UNIVERSITÉ
Concordia
UNIVERSITY

COMP 6231

DISTRIBUTED SYSTEMS DESIGN

Summer 2016

Instructor: Sukhjinder K. Narula

Project: Highly Available CORBA Distributed Staff Management System

Team members:

Yang Dong
Huaqiang Kang
Meng Yao
Xingjian Zhang

CONTENT TABLE

Design and Architecture

The primary goal of the distributed stuff management system is to tolerate benign process failure. The following techniques are used to achieve this:

- Replicated Servers
- A failure detection subsystem
- Subsystems (Leader Election) to restore system integrity after a server failure.

However the system performance could be affected during critical periods related to process failure and initialization.

Overview

The client and the front end communicate using CORBA. The front end and the leader communicate using UDP. The client will send request to front end and front end will send the request to leader process. The leader process will execute the operation in its own host, also will broadcast the client request atomically to all the servers replicas, receive the responses back from the servers, compares the results and sends a single response back to the client. The leader process and the front end communicate using the reliable UDP protocol. The unreliable UDP protocol has to be made reliable using some approaches. We would be using the Acknowledgement technique to achieve this:

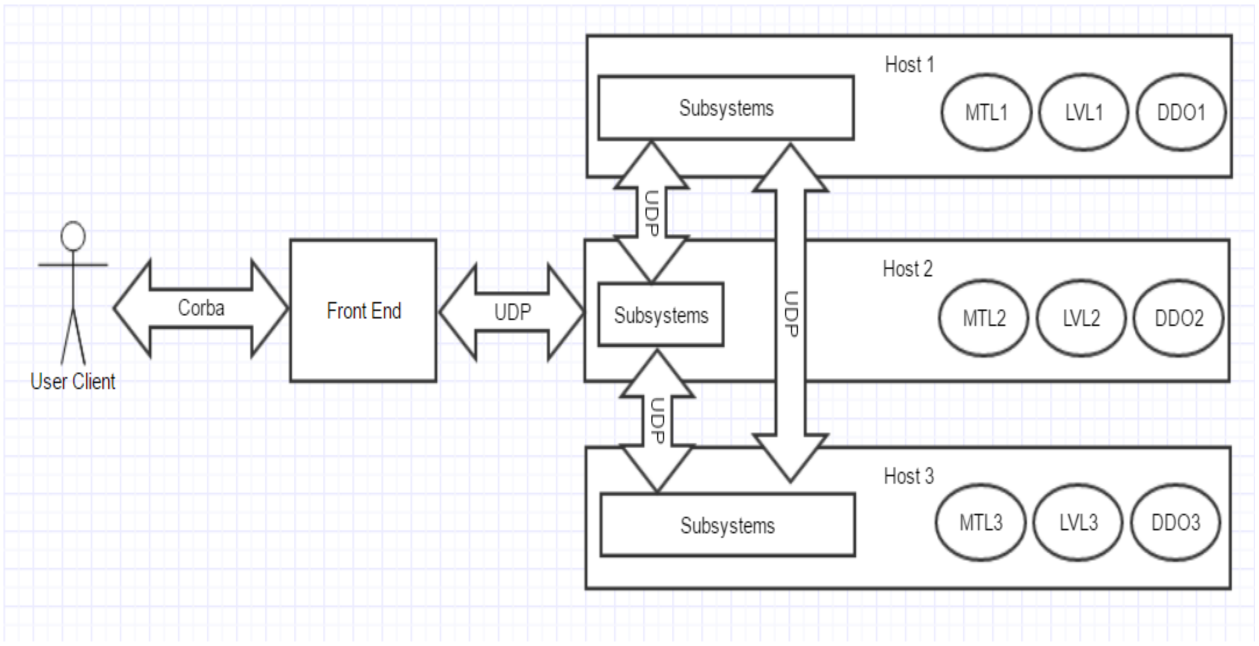


Figure 1 overview of DSMS system architecture

Subsystems explanation

This comprises of the following mechanisms:

- Replicated Servers
- Reliable FIFO broadcast subsystem.
- Leader election subsystem.
- Failure detection subsystem.

These subsystems will be a part of each replica server process. The failure detection subsystem residing in each host will periodically check with each other for the host availability and remove the failed server process. It will trigger the leader election subsystem in case of group leader process failure.

The FIFO broadcast mechanism (IP multicast) residing in the leader process will forward the request to other server processes in the group whenever it will receive the request from the Front End.

Use case diagram

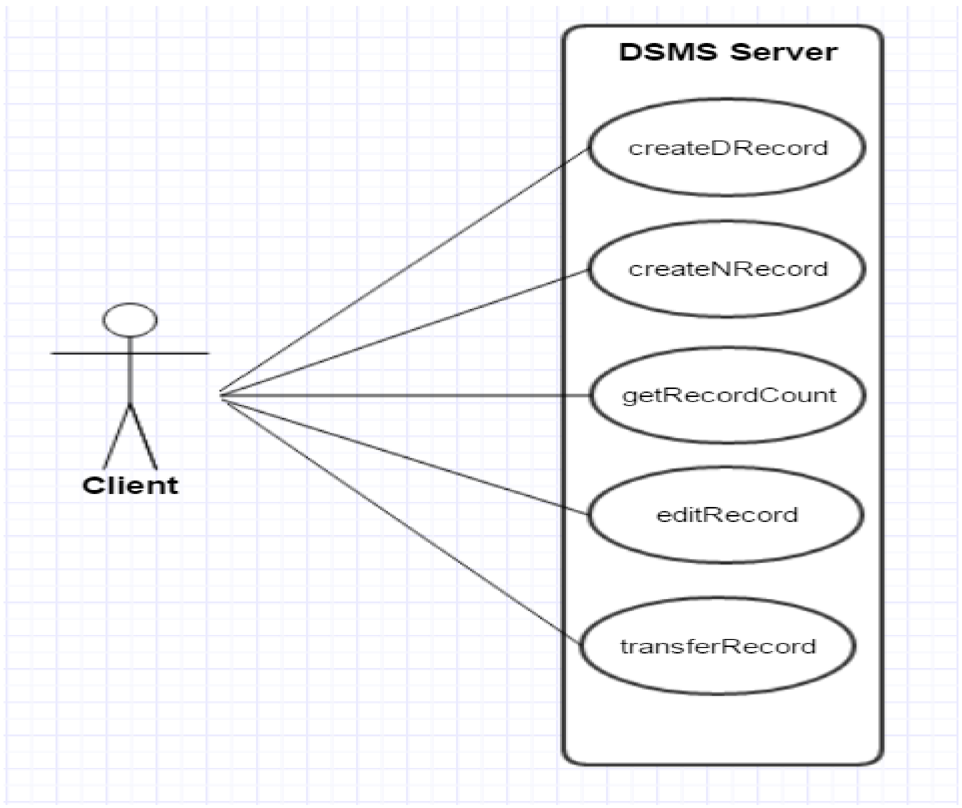


Figure 2 Use case of DSMS

Each client can send requests to server and receives the result without knowing in which server or host the processes have been performed. So the system will be transparent from the client view.

The operations that a client can ask for are:

- Create a Doctor/Nurse record for any clinic with an unique ID.
- Edit an existing record and update its status.
- Get the records counts for each clinic.
- Transfer an existing record from one clinic to another

System Sequence

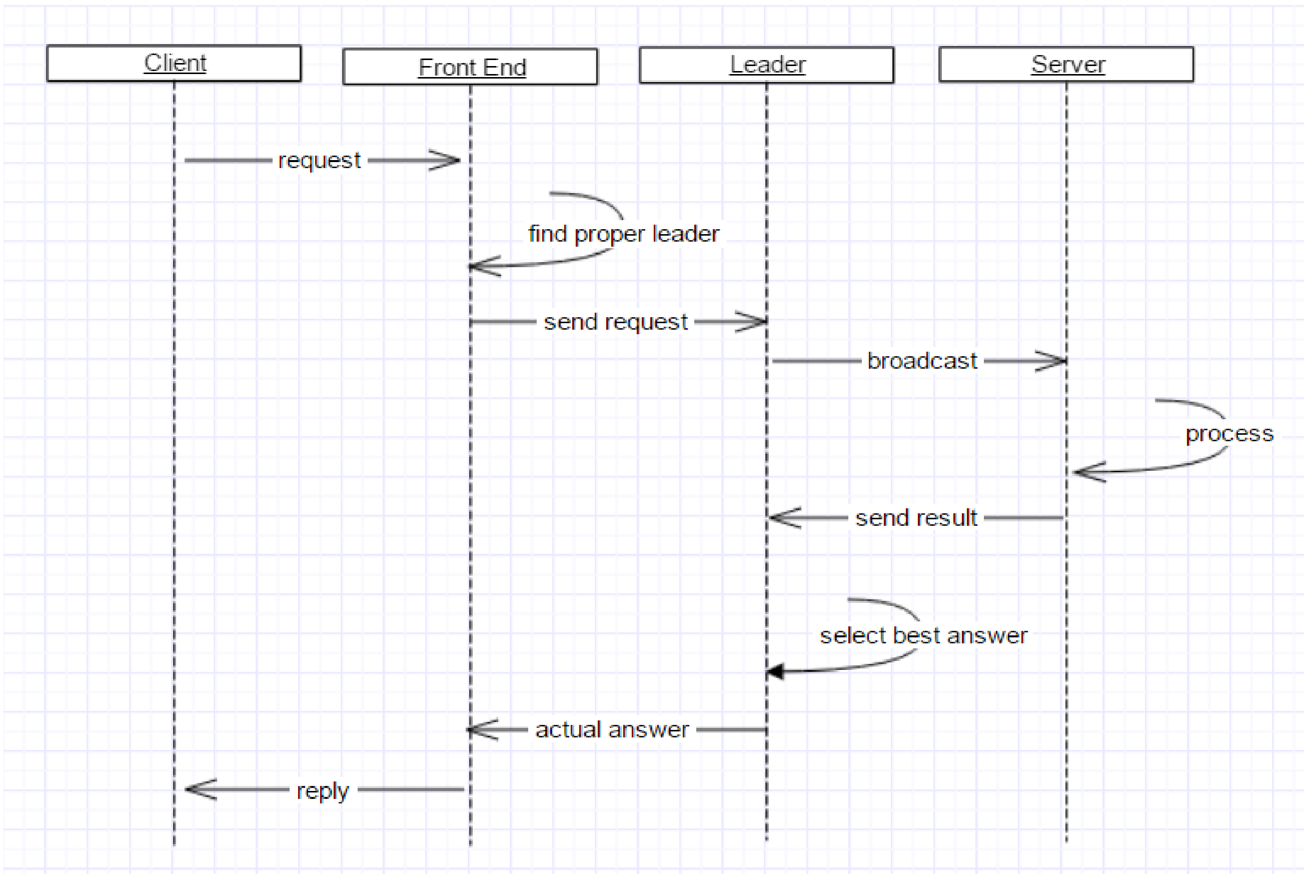
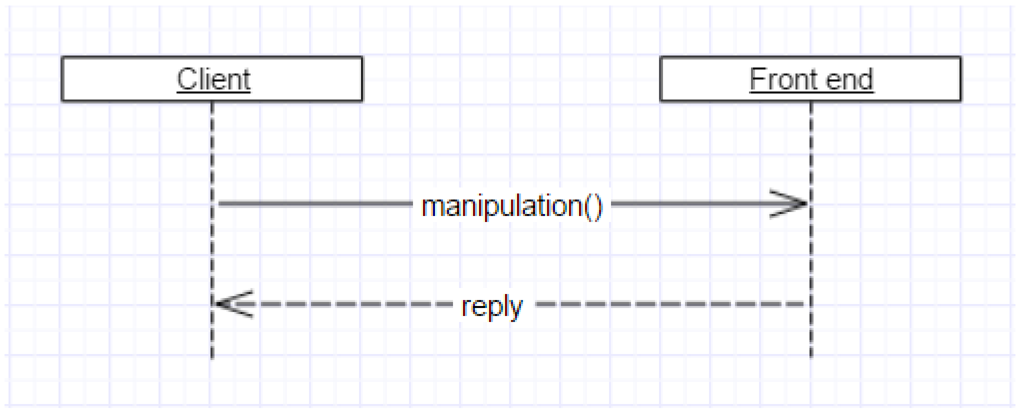


Figure 3 Sequence diagram of DSMS

Client invokes a remote object through the front end; Client and front end communicate with each other through CORBA. Actually, Front end is the entry point of the server.

Detailed Functional Implementation

All manipulations will send by clients to servers through the front end just as diagram shows



:

1. Creating a Doctor/Nurse Record:

When a client sends a request to create a Doctor or Nurse record for one of the clinics through the front end. Front end sends the request to right leader and waits for the reply. After receiving the request response, the front end sends result to the Client.

2. Get the record counts.

When a client sends a request to retrieve the count of records from one of the clinics through the front end, front end sends the request to right leader and waits for the reply. After receiving the request response, the front end sends result to the Client.

3. Edit an existing record.

When a client sends a request to edit a record for one of the clinics through the front end, front end sends the request to right leader and waits for the reply. After receiving the request response, the front end sends result to the Client.

4. Transfer a record from one clinic to another.

When a client sends a request to transfer a record for one of the clinics through the front end, front end sends the request to right leader and waits for the reply. After receiving the request response, the front end sends result to the Client.

Limitations and Assumptions during implementation

Assumptions

Before designing the distributed systems we have made certain assumptions that are out of the scope of this project.

1. The CORBA architecture will not fail.

   The whole system works on the CORBA architecture and if it fails the application will not work correctly.

2. Processor failures are benign.

   We need to design a system which handles process crashes only.

3. No partition network failure.

   We cannot distinguish between the broken network and server crash and this is an assumption in the project description. There will be no UDP message loss during the subsystem communication is going on. Thus the test case for starting the election subsystem due to a UDP packet loss will not be handled.

4. The client request that comes in when a current leader crashes is not buffered.

   The client request that appears during the time interval between a current leader crash and the election of a new leader will not appear again in the system, i.e. it will be lost.

Limitation

When the leader crashes the processes start electing to choose an alternative leader. During this period every request, which is sent to group, will be failed, because the FE sends the request only to leader and during this time there is no leader in the group.

To solve this problem we can have a mechanism in the FE to send the request again at a later time when it receives notification that a new leader has been elected. So the system remains highly available.

Data Flow Description and Module Interaction

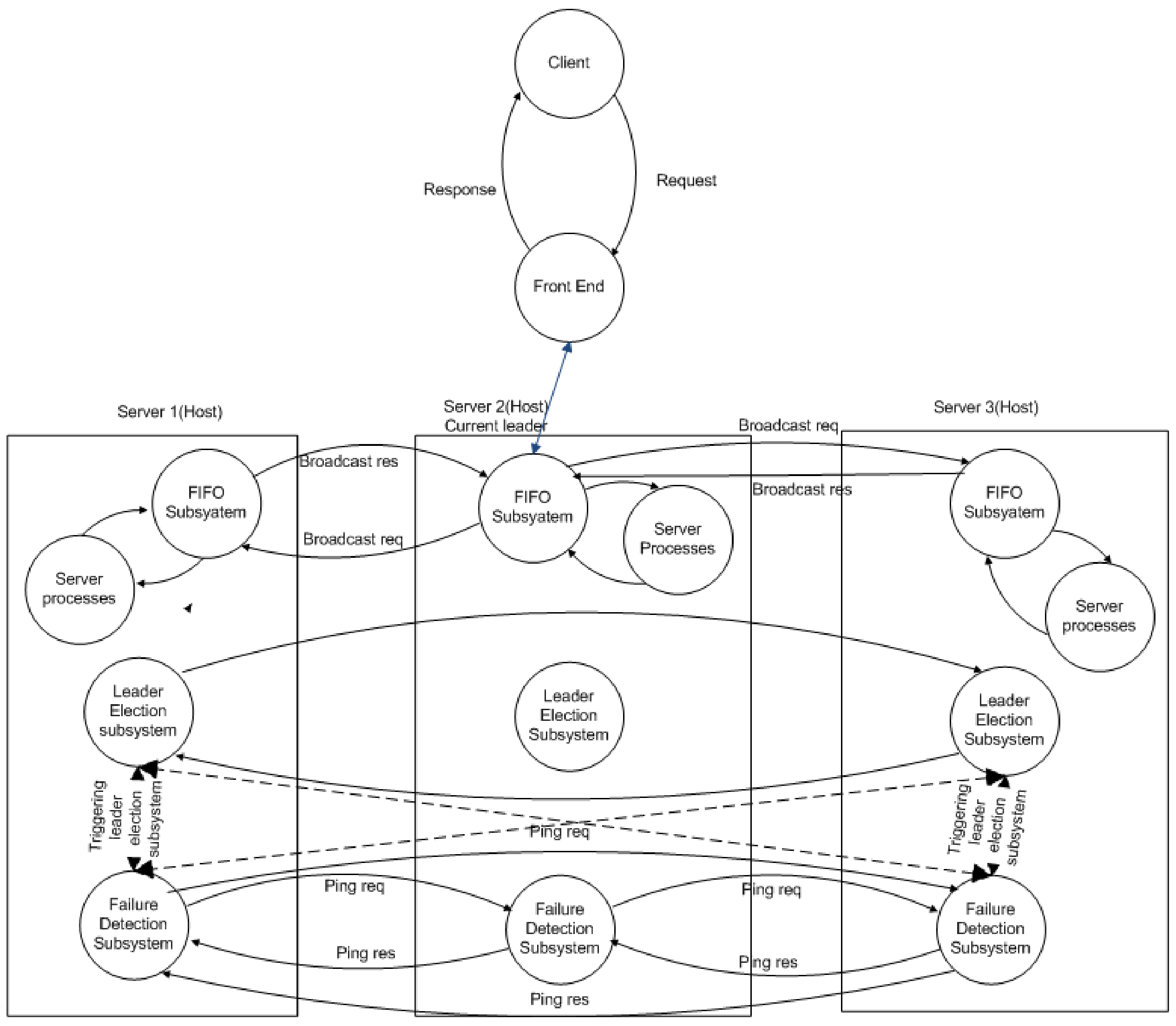


Figure 4 Data flow diagram of DSMS

Client

A client in this system represents an officer client who sends a request to the Front End and receives reply from the Front-End for the same request.

Front End

A client sends a request to the server through the front end. Front end has its own repository where station server names, contact information are stored. In addition to this the front end will also be storing the information about who is the current leader by maintaining the processID in the repository. After receiving the response, the front end sends result to the Client.

Actually front end communicates with client directly and it is a bridge between client and group leader. It receives the requests from clients by CORBA invocation and sends the request to the appropriate group leader through UDP.

Front end functionalities:

1. FE receives concurrent requests from the client including the manager ID and other parameters required to fulfill the operations.
2. The front end will generate a sequence ID for each new request and add this sequence ID as a key and parameters as values in the request buffer. Front end will forward the client request to the group leader process (FIFO broadcast subsystem who will order the concurrent requests by prioritizing the request operation) in the group and wait for the response.
3. The front end will be having the knowledge of the current leader from its repository and in case of group leader process failure the front end will be notified with new leader information by the leader election subsystem.
4. The front end will be communicating over the reliable UDP communication with the leader process with the help of ACK or NAK piggybacked in the response message. With this ACK the front end will match the request to the exact response thus using the asynchronous Request-Reply protocol.

Replica: (Individual Host)

It consists of server process for each station server and subsystems.

Server Processes

There are three servers, each has one replica processes. They all have the same functions. When it is decided to be the leader server, it will run the leader process, otherwise it runs the team member process which receives the requests from the host leader. The data will be handled in each replicated server and return the result to the leader server to get the valid result.

Reliable UDP FIFO broadcast subsystem

Principle and algorithms for a reliable broadcast:

Integrity, validity and agreement. There are two ways to achieve this. One is deploying the reliable broadcast algorithm based on basic broadcast. Another is broadcast based on IP and piggybacked acknowledgements.

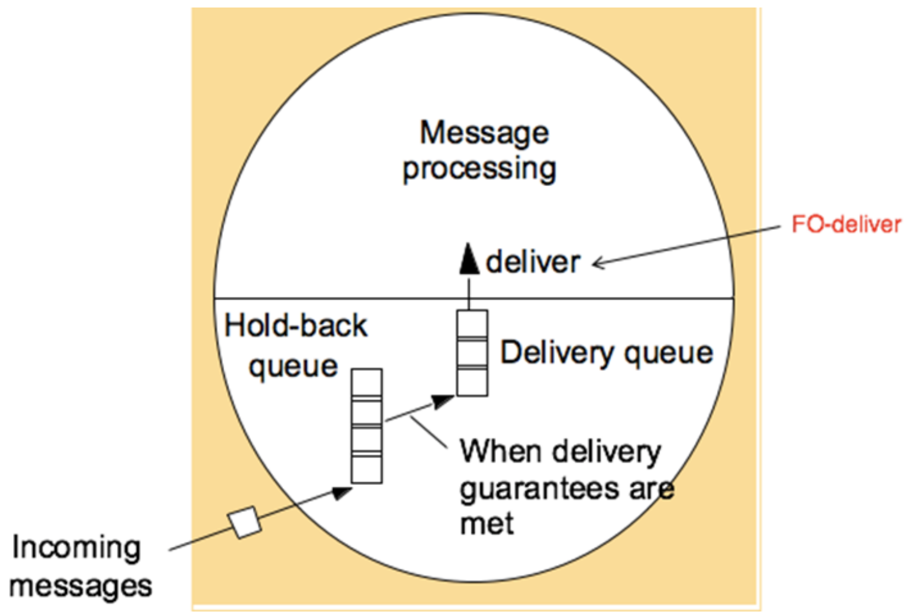
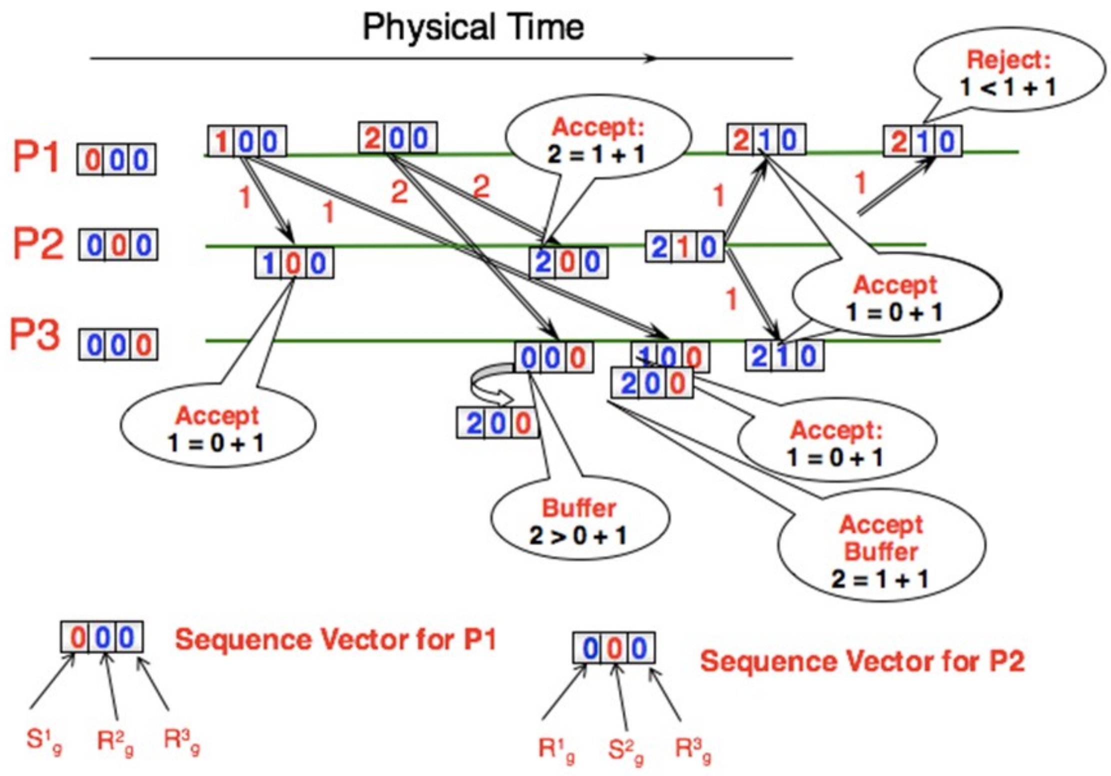


Figure 5 Message processing in FIFO

Sequence acknowledgements guarantee the FIFO ordering. Each process keeps a sequence number for each other process q for group g, as R^q_g. Each process p also keeps a sequence to record the messages it has sent to group g marked as S^p_g. This forms a victor<R^1_g R^2_g … $S^n_{g\ldots}$ R^{p-1}_g, R^p_g >. When one sender wants to send a message, it piggybacks the sequence and the latest R received from a sender q's acknowledgement in form of <q,R^q_g> onto the message.

When a message m sent from q arrived at process p, p checks whether $S = R^q_g +1$. If so, p FO-delivers m and increments R^q_g.

If $S > R^q_g +1$, p places the message in the hold-back queue until the intervening messages have been delivered and $S = R^q_g +1$. At the same time, a separate response message will be sent to sender or process q because of lost messages. This can be called a negative acknowledgement. If $S < R^q_g +1$, p drops the message because it has delivered that message.



Coding implementation:

1. Sequence and records

First, a sequence generator will be created for generating sequence number. It should be synchronous, protected by lock or mutex. This enables two concurrent threads of a process won't get the same sequence number when they enter the critical section at the same time.

```
Sender
UDPPkt = newMessage(Sequence,ack<lastRevPID,Rec[g][lastRevAdd]>,msg);
Broadcast(UDPPkt);
sequence++;


Receiver
senderAdd,senderSeq,ack<PID,Seq>,msg = unpack(UDPPkt);
if(msg == negACK) {
  broadcast(g,senderBuff[ack<Seq>+1])
  } else {
  if(senderSeq >= Rec[g][senderAdd]+1 ) {
      put(msg,senderAdd,senderSeq-Rec[g][senderAdd]);      //insert the message to the hold
back queue, the last arg is the offsite to the head of queue
      sequence = sequence + move(senderAdd);        // move ahead for hold back queue, move
returns the number of messages moving ahead
  }
  if(Rec[g][getAdd(PID)] < ack<Seq> or length(holdbackQueue) == threshold or timeoutEvent )
{                 //when S > Rpg+1 or R > Rqg a negACK to send
      sendNegACK(getAdd(PID),Rec[g][getAdd(PID)]);
  }
}
```

2. Hold-back queue

The hold –back queue will be implemented by ArrayList. ArrayList supports dynamic length arrays. When the size of ArrayList is shorter than required, the contents will be copied to a new longer array before insertion.

Failure Detection Subsystem

Goal: Find a failed server

Assumptions:

- There are three servers (Servers that are responsible for managers request)
- Only one failure happens in the system and it cannot be front end and leader election subsystem.

Algorithm design:

Every server is responsible for testing one other server and if that server fails it should report the failure to other two servers. In every time interval the node checks each other, this indicates the timeout for ping response. After timeout other servers will send ping request three times and if it fails to do so they declare about the failure of that node.

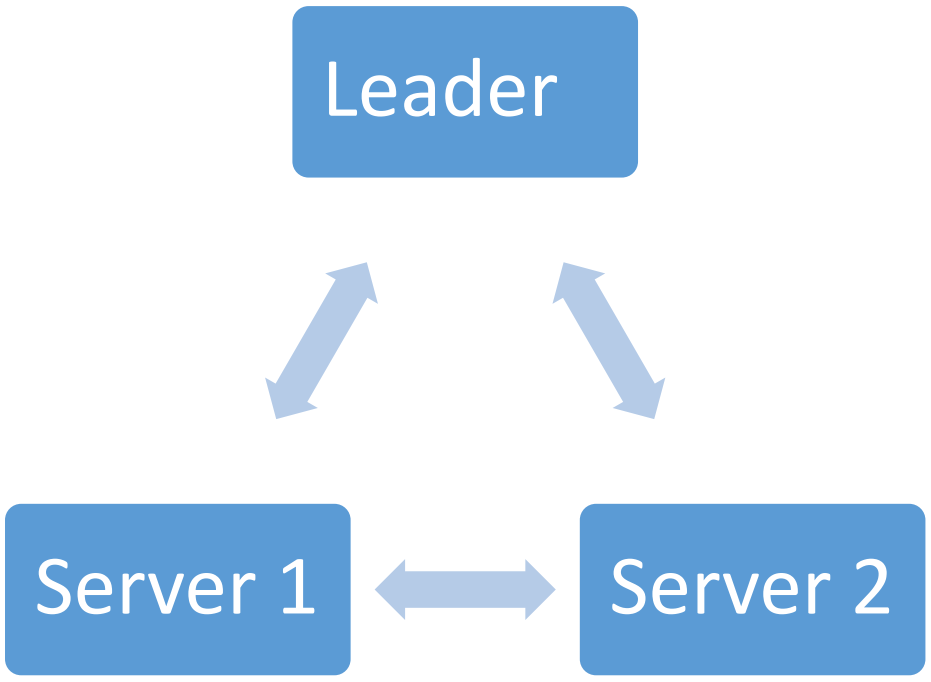


Figure 6 failure detection subsystem view

If the failed server is leader, then any other server node will start the leader election subsystem module. The checking process of each node is done through UDP connection with empty message and acknowledgement.

Leader Election Subsystem

The bully algorithm is a method in distributed computing for dynamically electing a coordinator by process ID number. The process with the highest process ID number is selected as the coordinator. Assumptions for Bully Algorithm:

- The system is synchronous and uses timeout for identifying process failure
- Allow processes to crash during execution of algorithm
- Message delivery between processes should be reliable
- Prior information about other process id's must be known

Algorithm Design:

1. Leader initialization

According to the assumption before, the subsystem inside every host will gather all the process IDs after setting up every host. Comparing each of the IDs, the subsystem will then select the one with highest process ID to become the coordinator (leader process).

2. Leader election using Bully Algorithm

When a process (one of non-leader host) noticed the leader crash through failure detection subsystem, it will hold an election

- Current process sends an ELECTION message to all processes with higher numbers
- If no one responds, this process wins the election and becomes a coordinator
- If one of the higher-ups answers, it takes over the election.

When a process receives an ELECTION message from one of the lower-numbered hosts

- Receiver sends an OK message back to the sender to indicate that it is alive and will take over the election.
- Receiver holds an election, unless it is already holding one.
- Eventually, only one process wins all elections and becomes the new coordinator.
- The new coordinator sends all processes a message telling them it is the new coordinator starting immediately.

When a process that was down previously returns alive, it will hold an election

- The process sends an ELECTION message to all processes with higher numbers
- If its process ID happens to be the highest, it will become the new coordinator

Figure 6 shows an example of Bully Algorithm, suppose host 1 is initialized as leader and host 3 is higher-numbered compared to host 2.

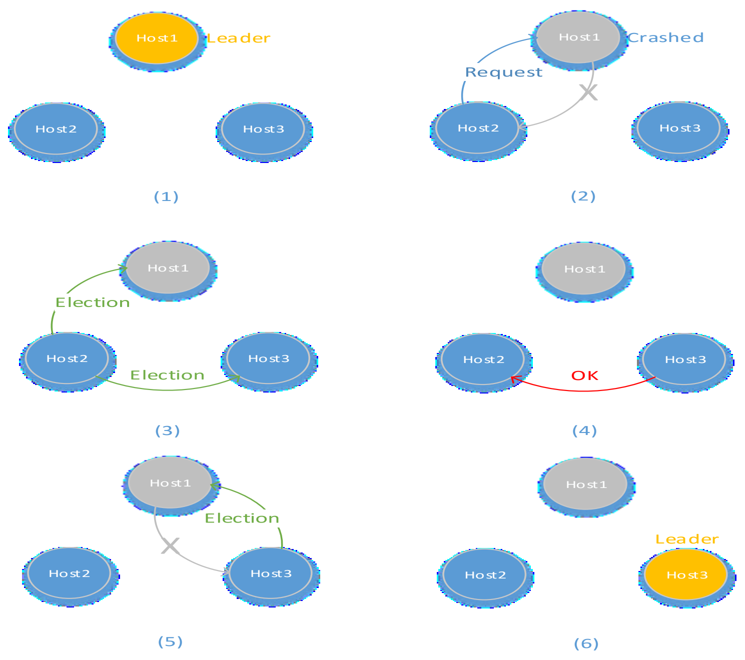


Figure 7 example of leader election subsystem

Test Cases:

Will be added when the project has been implemented.

Team tasks allocation

Student name	Student ID	Specific tasks
Yang Dong	40003546	Design and implement the group leader process which receives a request from the front end, FIFO broadcasts the request to all the server replicas in the group using UDP datagrams, receives the responses from the server replicas and sends a single response back to the front end as soon as possible.
Huaqiang Kang	40018914	Design and implement a reliable FIFO broadcast subsystem over the unreliable UDP layer
Meng Yao	26847005	Design and implement a failure 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.
Xingjian Zhang	40010222	Design and implement a distributed leader election subsystem (based on the bully algorithm), which will be called when the current leader has crashed to elect a new leader for the process group.