replica Manager: 3;Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 3;Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.1
Message from Replica Manager: 3;1Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.170
Message from Replica Manager: 3;Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.176
Software Failure Frequency: 3 in 192.168.54.170
sending response to Replica Manager for Software Failure: 3 to 192.168.54.170
sent response to Replica Manager for Software Failure: 3 to 192.168.54.170
```

Detection at Replica Manager:

```
**************************************************
Receiving Request from Handler
Software failure Message from FE : 0:3
Starting recovery from Software failure
sending request to Handler: 0;0;00;kill;null;null;null;null;null;0
sent request to Handler
Waiting for recovery
Socket Exception in recieveResponseHandler(): socket closed
Receiving Request from Handler
Waiting  more for recovery
Started new Replica
**************************************************
Preparing Montreal city Server
Montreal Server ready.
**************************************************
Initialized hashmap for MTL
MTL :  UDP port started at 1563
{Dentist={}, Physician={}, Surgeon={}}
Adding dummy data in hashmap for MTL
{Dentist={MTLM010522=[15]}, Physician={MTLM010522=[12]}, Surgeon={MTLM010522=[10]}}
**************************************************
Preparing Quebec city Server
Quebec city Server ready.
**************************************************
Initialized hashmap for QUE
```

Data consistency during recovery :

```
{Dentist={MTLM010522=[15]}, Physician={MTLM010522=[12], MTLM010622=[12]}, Surgeon={MTLM010522=[10]}}
**********************************************************
addAppointment - reply:
Success: Appointment MTLM010622 (Physician) is added successfully
sending response to RM: 1;Success: Appointment MTLM010622 (Physician) is added successfully;
sent response to RM
Message from Handler :
 1;Success: Appointment MTLM010622 (Physician) is added successfully;
No Response Sent
**********************************************************
Receiving Request from Handler
Add an appointment : Added appointment type  Surgeon with appointment id MTLA020622 with capacity 14
MTL
{Dentist={MTLM010522=[15]}, Physician={MTLM010522=[12], MTLM010622=[12]}, Surgeon={MTLM010522=[10], MTLA020622=[14]}}
**********************************************************
addAppointment - reply:
Success: Appointment MTLA020622 (Surgeon) is added successfully
sending response to RM: 2;Success: Appointment MTLA020622 (Surgeon) is added successfully;
sent response to RM
Message from Handler :
 2;Success: Appointment MTLA020622 (Surgeon) is added successfully;
No Response Sent
**********************************************************
Receiving Request from Handler
Add an appointment : Added appointment type  Dentist with appointment id MTLE030622 with capacity 23
MTL
{Dentist={MTLM010522=[15], MTLE030622=[23]}, Physician={MTLM010522=[12], MTLM010622=[12]}, Surgeon={MTLM010522=[10], MTLA020622=[14]}}
**********************************************************
addAppointment - reply:
Success: Appointment MTLE030622 (Dental) is added successfully
sending response to RM: 3;Success: Appointment MTLE030622 (Dental) is added successfully;
sent response to RM
**********************************************************
Receiving Request from RM
Message from Handler :
 3;Success: Appointment MTLE030622 (Dental) is added successfully;
No Response Sent
**********************************************************
Status file is reset
Receiving Request from Handler
```

```
Message from Sequencer:
 4;192.168.54.67;00;addAppointment;null;MTLM040622;Physician;null;null;23
{1=1;192.168.54.67;00;addAppointment;null;MTLM010622;Physician;null;null;12, 2=2;192.168.54.67;00;addAppointment;null;MTLA020622;Surgeon;null;null;14, 3=3;192.168.54.67;00;addAppointment;null;MTl
sending request to Handler: 4;192.168.54.67;00;addAppointment;null;MTLM040622;Physician;null;null;23
sent request to Handler
Receiving Request from Sequencer
Socket Exception in recieveResponseHandler(): socket closed
Receiving Request from Handler
Message from RM:
 4;192.168.54.67;00;addAppointment;null;MTLM040622;Physician;null;null;23
{1=1;192.168.54.67;00;addAppointment;null;MTLM010622;Physician;null;null;12, 2=2;192.168.54.67;00;addAppointment;null;MTLA020622;Surgeon;null;null;14, 3=3;192.168.54.67;00;addAppointment;null;MTl
Inside processRequestFromUser()
Add an appointment : Added appointment type  Physician with appointment id MTLM040622 with capacity 23
MTL
{Dentist={MTLM010522=[15], MTLE030622=[23]}, Physician={MTLM040622=[23], MTLM010522=[12], MTLM010622=[12]}, Surgeon={MTLM010522=[10], MTLA020622=[14]}}
**********************************************************
addAppointment - reply:
Success: Appointment MTLM040622 (Physician) is added successfully
sending response to RM: 4;Success: Appointment MTLM040622 (Physician) is added successfully;
sent response to RM
**********************************************************
Receiving Request from RM
Message from Handler :
 4;Success: Appointment MTLM040622 (Physician) is added successfully;
sending response to FrontEnd: 4;Success: Appointment MTLM040622 (Physician) is added successfully;
sent response to FrontEnd
**********************************************************
Receiving Request from Handler
```

Responses at Front-End After Software failure recovery:

```
Receiving Request from RM
Message from Replica Manager: 3;Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 3;Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.1
Message from Replica Manager: 3;1Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.170
Message from Replica Manager: 3;Success: Appointment MTLE030622 (Dental) is added successfully;
Replica Manager IP Address:/192.168.54.176
Software Failure Frequency: 3 in 192.168.54.170
sending response to Replica Manager for Software Failure: 3 to 192.168.54.170
sent response to Replica Manager for Software Failure: 3 to 192.168.54.170
In FrontEndImpl:
192.168.54.67;00;addAppointment;null;MTLM040622;Physician;null;null;23
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Message from Replica Manager: 4;Success: Appointment MTLM040622 (Physician) is added successfully;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 4;Success: Appointment MTLM040622 (Physician) is added successfully;
Replica Manager IP Address:/192.168.54.170
Message from Replica Manager: 4;Success: Appointment MTLM040622 (Physician) is added successfully;
Replica Manager IP Address:/192.168.54.1
Message from Replica Manager: 4;Success: Appointment MTLM040622 (Physician) is added successfully;
Replica Manager IP Address:/192.168.54.176
```

Crash Failure:

Front end informs RM about Crash failure:

```
Receiving Request from RM
Message from Replica Manager: 2;Success: you successfully booked the appointment for (Physician) with id MTLM010722;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 2;Success: you successfully booked the appointment for (Physician) with id MTLM010722;
Message from Replica Manager: 2;Success: you successfully booked the appointment for (Physician) with id MTLM010722;
Replica Manager IP Address:/192.168.54.1
Replica Manager IP Address:/192.168.54.176
Closing listener because of timeout.
Message from Replica Manager:
Replica Manager IP Address:null
Socket Exception in receiveResponseFromReplicaManager(): For input string: ""
Crash Failure Frequency: 2 in 192.168.54.170
In FrontEndImpl:
192.168.54.67;00;addAppointment;null;MTLM020722;Surgeon;null;null;10
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Message from Replica Manager: 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.176
Message from Replica Manager: 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.1
Closing listener because of timeout.
Message from Replica Manager:
Replica Manager IP Address:null
Socket Exception in receiveResponseFromReplicaManager(): For input string: ""
Crash Failure Frequency: 3 in 192.168.54.170
sending response to Replica Manager for Crash Failure: 3
sent response to Replica Manager for Crash Failure
```

Detection at Replica Manager:

```
Hashmap for userRequestList : {}
Receiving Request from Sequencer
Receiving Request from FrontEnd for Software Failure
Receiving Request from FrontEnd for Crash Failure
Receiving Request from Handler
Message from Sequencer:
 1;192.168.54.67;00;addAppointment;null;MTLM010722;Physician;null;null;12
{1=1;192.168.54.67;00;addAppointment;null;MTLM010722;Physician;null;null;12}
sending request to Handler: 1;192.168.54.67;00;addAppointment;null;MTLM010722;Physician;null;null;12
sent request to Handler
Receiving Request from Sequencer
Socket Exception in recieveResponseHandler(): socket closed
Receiving Request from Handler
Message from Sequencer:
 2;192.168.54.67;00;bookAppointment;MTLP1235;MTLM010722;Physician;null;null;0
{1=1;192.168.54.67;00;addAppointment;null;MTLM010722;Physician;null;null;12, 2=2;192.168.54.67;00;bookAppointment;MTLP1235;MTLM010722;Physician;null;null;0}
sending request to Handler: 2;192.168.54.67;00;bookAppointment;MTLP1235;MTLM010722;Physician;null;null;0
sent request to Handler
Receiving Request from Sequencer
Socket Exception in recieveResponseHandler(): socket closed
Receiving Request from Handler
Message from Sequencer:
 3;192.168.54.67;00;addAppointment;null;MTLM020722;Surgeon;null;null;10
{1=1;192.168.54.67;00;addAppointment;null;MTLM010722;Physician;null;null;12, 2=2;192.168.54.67;00;bookAppointment;MTLP1235;MTLM010722;Physician;null;null;0, 3=3;192.168.54.67;00;addAppointment;null;MTLM0
sending request to Handler: 3;192.168.54.67;00;addAppointment;null;MTLM020722;Surgeon;null;null;10
sent request to Handler
Socket Exception in recieveResponseHandler(): socket closed
Receiving Request from Sequencer
Receiving Request from Handler
Crash Message from FE : 3:0
sending request to Handler: 0;0;00;checkAlive;null;null;null;null;null;0
sent request to Handler
Socket Exception in recieveResponseHandler(): socket closed
Receiving Request from Handler
Starting recovery of crash failure
Started new Replica
*******************************************************
Preparing Montreal city Server
Montreal Server ready.
*******************************************************
Initialized hashmap for MTL
MTL :  UDP port started at 1563
```

Data consistency during recovery :

```
addAppointment - reply:
Success: Appointment MTLM010722 (Physician) is added successfully
sending response to RM: 1;Success: Appointment MTLM010722 (Physician) is added successfully;
sent response to RM
Message from Handler :
 1;Success: Appointment MTLM010722 (Physician) is added successfully;
No Response Sent
*******************************************************
isCrashed : true : isSWFailed false : lastSuccessfulResponse_CrashFailure 3 : lastSuccessfulResponse_CrashFailure 0
Book an appointment : Success!! MTLP1235 booked appointment type Physician with appointment id MTLM010722.
Receiving Request from Handler
MTL
{Dentist={MTLM010522=[15]}, Physician={MTLM010722=[11, MTLP1235], MTLM010522=[12]}, Surgeon={MTLM010522=[10]}}
*******************************************************
bookAppointment - reply:
Success: you successfully booked the appointment for (Physician) with id MTLM010722
sending response to RM: 2;Success: you successfully booked the appointment for (Physician) with id MTLM010722;
sent response to RM
Message from Handler :
 2;Success: you successfully booked the appointment for (Physician) with id MTLM010722;
No Response Sent
*******************************************************
isCrashed : true : isSWFailed false : lastSuccessfulResponse_CrashFailure 3 : lastSuccessfulResponse_CrashFailure 0
Receiving Request from Handler
Add an appointment : Added appointment type  Surgeon with appointment id MTLM020722 with capacity 10
MTL
{Dentist={MTLM010522=[15]}, Physician={MTLM010722=[11, MTLP1235], MTLM010522=[12]}, Surgeon={MTLM020722=[10], MTLM010522=[10]}}
*******************************************************
addAppointment - reply:
Success: Appointment MTLM020722 (Surgeon) is added successfully
sending response to RM: 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
sent response to RM
*******************************************************
Message from Handler :
 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
Receiving Request from RM
No Response Sent
*******************************************************
isCrashed : true : isSWFailed false : lastSuccessfulResponse_CrashFailure 3 : lastSuccessfulResponse_CrashFailure 0
Status file is reset
Receiving Request from Handler
```

```
isCrashed : true : isSWFailed false : lastSuccessfulResponse_CrashFailure 3 : lastSuccessfulResponse_CrashFailure 0
Status file is reset
Receiving Request from Handler
%essage from Sequencer:
 4;192.168.54.67;00;bookAppointment;MTLP1235;MTLM020722;Surgeon;null;null;0
{1=1;192.168.54.67;00;addAppointment;null;MTLM010722;Physician;null;null;12, 2=2;192.168.54.67;00;bookAppointment;MTLP1235;MTLM010722;Physician;null;null;0, 3=3;192.168.54.67;00;addAppointment;null;MTLM0
sending request to Handler: 4;192.168.54.67;00;bookAppointment;MTLP1235;MTLM020722;Surgeon;null;null;0
sent request to Handler
Receiving Request from Sequencer
Socket Exception in recieveResponseHandler(): socket closed
%essage from RM:
 4;192.168.54.67;00;bookAppointment;MTLP1235;MTLM020722;Surgeon;null;null;0
{1=1;192.168.54.67;00;addAppointment;null;MTLM010722;Physician;null;null;12, 2=2;192.168.54.67;00;bookAppointment;MTLP1235;MTLM010722;Physician;null;null;0, 3=3;192.168.54.67;00;addAppointment;null;MTLM0
Inside processRequestFromUser()
Receiving Request from Handler
Book an appointment : Success!! MTLP1235 booked appointment type Surgeon with appointment id MTLM020722.
%TL
{Dentist={MTLM010522=[15]}, Physician={MTLM010722=[11, MTLP1235], MTLM010522=[12]}, Surgeon={MTLM020722=[9, MTLP1235], MTLM010522=[10]}}
**********************************************************
bookAppointment - reply:
Success: you successfully booked the appointment for (Surgeon) with id MTLM020722
sending response to RM: 4;Success: you successfully booked the appointment for (Surgeon) with id MTLM020722;
sent response to RM
**********************************************************
Receiving Request from RM
%essage from Handler :
 4;Success: you successfully booked the appointment for (Surgeon) with id MTLM020722;
sending response to FrontEnd: 4;Success: you successfully booked the appointment for (Surgeon) with id MTLM020722;
sent response to FrontEnd
**********************************************************
isCrashed : false : isSWFailed false : lastSuccessfulResponse_CrashFailure 0 : lastSuccessfulResponse_CrashFailure 0
Receiving Request from Handler
```

Responses at Front-End After Crash failure recovery:

```
In FrontEndImpl:
192.168.54.67;00;addAppointment;null;MTLM020722;Surgeon;null;null;10
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Message from Replica Manager: 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.176
Message from Replica Manager: 3;Success: Appointment MTLM020722 (Surgeon) is added successfully;
Replica Manager IP Address:/192.168.54.1
Closing listener because of timeout.
Message from Replica Manager:
Replica Manager IP Address:null
Socket Exception in receiveResponseFromReplicaManager(): For input string: ""
Crash Failure Frequency: 3 in 192.168.54.170
sending response to Replica Manager for Crash Failure: 3
sent response to Replica Manager for Crash Failure
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Receiving Request from RM
Message from Replica Manager: 4;Success: you successfully booked the appointment for (Surgeon) with id MTLM020722;
Replica Manager IP Address:/192.168.54.67
Message from Replica Manager: 4;Success: you successfully booked the appointment for (Surgeon) with id MTLM020722;
Replica Manager IP Address:/192.168.54.170
Message from Replica Manager: 4;Success: you successfully booked the appointment for (Surgeon) with id MTLM020722;
Replica Manager IP Address:/192.168.54.176
Message from Replica Manager: 4;Success: you successfully booked the appointment for (Surgeon) with id MTLM020722;
Replica Manager IP Address:/192.168.54.1
```

Test Cases:

#	Method Name	Scenario	Cases
1		Hashmap Initialization	Hashmap should be initialized with pre defined dummy data
2		Client Connection	Client program should run and ask for user ID
			Client program should be able to determine the type of client and the server to which the user belongs
			Client should get proper options based on its type
		Log check	Logs generated for server and clients connected
3	addAppointment()	Admin Client	New Appointment ID -> Added
			Duplicate Appointment ID --> Not Added
			Fails with Appointment ID of other servers
		Log check	Appropriate logs generated
4	listAppointmentAvailability( )	Admin Client	List all appointments avialable on all server
			Return appoinments with 0 capacity as well found on a server
		Logs check	Appropriate logs generated
5	removeAppoinment()	Admin Client	remove appointment with no patient booking
			remove appointment with rebboking of patient appointments
		Logs check	Appropriate logs generated
6	bookAppointment	Patient and admin client	Patient can book appointment if it exists and capacity is not full
			Appointment booking should fail if patient already has same appointment type and appointment id
			Appointment booking should fail if appointment ID doesnot exist
			Appointment booking should fail if capacity is full
			Appointment booking should fail if user tries to book a second appointment for a appointment type in same day
			Appointment booking on different server with same above conditions
			Booking fails for other servers if 3 bookings are already present on other servers in the same week
		Log check	Appropriate logs generated

7	getAppointmentSchedule	Patient and admin client	All appoinments booked in all cities under user are listed
			Returns empty string if no appointment booked by user
		Log check	Appropriate logs generated
8	cancelAppointment	Patient and admin client	Patient can cancel appointment booked under their name
			Cancel fails if appointment does not exist
			Cancel works for multiple appointment ID under different Appointment types
		Log check	Appropriate logs generated
9	swapAppointment	Patient and admin client	Swap completes if old appointment exists and new booking capacity is not full
			The swap fails if above case is satisfied but bookAppointment conditions fail
			The swap fails if old appointment does not exist
			Swap fails if new appointment does not exist or capacity is full in case it exists
		Log check	Appropriate logs generated
10	Front End	Front end Checks	Front-end captures the request from client
			FE compares all the results and sends correct response to client
			FE reports the server which sends incorrect responses to RM
11	Sequencer	Sequencer Checks	Sequencer assigns unique id to each request coming from front end
			Sequencer multicasts all the requests with their unique Id to RM
12	RM	RM Checks	RM boots up the server replicas
			RM sends the requests to replicas
			RM receives the response from replicas and send it to front end
			RM checks for server crash
			RM performs the recovery of crashed servers
			RM receives the notification from FE about software failure
			RM performs software failure recovery

Task Distribution:

Task 1 : Yash Chiragkumar Patel
Task 2 : Anurag Shekhar
Task 3 : Shreyas Kishorkumar Patel
Task 4 : Rahul Ashwinkumar Patel