Boney Bank

Design and Implementation of Distributed Applications 2022-2023
IST (MEIC-A / MEIC-T / METI)
Project Description

Version 1.2

1 Introduction

The goal of the Boney Bank project is to develop a three tier system that emulates the behaviour of simple but reliably fault-tolerant bank application. Boney Bank has a first tier composed of clients who deposit and withdraw money from a banking account. The operations are submitted to a set of bank server processes, the second tier, that use primary-backup replication to ensure the durability of the account information. Finally, in the third tier, is the Boney system, a distributed coordination system that the bank server processes use to determine which one of the bank servers in the second tier is the primary server.

2 Architecture

2.1 Processes

Processes (both the bank and the Boney processes) never fail but may be "*frozen*". A frozen process does not process and/or acknowledge messages received from other processes and does not send messages. Frozen process may become "*normal*". The transition from "normal" to "frozen" and from "frozen" to"normal" is triggered via a control interface.

2.2 Perfect Channels

Perfect channels connect the processes in the system and ensure that a frozen process eventually receives and sends all messages transmitted when it was frozen. Perfect channels must queue messages and re-transmit them when needed to ensure that all processes eventually receive all messages that were sent to them.

All processes in the system communicate using perfect channels.

2.3 Failure Detection and Leader Election

Processes may be part of a "group". For each group, every member keeps a"*not-suspected*" or"*suspected*" flag, that captures its own guess about the state of other processes, i.e., if it believes other members of the group are "normal" or"frozen". The "suspected"/"not-suspected" may be inaccurate (for instance, a process may "suspect" that another process is "frozen" when it is not). How processes "guess" the state of other processes is described later in the text.

Each process keeps a guess of who is the group leader. The group leader is the process with lower ID that it is "not-suspected".

2.4 Paxos (partial implementation)

Paxos is implemented by the group of processes running the Boney service. Each process in a group is both a proposer, an acceptor, and a learner. Paxos members receive request from clients, order these requests, and execute each request (and respond to clients) as soon as the request has been ordered.

Clients send the request to all group members and may receive (identical) responses from all group members.

Members of a Paxos group operate as follows. Processes run multiple consensus instances, one after another. Each consensus instance outputs a set of commands, ordered in a deterministic way. This determines a total order of all commands. Learners process commands in this total order. I.e, a learner executes the first command output by instance 1 of consensus, then the second command output by instance 1, etc, until all commands output by instance 1 of consensus are executed. Then it executes the first command output by instance 2 of consensus, etc. After a command is executed, a reply is sent to the client.

When a proposer receives a command from a client they are stored in a list of "unordered" commands. As soon as instance-n of consensus is terminated, the commands that are in the set of accepted commands for instance-n are marked as "ordered". Then proposer initiates instance-$(n+1)$, proposing all commands in the unordered list that have not yet been marked as "ordered" (if the set is empty, it simply waits until there is at least on unordered command). The output of an instance is the accepted set of commands among the ones that may have been proposed for that instance. Note that different proposer may propose different set of commands for a given instance of consensus.

Proposers, acceptors, and learners run the classic Paxos algorithm as will be described in theory classes.

2.5 Boney

Boney is a service inspired by Chubby. Its role is to assign values to"slots". Slots and numbered from 1 to infinite. Every slot is assumed to have an initial value of "*null*".

Clients of Boney use a single command:

```
outvalue = compareAndSwap (slot, invalue).
```

The first command to be accepted for a given slot wins the slot and changes the value assigned to the slot from *null* to `invalue`. Clients that are ordered after the first command read the value proposed by the winner.

Boney is implemented using the Paxos algorithm above. Learners keep the value assigned for each slot and respond to the client with the value assigned to the slot (i.e., the first non-null assigned value). Note that the value returned by `compareAndSwap` is never null.

2.6 Bank

Bank is a distributed service, implemented by a group of processes that accepts three commands:

```
balance = deposit (value)
(value, balance) = withdrawal (value)
balance = readBalance ()
```

Clients send the request to all replicas of the service and receive a reply from at least one of them. If a `withdrawal` would bring the balance lower than zero, the operation is not executed (and the `value` returned is zero).

Bank is executed by a primary-backup protocol that goes through a series of consecutive time "*slots*". A Bank process executes for some time in slot n, then moves to slot $n + 1$, etc. In each slot, there is a primary process that is responsible for assigning sequence numbers to requests. Sequence numbers are assigned using a two-phase commit protocol. The primary sends the sequence number to all replicas as "*tentative*" and waits for an acknowledgement from a majority of replicas. A backup only acknowledges sequence numbers assigned in slot n if the sender was the primary for that slot and as not changed until the current slot. When the primary changes, messages from the old primary are no longer accepted or acknowledged. If the primary obtains a majority of acknowledgements, it sends a "commit" to the replicas. Commands are executed in the order of committed sequence numbers.

At the beginning of a "slot", each process checks who the is leader of the group for that slot. Note that different replicas may believe different processes to be the leader of the group. Then, they use the "Boney" service to elect a primary. They do so by executing

```
primary=compareAndSwap (slot, leader)
```

If the primary of slot n+1 is different from the primary of slot n, before it starts assigning sequence numbers to new commands it executes the following "*cleanup*" procedure:

- It sends a "`listPendingRequests (lastKnownSequenceNumber)`" to all other replicas, where "`lastKnownSequenceNumber`" is the highest sequence number the (new) primary knows to have been committed. Nodes only reply to this request after they have moved to the new slot and the request comes from the primary assigned for that slot.
- It then waits for a majority of replies.
- Using the replies it collects all sequence numbers that have been proposed by the previous primary (or primaries), but that may not have not been committed yet.
- Then it proposes and commits all those sequence numbers, using the same protocol used to commit new sequence numbers.
- Finally it starts assigning new sequence numbers to commands that have not been assigned sequence numbers yet.

If a process receives different sequence numbers to the same command from different primaries, it will accept and adopt the sequence number from the most recent slot. A sequence number that has been committed should never change, but can be proposed again by a new primary. This may happen if the new primary it is not sure if the sequence number was commited by the previous primary.

2.7 Bank Clients

Clients execute in a loop where they execute a sequence of "deposit" (`D amount`), "withdrawal" (`W amount`), and "readBalance" (`R`) requests. Each request has an unique identifier, that is a tuple ⟨`clientId`, `clientSequenceNumber`⟩. The sequence of operations executed by a client in each iteration of the loop is specified in a script. The script may also include a "wait" instruction (S milliseconds) in order to insert a waiting time between other commands. Clients should print out in their consoles the result of each operations that is concluded. When they receive replies from multiple bank servers, they should print out all replies received and the information of whether each server is replying as a primary or a backup server.

2.8 Time Slots

The system assumes that all processes have access to loosely synchronised clocks. Slots are attributed to (configurable) δ second intervals, starting at a given (configurable) instant, that is known by all participants at bootstrap. All processes maintain a timer that periodically call an handler at the beginning of each slot.

For most processes, "slots" are only used by the "control" plane of the project, to freeze/unfreeze processes and to change the output of the failure detector (i.e, to define which processes are suspected by other processes). Slots are *not* explicitly used in the code of the Paxos/ Boney implementation or by the Bank clients.

Bank processes are the exception to this rule. In Bank, a process needs to obtain a *lease* (with the help of Boney) to operate as primary during a given slot. At the end of each slot the lease expires and primary must renew the lease to continue to operate as primary. So, in the Bank operation, slots are explicitly used to elect and check the validity of primaries.

2.9 Failures

To simplify the project, processes never fail and eventually send/receive all messages. However, processes may *freeze* and delay processing and sending messages during one or more slots. Both Paxos/Boney and Bank services require a majority of non-frozen process to make progress. If a majority of group members is frozen in a given slot, the protocols will not make progress on that slot. Due to this, it is possible that a Bank server performs `compareAndSwap` to elect a primary for a slot but only obtains the response in a future slot (this may happen, if a majority of Boney servers is frozen). In this case, the Bank server will need to issue a new `compareAndSwap` to elect the primary for the current slot. Finally, it is suggested that Bank servers elect primaries for all slots, even for the slots that have expired; this makes sure all nodes have a consistent view of the sequence of primaries. For instance, if a process uses `compareAndSwap` to elect a primary for slot n but only receives the response in slot $n+2$, it will still elect a primary for slot $n+1$ before proposing a primary for the current slot (i.e., $n+2$).

2.10 Processes

A typical configuration of the system could be a set of 9 processes:

- 3 Paxos/Boney servers
- 3 Bank servers (Bank Servers are clients of Boney)
- 2 Bank clients
- 1 puppet master

2.11 Global Adversary/Scheduler

There is a shared configuration file that indicates:

- The identifiers and roles of each process (command P). You can assume that Boney and Bank server processes are listed before bank clients. For the case of Boney and Bank server processes, this command includes a URL where they can be contacted.
- For how many slots the system is supposed to operate (command S).
- The value of δ that specifies the duration of a slot in milliseconds (command D).
- The global "wall" time of the beggining of the first slot (command T)
- Furthermore, for each slot, the system specifies if each node (in the Boney and Bank server groups) is "normal" or "frozen" and "suspected" or "not-suspected" by each other member of its group (command F). A list of triples containing the process id, an N or F and a S or NS indicate the state of each of the set of processes with the lowest IDs in the system, i.e. the Boney and Bank processes.

To simplify the project, server state data is stored *exclusively* in volatile memory.

3 Implementation

The project should be programmed using C# and use gRPC for remote communication. For simplicity, it can be assumed that the configuration files and scripts are available on all nodes. It is also important to note that, although the scripts syntax should not be altered, additional parameters can be sent in the communication between servers.

4 Final Report

Students should prepare a final report describing the developed solution (max. 6 pages). In this report, students should follow the typical approach of a technical paper, first describing very briefly the problem they are going to solve, the proposed implementation solutions, and the relative advantages of each solution. Please avoid including in the report any information already mentioned in this project description. The report should include an explanation of the algorithms used and justifications for the design decisions. The final reports should be written using LaTeX. A template of the paper format will be provided to the students.

5 Checkpoint and Final Submission

The grading process includes an intermediate optional checkpoint and a mandatory final submission. For the checkpoint the students may submit a preliminary implementation of the project; if they do so, the checkpoint grade may improve their final grade. The goal of the checkpoint is to control the evolution of the implementation effort.

The checkpoint should include at least the implementation of Paxos/Boney. If the students have not implemented the Bank for the checkpoint (not required), Bank clients do not need to be running and Bank server just issue periodic `compareAndSwap` requests to Boney.

The final submission should include the source code (in electronic format) and the associated report (max. 6 pages).

6 Relevant Dates

- October 14^{th} - Electronic submission of the checkpoint code;
- October 17^{th} to October 21^{st} - Checkpoint evaluation during the lab sessions;
- November 4^{th} - Electronic submission of the final code and report.

7 Grading

The project grading will depend on a discussion at the end of the semester where all members of the groups must be present and where individual grades will be determined. That grade will depend on, besides the quality of the project, the individual performance in the discussion and the lecturer's evaluation.

The project grade (45% of the course's grade) is the *best* of the following two:

- Final_Project_Grade
- 85% of the Final_Project_Grade + 15% of Checkpoint_Grade

8 Cooperation among Groups

Students must not, *in any case*, see the code of other groups or provide their code to other groups. If copies of code are detected, all groups involved will fail the course.

9 "Época especial"

Students being evaluated during the "Época especial" will be required to do a different project and an exam. The "Época especial" project will be announced on the first day of the "Época especial" period, must be delivered on the day before last of that period, and will be discussed on the last day.