COMP 6231- Distributed System Design

Project on

Software Failure Tolerant / Highly Available CORBA Distributed Class Management System (DCMS)

Date:	July 28th, 2018.
Professor:	Mr. Mohamed Taleb
Team No:	#15
Student Name:	Sagar Vetal (40071979) Himanshu Kohli (40070839) Khyatibahen Chaudhary (40071098) Zankhanaben Ashish Patel (40067635)

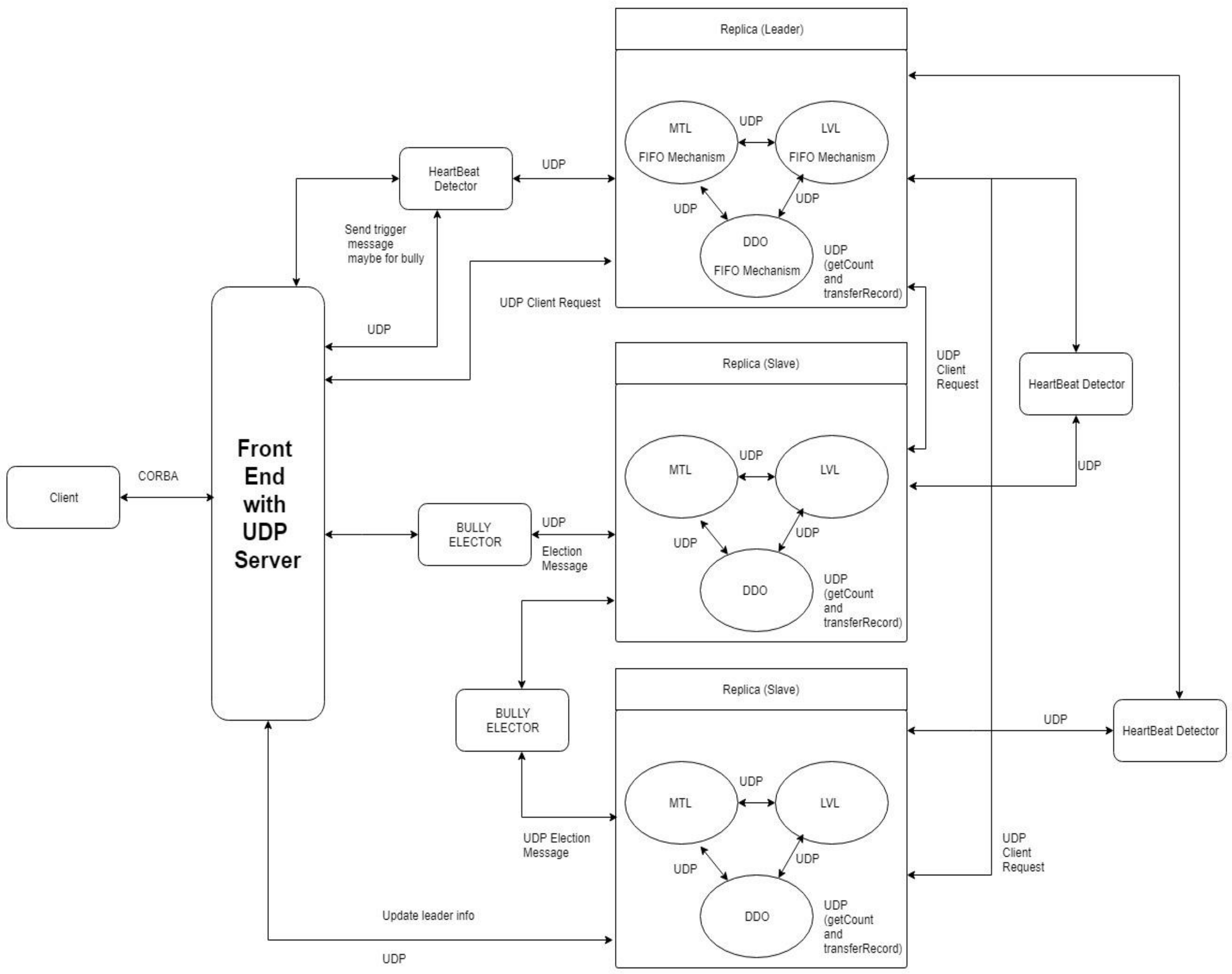
Contents

1. Objective

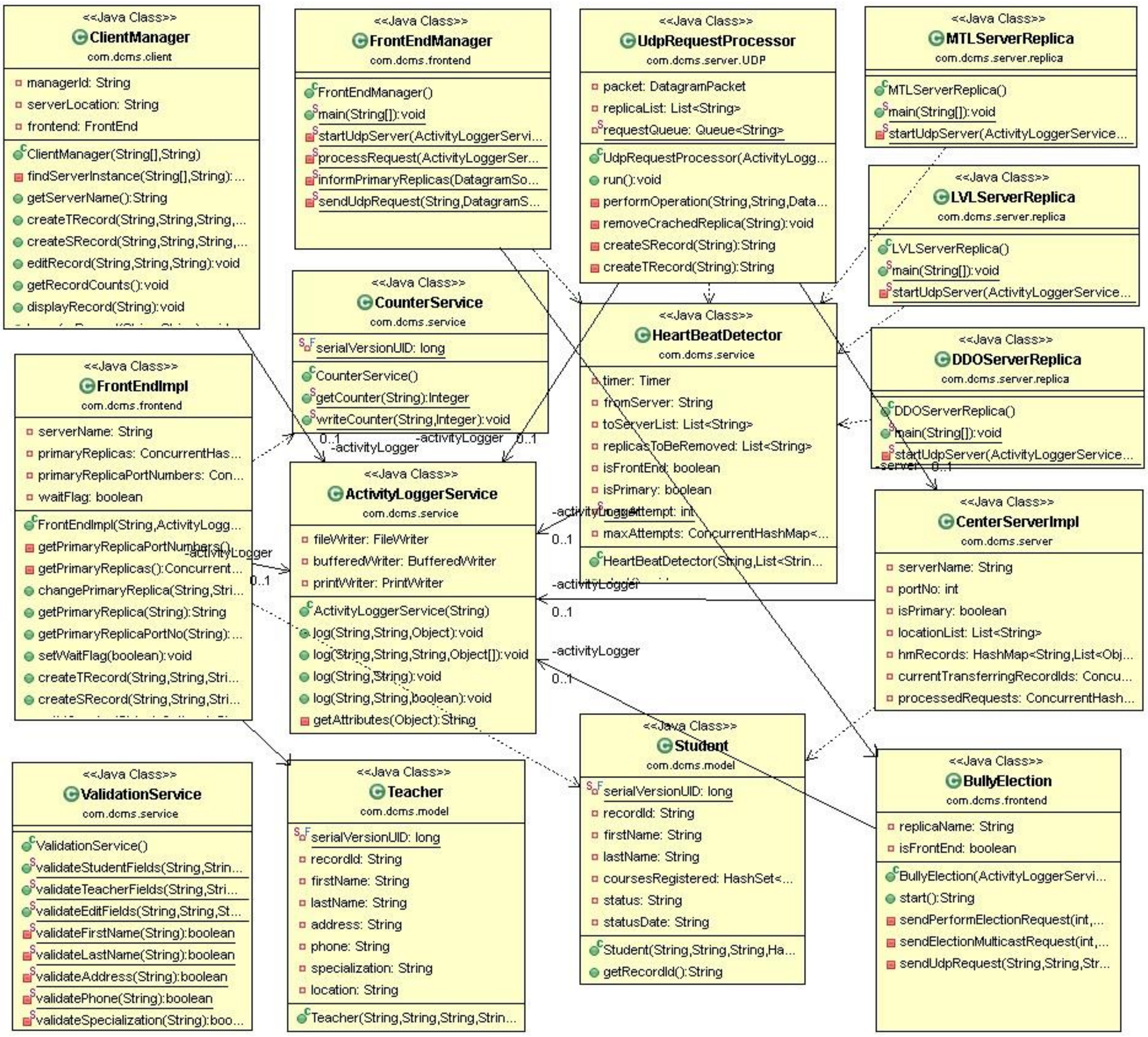
To implement a highly available CORBA Distributed Class Management System, which tolerates process crashes only (no software bugs) using unreliable failure detection. There are group of three servers processes providing redundancy for highly available and periodically checking each other for failure detection. We also have to implement bully algorithm to elect new leader from slave replicas in case of primary replica failure. Also, implement FIFO broadcast mechanism in which leader replicas atomically send client request to slave replicas in the same sequence in the order of received operation, receives response from them and sends single response to the client as soon as possible. The servers communicate via unreliable UDP protocol.

2. Design

2.1 System Architecture



2.2 Class Diagram



<<Java Class>>
ClientManager
com.dcms.client
managerId: String
serverLocation: String
frontend: FrontEnd
ClientManager(String[],String)
findServerInstance(String[],String):...
getServerName():String
createTRecord(String,String,String,...
createSRecord(String,String,String,...
editRecord(String,String,String):void
getRecordCounts():void
displayRecord(String):void
<<Java Class>>
FrontEndManager
com.dcms.frontend
FrontEndManager()
main(String[]):void
startUdpServer(ActivityLoggerServi...
processRequest(ActivityLoggerSer...
informPrimaryReplicas(DatagramSo...
sendUdpRequest(String,DatagramS...
<<Java Class>>
UdpRequestProcessor
com.dcms.server.UDP
packet: DatagramPacket
replicaList: List<String>
requestQueue: Queue<String>
UdpRequestProcessor(ActivityLogg...
run():void
performOperation(String,String,Data...
removeCrachedReplica(String):void
createSRecord(String):String
createTRecord(String):String
<<Java Class>>
MTLServerReplica
com.dcms.server.replica
MTLServerReplica()
main(String[]):void
startUdpServer(ActivityLoggerService...
<<Java Class>>
LVLServerReplica
com.dcms.server.replica
LVLServerReplica()
main(String[]):void
startUdpServer(ActivityLoggerService...
<<Java Class>>
CounterService
com.dcms.service
serialVersionUID: long
CounterService()
getCounter(String):Integer
writeCounter(String,Integer):void
<<Java Class>>
HeartBeatDetector
com.dcms.service
timer: Timer
fromServer: String
toServerList: List<String>
replicasToBeRemoved: List<String>
isFrontEnd: boolean
isPrimary: boolean
pingAttempt: int
maxAttempts: ConcurrentHashMap<...
HeartBeatDetector(String,List<Strin...
<<Java Class>>
DDOServerReplica
com.dcms.server.replica
DDOServerReplica()
main(String[]):void
startUdpServer(ActivityLoggerService...
-server 0..1
<<Java Class>>
FrontEndImpl
com.dcms.frontend
serverName: String
primaryReplicas: ConcurrentHas...
primaryReplicaPortNumbers: Con...
waitFlag: boolean
FrontEndImpl(String,ActivityLogg...
getPrimaryReplicaPortNumbers()...
getPrimaryReplicas():Concurrent...
changePrimaryReplica(String,Stri...
getPrimaryReplica(String):String
getPrimaryReplicaPortNo(String):...
setWaitFlag(boolean):void
createTRecord(String,String,Stri...
createSRecord(String,String,Stri...
-activityLogger 0..1
0..1 -activityLogger
-activityLogger 0..1
<<Java Class>>
ActivityLoggerService
com.dcms.service
fileWriter: FileWriter
bufferedWriter: BufferedWriter
printWriter: PrintWriter
ActivityLoggerService(String)
log(String,String,Object):void
log(String,String,String,Object[]):void
log(String,String):void
log(String,String,boolean):void
getAttributes(Object):String
-activityLogger 0..1
-activityLogger 0..1
-activityLogger 0..1
<<Java Class>>
CenterServerImpl
com.dcms.server
serverName: String
portNo: int
isPrimary: boolean
locationList: List<String>
hmRecords: HashMap<String,List<Obj...
currentTransferringRecordIds: Concu...
processedRequests: ConcurrentHash...
<<Java Class>>
ValidationService
com.dcms.service
ValidationService()
validateStudentFields(String,Strin...
validateTeacherFields(String,Stri...
validateEditFields(String,String,St...
validateFirstName(String):boolean
validateLastName(String):boolean
validateAddress(String):boolean
validatePhone(String):boolean
validateSpecialization(String):boo...
<<Java Class>>
Teacher
com.dcms.model
serialVersionUID: long
recordId: String
firstName: String
lastName: String
address: String
phone: String
specialization: String
location: String
Teacher(String,String,String,Strin...
<<Java Class>>
Student
com.dcms.model
serialVersionUID: long
recordId: String
firstName: String
lastName: String
coursesRegistered: HashSet<...
status: String
statusDate: String
Student(String,String,String,Ha...
getRecordId():String
<<Java Class>>
BullyElection
com.dcms.frontend
replicaName: String
isFrontEnd: boolean
BullyElection(ActivityLoggerServi...
start():String
sendPerformElectionRequest(int,...
sendElectionMulticastRequest(int,...
sendUdpRequest(String,String,Str...

3. Implementation

3.1 Classes Description

- **ClientManager.java**
  ClientManager communicate with FrontEnd through CORBA. It can perform six different operations: createTRecord, createSRecord, editRecord, getRecordCounts, displayRecord and transferRecord. Clients send multiple requests, which are handled in parallel. From the client's end, the whole system's functionalities are encapsulated while the complexities are hidden from the end users.

- **FrontEnd.java**
  FrontEnd is the mediator between the clients and the replicas. It is an interface which declares below methods that client can invoke on the service.

1. *createTRecord (firstName, lastName, address, phone, specialization, location, managerId);*
2. *createSRecord (firstName, lastName, coursesRegistered, status, statusDate, managerId);*
3. *editRecord (recordId, fieldName, newValue, managerId);*
4. *getRecordCounts (managerId);*
5. *displayRecord (recordId, managerId);*
6. *transferRecord (managerId, recordId, remoteCenterServerName);*

- **FrontEndImpl.java**
  This class implements methods for all operations declared in FrontEnd interface. When a client needs a service from the DSMS, it sends a request over the CORBA to the FE. The FE will validate request, generate request id, append it to request and send it to the primary replica using a UDP connection. In case of primary replica crashing FrondEnd will perform Bully Algorithm to elect new primary replica from slave replicas.

- **MTLServerReplica.java, LVLServerReplica.java and DDOServerReplica.java**
  These classes are used as primary replica as well as slave replica for their respective location. To differentiate between primary and slave replica one Boolean flag has been used named as "isPrimary". These classes start their respective UDP server. Primary replicas send heartbeat to FronEnd and also receive heartbeat from slave replicas. Each replica file has to be run 3 times in order to start 1 primary replica and 2 slave replica using 3 command line arguments like below,
  replicaName<space>portNo<space>isPrimaryFlag
  E.g. To start primary and slave replicas for Montreal we have run MTLServerReplica.java 3 times using below 3 arguments respectively,
  MTL_Replica_1 9071 true (for primary)
  MTL_Replica_2 9072 false (for slave 1)
  MTL_Replica_3 9073 false (for slave 2)

- **CenterServerImpl.java**
  This class is used to perform all operations at server side for each replica.

- **HeartBeatDetector.java**
  This class is used to detect the heartbeat of primary replicas for FrontEnd and to detect the heartbeat of slave replicas for primary replica, so that FrontEnd and primary replica can identify that which replica is crashed.

- **BullyElector.java**
  This class is used to perform the bully algorithm. It will find superior processes of current process using process id and send election message to them. If it does not receive any response from its superior process, it will make itself as primary replica and inform FrontEnd about it.

- **CounterService.java**
  This class is serializable class used to generate the record id for student and teacher and it will help to keep record id unique among all servers. This class will read latest counter for record from one of below text files,
  StudentCounter.txt

TeacherCounter.txt
RequestCounter.txt

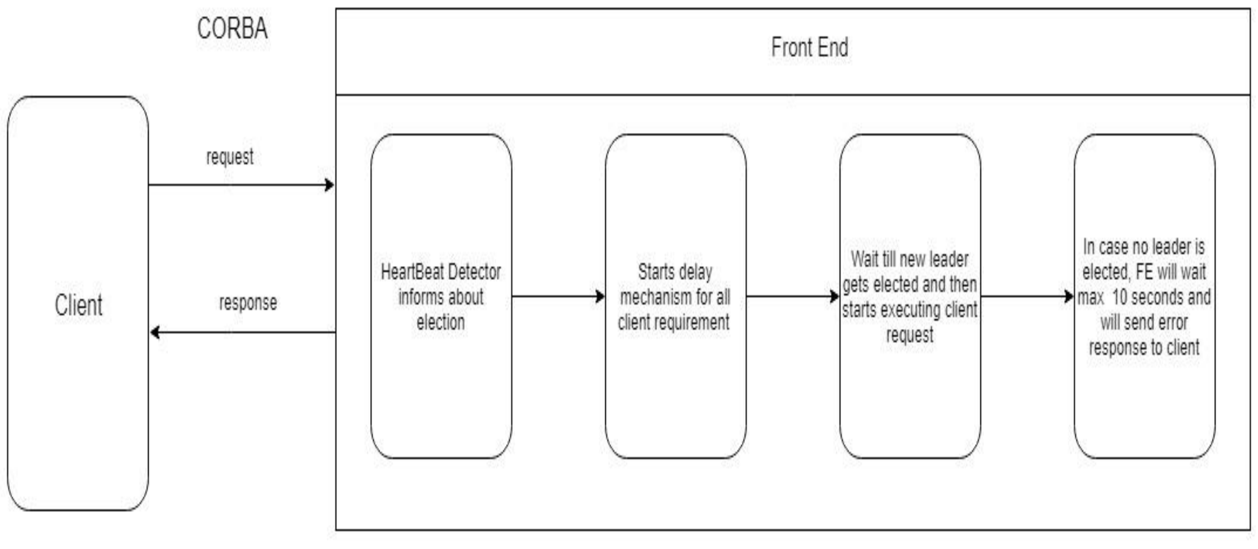
- **Student.java (Model)**
  This is a java bean class to hold following data of students,
  First Name, Last Name, Courses Registered, Status and Status Date.

- **Teacher.java (Model)**
  This is a java bean class to hold following data of teacher,
  First Name, Last Name, Specialization, Location, Phone and Address.

- **ValidationService.java**
  This class is used to perform following validations,
  1. First name, Last name and specialization fields should include only characters.
  2. Address field can be alphanumeric.
  3. Phone no field should be numeric of length 10.
  4. Location field should have either MTL, LVL or DDO only as a value.
  5. Status field should have either active or inactive as a value.
  6. Status Date format is dd-MM-yyyy.

3.2 Front End

When a client needs a service from the DSMS, it sends a request over the CORBA to the FE. The FE will validate request, generate request id, append it to request and send it to the primary replica using a UDP connection.
In case of primary replica crashing HeartbeatDetector will inform FronEnd about election. So FrontEnd will start delay mechanism using one 'waitFlag' for all upcoming client request till new primary replica got elected and then will perform Bully Algorithm to elect new primary replica from slave replicas. In case no new primary replica got elected FrontEnd will wait max 10 seconds and send error response to client. Here we have used socket timeout and maximum attempt functionality to make the UDP reliable.

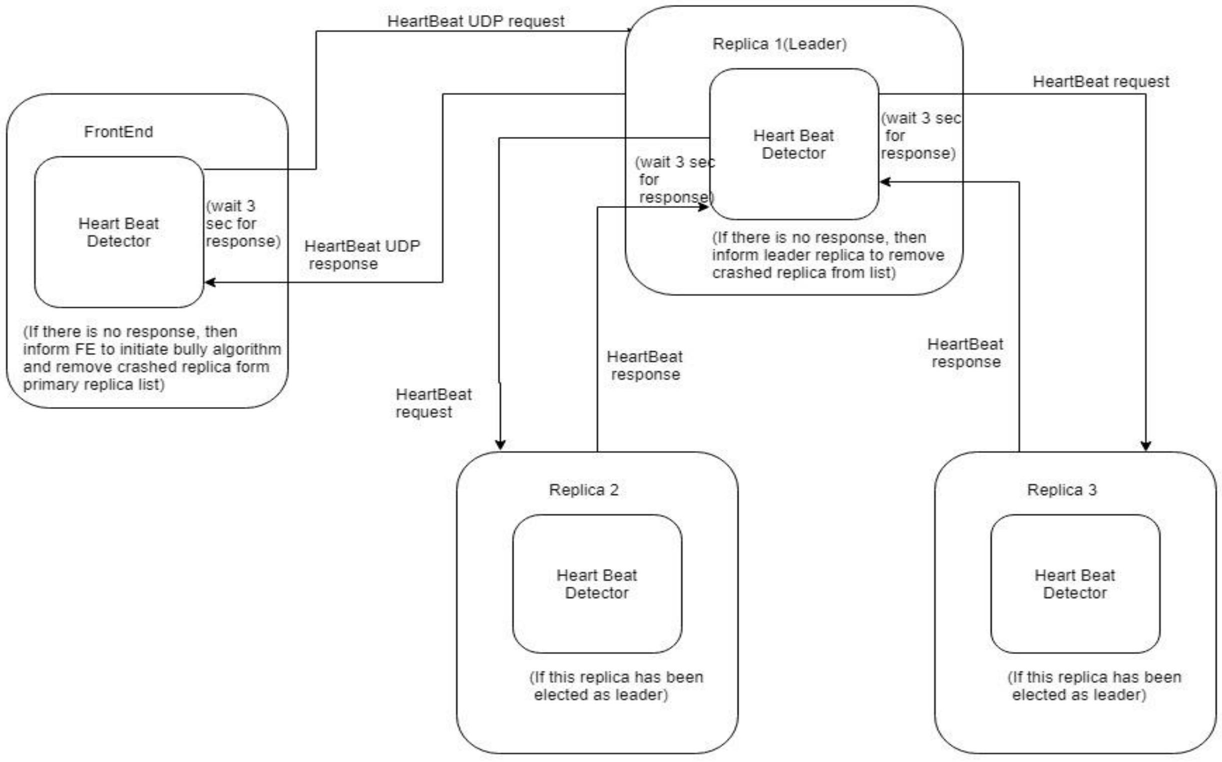

CORBA
Front End
request
response
Client
HeartBeat Detector informs about election
Starts delay mechanism for all client requirement
Wait till new leader gets elected and then starts executing client request
In case no leader is elected, FE will wait max 10 seconds and will send error response to client

3.3 Heartbeat Detection

Here we have HeartBeatDetector to detect the heartbeat of primary replicas for FrontEnd and to detect the heartbeat of slave replicas for primary replica, so that FrontEnd and primary replica can identify that which replica is crashed. Heartbeat detector will wait maximum 3 seconds for receiving heartbeat from slave replicas (for primary) and primary replicas (for FrontEnd). If there is no heartbeat received from any replica, HeartBeat detector will consider that replica as crashed.

In case of FrontEnd it will inform FronEnd to initiate bully algorithm and remove crashed replica from their primary replica list. In case of primary replica it will inform primary replica to remove crashed replica from their slave replica list. Slave replicas will also have their Heartbeat detector, but it will initiated once that slave replica has been elected as primary.

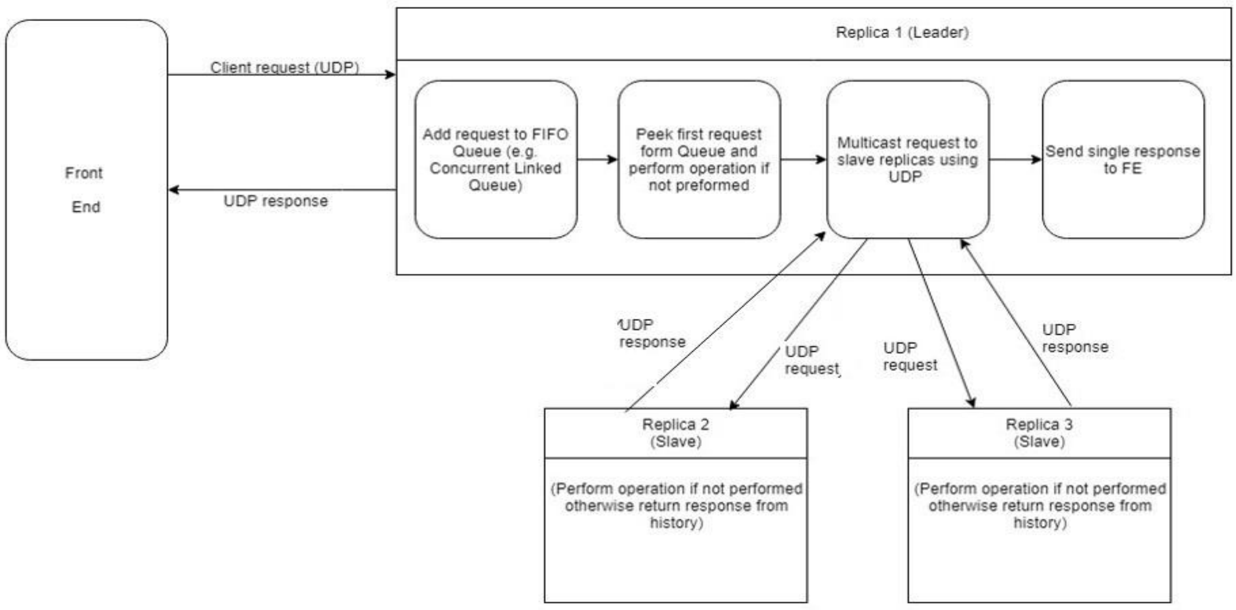
Here we have used socket timeout and maximum attempt functionality to make the UDP reliable.



HeartBeat UDP request
Replica 1(Leader)
HeartBeat request
FrontEnd
Heart Beat Detector
(wait 3 sec for response)
(wait 3 sec for response)
Heart Beat Detector
(wait 3 sec for response)
HeartBeat UDP response
(If there is no response, then inform leader replica to remove crashed replica from list)
(If there is no response, then inform FE to initiate bully algorithm and remove crashed replica form primary replica list)
HeartBeat response
HeartBeat response
HeartBeat request
Replica 2
Heart Beat Detector
(If this replica has been elected as leader)
Replica 3
Heart Beat Detector
(If this replica has been elected as leader)

3.4 FIFO Mechanism

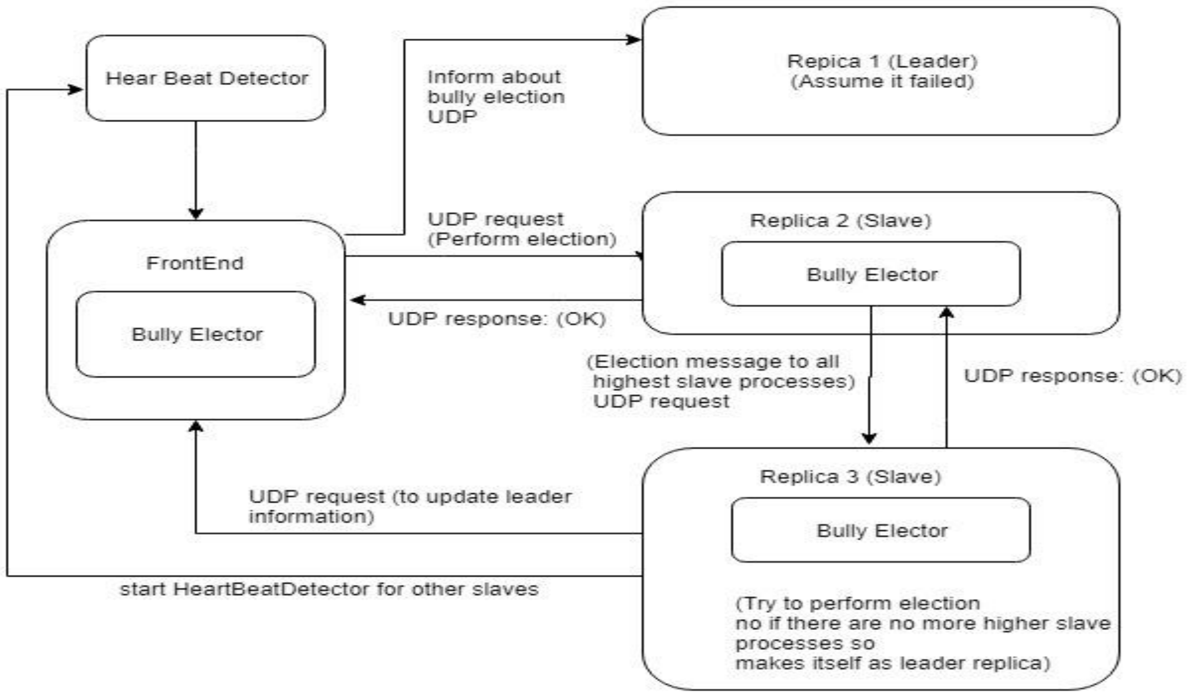
Whenever FrontEnd will send client request to primary replica, it will add that request into FIFO queue. Here we have used *ConcurrentLinkedQueue* data structure to create FIFO queue as it is more thread safe in multithreading environment. Primary replica will peek head request from the queue and perform the operation and simultaneously it will multicast that request to slave replicas through UDP, so after receiving response from slave replicas primary replica will send single response to FrontEnd.



Front End
Client request (UDP)
UDP response
Replica 1 (Leader)
Add request to FIFO Queue (e.g. Concurrent Linked Queue)
Peek first request form Queue and perform operation if not preformed
Multicast request to slave replicas using UDP
Send single response to FE
UDP response
UDP request
UDP request
UDP response
Replica 2 (Slave)
(Perform operation if not performed otherwise return response from history)
Replica 3 (Slave)
(Perform operation if not performed otherwise return response from history)

3.5 Bully Algorithm

The Bully algorithm is a method for dynamically electing a coordinator or leader from a group of distributed computer processes. In our project when HeartbeatDetector informs FrontEnd about primary replica crash, FrontEnd will inform other running primary replica about election and send election message to immediate slave replica having higher process id(i.e. replica 2). Now replica 2 will send election message to all its superior replicas and after receiving response it will ask replica 3 now to send election message to its superior replica. But here there are no superior replicas having greater process id, so replica 3 will make itself as a primary replica. Replica 2 then start its HeartBeatDetector to check slave replicas and it will send message to FrontEnd to update new primary replica information.



Hear Beat Detector
Inform about bully election UDP
Repica 1 (Leader) (Assume it failed)
FrontEnd
Bully Elector
UDP request (Perform election)
UDP response: (OK)
Replica 2 (Slave)
Bully Elector
(Election message to all highest slave processes) UDP request
UDP response: (OK)
Replica 3 (Slave)
Bully Elector
UDP request (to update leader information)
start HeartBeatDetector for other slaves
(Try to perform election no if there are no more higher slave processes so makes itself as leader replica)

3.6 UDP Reliability

In our project we have used socket timeout and maximum 3 attempt functionality to make the UDP reliable. While sending UDP request we have set socket timeout of 2 seconds, so that source server will wait for 2 seconds for response and after that throw timeout exception. In socket timeout exception block we have decreased the attempt count and resend that request. This process will repeat 3 time as maximum attempts are set to 3. This will help UDP server to maintain its reliability and avoid one UDP call of sending acknowledgement from destination server to source server which in turn reduce the delay of UDP calls.

4. Data Structures

4.1 Record Storage

We have used simple *HashMap* like data structure for the storage of records which allows to perform adding, retrieval and updating operations on both type of records i.e. teacher and student.

Syntax:
HashMap<String, List<Object>> hmRecords = new HashMap<String, List<Object>>();

Here, key is first letter of the last name of student/teacher and value is an object which is the parent class which allow to choose object type i.e. Student or Teacher.

4.2 FIFO Mechanism

We have used *ConcurrentLinkedQueue* data structure to create FIFO queue as it is more thread safe in multithreading and concurrent process environment.
Syntax:
Queue<String> recordQueue = new *ConcurrentLinkedQueue<String>();*

4.3 Response History

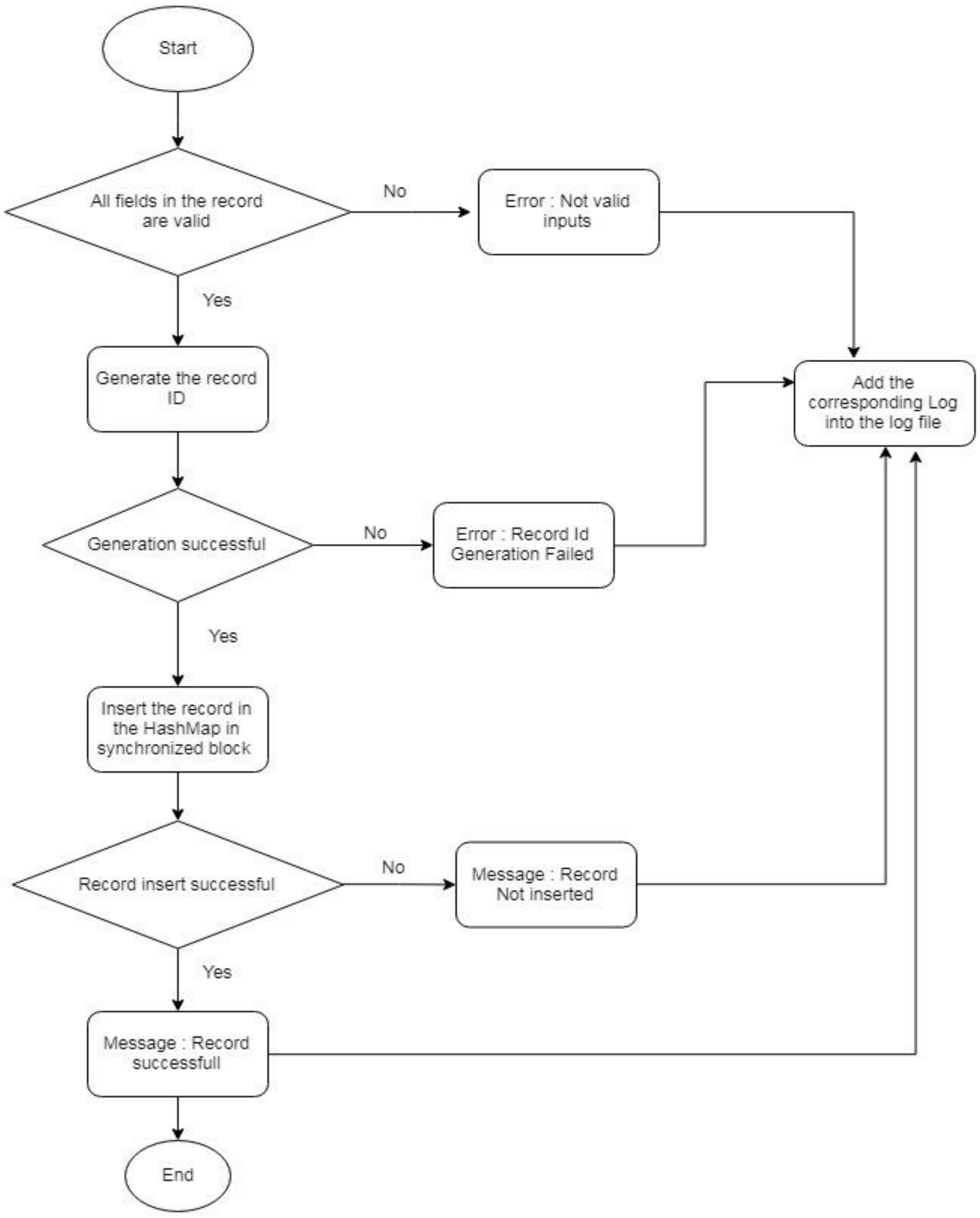
We have maintaining response history at each replica to avoid duplicacy of records. To achieve this we have used *ConcurrentHashMap* data structure.
Syntax:
ConcuurentHashMap<String, String> processedRequests = new ConcuurentHashMap <String, String>();

Here, key is request id and value is response of the respective request.

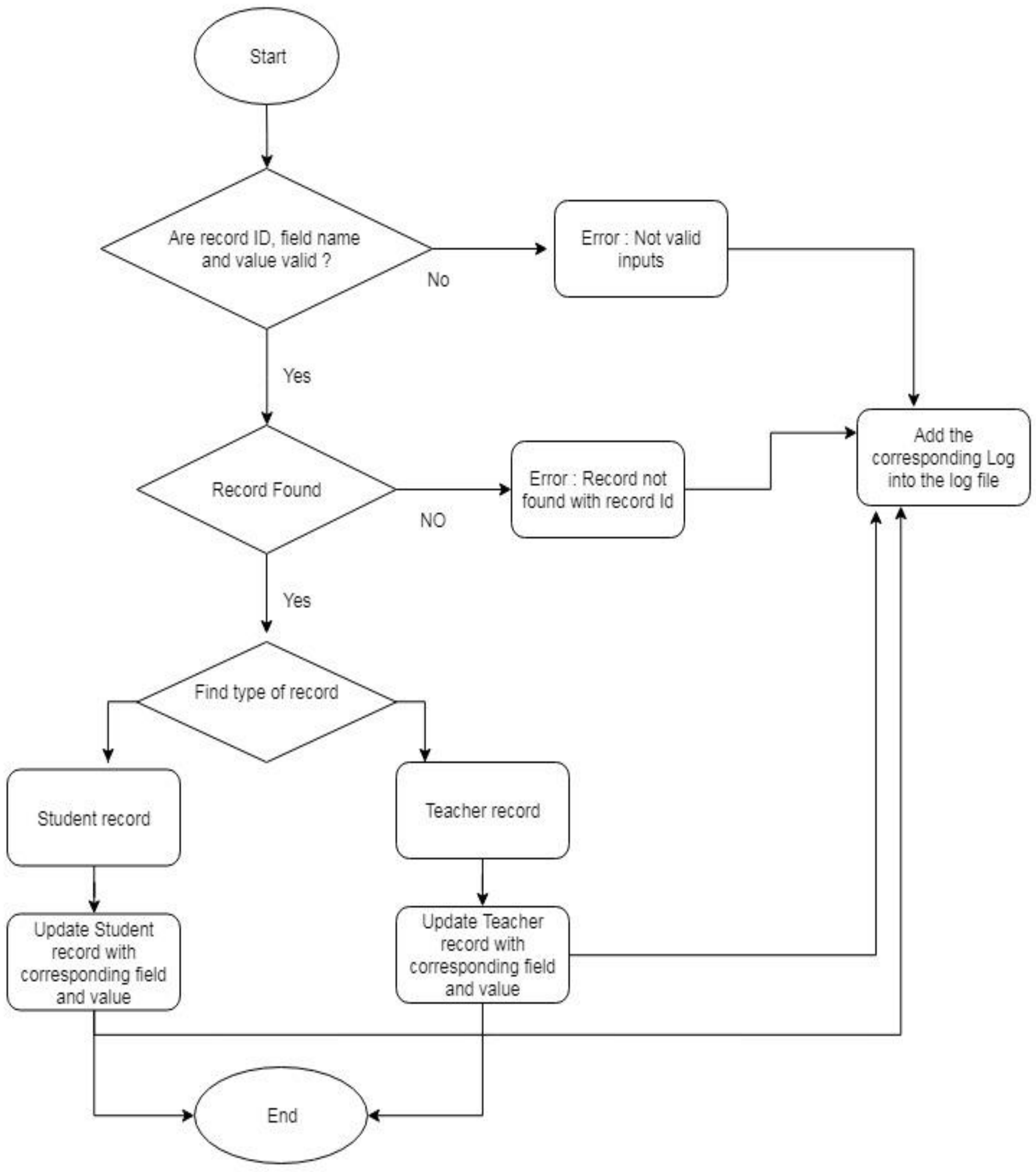
5. Operations:

- **Create Record:**
  - While creating new record first all the fields of the record are validated at server side by *ValidationService* class. Validations are as follow,
    1. First name, Last name and specialization fields should include only characters.
    2. Address field can be alphanumeric.
    3. Phone no field should be numeric of length 10.
    4. Location field should have either MTL, LVL or DDO only as a value.
    5. Status field should have either active or inactive as a value.
    6. Status Date format is dd-MM-yyyy.
  - If all fields are valid, record will be stored in hashmap using first letter of the last name of student/teacher as a key.
  - Otherwise corresponding error will be returned to client.

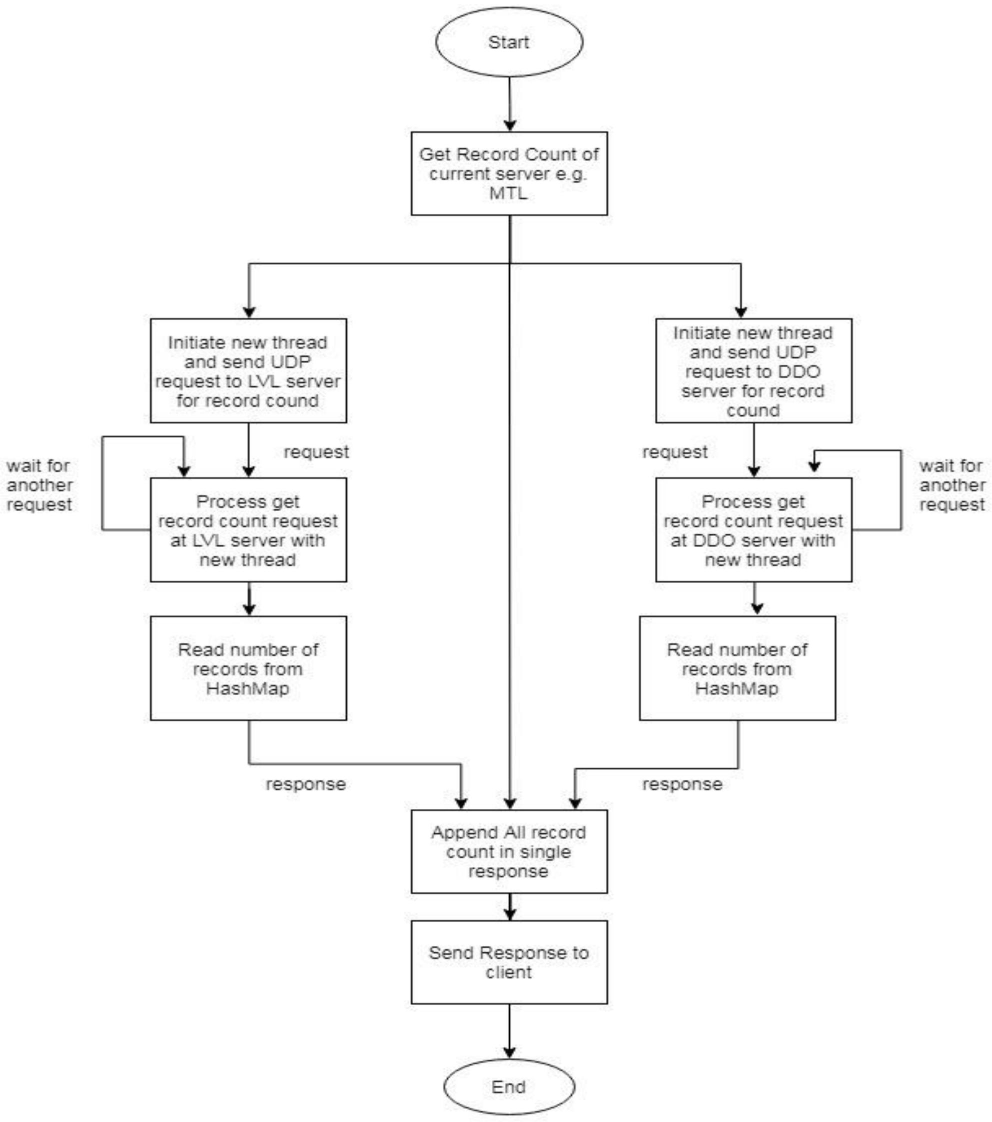
- All success/error messages will be logged into respective log files.


Start
All fields in the record are valid
No
Error : Not valid inputs
Yes
Generate the record ID
Add the corresponding Log into the log file
Generation successful
No
Error : Record Id Generation Failed
Yes
Insert the record in the HashMap in synchronized block
Record insert successful
No
Message : Record Not inserted
Yes
Message : Record successfull
End

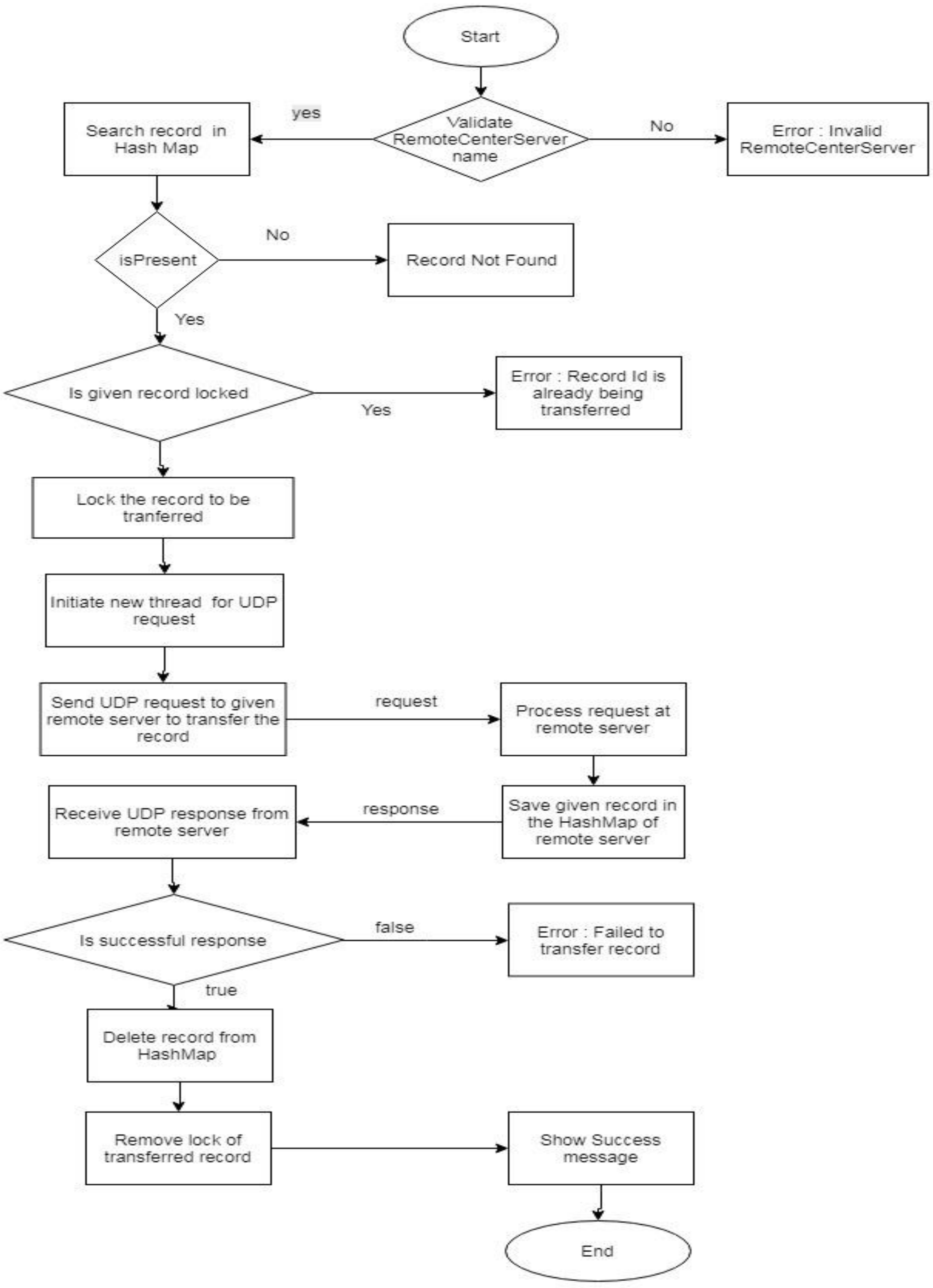
- **Edit Record:**
  - While editing records *ValidationService* class will take care of the validation of allowed fields to be modified and type data allowed for those fields.
  - If all fields are valid, the record will be searched in the hashmap using RecordId.
  - If record is found, entered field will get updated with new value.
  - Otherwise corresponding error will be returned to client.
  - All success/error messages will be logged into respective log files.


Start
Are record ID, field name and value valid ?
No
Error : Not valid inputs
Yes
Record Found
NO
Error : Record not found with record Id
Add the corresponding Log into the log file
Yes
Find type of record
Student record
Teacher record
Update Student record with corresponding field and value
Update Teacher record with corresponding field and value
End

- **Get Record Count:**
  - First current server will fetch the number records from hashmap.
  - Then it will initiate separate thread for each UDP server and send request to fetch record count from each server.
  - Then each server will process get record count request in separate thread and return response to current server.
  - On successful response current server will append all record counts in single response and return it to client.
  - Otherwise corresponding error will be returned to client.
  - All success/error messages will be logged into respective log files.


Start
Get Record Count of current server e.g. MTL
Initiate new thread and send UDP request to LVL server for record cound
Initiate new thread and send UDP request to DDO server for record cound
request
request
wait for another request
wait for another request
Process get record count request at LVL server with new thread
Process get record count request at DDO server with new thread
Read number of records from HashMap
Read number of records from HashMap
response
response
Append All record count in single response
Send Response to client
End

- **Transfer Record:**
  - First remote server name will be validated whether is correct or not. If it is wrong, server will returned error "Invalid remote server".
  - If it is correct, the record will be searched in the hashmap using RecordId.
  - If record is found, current server will put lock on that record, so that other thread cannot do any operation on that thread. Then server will initiate new thread and send UDP request to remote server and remote server will process that request in separate thread and store that record and return response.
  - On successful response current server will release the lock and delete that record from Hashmap.
  - Otherwise corresponding error will be returned to client.
  - All success/error messages will be logged into respective log files.



Start
Validate RemoteCenterServer name
yes
No
Search record in Hash Map
Error : Invalid RemoteCenterServer
isPresent
No
Record Not Found
Yes
Is given record locked
Yes
Error : Record Id is already being transferred
Lock the record to be tranferred
Initiate new thread for UDP request
Send UDP request to given remote server to transfer the record
request
Process request at remote server
Save given record in the HashMap of remote server
response
Receive UDP response from remote server
Is successful response
false
Error : Failed to transfer record
true
Delete record from HashMap
Remove lock of transferred record
Show Success message
End

6. Multithreading, Synchronization and Concurrency:

We have implemented multithreading in UDP server communication and multiple client testing. In *getRecordCounts() and transferRecord()* methods, a separate thread is initiated to send UDP (getRecordCount and transferRecord respectively) requests to remaining servers. Also above mentioned requests are processed in separate thread at each UDP server. Also to *multicast client request to slave replicas* and for *bully algorithm* multithreading functionality has been implemented.

We have used the keyword **synchronized** to synchronize thread execution so that only one thread gets the data access and response from server at a time and others wait while execution complete. Synchronization is a procedure which is used to avoid interference with the other threads and memory consistent errors. Moreover, each thread has its own memory pool so there are no shard resources among other threads.

We have used following Synchronization in the project:

- **Block level Synchronization:**
  In the methods like **createSRecord()**, **createTRecord() and editRecord()**, we have made specific block of code synchronized. That code block contains actual insertion/updation into and retrieval from HashMap, and other part of the method don't need synchronization while performing the record addition.
- **Method level Synchronization:**
  Methods like **searchRecord() and getRecordCount()**, contains only retrieval code of HashMap, hence we have entirely made these methods synchronized. Also in order to avoid duplicate record id we have made **getCounter() and writeCounter()** method synchronized in **CounterService** class.

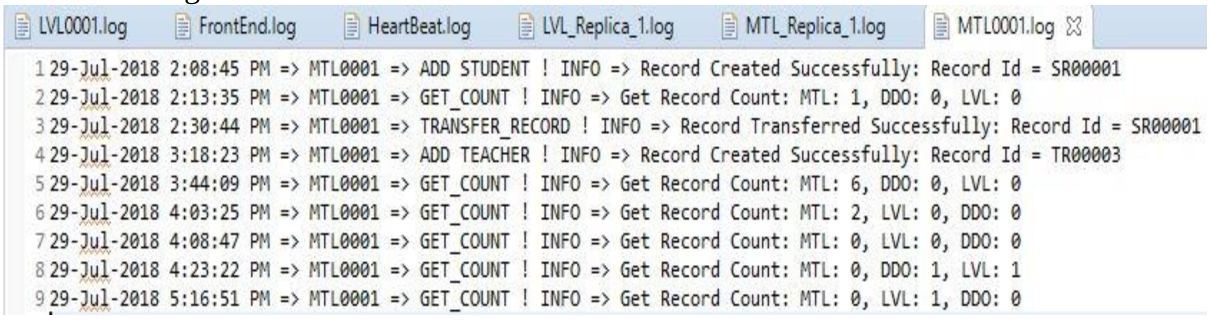
The Fine-grained locking is used to put intrinsic lock while doing parallel access to data resources so that data is accessed as required without infecting the data. This increases the efficiency and maximizes the concurrency to perform concurrent operations.

Concurrency is implemented by using the synchronized () {} block in java which puts temporary lock on the object or list passed to it as parameter in this implementation.

7. Activity Logger

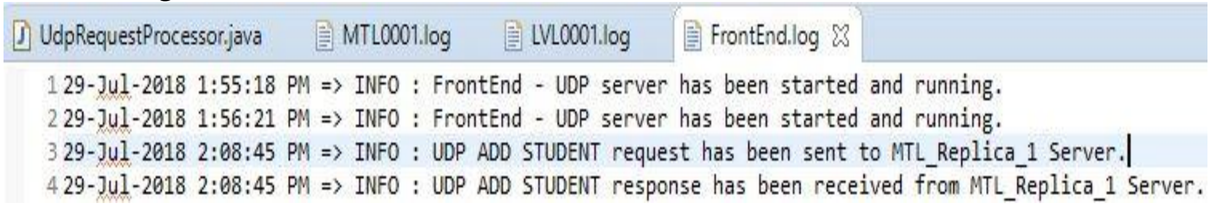
We have created our own customized logger service class named as "**ActivityLoggerService**" which has three overloaded methods to log the success messages and error messages in respective activity log file. There will be log files at frontend, server and client side.
Following will be log file names,
For FrontEnd: *frontend.log*
For Server: *<replicaName>.log*
For Client: *<managerId>.log*
For HeartBeat: *heartbeat.log*

MTL00001.log file:



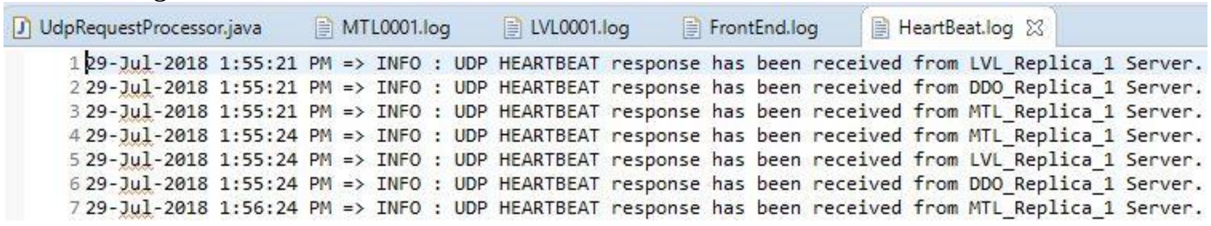
```
29-Jul-2018 2:08:45 PM => MTL0001 => ADD STUDENT ! INFO => Record Created Successfully: Record Id = SR00001
29-Jul-2018 2:13:35 PM => MTL0001 => GET_COUNT ! INFO => Get Record Count: MTL: 1, DDO: 0, LVL: 0
29-Jul-2018 2:30:44 PM => MTL0001 => TRANSFER_RECORD ! INFO => Record Transferred Successfully: Record Id = SR00001
29-Jul-2018 3:18:23 PM => MTL0001 => ADD TEACHER ! INFO => Record Created Successfully: Record Id = TR00003
29-Jul-2018 3:44:09 PM => MTL0001 => GET_COUNT ! INFO => Get Record Count: MTL: 6, DDO: 0, LVL: 0
29-Jul-2018 4:03:25 PM => MTL0001 => GET_COUNT ! INFO => Get Record Count: MTL: 2, LVL: 0, DDO: 0
29-Jul-2018 4:08:47 PM => MTL0001 => GET_COUNT ! INFO => Get Record Count: MTL: 0, LVL: 0, DDO: 0
29-Jul-2018 4:23:22 PM => MTL0001 => GET_COUNT ! INFO => Get Record Count: MTL: 0, DDO: 1, LVL: 1
29-Jul-2018 5:16:51 PM => MTL0001 => GET_COUNT ! INFO => Get Record Count: MTL: 0, LVL: 1, DDO: 0
```

FrontEnd.log:



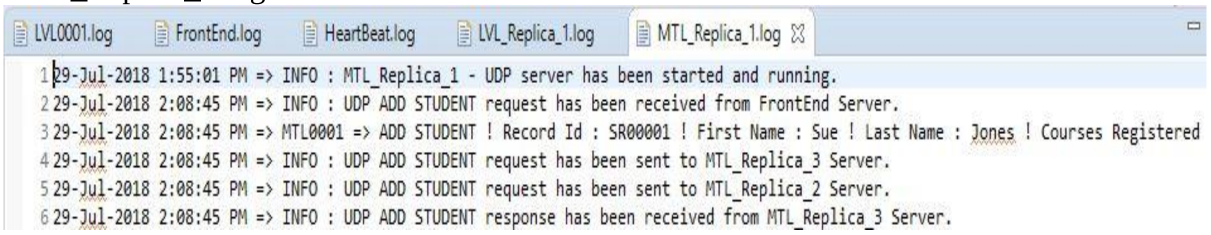
```
29-Jul-2018 1:55:18 PM => INFO : FrontEnd - UDP server has been started and running.
29-Jul-2018 1:56:21 PM => INFO : FrontEnd - UDP server has been started and running.
29-Jul-2018 2:08:45 PM => INFO : UDP ADD STUDENT request has been sent to MTL_Replica_1 Server.
29-Jul-2018 2:08:45 PM => INFO : UDP ADD STUDENT response has been received from MTL_Replica_1 Server.
```

HeartBeat.log



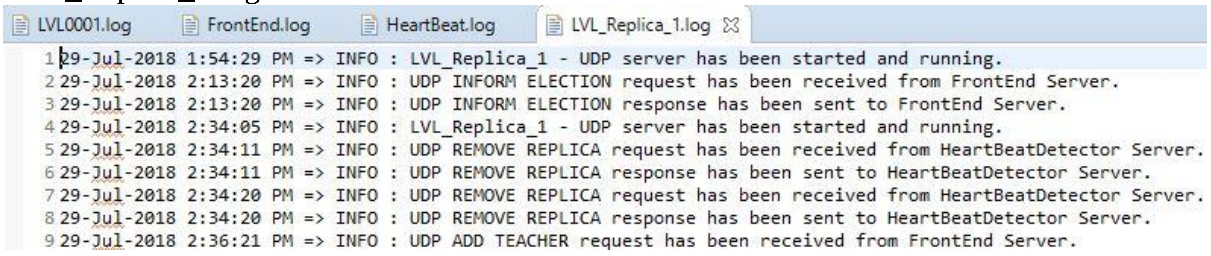
```
29-Jul-2018 1:55:21 PM => INFO : UDP HEARTBEAT response has been received from LVL_Replica_1 Server.
29-Jul-2018 1:55:21 PM => INFO : UDP HEARTBEAT response has been received from DDO_Replica_1 Server.
29-Jul-2018 1:55:21 PM => INFO : UDP HEARTBEAT response has been received from MTL_Replica_1 Server.
29-Jul-2018 1:55:24 PM => INFO : UDP HEARTBEAT response has been received from MTL_Replica_1 Server.
29-Jul-2018 1:55:24 PM => INFO : UDP HEARTBEAT response has been received from LVL_Replica_1 Server.
29-Jul-2018 1:55:24 PM => INFO : UDP HEARTBEAT response has been received from DDO_Replica_1 Server.
29-Jul-2018 1:56:24 PM => INFO : UDP HEARTBEAT response has been received from MTL_Replica_1 Server.
```

MTL_Replica_1.log



```
29-Jul-2018 1:55:01 PM => INFO : MTL_Replica_1 - UDP server has been started and running.
29-Jul-2018 2:08:45 PM => INFO : UDP ADD STUDENT request has been received from FrontEnd Server.
29-Jul-2018 2:08:45 PM => MTL0001 => ADD STUDENT ! Record Id : SR00001 ! First Name : Sue ! Last Name : Jones ! Courses Registered
29-Jul-2018 2:08:45 PM => INFO : UDP ADD STUDENT request has been sent to MTL_Replica_3 Server.
29-Jul-2018 2:08:45 PM => INFO : UDP ADD STUDENT request has been sent to MTL_Replica_2 Server.
29-Jul-2018 2:08:45 PM => INFO : UDP ADD STUDENT response has been received from MTL_Replica_3 Server.
```

LVL_Replica_1.log



```
29-Jul-2018 1:54:29 PM => INFO : LVL_Replica_1 - UDP server has been started and running.
29-Jul-2018 2:13:20 PM => INFO : UDP INFORM ELECTION request has been received from FrontEnd Server.
29-Jul-2018 2:13:20 PM => INFO : UDP INFORM ELECTION response has been sent to FrontEnd Server.
29-Jul-2018 2:34:05 PM => INFO : LVL_Replica_1 - UDP server has been started and running.
29-Jul-2018 2:34:11 PM => INFO : UDP REMOVE REPLICA request has been received from HeartBeatDetector Server.
29-Jul-2018 2:34:11 PM => INFO : UDP REMOVE REPLICA response has been sent to HeartBeatDetector Server.
29-Jul-2018 2:34:20 PM => INFO : UDP REMOVE REPLICA request has been received from HeartBeatDetector Server.
29-Jul-2018 2:34:20 PM => INFO : UDP REMOVE REPLICA response has been sent to HeartBeatDetector Server.
29-Jul-2018 2:36:21 PM => INFO : UDP ADD TEACHER request has been received from FrontEnd Server.
```

8. Test Cases

We have created two separate classes named *SingleClientTest and MultiClientTest* to test the functionality with single client and multiple clients respectively. We have created multiple threads in the *MultiClientTest* class for each server to test the functionality. Each thread performs actions on multiple records maintaining concurrency and are synchronized with each other. The threads do not access the same data resources at the same time.

We also have perform multiple client testing by running project on 3 different machines.

Below excel sheet (***"TestCases.xlsx"***) includes multiple test cases for single client and multiple client test.

TestCases .xlsx

11. Group Work Assignment:

Sr No	Task	Performed By
1	Requirement Gathering	Sagar Vetal, Himanshu Kohli, Khyatibahen Chaudhary, Zankhanaben Patel
2	Analysis & Project Design	Sagar Vetal
3	FrontEnd Implementation	Khyatibahen Chaudhary
4	Heartbeat Detector Implementation	Zankhanaben Patel
5	FIFO Mechanism	Himanshu Kohli
6	Bully Algorithm Implementation	Sagar Vetal
7	UDP Servers and Request Processor	Sagar Vetal
8	Activity Logger	Sagar Vetal
9	Project Report	Sagar Vetal, Himanshu Kohli, Khyatibahen Chaudhary, Zankhanaben Patel
10	Test case preparation	Sagar Vetal, Himanshu Kohli, Khyatibahen Chaudhary, Zankhanaben Patel
11	Testing	Sagar Vetal, Himanshu Kohli, Khyatibahen Chaudhary, Zankhanaben Patel

12. Difficulties Faced

1. Managing Concurrency amongst three replicas
2. Implementing synchronization among shared data and avoiding deadlocks
3. Testing all the scenarios in a real system
4. Implementation of Bully algorithm.
5. Problem with heart beat while the primary/leader is changed.

13. Conclusion

The project was successfully implemented and represents the real world scenario of a system having three replicas having same data without violating the integrity of the system. Managing concurrency, synchronization and message passing amongst servers.

14. References

1. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentLinkedQueue.html
2. https://stackoverflow.com/questions/9580457/fifo-class-in-java
3. https://stackoverflow.com/questions/24085571/synchronizing-queue-in-java-on-multiple-threads
4. https://stackoverflow.com/questions/10210516/suspicious-code-output-for-bully-algorithm