Distributed Event Management System

Software Failure Tolerant or Highly Available Distributed Event Management System

Submitted By:

Gursimran Singh - 40080981
Natheepan Ganeshamoorthy - 29335838
Ubor Ufuoma - 40072909

Table of Contents

Introduction

Distributed Event Management system is a distributed system which can be used by an Event Manager to add/remove/book or delete events and permits customers to book/cancel and swap events. This system is designed in such a way that it can tolerate either a single software (non-malicious Byzantine) failure or be highly available under a single process crash failure using active replication. In order to maintain location transparency, we use Java CORBA from client to the Front-end. For communication between the other modules (Sequencer, Replicas, Replica Manager), UDP communication is used. The goal of the project is to handle component crash failure or software failure and serves the user.

Techniques Used

(a) UDP (User Datagram Protocol):

User Datagram Protocol is a transport layer protocol used by application layer to send and receive messages which we refer them as datagram packets, to other hosts connected in a network.

UDP supports multicast, where a single datagram packet can be automatically routed without duplication to very large numbers of subscribers. UDP is lightweight, as it doesn't require any prior communication to set up the connection between two applications and also it avoids the overhead of error checking and correction.

Implementation:
We can implement a UDP protocol using a Datagram Packet class of java. A datagram packet has the following structure:

Source Port : A source port number consist of source IP address along with port number using which if a necessary receiver can respond to the sender and is optional.

Destination Port: A destination port number consist of source IP address along with a port number of the particular application using which router forward datagram packet to the source.

Length: This field specifies the length of UDP message along with header size which is 8 bytes and data can be of length 65535 bytes maximum.

CheckSum: This field is used to maintain data integrity of the UDP datagram packet.

Data: This field contains actual data of the UDP datagram packet that destination application will receive.

(b) Common Object Request Broker Architecture (CORBA):

The CORBA architecture mainly implemented to make client enable to invoke remote calls when a client and a server are implemented in different programming languages.

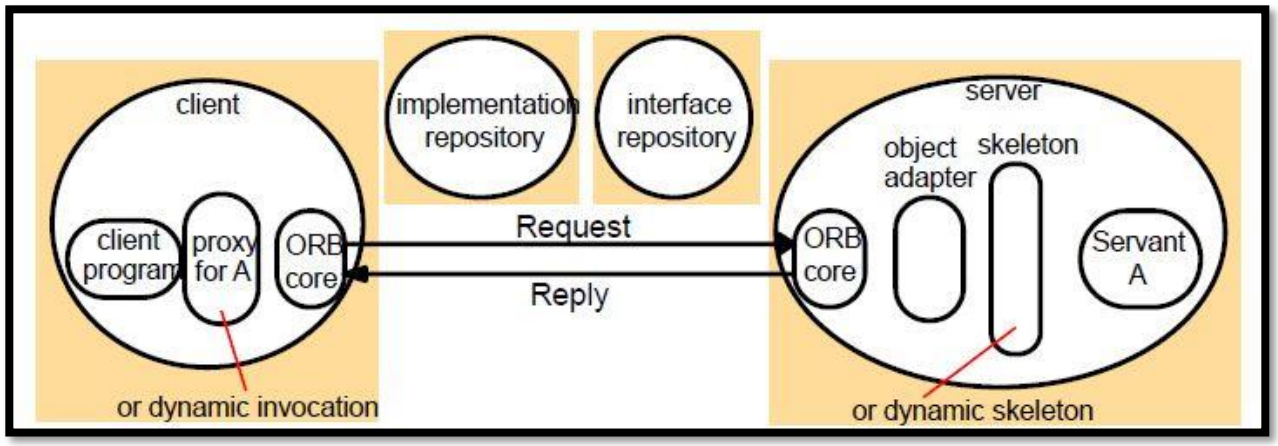

Figure 1 Corba Architecture

CORBA Architecture contains the following components:

- Object Adapter
  - Object Adapter connects the IDL interface of CORBA object and the programming language interface
  - Object Adapter generated remote object reference, it starts and stops
  - servants and send RMI to the respective servant.
  - Portable Object Adapter
    1) POA allows client and server to run on ORBs which are developed in different programming languages. It supports CORBA objects to be instantiated transparently.
    2) POA differentiates creation of CORBA objects with the creation of servants which creates those objects

- Skeleton
  - Skeleton class is implemented on server-side. An RMIs are dispatched to a server via skeleton and skeleton is responsible for unmarshalling of incoming requests and of outgoing responses.

- Client Stub
  - Client stubs are implemented on client-side. The client stub behaves like a server for that particular client. Client stubs also responsible for marshalling of outgoing requests and unmarshalling of incoming replies.

- Implementation repository
  - The implementation repository works to active servers on request and locates server where they are running. Here, the object adapter name is used to identify a server.
  - The implementation repository used to map object adapter name and path of a file in which the object is implemented. When object implementation started, the hostname and port number of a server also mapped to the implementation repository.

- Interface repository
  - Interface repository store the name of method and type of arguments with an exception.

- If a client has not a proxy of CORBA remote object, then interface repository provides the information about methods and type of parameters.

(c) Reliable Multicast Algorithm:

Process can belong to several closed groups

1. On initialization, Received := { };
   For process p to Reliable-multicast message m to group g
   B-multicast(g,m); //p ∈ g is included as destination

2. On B-deliver(m) at process q with g = group(m) if(m ∉ Received)
   Then End if Received := Received U { m };
   If (q ≠ p) then B-multicast(g,m); end if R-deliver m;

In the reliable multicast algorithm, the sender multicast the message to the group and to make the multicast reliable, each process in the group upon receiving the message they will again broadcast the message to the group so that if one process gets the message all other processes in the group also gets the message. The processes in the group ignore duplicate messages.

Code snippet for Multicast:

```
// args give message contents & destination multicast group (e.g. "228.5.6.7")
MulticastSocket s =null;
try {
        InetAddress group = InetAddress.getByName(args[1]);
        s = new MulticastSocket(6789);
        s.joinGroup(group);
        byte [] m = args[0].getBytes();
        DatagramPacket messageOut =
                new DatagramPacket(m, m.length, group, 6789);
        s.send(messageOut);
```

(d) Total Ordering Using Sequencer:

1. Algorithm for Front End
   a) On Initialization: Received := { }; //Sequencer process To unicast
      message m to sequencer S
      B-unicast(sequencer(g),(m));
      On B-delivery(m) with sequencer S if(m ∉ Received)
      Then End if
      Received := Received U { m }; R-deliver m;
   b) Wait until <Ack> from replica manager RM; Unicast next message m to sequencer S
2. Algorithm for Sequencer of g: On initialization: S := 0;
   On B-delivery(m) with sequencer S
   R-multicast(g,<sequence number, m>); S := S+1;
   If B-delivery(Ack) with sequencer S from Front End S := 0;

To maintain total ordering, when the sequencer receives a request from Front End, it will store the request and multicast the request along with a sequence number to a group of replicas. All replicas will execute the message depending on the sequence number attached to the message and hence achieving total order execution.

When an acknowledgment is received from the respective front end then sequencer delete the stored request.

Design Architecture

The proposed architecture of the system consists of three parts, the Client layer, the CORBA Front End and Sequencer, and finally the server which consists of the Replica Managers. The Front End receives client's requests and forward the requests to the Sequencer. The sequencer assigns a unique sequence number and reliably multicasts the request to all the replicas. In order to handle a software failure, the three replicas execute client requests in total order and return the results back to the FE which in turn returns a single correct result back to the client as soon as two identical (correct) results are received from the replicas.

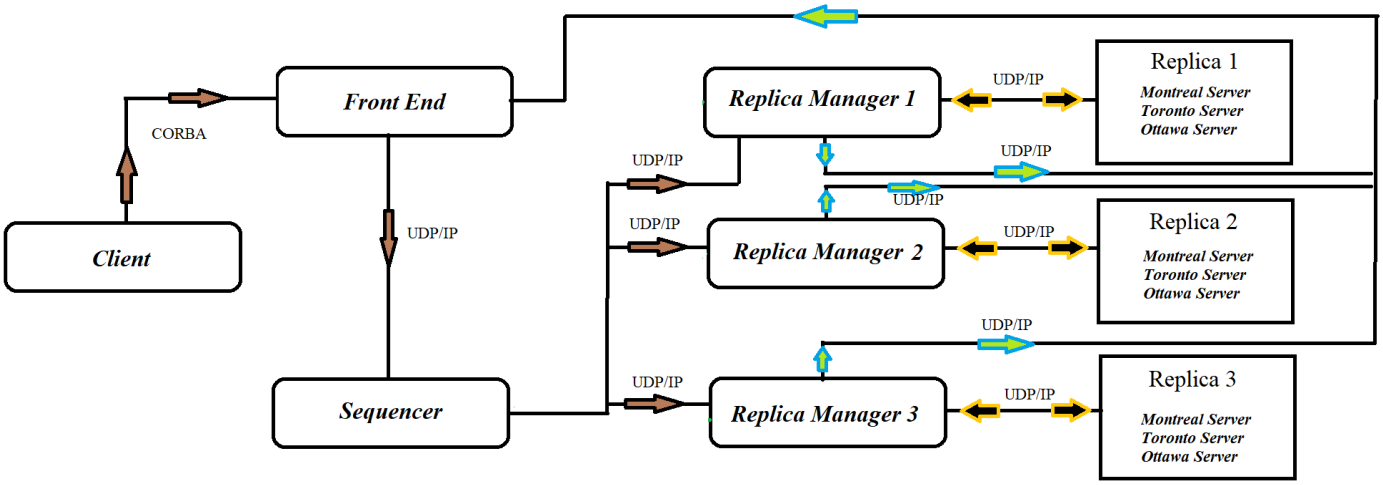

Figure 2 Project Architecture

(a) Client

This is the only module that interacts with the user. It accepts user input and provides a desirable output. The client will collect the necessary information from the user such as Customer ID, Manager ID, Event ID, Event Type, Booking Quantity and the operation to be performed and sends the request as a CORBA request to the front end. After the operation is performed by the system, the client gets the response back from the front end.

(b) Front End

After receiving the client request, it sends a UDP request to the sequencer. The front end also receives responses from each Replica Manager, checks for correctness of the responses and then reply back to the client with a response. It selects the majority of the responses and sends back to the client. In case of failure, it informs the Replica Managers about it. The front end will send an acknowledgment to the sequencer on successfully receiving the responses from the replicas and the sequencer will then multicast the subsequent requests.

(c) Sequencer

The sequencer will be waiting for UDP message from Front End and upon receiving the message it will append a unique sequence id to the message and store the message in a queue. It will broadcast the message to all the 3 replicas and wait for an acknowledgment from all the replicas. If it receives an acknowledgment, it will remove the message from the queue and broadcast the next message waiting in the queue. If the replica does not get acknowledgment from any replica within the timeout, it will again multicast the same message to that particular replica, this mechanism continues until it receives an acknowledgment from all the three replicas. The replicas will drop the duplicate messages from the sequencer by keeping track of the past message sequence ID. In this way, the sequencer achieves Total ordering. It also alerts the other replicas, if from a particular replica it doesn't get any acknowledgement.

(d) Replica Managers

The Replica Manager forwards the operation request to the replicas. All the replicas process the requests in the same order. Replica Manager maintains a counter each time there is an incorrect response from Replicas to the front end and then Replica manager decides how to handle that particular error.

(e) Data Flow

The client sends a CORBA request to the Front-end. It sends the encoded message as a UDP request to the Sequencer and starts a Timer for maintaining the response time from the replicas.

On receiving the request from Front-end, the Sequencer then sends an acknowledgment back to the Front-end. It attaches a unique sequence number to the request and then multicasts it to all the Replicas.

The Replicas performs the operation (like add event, remove an event, list all events, book event, cancel event, list customer schedule, swap event) and encodes the response. The Replicas then sends the response back to the Front-end as a UDP message.

The Front-end stops the timer on receiving the responses from Replicas and determines the correct response from the responses.

In case of different responses from all the replicas, the majority of all the responses (at least 2) will be selected by the Front-end and is sent back to the client. The Front-end then checks back with the Replica Manager with the incorrect result and the Replica Manager keeps a counter for the same. Once that counter reaches a threshold of 3, the Replica Manager replaces its corresponding replica.

In case of not receiving a response from any of the replica by the front end within a certain time. It will inform to replica manager. The replica manager reboots its corresponding replica with data consistency.

(f) Recovery Management:

- Software Bug Recovery:
  When the front end gets wrong answer from any replica. It sends a message to the particular replica manager about the software bug. The correct method is run after the replica manager which has the bug knows that it is having a software bug.
- Crash Failure Recovery:
  When the front end doesn't get any output from a particular replica after the 2 times of the slowest response from the replica, then it sends the message to the other replicas informing a failure in the replica. The other replicas send a message to the crashed replica and confirm whether it failed or not.
- Recovery:
  When the crashed replica restarted then it gets the data from the other replica for data consistency as a java object, this java object will be parsed by the replica and gets ready for the next operation to perform which will be received from the sequencer.

ASSUMPTIONS:

- Sequencer doesn't crash.
- Replica Managers don't crash.
- The network doesn't congest and die.

Test Scenarios

(a) Client

Preconditions Client and FE should communicate using CORBA.		
Action	**Expected Result**	**Actual Result**
Send request to FE.	A request should be sent successfully.	Pass
Receive response from FE.	A response should be received successfully.	Pass

(b) Front End

Preconditions Client and FE should communicate using CORBA. Only FE should communicate with Sequencer using UDP. FE and Replicas, Replica Managers communicates using UDP.		
Action	**Expected Result**	**Actual Result**
Receive requests from the client	each request sent by client must be received successfully by the FE	Passed
Forward each client request to the sequencer	each request must be sent successfully to the sequencer.	Passed
Receive responses from all replicas	If there is no crash failure, FE should receive the response from all replicas.	Passed
Check for software failure	If all the received responses are different, Check the response which is in minority. FE should consider that replica can have a software failure.	Passed
Check for crash failure	If a response from any replica will not be received in a certain time period, FE should consider that replica might be crashed.	Passed
Inform to replica managers about failure.	If FE detects any of the failure software failure or crash failure, It should inform to replica manager about the failure	Passed

(c) Sequencer

Preconditions Sequencer should be failure free. Only FE should communicate with Sequencer using UDP. Only Sequencer can communicate with Replicas by also using UDP.		
Action	**Expected Result**	**Actual Result**
Receive request from FE	Sequencer should able to receive request successfully sent by the FE.	Passed
Set sequence ID	Sequencer should set unique sequence ID for each request.	Passed
Forward request to all the replicas	Sequencer should forward request message with respective sequence ID.	Passed

(d) Replica Managers

Preconditions Only FE can communicate with Replica managers using UDP. Replica managers should be connected to their own replica. All the replica managers can communicate among themselves using UDP.		
Action	**Expected Result**	**Actual Result**
Get informed by the FE	Replica managers should get informed while any replica has failure.	Passed
Keep count for software failure	If replica manager get informed by the FE about software failure consecutive three times, then it should take action for a recovery.	Passed
Manage software failure	To manage software failure, replica manager should replace failed replica.	Passed
Manage crash failure	While any replica get crashed, its replica manager should restart the	Passed

	replica and also manage data consistency.	
Get data from all other replica manager	In case of crash failure, The replica manager should able to get data from all other replica though replica manager.	Passed

(e) Replicas

Preconditions - Only Sequencer can communicate with Replicas using UDP. - Replicas should be connected to their own replica managers. - Only Replicas can communicate FE using UDP.		
Action	Expected Result	Actual Result
Receive request from the sequencer	All replicas should able to receive a request from the sequencer with sequence ID for each message.	Passed
Send response back to the FE	Replicas should send a response back to FE and it should maintain total ordering.	Passed
Manage data structure	Each replica should have their own data structure and they should maintain the data.	Passed

S. No	Test Cases	Set Up	Expected Result	Actual Result
	Add an event	Enter manager id	-Logs are	-Same as expected

1		Manager id is valid, list of operations to be perform display Select 1 to add event Enter eventID, eventType and capacity	generated -Event added	-Same as expected result
2	Remove an event	Enter manager id Manager id is valid, list of operations to be perform display - Select 2 to remove event -Enter eventID	-Logs are generated -event removed - event capacity decreased	-Same as expected result
3	List all available events	-Enter manager id Manager id is valid, list of operations to be perform display -Select 3 to list event	Logs are generated List of all available events	-Same as expected result
4	Get Booking Schedule from all servers	Enter customer id customer id is valid, list of operations to be performed display	-Logs are generated -Booked event should come as output from all 3 servers	-Same as expected result
5	Swap Events	-Enter customer ID -Enter old eventID and type -Enter new eventID and type	-Logs are generated -Swapped event confirmation should come.	-Same as expected result
6	Book Event	-Enter customer ID -Enter eventID and type -Enter eventType	-Logs are generated -Swapped event confirmation should come.	-Same as expected result
7	Cancel Event	-Enter customer ID -Enter eventID and type -Enter eventType	-Logs are generated -Swapped event confirmation should come.	-Same as expected result

Team Responsibility

1. (Gursimran Singh - 40080981)
   Design and implement the front end (FE) which receives a client request as a CORBA invocation, forwards the request to the sequencer, receives the results from the replicas and sends a single correct result back to the client as soon as possible. The FE also informs all the RMs of a possibly failed replica that produced incorrect result.

2. (Natheepan Ganeshamoorthy - 29335838)
   Design and implement the replica manager (RM) which creates and initializes the actively replicated server system. The RM also implements the failure detection and recovery for the required type of failure.

3. (Ubor Ufuoma - 40072909)
   Design and implement a failure-free sequencer which receives a client request from a FE, assigns a unique sequence number to the request and reliably multicast the request with the sequence number and FE information to all the three server replicas

Bibliography

[1] "Getting Started Using Java™ RMI," 2019. [Online]. Available: https://docs.oracle.com/javase/7/docs/technotes/guides/rmi/hello/hello-world.html.

[2] "Course WebSlides," 2019. [Online]. Available: https://moodle.concordia.ca/moodle/pluginfile.php/3595968/mod_resource/content/1/Tutorial%202%28UDP TCP%29.pdf.

[3] "UDP (User Datagram Protocol)," 2018. [Online]. Available: https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol.

[4] "Multithreading in JAVA," 2019. [Online]. Available: https://www.geeksforgeeks.org/multithreading-in-java/.