Software Failure Tolerance Distributed Appointment Management System

For

Distributed System Designs (COMP 6231)

Winter 2022

Master of Applied Computer Science

(MACompSc)

Submitted By:

Anam Ayesha Shaikh 40205690

Kawsar Ahmed 40192329

Gagandeep Kaur 40203768

Instructor:

Rajagopalan Jayakumar

Table Of Contents

1. INTRODUCTION

Distributed Appointment Management System (DAMS) is a designed application used by patients to book appointments in Montreal, Quebec and Sherbrooke Hospitals. Another user, admin is the administrator who manages all the appointment of the patients. All three cities are the three servers, Montreal, Quebec and Sherbrooke. We have already implemented the DAMS system using RMI, CORBA and Web-Services Distributed System Implementation Techniques. As we have all distributed implementation technique of DAMS we now focus to characterize our DAMS system using very important Distributed System Characteristic known as Failure Tolerance.

Failure Tolerance is the ability of the distributed system to produce correct results even if certain number of components fail. Why do we need Failure Tolerance in Distributed System? Distributed System must maintain 100% (most of the time 99.99% practically) availability even if any hardware, software or network fails. Which means if one component fails there must be another component to provide the backup for the failed component and process the request successfully this is achieved by Replication. Replication is one of the important features of the Distributed Systems and is implemented by creating multiple replications of the servers known as replicas performing the same functionality as requested by client individually.

In this project, we are designing our DAMS application as Software Failure Tolerance using Active Total Order Replication with Reliable Multicast using Kaashoek's Protocol.

2. TECHNOLOGY USED

2.1. REPLICATION

Replication in computing involves sharing information so as to ensure consistency between redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or accessibility.

Replication can be implemented as Passive Replication and Active Replication. Where the requests is sent from the client to Front-end (FE) which acts as the middleware between client and the severs. Each servers have replicas which is managed by the Replica Manager (RM). RM is responsible for failure detection and failure recovery.

2.1.1. Passive Replication:

In Passive Replication, there is a primary replica which accepts and executes the client request through FE. This replication is passive because only the primary replica executes the client request other replicas acts as the backup replica.

After the primary replica processes the requests it sends the update to the backup replicas which in turn update themselves. This replication is implemented for crash detection and provide high availability.

Problem with Passive Replica:

If the primary replica fails a backup is chosen to be the primary replica during this process the system will not be available and hence it is not highly available replication system. This replica does not handle software failure as the primary replica only processes client request. Thus, we will implement DAMS for Software Failure Tolerance using Active Replication.

2.1.2. Active Replication:

This replication is implemented to detect and recover from software bugs and crash failure. In Active Replication, all the RM processes the client request and produces the result which is sent back to the FE. The RM executes the client request in Total Order.

To detect software bugs in the replica the Active Replication uses Total Order Implementation. Where, all the RM will process the result based on the order in which client request which ensures Sequential Consistency of the data on each replica. Thus, producing identical data on each replica after processing each client requests.

2.2. Total Order Reliable Multicast:

In Active Replication as the front-end sends the request to all the replica managers (RM) it ensures that all the replica manager receives the request to perform failure detection because if one of the replica's does not receive the request then we will not be able to detect software failure in the system. Thus, we will use Reliable Multicast. Using reliable multicast our front-end is sure that the multicast was received successfully by all the replicas using sequence number.

Total Order defines that the replicas must work in synchronization based on the order of requests requested from the clients. If there are multiple requests from clients the front-end makes sure that each request is ordered and then all the replicas implements the requests in order synchronization.

This is Total Order Reliable Multicast which ensures same operation in same order which guarantees data sequential consistency.

2.3. Total Ordering using Sequencer:

To ensure Total Ordering to all the RM the request is transmitted from the FE to the sequencer. Sequencer then generates a unique sequence number and replies it to the FE which then multicasts to all the RM.

Here, client request is multicast twice. Once, from FE to the RM without the sequence number and other time it generates the sequence number using sequencer for each request and multi-cast to RM again. This cause bandwidth overhead on the network as the request is multi-cast twice to the RM.

2.4. Kaashoek's Protocol:

To avoid this problem, we use Kaashoek's Protocol. In Kaashoek's protocol the FE will just send the request from client to the sequencer where sequencer will generate sequences of the client request and the it will multi-cast it to the RM.

3. DAMS Architecture Design as Software Failure Tolerance System using Kaashoek's Algorithm

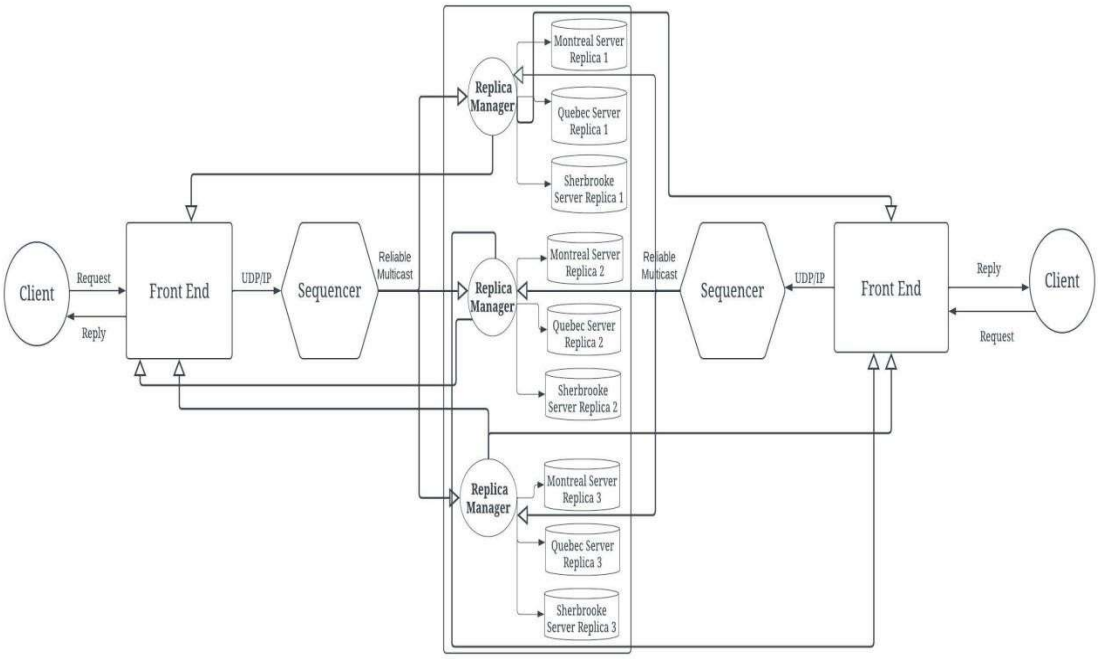


Fig 1. DAMS Architecture as Software Failure Tolerance using Kaashoek's Algorithm

In this project we are implementing DAMS with Software Failure Tolerance. In our architecture we have Front-End (FE), Sequencer and Replica Manager (RM). As in the above architecture we have three Replica Manager for each city namely Montreal, Quebec and Sherbrooke. Each city has three replicas e.g for Montreal Server we will create Montreal Replica 1, Montreal Replica 2 and Montreal Replica 3 and so for Quebec and Sherbrooke as Quebec Replica 1, Quebec Replica 2, Quebec Replica 3, Sherbrooke Replica 1, Sherbrooke Replica 2 and Sherbrooke Replica 3.

We have implemented Web-Services for Client-Server communication in this Distributed System.

Client:

Sends a request to the Front- End (FE) and wait for the reply.

Front-End (FE):

Front-end acts as the middleware between the client and replica manager (RM).

It is a server-side object which takes the client request and forward it to the sequencer using UDP/IP unicast.

After, the request is processed, all the RM reply with their processed result to the FE. FE then calculates the precise result and notify the RM about the incorrect result received from any of the replica.

Sequencer:

After receiving the request from FE the sequencer will generate a send sequence number Sg for each request by piggybacking the send sequence number in the header of the requests and multi-casts to all the replica managers using UDP/IP reliable multi-cast.

Replica Manager (RM):

Replica Manager is responsible for failure detection and recovery.

After receiving the client request from sequencer, it forwards it to all the replicas using UDP/IP for processing based on the sequence received from the sequencer.

It replies back to the FE where FE notifies the RM about incorrect result producing replica then RM will perform failure recovery by replacing the replica. Failure recover is done based on the number of times a certain replica has failed. If a replica has failed more than three times then it will replace the replica with the correct implementation.

Replica:

Each Replica is a JAVA implementation which contains the services which is provided to the client.

4. Data-flow Diagram:

Dataflow provides a visual representation of the operation of each process and how the data is transmitted in the DAMS system to achieve Software Failure Tolerance in DAMS.

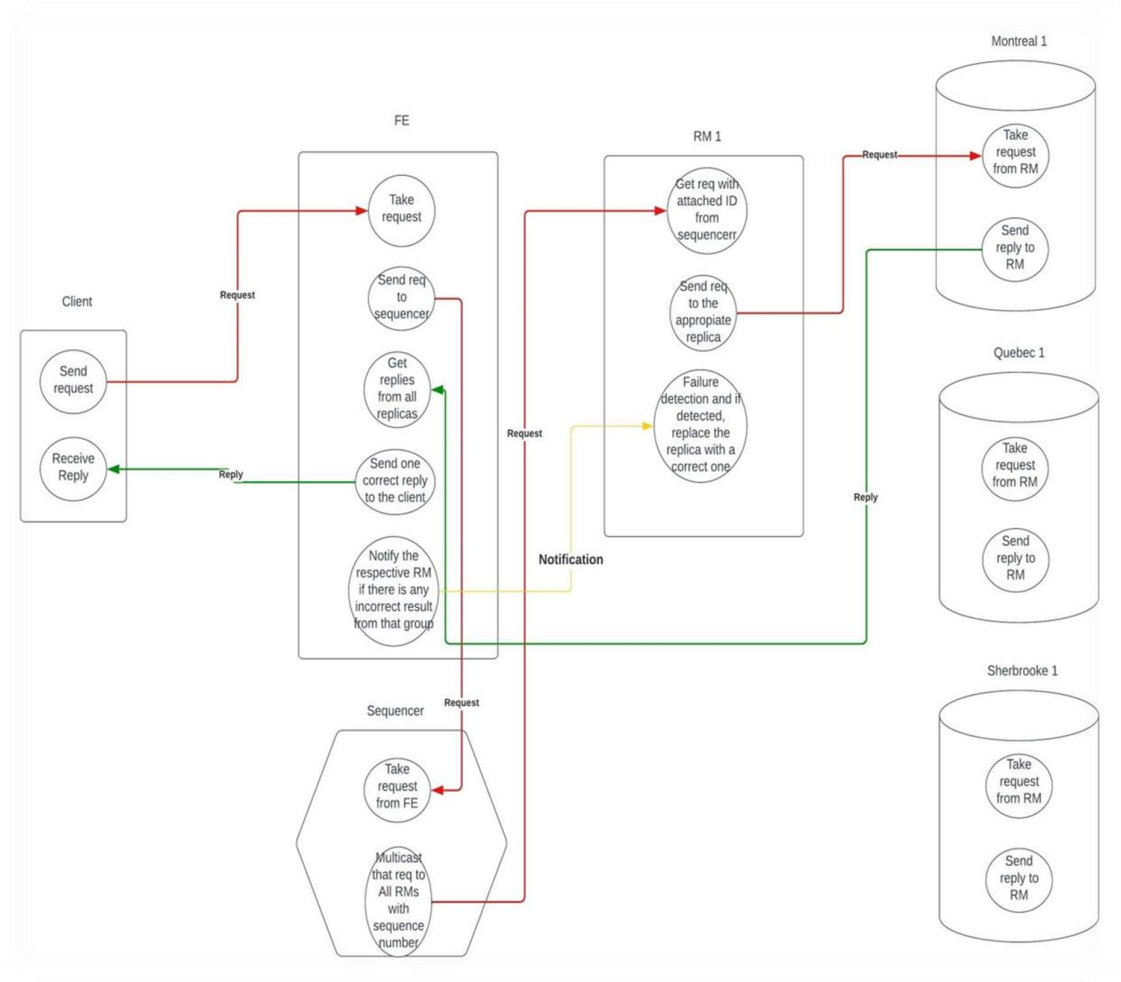
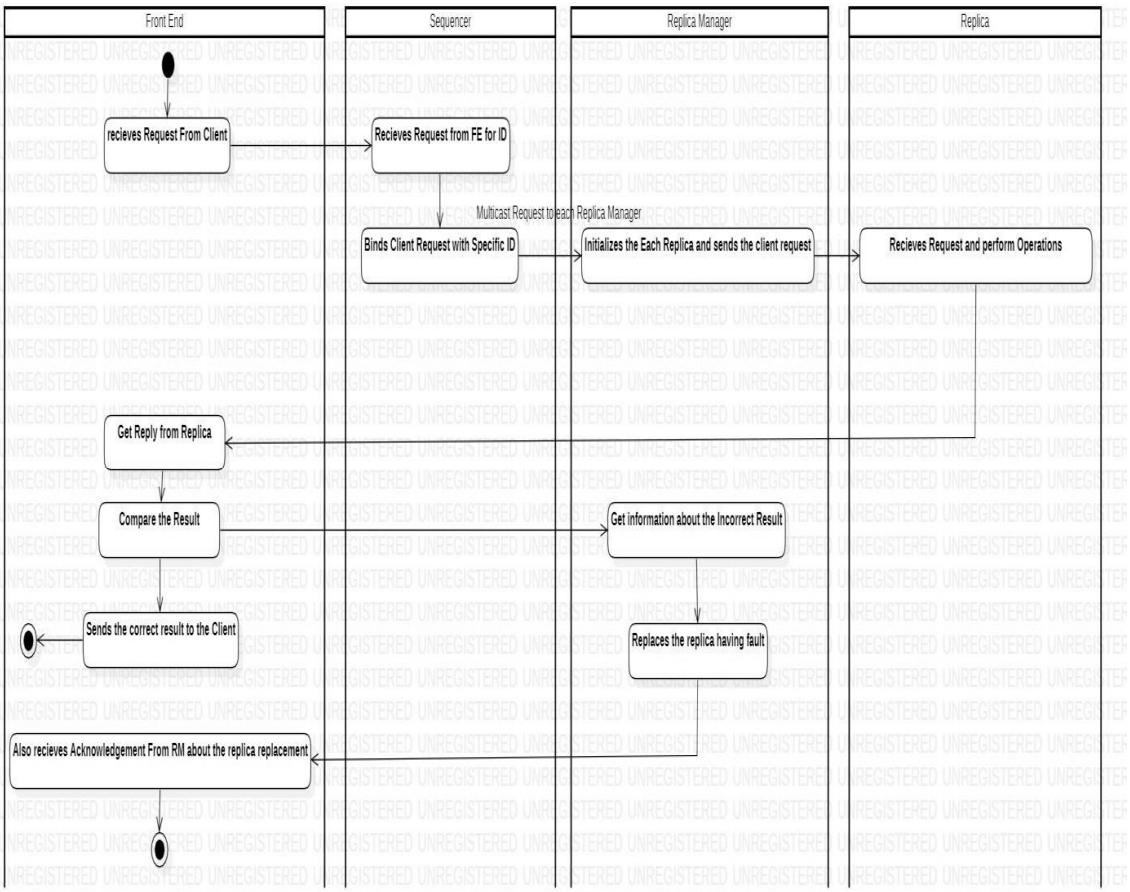


Fig 2. Dataflow Diagram of Software Tolerance Distributed Appointment Management System using Kaashoek's Algorithm

5. Activity Diagram

An activity diagram is a behavioral diagram i.e. it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.



6. Team and Individual Task

Student ID	Student Name	Student Task
40192329	Kawsar Ahmed	Replica Manager (RM)
40203768	Gagandeep Kaur	Sequencer
40205690	Anam Ayesha Shaikh	Front-End (FE)

Front End- The Front end also known as the web services server to the web services client. That is when clients send request it first lands on front end. Front end then invokes sequencer for assigning sequence number to the request. After getting the result from sequencer then it unicast the request to multiple replica managers for results. After receiving results from Replica manager front end compares the result. It takes the majority results as correct result and if there is any incorrect result from any replica then informs that replica manager to replace that replica.

Sequencer- This has been implemented for sending the requests in order so that Each replica should implement the same request. This is done simple by adding incrementing the sequence number each time the front end gets and request.

Replica Manager- The replica manager receives the request from front end and forward the request to respective replica. Replica Manager also queues the client request according to sequence number if it receives multiple client requests. Replica manager also changes the faulty replica after it gets the information about the defect from front end.

7. Difficult part/Problems Faced

Implementing multicasting on Local Area Network (LAN) was most difficult part for this project. Multicast request was only sent to one replica manager at time, and it was very unpredictable that which replica manager is receiving the request or will receive the request next. Thus, to be sure every replica manager receives request and send reply back instead of using multicasting we have used multicast sockets to send the UDP request to every replica. By implementing this we were able to invoke all the replica managers, send requests and receive replies.

8. Test-Scenario

Test Case number	Test Scenario	Inputs	Result
T_1	Add Appointment	MTLA1234 Physician MTLE210422 2	Appointment Added
T_2	Add Appointment	MTLA1234 Physician MTLE210422 2	Add Appointment unsuccessful
T_3	Remove Appointment	MTLA1234 Physician MTLE210422	Appointment Removed successfully
T_4	Remove Appointment	MTLA1234 Dental MTLE210422	No such Appointment Appointment Remove Unsuccessful
T_5	List Available Appointment	MTLA1234 Dental	{List All the available appointments from All the servers}
T_6	Add Appointment	MTLA1234 MTLP1234 Physician MTLE210422	Appointment Booked Successfully
T_7	Add Appointment	MTLA1234 MTLP1234 Physician MTLE210422	Appointment Already Booked
T_8	List Appointment Schedule	MTLA1234 MTLP1234	{List All the appointments booked by patient on all the servers}
T_9	Swap Appointment	MTLA1234 MTLP1234 Physician MTLE210422 Physician MTLM210422	Appointment Swapped Successfully
T_10	Cancel Appointment	MTLA1234 MTLP1234 MTLM210422	Appointment Cancelled Successfully
T_11	Cancel Appointment	MTLA1234 MTLP1234 MTLM210422	No such Appointment. Appointment Cancelled Unsuccessful

9. References:

i. Replication
https://www.slideshare.net/KavyaBarnadhyaHazari/replication-in-distributedsystems#:~:text=Replication%20in%20computing%20involves%20sharing,fault%2Dtole rance%2C%20or%20accessibility

ii. Activity Diagram

https://www.geeksforgeeks.org/unified-modeling-language-uml-activitydiagrams/#:~:text=An%20activity%20diagram%20is%20a,the%20activity%20is%20bein g%20executed.