CMSC 621: PROJECT 3

Fault Tolerant Banking Service

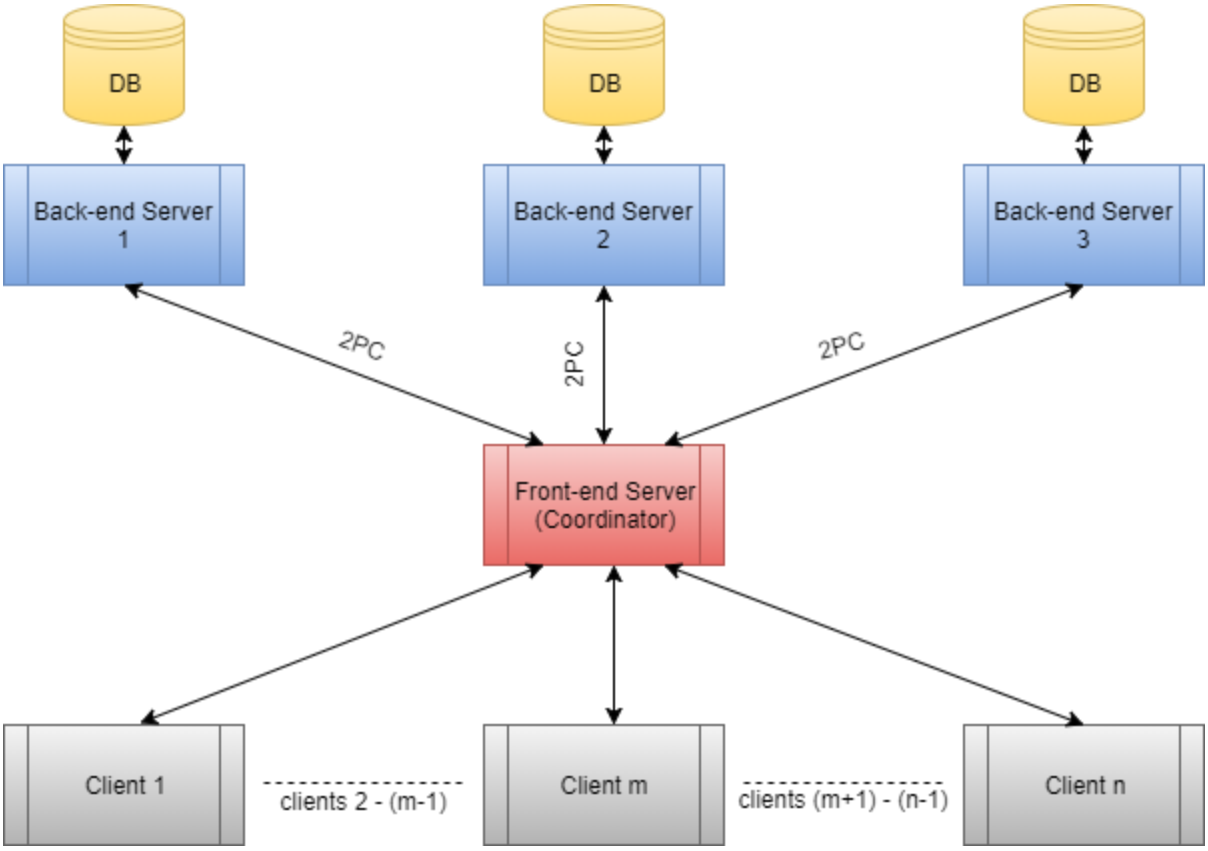
(Capable of Multiple Concurrent Operations)

Name: Intekhab Naser **ID:** ZC11577

README:

The following architecture is used to implement the system structure.

1) **Multi-tiered Architecture:**



Project Fundamentals:

This project demonstrates **fault tolerance** with replicated data of bank accounts over 3 bank servers. **Atomicity** of transactions is attained by using **2 phase commit protocol (2PC)** and **locking** mechanism is implemented to maintain consistency in the database/transactions.

Assumption: Front-end server never crashes.

Fault Tolerance: The bank servers are capable of handling multiple simultaneous requests and continue to process if the condition (2f+1) is satisfied, where 'f' is the number of faulty servers.

- From the three back-end servers, a failure/crash of one server can be tolerated.
- If two servers crash i.e. it becomes a single server application, then, even though the system can function normally, the two-phase commit protocol will not have any significance. That is why more than one server crashes are explicitly not tolerated.
- Needless to mention, if all three servers crash, the system halts.

Two-Phase Commit Protocol: Two-Phase Commit Protocol (2PC) is used to maintain atomicity of transactions. The front-end server acts as coordinator and all back-end servers act as cohorts/participants in the Two-Phase Commit Protocol (2PC). The process happens as follows:

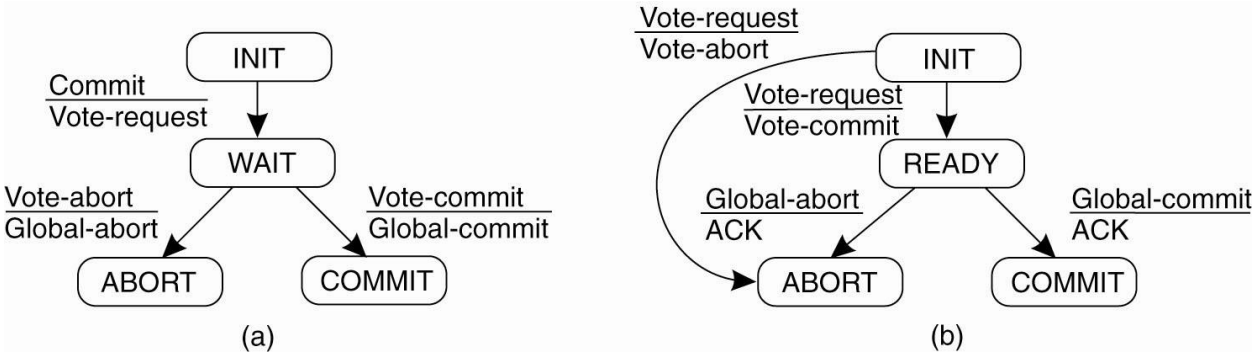
Phase 1:

- Along with client's request, the Coordinator sends a vote request to all cohorts to check if all participants are prepared to commit. Front-end server sends a message "READY" to back-end servers.
- If the back-end server is ready, it replies with "OK" message to front-end server

Phase 2:

- When the coordinator (front-end server) receives "OK" response from all the cohorts, it sends a message "COMMIT" to all back-end servers. Otherwise, it sends "ABORT".
- On receiving "COMMIT", backend servers reply to the coordinator with processed response to the client request. Otherwise, they abort the request by replying with a message as "Request Aborted!".

The finite state machine for the (a) coordinator and (b) participants in 2PC



Locking Mechanism:

Mutex locks are used to maintain consistency and avoid race conditions in case of concurrent access of a shared resource by multiple threads.

In this scenario, the **thread connection between the front-end and back-end processes is a shared resource** for multiple concurrent client requests/threads. Thus, in this architecture, the locking mechanism is implemented in the front-end process. Each client thread at the front-end process needs to lock the thread to the back-end processes before starting the transaction request and two=phase commit protocol. It then needs to unlock this thread after receiving the corresponding response from back-end servers.

The accounts in the back-end bank database records will be consistent as the client requests are already serialized in the front-end process.

-x-

Compiling and Running the Programs:

Instructions on compiling and running the code base:

1) Compiling the program:

To compile the back-end server, front-end server and client programs, I use Makefile as follows:

```
compile: bserver.o fserver.o client.o
bserver.o: BackendServer.cpp
    g++ -o back.o BackendServer.cpp -lpthread
fserver.o: FrontendServer.cpp
    g++ -o front.o FrontendServer.cpp -lpthread
client.o: Client.cpp
    g++ -o client.o Client.cpp
clean:
    rm -rf *.o
```

The following command compiles all the programs:

- ⇨ Make compile

The following command deletes the object files:

- ⇨ Make clean

2) Running the program:

It should be taken into consideration that, while running the program:

- Strictly 3 back-end server processes should be executed initially (one server crash can be tolerated)
- Only one front-end server process should be executed
- While multiple client processes may be connected to front-end server.

To execute back-end server program, use the following command:

⇨ ./back.o <unique port number>
- Start three such processes

To execute front-end server program, use the following command:

⇨ ./front.o <unique port number>
- After this, Front-end Server (Coordinator) asks three times:

⇨ “Enter the port number of Machine to connect: ”

We should specify the three back-end server process’s port numbers one by one.

To execute client program, use the following command:

⇨ ./client.o <front-end server port number> localhost

Run Multiple Clients programs.

A brief overview of the project’s working may be obtained from this screenshot (A .png file of this image is added in the repository as it’s difficult to read from this screenshot)