DAD Project Report - Group 31

Cristi Savin - 95549 Gonçalo Guerreiro - 95581 Joan Evangelisti Vadell - 105313

1. Introduction

This project consists of a distributed system composed by three tiers, emulating the behaviour of a simple bank application.

The clients (first tier) submit operations to a set of bank server processes (second tier) that use a primary-backup replication protocol, where the leader in each slot runs a two-phase commit protocol to order the operations.

To ensure coherence in the choice of each slot's leader, the servers use the Boney service (third tier), a distributed coordination system that runs Paxos to elect a primary server.

2. Boney

The Boney processes represent the third and final layer of this application. It's main role was to provide the Bank Servers with a simple interface where they can submit CompareAndSwap commands allowing them to check the leader for certain time slot, and, if none had been elected, kick-start an election.

2.1 Compare & Swap

This represented the interface that communicated with the Bank Servers. It is through the compareAndSwap commands that the above processes propose themselves as leaders and as return they would receive the id of the elected leader.

Its implementation is fairly simple because it only serves as more of an interface. Besides basic checks for compareAndSwap requests pertaining to already elected slots (where it simply returns the elected value to the requesting process) its only other function is to pass control to the Paxos part of Boney to start a new consensus instance.

2.2 Paxos

Paxos represents the heavier and trickier part of the Boney processes. It implements the classical Paxos algorithm that, as mentioned before, can be start with a compareAndSwap request for time slots where no leader is elected yet.

Paxos was implemented with the following division in mind - a front-end that would be the client of other Paxos processes and a server service implementation that dealt with those clients. The Paxos proposer was implemented as a client(front-end) while the acceptor and listener roles reside in the server service.

Messages exchanged between clients and servers of Paxos always include the timestamp to which they refer to in order to not generate new mistaken instances of Paxos.

2.2.1 Paxos State

In order to facilitate the handeling of state information of a Boney process that is devided into Compare and Swap, Paxos Front-End and Paxos Server Service, we thought best to create a class just to keep state and give each of the three parts mentioned access to it.

Access has to be done by aquiring a lock to it, guaranteeing that information is safely handled when all these three parts are running in parallel.

3. Bank

The bank is a distributed service implemented by a group of processes that accepts commands and replies to the clients. These processes use primary-backup replication to ensure the durability of the account information.

The bank execution is divided in time slots. At the beginning of each time slot, the bank processes determine if they are frozen and if they suspect the others, and then use the Boney service to elect a leader for that slot.

3.1. Two-Phase Commit Protocol

Since bank clients can send multiple concurrent commands, the bank servers use a two-phase commit protocol to assign a unique sequence number to each operation, guaranteeing that all the processes execute the commands in the same order.

3.2. Primary Server

In each slot, the elected primary server is responsible for assigning a unique sequence number to each received command and send that command to the backup servers.

When the primary server receives a command, it assigns a tentative sequence number, equal to the highest known committed sequence number plus one, sends it to all the replicas and waits for an acknowledgement from a majority.

After it receives a response from a majority of replicas, the primary server sends a commit message to all servers and executes the command.

3.3. Backup Servers

When a backup server receives a command, it just responds with a message informing that it is not the primary, since the primary is the only server responsible for processing commands.

When it receives a tentative message, it is only accepted if the sending server was the leader for that slot and if the backup server does not already have a tentative sequence number for that command from a more recent leader.

When a commit message is receives it is always accepted and the corresponding command is executed.

3.4. Cleanup

When a new leader is elected (i.e. the primary of slot n + 1 is different from the primary of slot n), it executes a cleanup procedure to check if there is unfinished job from the previous leaders.

First, it sends a listPendingRequests message to all replicas with the highest sequence number it knows to have been committed and waits for a majority of replies. The servers reply with their list of tentative commands with sequence number higher than the one sent by the new primary.

When a majority of responses is received, the new primary server proposes and commits all those sequence numbers using the same two-phase commit protocol described above. Once this procedure is completed, the new primary can start to assign new sequence numbers to commands received, as described in section 3.2.

4. Bank Client

The bank client executes an script that reads a document with all the petitions that the client will do to the bank and send all the petition to the bank.

First of all, we check if the number of arguments are correct, Second of all we identify the client, if everything is correct we can start reading the lines of the document that have all the petitions.

To send to the bank server each sequence, we need to analyze which operations want to do the client, after that depending on which type of operations we will send to the ClientFrontend to execute the operation and send it with a protobuf channel to the bank server. Finally we will wait until for the next operation.

5. Puppet Master

The project can be run using the Puppet Master, that receives as arguments the paths to the configuration file and to the Boney.exe, BankServer.exe and BankClient.exe executables. The Puppet Master parses the configuration file and launches all the processes with the necessary arguments (path to the configuration file and process id) using the correspondent executable file.