DISTRIBUTED SYSTEMS DESIGN

COMP 6231: SUMMER 2017

INSTRUCTOR: SUKHJINDER K NARULA

PROJECT 2: HIGHLY AVAILABLE DISTRIBUTED CLASS MANAGEMENT SYSTEM (DCMS)

AKRITI SAINI: 40039689
SHREYA BISWAS: 40018190
SHUAI CHEN: 40010960

DESCRIPTION:

In this project, we aim to design an application that implements a highly available Distributed Class Management System (DCMS), which tolerates process crashes using unreliable failure detection. There are three server processes running on three different hosts providing redundancy for high availability and periodically checking each other for failure detection. One of the processes in the group is the elected leader and receives the requests from the clients through a CORBA front end. The leader of the server group broadcasts the client request automatically to all the servers in the group using a reliable FIFO broadcast mechanism, receives the responses from them and sends a single correct response back to the client as soon as possible. The replicated servers communicate using UDP protocol.

REQUIREMENTS:

The application has the following requirements:

1. Design highly available active replication scheme using process group replication and reliable group replication.
2. Design and implement group leader process which receives request from the front end, FIFO broadcasts the request to all the server replicas in the group using UDP, receives the responses from the server replicas and sends the correct response back to the front end as soon as possible.
3. Design and implement a reliable FIFO broadcast subsystem over the unreliable UDP layer.
4. Design and implement a detection subsystem in which the processes in the group periodically check each other and remove a failed process from the group. If the group leader has failed, a new leader is elected using a distributed election subsystem.
5. If the Primary Backup fails, we choose the replica with the next ID as the new leader.

ARCHITECTURE DESIGN

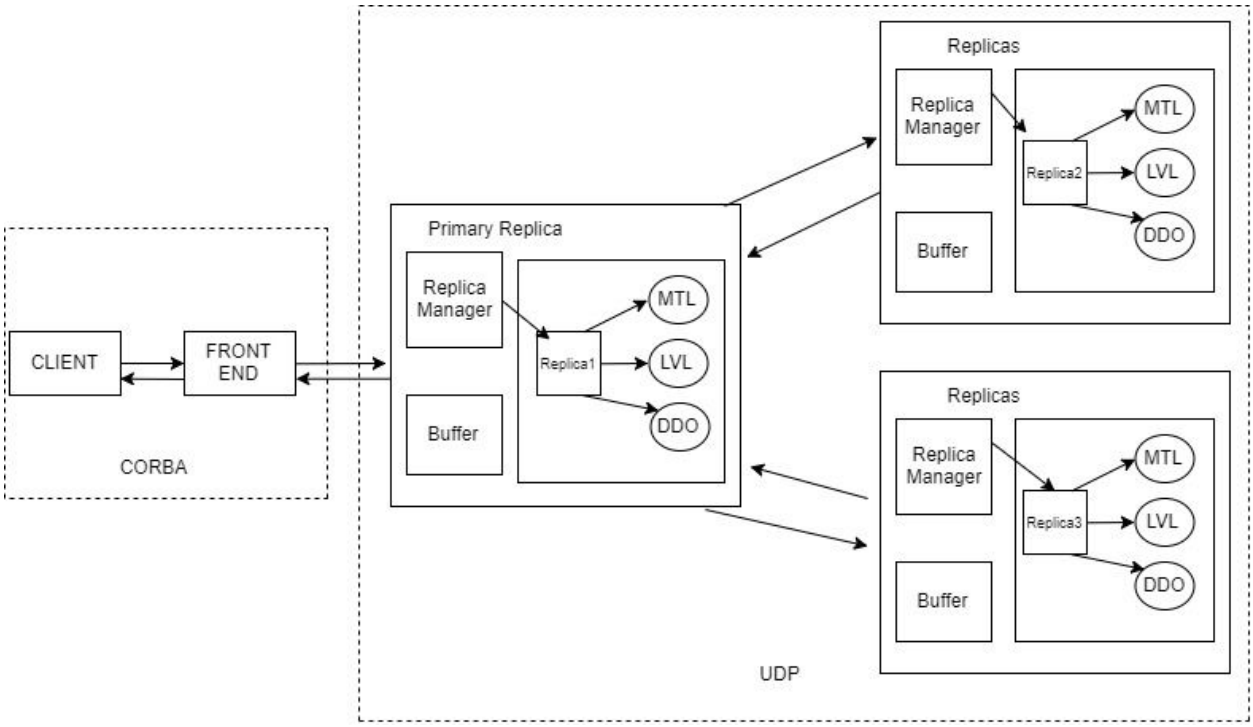

Fig 1: Overall Architecture

IMPLEMENTATION

1. We have one or more clients.
2. Client communicates with the Front End through CORBA.
3. The Primary communicates with the Replica which is connected to the Servers, which all are connected through UDP.
4. We have used three replica managers out of which 1 replica manager is primary replica and rest would be backup. The three replica managers communicate among themselves using UDP. One of the replica manager is elected leader which acts as the primary backup.
5. Here, we have One replica manager with respective to one Replica which is directly connected to the 3 servers. Similarly for the other two Replica Managers, as shown in Fig1.
6. There is one Primary Replica which is the elected leader among the replica managers.
7. Each Replica has its own buffer. If one of the replica managers fails(crashing), it need some time to get restarted, during this time the request is stored in the buffer till the replica manager restarts.
8. There are two backup replica managers. They receive the forwarded request from the primary backup, processes them and sends the acknowledgement back to the primary backup.

9. FIFO broadcast is used by the primary to broadcast the requests to the other replica managers.
10. Each replica sends heartbeat to the other two Replica Managers every 5 seconds to ensure that it is alive. If a replica Manager does not receive the heartbeat of a replica within every 5 seconds, it assumes that the replica has failed and informs its respective Replica Manager to reboot the replica.

WORKFLOW

1. The client sends a request to the CORBA Front End.
2. The Front End then forwards the request to the Primary Replica Manager.
3. The Primary communicates with the Replica which is connected to the Servers. Each request is forwarded in this manner.
4. The Primary Backup FIFO broadcasts the request to the other two Replica Managers which in turn forwards the request to their respective replicas to process it.
5. After the request is processed by all the replicas, they send the reply to their respective Replica Manager, which in turn forwards the reply back to the Primary Backup.
6. After the Primary receives the response from the other Replica Managers, it forwards one correct response back to the Front end, which in turn sends the response back to the client.
7. If any of the replica fails, its respective Replica Manager is responsible for restarting the replica and restoring it back to its previous state..
8. Till the time the replicas are restarting, the request get added into the buffer of its respective Replica Manager.
9. If the Primary Backup fails, we choose the replica with the next ID as the new leader.
10. All these Replica Managers communicate with each other using reliable UDP.
11. Each replica sends heartbeat to the other two Replica Managers every 5 seconds to ensure that it is alive. If a replica Manager does not receive the heartbeat of a replica within every 5 seconds, it assumes that the replica has failed and informs its respective Replica Manager to reboot the replica.

MESSAGE FLOW

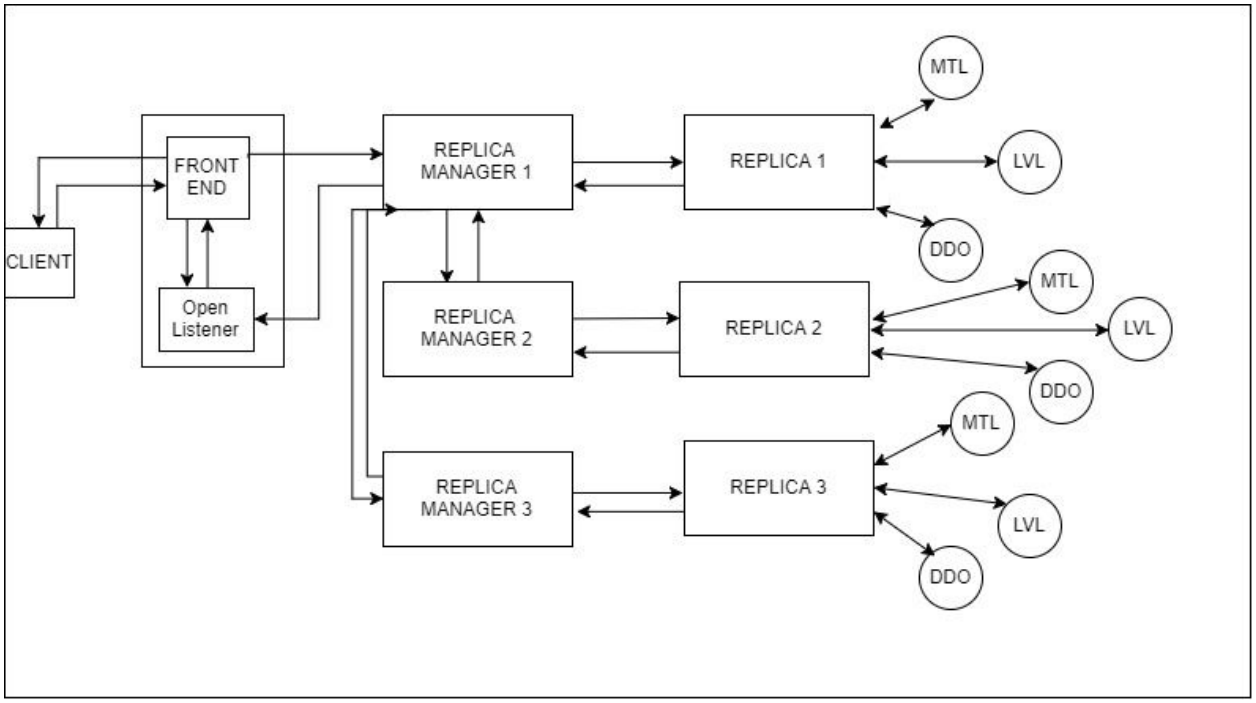

Fig 2: Message Passing

1. The client sends a request to the CORBA Front End.
2. The front end has a open listener which is always open to receive the response.
3. The Front End forwards the request to the Primary Replica Manager.
4. The Primary Replica Manager stores the request in a buffer.
5. Using FIFO to process the request, the Primary Replica Manager forwards every request to its own Replica and other Replica Managers.
6. Every Replica Manager has it own Replica which after receiving the requests processes it accordingly, sends it to the appropriate server.
7. The associated server then performs the action accordingly and then sends the response to the Replica which in turn sends the response back to the Replica Manager.
8. The Primary Replica Manager receives response from other Replica Managers.
9. The final response is forwarded to the Front End.
10. The Front End Send the final response to the Client.

RELIABLE UDP

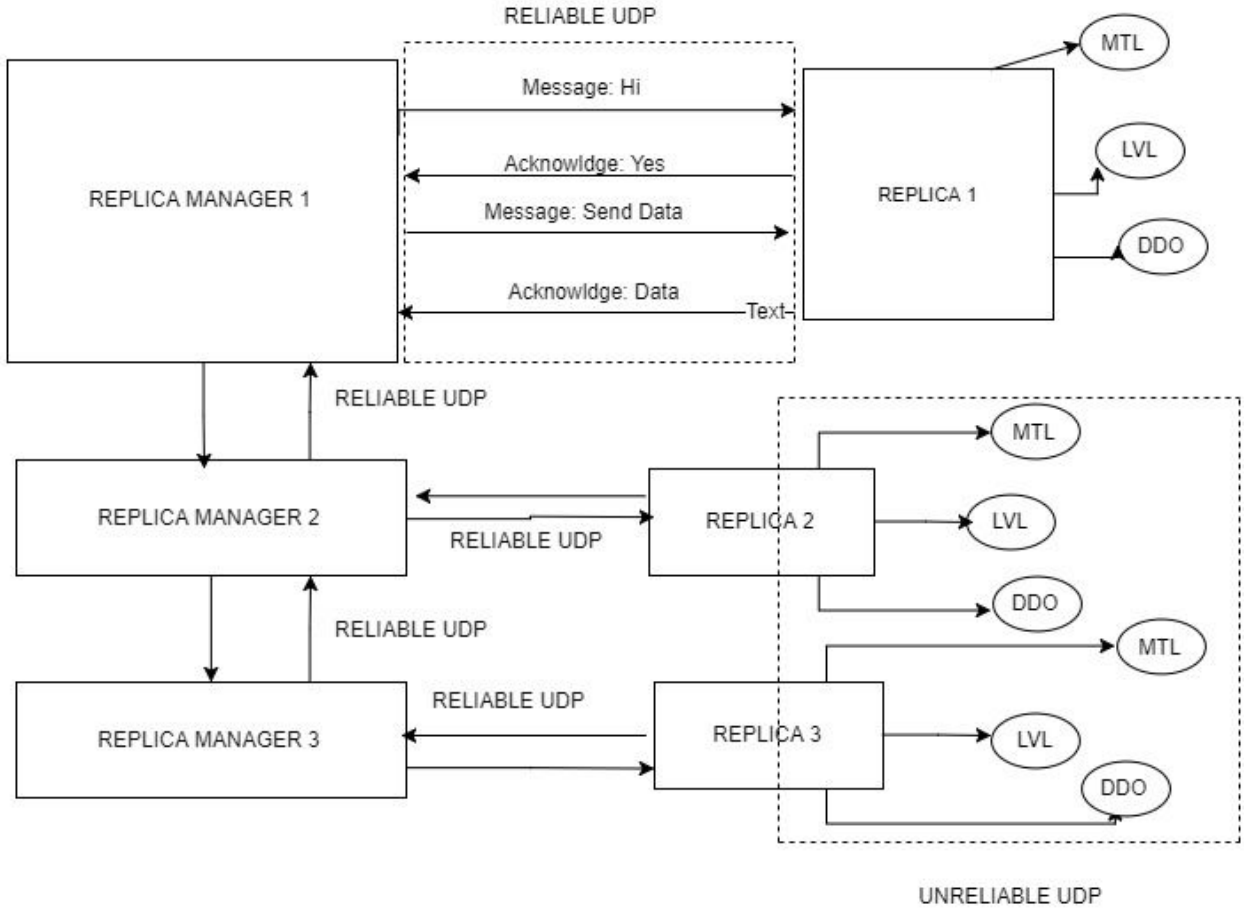

Fig 3 Reliable UDP

1. The Replica Manager communicates with the Replica through reliable UDP.
2. The Replica Manager sends the message to the Replica saying “Hi”, to build the connection.
3. The replica respond back with acknowledgment saying that it's ready for the request.
4. The replica manager then sends the request to the Replica.
5. The replica finally sends the acknowledgement data response back to the Replica Manager.
6. All of the Replica managers and replica are connected through reliable UDP.
7. All the replica managers are connected to the Server with unreliable UDP
8. The client sends a request to the CORBA Front End.
9. The Front End then forwards the request to the Primary Replica Manager.
10. The Primary communicates with the Replica which is connected to the Servers, the request is forwarded in this manner.
11. After the request is processed it is sent to the Front End in the same manner, from Servers to the Replica, Replica to the Replica Manager and then to the Front End and finally to the client.

HEART BEAT

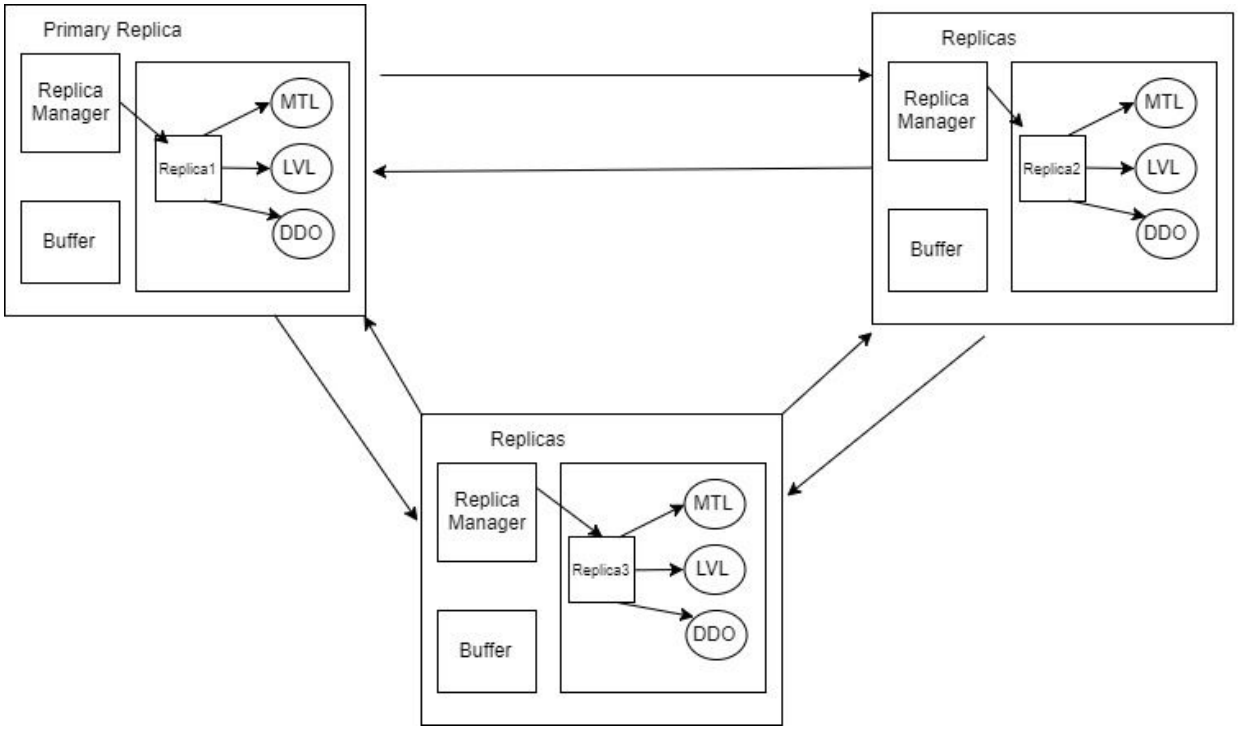

Fig 3 Heartbeat

1. All the Replica Manager communicates with their Replica through reliable UDP.
2. Now in order to check whether the Replicas are alive, The replica 1 send its alive message in every 5 seconds to the Replica Manager 2 and 3. If Replica Manager 2 or 3 does not receives alive message for more than 5 seconds from replica 1 then it communicates with Replica Manager 1 informing it about the replica 1 failure.
3. Now the Replica Manager 1 will reinstantiate its replica 1.
4. The same process takes place for all the replicas.
5. In this manner, all the Replica Managers keep a check on their replicas and if a process fails it is detected immediately.

FIFO

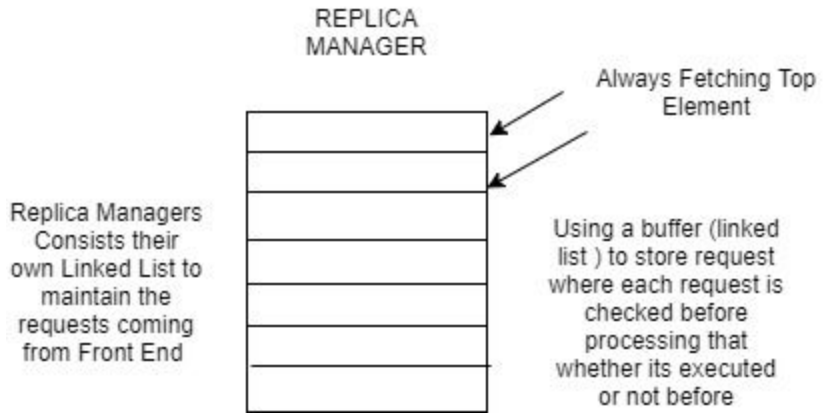

Fig 4. FIFO

1. The clients send request to the CORBA Front End.
2. Now Front End sends them to the Primary Replica Manager.
3. The Primary Replica Manager then stores them in a buffer(Linked list).
4. Replica Managers has their own Linked List buffer to maintain the Requests coming from the Front End.
5. When the Replica Manager gets a response for the current request from its replica as well as the other Replica Managers, then it forwards the next request.
6. In case of the Primary Replica manager's Replica fails, a new Leader is elected, and as long as it restores its own replica, the incoming requests are stored in its buffer.

DATA STRUCTURE:

1. All the records in a center server are stored in a hash map:

   HashMap<Character, LinkedList<Record>> hashMapMTL = new HashMap<Character, LinkedList<Record>>()

   The hash map looks like the following:

   <Key> C: (“Zac”, “Chen”, “de maisonneve”, “5147719184”, “chinese”, “mtl”, “TR1111”)

2. Record: Stores all the record. It has the following variables:

   - String m_ID;
   - StudentRecord m_student;
   - TeacherRecord m_Teacher;

3. Studentrecord: Consists information about a student. It has the following fields:

   - First name
   - Last nameCoursesRegistered (maths/french/science, note that student could be registered for multiple courses)
   - Status (active/inactive)
   - Status Date (date when student became active (if status is active) or date when student became inactive (if status is inactive))

4. TeacherRecord: Consists information about a teacher. It has the following fields:

   - First name
   - Last name
   - Address
   - Phone
   - Specialization (e.g. french, maths, etc)
   - Location (mtl, lvl, ddo)

5. Buffer: LinkedList<String>
The buffer stores the incoming requests coming into the Replica Manager. IN case of a replica failure, the replica needs to be rebooted and restored back to its previous state. After the replica is re-instantiated, it is restored back to its previous state using the requests stored in the buffer.

TEST SCENARIOS

Case 1: Primary RM is working, two backup RMs are working.
Result: All the requests and responses are delivered properly.

Case 2: Primary RM fails while two other RMs are working properly.
Result: The system will elect a new leader RM among other two backup RMs by selecting the RM with the next ID.

Case 3: Primary RM is working properly, however one of the backup RMs fails.
Result: The respective RM will restart its replica and restore it back to its previous state.

Case 4: Incorrect Manager ID or incorrect request.
Result: The system does not respond to the incorrect request and informs the client that the request is incorrect.

RESPONSIBILITIES

AKRITI SAINI: 40039689
Modules: Test Scenarios, Data Structure, FIFO, Reliable UDP

SHREYA BISWAS: 40018190
Modules: Requirements, Architecture Design, Workflow, Integration, Heartbeat implementation, Electing new Leader

SHUAI CHEN: 40010960
Modules: Workflow, Test Scenarios, Description, Integration, Front End Design and implementation , Message Passing Design and implementation, Electing Leader.