Option B: Paxos

ASD TP2 2018/2019

Manuel Henriques 42546, Sebastião Pamplona 42735

1. Introduction

The project consisted in implementing a *Paxos* protocol for replicating data on different servers. We used *Scala* to implement our project, and decided to structure of our implementation in two different actors: *Replica* and *Paxos* (*Paxos* being a child of *Replica*).

2. Overview

In this section, we cover the essentials of each component of the project.

2.1 Replica

A replica is responsible for creating the *Paxos* actor. It manages client requests (writes and reads) using Paxos to agree on the ordering of operations, executed throughout all replicas.
Replicas can also handle other replicas requests, such as add and remove a replica from the system, react to replicas' crash messages and to synchronize missed operations.

2.2 Paxos

The Paxos actor is responsible for the global decision making. Each replica must compute the same operation, decided by the current Paxos instance. Every actor acts has a Proposer, an Acceptor and a Learner.

2.3 Fault tolerance

Our solution supports a Fail-Stop model. As such, when a replica crashes it notifies another replica from the system. That replica then proposes, with higher priority, the removal of the failed replica.

3. Specification and Arguments

3.1 Replica

3.1.1 Handling requests

A replica is responsible for handling clients' requests, as well as other replicas' requests (add and remove replica). When a replica receives a request, it stores it in a queue of operations, because it can be handling an other request, meaning it is waiting for the current Paxos instance to decide the operation.
Since replicas' requests have priority over clients', they are stored in another queue, treated with higher priority.

3.1.2 Executing operations

Operations that do not modify the state machine are executed instantly, without being stored in the operation sequence, nor checking if there there are missing operations (weak consistency); in the context of this project, this only applies to the read operation.
On the other hand, operations that modify the state machine can be executed in two different cases: 1) when Paxos decides an operation and there are no missing operations in the operation sequence, or 2) when a replica has successfully recovered all of its missing operations.

3.1.3 Synchronizing missed operations

A decide message can be lost due to a few different reasons, resulting in a gap in the operation sequence.
Every time a replica receives a decide from Paxos (with an operation and its respective position), it verifies if there is a gap in the operation sequence, in order to safely execute the decided operation (this is done by simply comparing two integers). If a gap is detected, a replica starts a synchronizing process, asking

each replica for the missed operations. If all lost operations are retrieved successfully, then the replica executes all the missed operations, in order. Otherwise, another synchronization is done upon the next decide.

3.1.4 Adding and removing replicas

After a replica joins the system, it starts Paxos before synchronizing the past operations, in order to participate in Paxos quorums as soon as possible.

A replica that leaves the system is simply removed from the membership.

3.2 Paxos

(Bellow, each replicas' Paxos is simply referred to as replica, for simplicity).

3.2.1 Notion of Paxos instance

An instance is represented by an integer, inside each replica. So every data related to deciding what's the next operation to be executed, is associated with this integer, via a hashmap.

3.2.2 Liveness

Paxos does not guarantee liveness, due to the sequence number logic (i.e., if I (as a replica) received a *PrepareOk* from a majority, I am ready to send an *Accept* to the membership; if in between, another replica Proposes a value with a higher sequence number, my *Accept* will be ignored, triggering my *AcceptOK* timeout, forcing me to *Propose* my operation again, with a higher sequence number. If the same happens to the other replica, we can fall into an endless loop).

3.2.3 Deciding an operation

As soon as a replica receives an *AcceptOK* from a majority, an operation is ready to be decided, meaning that a message *Decide* is sent to all of the membership, including itself.

4. Evaluation

Our evaluation took place in an environment with 6 replicas and 6 clients. We created an actor for the client, as well as for the tester. The tester is the last one to run, tasking each client with a number of operations to request. Regarding *Paxos' PrepareOK* and *AcceptOK* timeout, we used 500ms and 1000ms, respectively.

4.1 Correctness

4.1.1 State machine replication

We tasked each client with 800 write operations each, in several rounds, checking after each one if the operation sequence was the same in every replica, as well as the state machine.

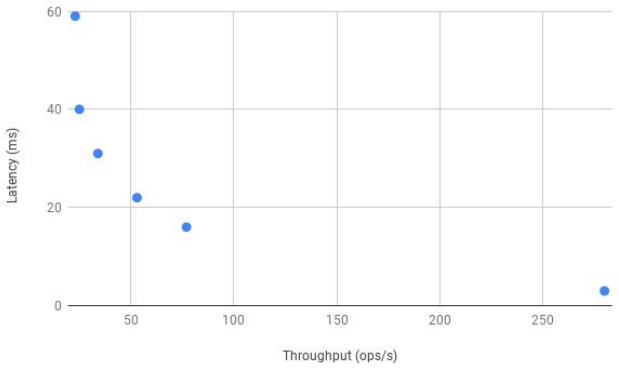
4.1.2 Membership

After some rounds, we experimented adding a new replica to the membership, and verifying that all the operations were synchronized successfully. Regarding removal, we used the same approach.

4.2 Performance

In terms of performance, we tasked each client with 50 write operations. A testing round consisted in assigning each client with 50 write operations, and waiting for all al them to finish, measuring the latency and throughput. We started with only one client, adding another after each testing round was over, until we reached the number of 6 clients.

After all 6 rounds were over, we plotted the latency vs the throughput of each round, to have a better notion of the effects of more operations in the server.



Graph 1. - Latency vs throughput of 50 write operations per client (the number of clients increases from right to left).

5. Conclusion

As a result of our evaluation, we were able to verify that Paxos circumvents the FLP, proving that the state machine was correctly replicated throughout all the replicas. We also concluded that as the number of clients increased, the performance of the server decreased exponentially. Maybe with Multi-Paxos, this decline would not be so drastic, given its performance optimization.

6. Pseudo-Code

(attached)

REFERENCES

[1] L. Lamport, Paxos Made Simple, 2001
[2] ASD Slides