CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

Project Title:

Software Failure Tolerant and Highly Available Distributed Health Care Management System (DHMS)

Session: **Winter 2024**

Date: **25th March 2024**

Course: **COMP 6231 *(Distributed System Design)***

Instructor: **R. Jayakumar**

By:

- *Naveen Rayapudi (40291526)*
- *Shanmukha Venkata Naga Sai Tummala (40289721)*
- *Alain Weng (40132842)*
- *Daniel Dan-Ebbah (40292504)*

Project Overview:

This project aims to design a Distributed Health Care Management System (DHMS) to simultaneously handle software failures and process crashes. This involves creating an actively replicated server system with four replicas, ensuring high availability and fault tolerance without selecting the type of failure at server initialization. The system should be able to detect and recover from both failure types when both failures happen simultaneously.

Clients to this application are of 2 types:

1. Admin
2. Patient

Patients can book, cancel, swap appointments, and view the booked appointments. *Admins* can additionally manage the appointments, i.e., add, remove, and view the list of available appointments. Admin can also book, cancel, swap or view his/her appointments like a patient.

Each admin/patient will only interact with the server of their city to perform any of the actions. Servers communicate with other servers and perform the required action for the user.

Multiple users should be able to perform actions concurrently on the distributed application.

The different hospitals to be handled by the application would be located in different cities:

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

Design Architecture:

Implementation Details:

The system would consist of the following components:

- 3 Clients
- 3 Front End
- 1 Sequencer
- 4 Replica Managers
- 4 Server Replicas

We are going to build the system such that should be able to fulfill the following criteria:

1. **Reliable Multicast:**
   Reliable multicast should be so that either all replicas receive the client request or none should receive it. This multicast will be implemented as multiple unicast in the software.

2. **Sequential consistency:**
   Our replicated system is not *linearizable*, because we cannot guarantee that the execution of the requests is consistent with the real-time issuing of the client's requests. However, the system is sequentially consistent because each client is *synchronized*, which means a client sends a request and waits for the response before sending another request. Hence, the execution order is consistent with the program order in which each client issues the operations.

3. **Ordering:**
   To achieve total ordering, we use the sequencer, common to the entire system, and assign a unique sequence number to requests from all the front ends. A server replica executes the requests it receives by following the ordering of sequence numbers. Because all the replicas receive the same

sequence numbers, the order in which requests are executed will be the same across all the replicas.

4. **Crash Detection & Recovery:**
   If the *Front End* does not receive the result from a replica within a reasonable time (twice the time taken for the slowest result so far), it suspects that the replica may have crashed and informs all the *Replica Managers* of the potential crash. Each RM then checks if the replica that did not produce the result is available. They do so by using a ping mechanism. If two or more (majority) replicas identify that the target replica is crashed, they remove it from the group and spin up a new replica with the updated data.

5. **Software Failure Detection & Recovery:**
   If any one of the replicas produces incorrect results, the *Front End (FE)* informs all the *Replica Managers (RM)* about that replica. Each RM keeps track of these incorrect results. If a replica produces incorrect results for three consecutive client requests, then one of the RMs will replace that replica with another correct one.

6. **Reliable UDP by Handling Issues:**
   a. <u>**Issue**</u>: The request from the FE does not reach the sequencer.
      <u>**Solution**</u>: Timeout mechanism - wait for some time for the response, after that, resend the request.

   b. <u>**Issue**</u>: A request from the sequencer does not reach some of the replicas.
      <u>**Solution**</u>: Each replica multicasts the request to all the other replicas. If a replica has already received the multicast message, then it simply ignores it. Otherwise, it will process the request and send the response to the front end.

c. **Issue**: Request has a `sequence number (n) > (1 + sequence number of the last executed request)`.
Solution: Queries other replicas to get the request with the previous sequence number and execute it first.

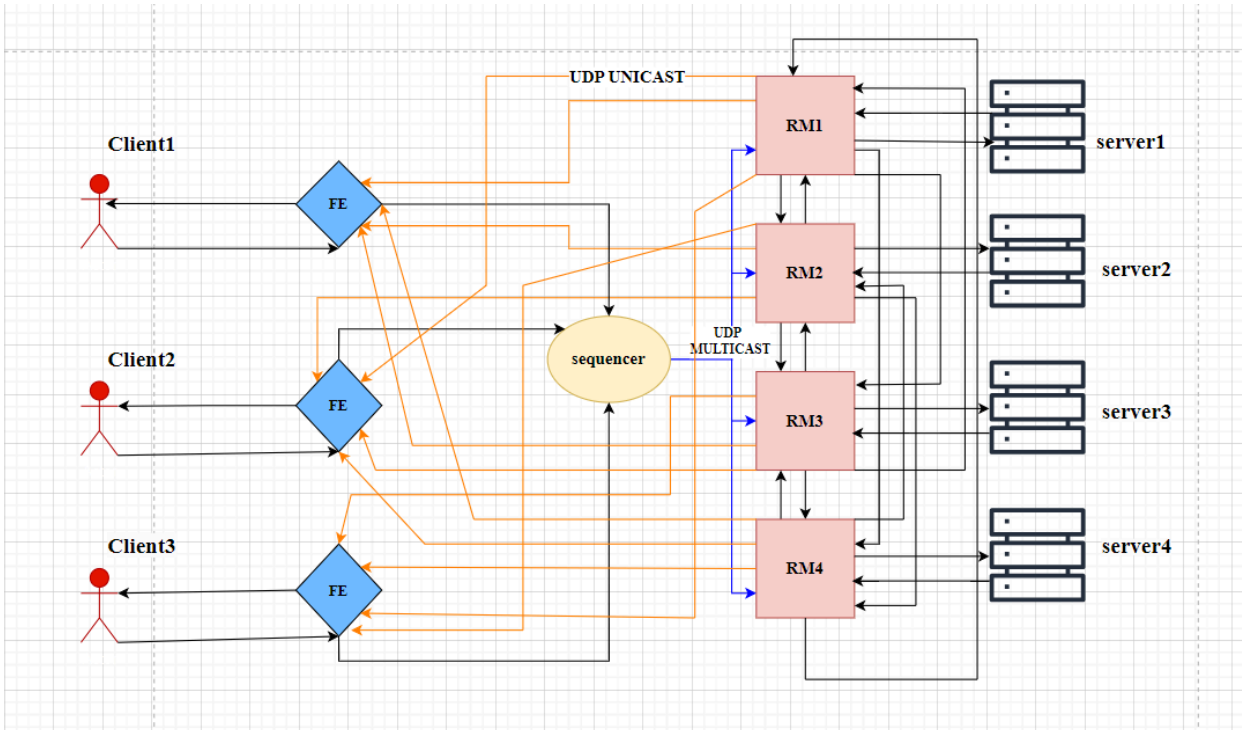
7. **Highly Available:**
One of the primary reasons for using active replication is to have a highly available system. The system should always be available even during crashes and software failures and serve the client(s) requests.

Component Design:

The system is designed to make use of active replication that includes:

- **Client:** Initiates requests to the DHMS (application).
- **Front End (FE):** This acts as a proxy between clients and the server replicas, it manages the request distribution and response aggregation.
- **Sequencer:** Assigns a unique sequence number to each client request to maintain the total order delivery across the replicas. It is implemented to be a failure-free component.
- **Replica Manager (RM):** Manages the lifecycle of the server replicas, where it monitors their health, handles failure detection and recovery
- **Server Replicas:** Executes the client request and maintains the application's state. Each server replica implements the three(3) different hospitals

Architecture Diagram:



Data Flow:

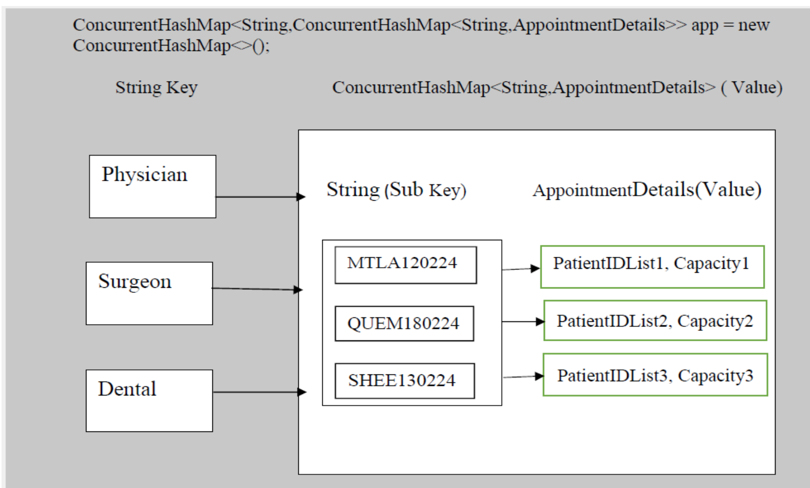
1. **Client → FE:** The client sends the request to the FE, which is one of the operations available to the respective client
2. **FE → Sequencer:** The FE forwards the request to the sequencer for sequencing
3. **Sequencer → Replicas:** The sequencer attaches a sequence number to the request and multicasts the request to all the server replicas
4. **Replicas → FE:** Each server replica executes the request and sends the result back to the FE
5. **FE → Client:** The FE aggregates the results from all the replicas and decides the final response, then sends it back to the client.

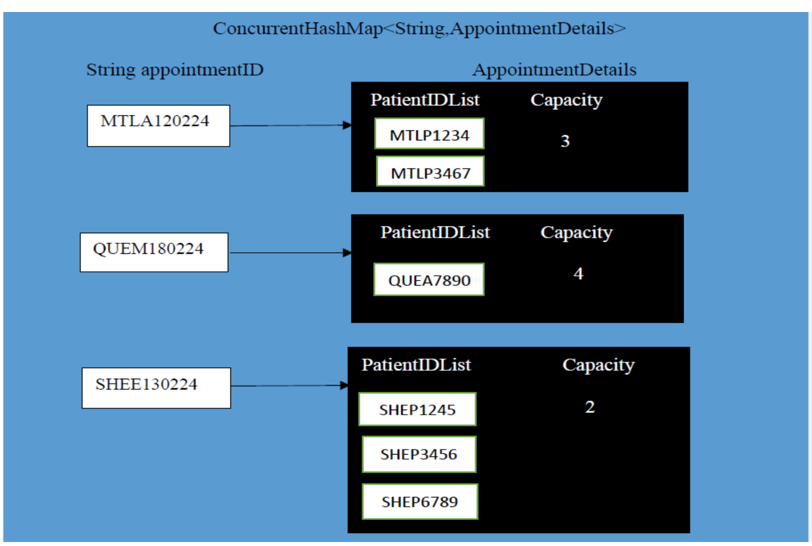
Data Structures:

The appointments data in each server program (MTLHospital, QUEHospital, SHEHospital) is stored in a ConcurrentHashMap data structure.

ConcurrentHashMap<String, ConcurrentHashMap<String, AppointmentDetails>> appointments;

AppointmentDetails is a Plain Old Java Object (POJO) that stores the list of all patient IDs and the available capacity (integer) of appointments.





We are planning to use Concurrent HashMaps, Lists and Arrays etc., to implement the RM, FE and Sequencer depending on the low-level design.

<u>Testing Scenarios:</u>

TEST	TEST CASES	EXPECTED RESULT
Login	Login with any ClientID	Success
Add Appointment	Invalid appointment ID	Fail
	Non-Admin client ID	Fail
	Valid appointment ID	Success
	Duplicate Appointment	Fail
	Appointment from other servers	Fail
Remove Appointment	Appointment ID doesn’t exist	Fail
	Appointment from other servers	Fail
	Non-Admin client ID	Fail
	Appointment has bookings with no later valid appointments	Fail
	Appointment has bookings with later valid appointments	Success
List Appointment Availability	Have appointments in all servers	Arraylist with all appointments
Book Appointment	Appointment doesn’t exist	Fail
	Already have same booking	Fail

	Already have booking at the same time for another type	Fail
	Booking an existing appointment with no conflict between booking times for other appointments	Success
	Same as above but for other hospitals	Success
Get Appointment Schedule	Client has appointments booked	Arraylist of booked appointments
	Client doesn't exist	Fail
Cancel Appointment	Appointment doesn't exist	Fail
	Appointment exists	Success
Swap Appointment	Old appointment doesn't exist/not booked	Fail
	New appointment doesn't exist	Fail
	New appointment not bookable	Fail
	New and old appointments valid	Success
	New and old appointments valid but on different servers	Success
Sequencing	Verify that the sequence number goes up	

	Verify that the server waits for the correct sequence number before running the command	
Reliable UDP multicast	Send packets don't make it to the replica manager or that don't receive a response	Packets are sent from the other replica managers and a response
	Send packets to a normal working replica manager	There is a timely response
Server failure	One or multiple servers crash but not all	Server(s) restarted and info restored
	One server returns incorrect info	Server info deleted and updated
	One server crashes and another returns incorrect info	Crashed server restarts and info restored, incorrect server has info deleted and updated

Team Task Breakdown:

Name	Student ID	Task
Shanmukha Venkata Naga Sai Tummala	40289721	Design and implement the Front End (FE)
Daniel Dan-ebbah	40292504	Design and implement the replica manager (RM)
Naveen Rayapudi	40291526	Design and implement a failure-free sequencer
Alain Weng	40132842	Design test cases for all possible failure situations and implement a client program to run them