PROJECT

Distributed Library Management System (DLMS)

Mostafa Marandi | 40085773
Yi Zhang | 27788584
Divyansh Thakar | 40086580

Introduction

As Distributed System should handle **Fault Tolerance** and it should be **highly available**, the project is to make **Distributed Library Management System (DLMS)** in such a way that new DLMS tolerate single **software (non-malicious Byzantine) failure** and highly available under a single process **crash failure** using **active replication**. The system must consist of three different Replicas implementation as well as three new modules Front End, Sequencer and Replica Manager in order to achieve this functionality.

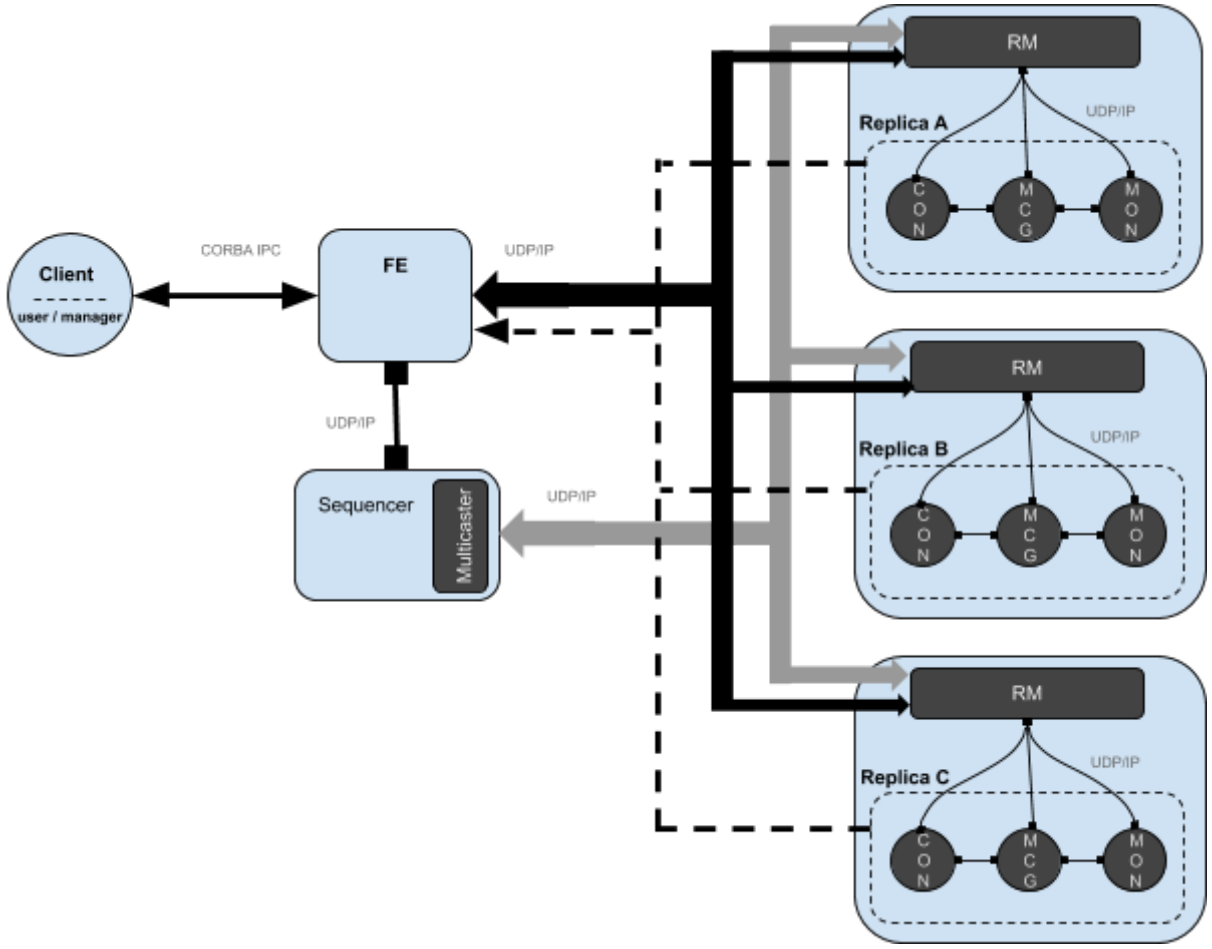
Objectives

- To implement software failure and crash failure tolerant CORBA Distributed Library Management System.
- To implement a CORBA Front End (FE) which receives client request and forwards it to a sequencer.
- To implement a Sequencer which assigns unique sequence number and use reliable FIFO UDP multicast subsystem.
- To implement the Replica Manager (RM) which is actively replicated server subsystem.
- System must concurrent
- Use multithreading client-server communication

System Requirements

- The system should be able to handle crash failure or software (non-malicious Byzantine) failure using Active Replication mechanism and failure type will be specified at FE initialization.
- All communication must be handled with UDP/IP protocol except the connection between client and front end which is CORBA IPC.
- All UDP communication must be reliable, so using 3 times sending mechanism can be considered.
- For software failure, if FE has informed RMs about receiving incorrect response from specific replica and this happened three times then the RMs must replace the faulty replica with new one.
- For crash failure, if FE has informed RMs about not receiving a message from a specific replica in a normal period of time then the RMs must be sure that the replica is truly crashed and after that replaced the faulty replica with new one.

Architectural design



Protocol and Algorithm

Communication between client and FE is achieved by using CORBA, and all other communications are achieved by using UDP/IP.

- **Total Order Algorithm**
  Sequencer assign unique id to each request, after RM receive the request, it first check for duplication, since we send message three times to achieve reliability. Then it compare received message id with expected id, if the two ids match, the message will be processed, otherwise, the request message will be put into a FIFO queue. Once the expected id number increase, the FIFO queue will be checked for matching request id.
- **Replica Replication/ Recover Algorithm**
  Replica recover is achieved by creating new servers for three libraries. Then data is updated by reading request records in the corresponding RM log and process these requests. Once recover finish, the replica is ready for executing new requests.

Detailed System Design

- **Front End**
  - Communication Module**:** This module is responsible for receiving client request within CORBA IPC and create new message upon the received request and send it throw UDP protocol to Sequencer, and also receive the final server UDP response. In this module the reply received from all 3 replicas is compared and give reply to the client which has majority.
  - Fault Detector Module: This module is responsible for detecting server crash failure or software failure and next informs RMs about the existence of the failure occurrence. It will check if one replica sends three incorrect results consecutively or do not respond in normal response time, then it detects the fault and tells replica manager that this replica is sending incorrect result.
- **Sequencer & Multicaster**
  - Sequencer Module**:** This module assign an incremental id to each received request and increase the sequence after each successful message send and then pass the message to the multicaster.
  - Multicaster Module: Multicaster will send the same message to all Replicas by UDP/IP which uses IP+Port number to uniquely identify each server in replicas. So in this architecture we use three different IP for each replica and use same port for same server in each replica ex. replica A IP:6060 for replica A CON server and replica B IP:6060 for replica B CON server ...
- **Replica Manager**
  - Fault Detector Module: There are two situations for replication: 1. for software failure, RM has the count to track failure times, once the count reaches three, RM will recover the failed replica; 2. for crash failure, if RM receive the "[cf]" tag, which means crash failure is detected, other RMs will send message to the crashed server to make sure its situation, and tell the result to the RM of the crashed replica. If it is agreed among the three RM that the replica really crashed, then the targeted RM will recover its replica.
  - Replication Module: If RMs decide that a specific replica must be replaced with new one, then this module is responsible for creating a new updated replica using replica recover algorithm described above.
  - Total Order Checker Module: Each replica manager must be sure that it received all the client request and also send requests to servers with specified order. The total order is maintained by following the total order algorithm explained above.
  - Receive and Send Message Module: RM receives messages from sequencer (request message), FE (failure message for both software failure and crash failure) and other RMs (check message for crash failure). RM is responsible for differentiate these messages and process them accordingly. For request

message, RM sends them to correct server based on user/manager ID by calling *sendMessage* function. *sendPingMessage* method is called in crash failure situation for checking whether the replica really crashed.

- **Replica**
  - Message Adaptor Module**:** As each replica used its own UDP messaging structure, this module is a requirement to convert shared FE UDP message to the replica specific UDP message
  - Server Cluster Module**:** This module consists of three library servers and run them concurrently which is equivalent to the system which is developed in previous assignments, however here the client request received in UDP. As soon as the request is processed, the produced response will be sent to FE.

Test Scenarios

- **Test *addItem* operation**

```
public static void tesAddItem() {
    DlmsClient MCGM1111 = new DlmsClient("MCGM1111", orbInfo);
    DlmsClient MONM1111 = new DlmsClient("MONM1111", orbInfo);
    MCGM1111.addItem("FRONTEND", "MCG6231", "DS", 3);
    MCGM1111.addItem("FRONTEND", "MCG6441", "AS", 1);
    MONM1111.addItem("FRONTEND", "MON6441", "DS", 1);
}
```

Result:

```
MCGM1111 > [succeed] Add new record succeed! MCGM1111 MCG6231 DS 3
MCGM1111 > [succeed] Add new record succeed! MCGM1111 MCG6441 AS 1
MONM1111 > MON server response = [succeed] the quantity of item with id (MON6441) and name (DS) increased by 1. Total quantity is 1
```

- **Test *listItemAvailability* operation**

```
public static void tesListItem() {
    DlmsClient MCGM1111 = new DlmsClient("MCGM1111", orbInfo);
    MCGM1111.listItemAvailability("FRONTEND");
}
```

Result:

```
MCGM1111 >
MCG6441 AS 1
MCG6231 DS 3[succeed]
```

- **Test *removeItem* operation**

```
public static void tesRemoveItem() {
    DlmsClient MCGM1111 = new DlmsClient("MCGM1111", orbInfo);
    MCGM1111.removeItem("FRONTEND", "MCG6231", 1);
    MCGM1111.listItemAvailability("FRONTEND");
}
```

Result:

```
MCGM1111 > [succeed] remove item success! MCGM1111 MCG6231 current quantity 2
MCGM1111 >
MCG6441 AS 1
MCG6231 DS 2[succeed]
```

- **Test *findItem* operation**

```
public static void tesFindItem() {
    DlmsClient MCGU0000 = new DlmsClient("MCGU0000", orbInfo);
    MCGU0000.findItem("FRONTEND", "DS");
}
```

Result:

```
MCGU0000 > MCG server response = [succeed] the list of items with the same name (DS) is provided
MCG6231,2
MON6441,1 .
```

- **Test *borrowItem* operation**

```
public static void tesBorrowItem() {
    DlmsClient MCGU0000 = new DlmsClient("MCGU0000", orbInfo);
    DlmsClient MCGM1111 = new DlmsClient("MCGM1111", orbInfo);
    MCGU0000.borrowItem("FRONTEND", "MCG6231", 1);
    MCGM1111.listItemAvailability("FRONTEND");
}
```

Result:

```
MCGU0000 > MCG server response = [succeed] the item with id (MCG6231) successfully borrowed
MCGM1111 >
MCG6441 AS 1
MCG6231 DS 1[succeed]
```

- **Test *exchangeItem* operation**

```
public static void tesExchangeItem() {
    DlmsClient MCGU0000 = new DlmsClient("MCGU0000", orbInfo);
    DlmsClient MCGM1111 = new DlmsClient("MCGM1111", orbInfo);
    MCGU0000.exchangeItem("FRONTEND", "MCG6441", "MCG6231");
    MCGM1111.listItemAvailability("FRONTEND");
}
```

Result:

```
MCGU0000 > [succeed] ExchangeItem Succeed
MCGM1111 >
MCG6441 AS 0
MCG6231 DS 2[succeed]
```

- **Test *returnItem* operation**

```
public static void tesReturnItem() {
    DlmsClient MCGU0000 = new DlmsClient("MCGU0000", orbInfo);
    DlmsClient MCGM1111 = new DlmsClient("MCGM1111", orbInfo);
    MCGU0000.returnItem("FRONTEND", "MCG6441");
    MCGM1111.listItemAvailability("FRONTEND");
}
```

Result:

```
MCGU0000 > [succeed] Return Item Succeed
MCGM1111 > MCG server response = [succeed] list of all items availability is provided:
MCG6441,AS,1
MCG6231,DS,2
```

- **Test atomicity of *exchangeItem* operation**
    - **Case 1: User tries already borrowed a book from other server and tries to exchange the same book from the same server.**

```
public static void tesExchangeAtomicityItem() {
    DlmsClient MCGM1111 = new DlmsClient("MCGM1111", orbInfo);
    DlmsClient MONM1111 = new DlmsClient("MONM1111", orbInfo);
    DlmsClient MONU0000 = new DlmsClient("MONU0000", orbInfo);
    MCGM1111.addItem("FRONTEND", "MCG6111", "DS1", 1);
    MCGM1111.addItem("FRONTEND", "MCG6112", "DS2", 1);
    MCGM1111.addItem("FRONTEND", "MCG6113", "DS3", 0);
    MONM1111.addItem("FRONTEND", "MON6111", "DS4", 1);

    MONU0000.borrowItem("FRONTEND", "MCG6111", 1);
    System.out.println("case 1 : exchange succeed" );
    MONU0000.exchangeItem("FRONTEND", "MCG6112", "MCG6111");
    MCGM1111.listItemAvailability("FRONTEND");
```

Result:

```
MCGM1111 > [succeed] Add new record succeed! MCGM1111 MCG6111 DS1 1
MCGM1111 > MCG server response = [succeed] the quantity of item with id (MCG6112) and name (DS2) increased by 1. Total quantity is 1
MCGM1111 > [succeed] Add new record succeed! MCGM1111 MCG6113 DS3 0
MONM1111 > [succeed] Add new record succeed! MONM1111 MON6111 DS4 1
MONU0000 > MON server response = MCG server response = [succeed] the item with id (MCG6111) successfully borrowed .
case 1 : exchange succeed
MONU0000 > [succeed] ExchangeItem Succeed
MCGM1111 > MCG server response = [succeed] list of all items availability is provided:
MCG6113,DS3,0
MCG6111,DS1,1
MCG6112,DS2,0
```

    - **Case 2: User again tries to exchange the book from other server which is not available so atomicity fails.**

```
System.out.println("case 2 : exchange failed because of unavailability" );
MONU0000.exchangeItem("FRONTEND", "MCG6113", "MCG6112");
MCGM1111.listItemAvailability("FRONTEND");
```

Result:

```
case 2 : exchange failed because of unavailability
MONU0000 > [failed] Fail exchange MCG6113 with MCG6112
MCGM1111 >
MCG6113 DS3 0
MCG6111 DS1 1
MCG6112 DS2 0[succeed]
```

- **Case 3: User again tries to exchange with the unborrowed book so atomicity fails.**

```
System.out.println("case 3 : exchange failed because of not yet borrowed book" );
MONU0000.exchangeItem("FRONTEND", "MCG6112", "MCG6111");
MCGM1111.listItemAvailability("FRONTEND");
```

Result:

```
case 3 : exchange failed because of not yet borrowed book
MONU0000 > [failed] ExchangeItem Failure
MCGM1111 >
MCG6113 DS3 0
MCG6111 DS1 1
MCG6112 DS2 0[succeed]
```

- **Case 4: User already borrowed a book from other server and tries to exchange a book from the same server.**

```
MONU0000.borrowItem("FRONTEND", "MON6111", 1);
System.out.println("case 4 : exchange failed because user can borrow only one book
MONU0000.exchangeItem("FRONTEND", "MCG6111", "MON6111");
MCGM1111.listItemAvailability("FRONTEND");
```

Result:

```
MONU0000 > [succeed] borrow succeed
case 4 : exchange failed because user can borrow only one book from other servers
MONU0000 > [failed] ExchangeItem Failure
MCGM1111 >
MCG6113 DS3 0
MCG6111 DS1 1
MCG6112 DS2 0[succeed]
```

- **Test software failure tolerance**

In software failure, RM will keep a count for how many times it happens. When the count reaches 3, RM will call recover method and replace the failed replica with a new one and update data to keep consistency.

Result:

```
received message is: 1_RM1_CON_[sf]
Receive [sf]: 1_RM1_CON_[sf]
count is: 1
received message is: 5_132.205.95.109:57444_addItem_CONM1111@CON6231@DS@2@_
processing request message...: 5_132.205.95.109:57444_addItem_CONM1111@CON6231@DS@2@_
Answer is: 5_[failed]
received message is: 2_RM1_CON_[sf]
Receive [sf]: 2_RM1_CON_[sf]
count is: 2
received message is: 6_132.205.95.109:57448_addItem_CONM1111@CON6231@DS@3@_
processing request message...: 6_132.205.95.109:57448_addItem_CONM1111@CON6231@DS@3@_
Answer is: 6_[failed]
received message is: 3_RM1_CON_[sf]
Receive [sf]: 3_RM1_CON_[sf]
count is: 3
```

Start replacement process after the count reaches 3.

```
count is: 3
Start recover
concordia server para: true
Socket CON: socket closed
Socket MON: socket closed
Socket MCG: socket closed
SOCKET ARE CLOSED
initialize con server
initialize mcg server
initialize mon server
SERVER ARE INITIALIZED
Start update data ...
MON Server ready and waiting ...
MCG Server ready and waiting ...
CON Server ready and waiting ...
Start update RM1 ...
processing request message...: 1_localhost:9999_addItem_MCGM1111@MCG6231@DS@3@
Answer is: 1_[succeed] Add new record succeed! MCGM1111 MCG6231 DS 3
processing request message...: 2_localhost:9999_addItem_MCGM1111@MCG6441@AS@2@
Answer is: 2_[succeed] Add new record succeed! MCGM1111 MCG6441 AS 2
processing request message...: 3_localhost:9999_addItem_MONM1111@MON6441@DS@1@
Answer is: 3_[succeed] Add new record succeed! MONM1111 MON6441 DS 1
processing request message...: 4_localhost:9999_addItem_CONM1111@CON6231@DS@1@
Answer is: 4_[succeed] Add new record succeed! CONM1111 CON6231 DS 1
processing request message...: 5_localhost:9999_addItem_CONM1111@CON6231@DS@2@
Answer is: 5_[succeed] Add item succeed! CONM1111 CON6231 DS 2
processing request message...: 6_localhost:9999_addItem_CONM1111@CON6231@DS@3@
Answer is: 6_[succeed] Add item succeed! CONM1111 CON6231 DS 3
Finish update data ...
```

- **Test crash failure tolerance**

In software failure, the RM will receive a message containing the tag "[cf]", then a string list *cfAck* is used to receive replies from other RM, if all three RMs agree that the replica really crashed, the replica's RM will call recover method and replace the failed replica with a new one and update data to keep consistency.

Result:

```
received message is: 1_RM1_MON_[cf]
Receive [cf]: 1_RM1_MON_[cf]
interrupt...
Socket: socket closed
recoverMessage: 0_localhost:4444_listItemAvailability_MONM1111_
received message is: 2_[cfack]
cfAck.size() == 2
request from: /132.205.95.109 : 56860
received message is: 3_[cfack]
cfAck.size() == 3
request from: /132.205.95.108 : 54778
cfAck.size() == 3
Start recover
```

Start recovery process after receiving three acknowledgements from other RMs.

```
Start recover
concordia server para: true
Socket CON: socket closed
Socket MCG: socket closed
SOCKET ARE CLOSED
initialize con server
initialize mcg server
initialize mon server
SERVER ARE INITIALIZED
MON Server ready and waiting ...
MCG Server ready and waiting ...
CON Server ready and waiting ...
Start update data ...
Start update RM1 ...
processing request message...: 1_localhost:9999_addItem_MCGM1111@MCG6231@DS@1@
Answer is: 1_[succeed] Add new record succeed! MCGM1111 MCG6231 DS 1
processing request message...: 2_localhost:9999_addItem_MONM1111@MON6231@DS@1@
Answer is: 2_[succeed] Add new record succeed! MONM1111 MON6231 DS 1
processing request message...: 1000_localhost:9999_MON_[crash]
processing request message...: 3_localhost:9999_addItem_MONM1111@MON62442@AS@1@
Answer is: 3_[succeed] Add new record succeed! MONM1111 MON62442 AS 1
Finish update data ...
RM1 ROCOVER
```

Responsibilities

- **Front-end:** Mostafa Marandi (40085773)
- **Replica manager:** Yi Zhang (27788584)
- **Failure-free sequencer:** Divyansh Thakar (40086580)

References

[1] https://en.wikipedia.org/

[2] Text book and the slides from Prof. M.L. Liu, California Polytechnic State University

[3] Lab tutorials and slides