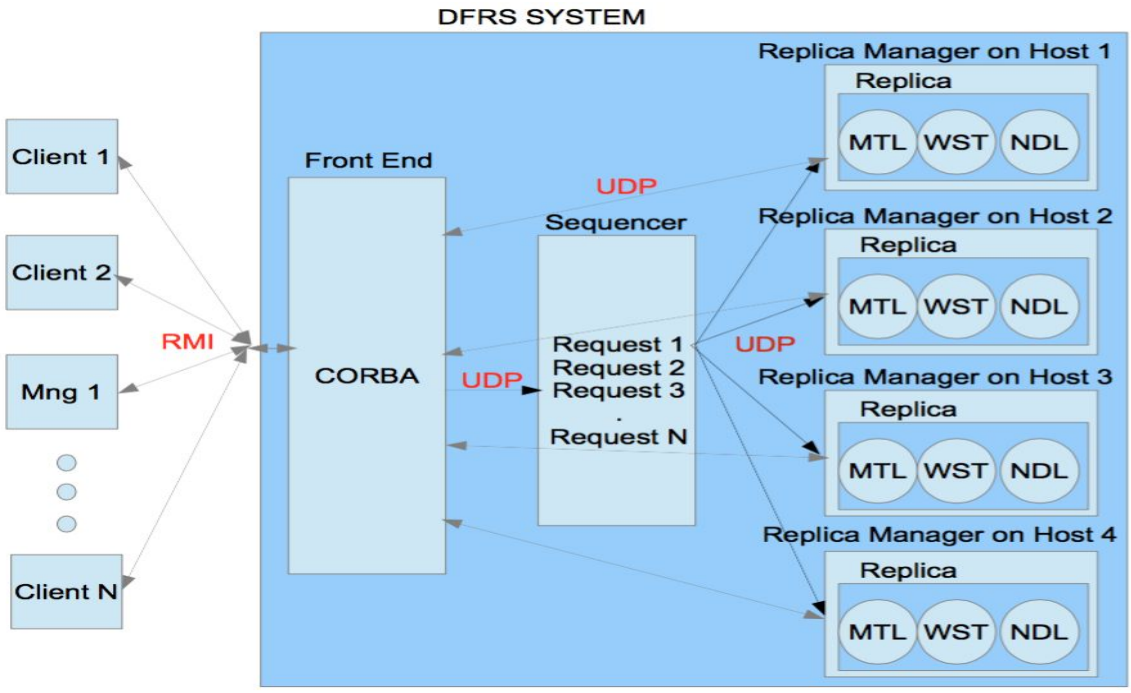
Concordia University
Faculty of Engineering and Computer Science
COMP 6231 Distributed System Design
Fall 2016

Project Design

Done by: Ulan Baitassov 40001592
Sajjad Ashraf 40012126
Muhammad Umer 40015021
Feras Younis 25344484

Project requirements:
Distributed Flight Reservation System (DFRS) should be software failure tolerant and highly available under process crash failure using process replication.

In order to tolerate the software failure and process crash failure at the same time, the system need 4 replicas running at the same time. In case if one server crashes and another one is returning incorrect results, the system is still is capable to verify the correct information using 2 more replicas, assuming these 2 replicas are working correctly.



Basic request-reply flow if no failures occur

The client or manager invokes the method on CORBA object which is implemented in Front End (FE). The FE receives the request, sets the timer for request and forwards it to the Sequencer using UDP/IP message. The Sequencer receives the message, adds the sequence number, FE ip address and port number to the message and forwards the message to all Replicas using UDP/IP. The Server replicas receive the message, converts the message to execution command and execute the request. After obtaining the result, servers send the result directly to the FE using UDP messages. The FE receives the messages, compares the results and sends correct result to the client or manager.

Front End (CORBA)

Front End (FE) is a process which may run on the same or different host. It receives the client request as a CORBA invocation and forwards the request to the Sequencer using UDP/IP Protocol. The FE has IP address and port number in order to send and receive UDP messages. The FE receives the results from the replicas via UDP/IP message and sends a single correct result back to the client as soon as possible. It compares the received results from different replicas in order to verify the correct result which will be sent to the client. By doing so, FE bypass the software failures. The FE will also have a timer for each client request. The timer will help FE determine how long it should wait for replica to reply. In case when one replica is down, so that FE doesn't wait for reply forever and sends the result acquired from other replicas to the client when timer is out. If any one of the replicas produces incorrect result the FE informs the Replica Manager of a possibly failed replica that is not sending result or producing incorrect results.

Sequencer

Sequencer is a process, which provides both the total ordering of the clients requests and reliable group communication using reliable UDP multicast to a group of replicas. It performs following major steps:

1. The Sequencer receives client requests from Front End(FE) via UDP message.
2. For each client request, it assigns a unique sequence number in total order.
3. The Sequencer adds this sequence number along with ip address and port number of FE to the message.
4. The Sequencer multicasts this message to the server replicas using reliable UDP/IP protocol.
5. The Sequencer also piggybacks the sequence number of the client request to the acknowledgement to FE, so that later FE is able to identify the message from the replica to corresponding client request.

Total Order

The Sequencer ensures that messages are received reliably and in the same order by server replicas. It multicasts one request and ensures that all server replicas receives the message before multicasting the next one. This achieved by reliable UDP protocol that is described below.

UDP Multicast

The multicast will be implemented by group communication where every server replica joins the group in order to receives the message requests. The Sequencer will use the group to broadcast the message to corresponding servers.

Replica Manager

Replica manager (RM) will be responsible for managing the replicas of our flight reservation servers. The Replica Manager creates and initializes the server replicas. It makes sure that server replicas are atomic in nature and recoverable in case of any failure. The replica manager is also responsible for making sure that servers are UP & Running by continuously receiving a "Heart Beat" messages from each server at the interval of every 30 seconds to make sure each server is running and ready to respond to any request. If it fails to receive the "Heart Beat" signal from replica's then it checks whether the replica was already running or not, if it was not running then RM can start the replica. Replica manager receives the message (feedback) from Front End if the server does not provide the result or sends incorrect result. Replica manager performs its replica recovery mechanism corresponding to the problem occurred. Replica manager implements the failure detection by maintaining the count number of incorrect results sent to FE by server replicas, if it reaches three, then replica manager will restart the faulty server replica. If the message from FE says that server replica failed to send the result in time, then replica manager immediately checks the status of this replica by sending UDP message to it. If the server replica is unreachable, then RM starts new replica.

Design reliability for UDP messaging

UDP protocol communication should be made reliable in order to avoid message loss. As we know, UDP protocol is unreliable and works on best effort. The messages that are sent may be corrupted, lost or delivered out of order. The UDP protocol does not provide any kind of reliability in message delivery except simple error checksum mechanism. The protocol does not retransmit message if the message is lost. Therefore, we are going to implement the reliability on top of the UDP protocol.

The reliability is achieved by providing the integrity and validity to message delivery. Integrity is making sure that the message is not corrupted and not duplicated. The validity is making sure that the message is delivered to the recipient.

Design for integrity:

1. The messages will be delivered at most one. Will add a filter to the receiver to drop any duplicate packet using sequence number attached to the message.
2. IP UDP got an option for checksums and will reject any corrupt messages.

Design for validity:

1. Acknowledgment of the received packets.
2. Timer for reply to come back. If no reply in certain time, retransmit the message.

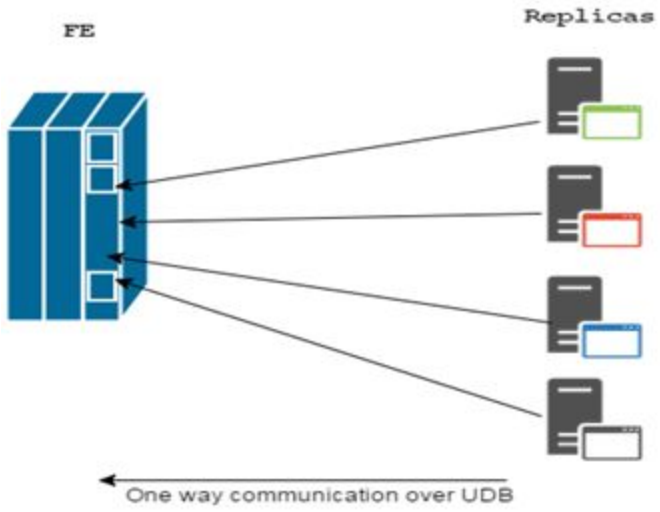
TCP provides all reliability needed to send the message, Why not to use TCP then ?

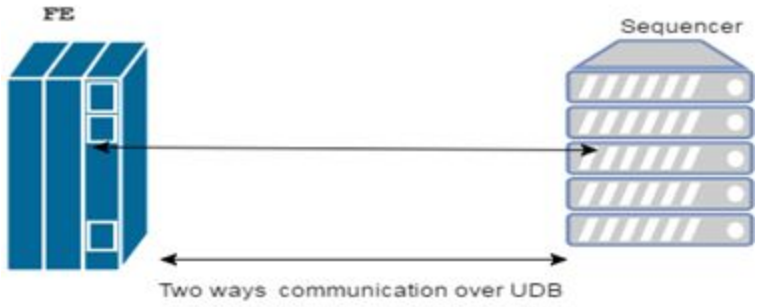
TCP adds too much complexity/overhead. TCP is slower, because it requires the connection establishment. TCP is used for streaming data, which in our case not required because, we send small size messages.

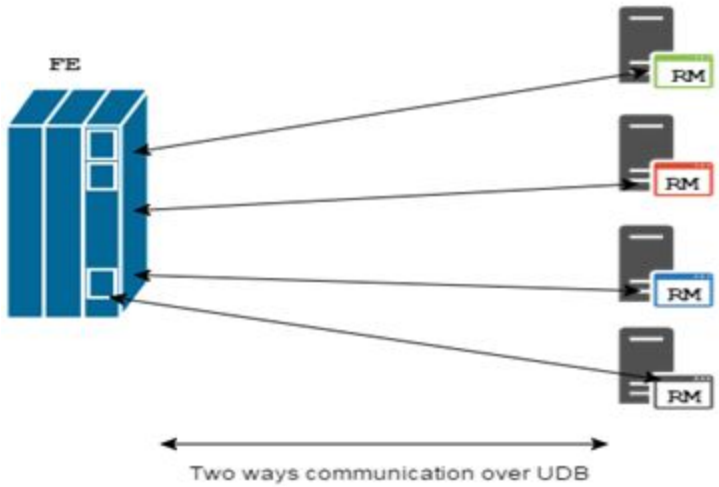
The step we will perform to a make UDP reliable as follows:

1. Sender will compute a checksum of the message and send it with the message and sequence number to recipient.
2. After sending, sender will wait for reasonable amount of time to receive an acknowledgment from the recipient with sequence number +1. If recipient fails to reply or the acknowledgement sequence number is same as current sequence number, then sender retransmits the same message again.
3. On receiver side, recipient receives the message, extracts the sequence number and checksum from the message. The recipient validates the message by computing the checksum. If the computed checksum match with received checksum, it means message is not corrupted. Then it checks the sequence number. If expecting sequence number is different, then packet is dropped. If it is same, then recipient sends the acknowledgement with next sequence number. In case message is corrupted, then recipient sends the acknowledgement with same sequence number.

Reliable User Datagram Protocol (RUDP) sockets will be used to pass messages between our system components (FE, RM, Sequencer, and Replica) as illustrated in below diagram.







Test Cases

Test case #1 kill one of the replica and see if the RM replace it again. This can be done by stopping that replica processor.

Test case #2 make one of the replica's function return incorrect result and invoke that particular function in all of the replicas by the client and see if the Front End catches the bug and informs the Replica Manager. Repeat this call for another three times and see if RM would replace faulty replica with a new one.

Test case #3 kill one of the replica and call the function from Test case#2 on another replica and see if the system is able to recover from both failures.

Test case #4 this test will be a white test where we send the same message from UDP socket and see if the receiver socket drop it.

Test case #5 Front End test of result comparison. Test the method which performs the correct result selection. For different inputs, make sure that selection works correctly.

In addition, we will create test cases which aim to test a simple request with best possible arguments called Happy test. This test give us conclusion that an application is deployed and configured correctly.

Project work division

Module	Ulan	Sajjad	Umer	Feras
Modify server to replica	+	+	+	+
Front end and software failure	+			
Sequencer		+		
Replica manager and crash failure			+	
Design reliability for UDP messaging				+
Design of test cases	+	+	+	+
System deployment to the local network	+	+	+	+