Assignment 3: Paxos Replicated State Machine

Distributed Systems Class

1 Overview

In order to build an highly available service such as Facebook, one of the most important building blocks is to provide a way in which the underlying data is replicated and can be accessed even in the presence of failures. To this end, this assignment will require you to build a replicated state machine based on Paxos.

The focus of this assignment will be on the backend Paxos algorithms and in designing and building a highly reliable distributed storage system. More importantly, we would like to provide you with substantial flexibility so that you can choose any of the following options to demonstrate your work on the Paxos implementation. (This is to address concerns that the framework is inflexible and that layering complexity upon earlier assignments means that you are likely to spend some of your cycles fighting with either the framework or assumptions from earlier parts of the project. On a related note, we have received some valuable feedback regarding the framework, so thanks!)

- You can build on the code from Assignment 1 and make some RPC operation more reliable than what it would be with the code from Assignment 1. For instance, you can take a simple wall post RPC and make it reliable in the face of server failures by using Paxos. This would require you to take a single Facebook server and represent it now as a Paxos group of multiple servers. A client can then submit a wall post to any of the servers in the Paxos group and the Paxos group would use consensus algorithm to determine a consistent and reliable ordering of wall updates.

- You can build on the code from Assignment 2 and make the 2PC operations from assignment 2 more reliable, again by having each Facebook server replicated in a Paxos group. Note that this would require every 2PC operation (such as tx begin, rpc to make wall post, tx end, etc.) to become a Paxos operation on the corresponding Paxos group. This is strictly harder than the first option, so you might prefer to choose the first option over this.

- Or you can do this assignment as a standalone piece of work that does not build on assignments 1 or 2, use any language or framework of your choice, and build a highly available lock server to demonstrate your implementation of Paxos replicated state machine. More details on the API that the lock server should support will be provided at the end of this document.

2 Paxos Implementation

Irrespective of the option you choose to take, the core of the assignment is to develop a Paxos replicated state machine. The basic idea of the Paxos RSM is described in Section 3 of the "Paxos made simple" paper by Lamport. A quick summary of the implementation strategy is as follows:

- Model the service that you are implementing as a deterministic state machine whose behavior or output is simply a function of its previous state and any inputs received from the user or environment in the current step.
- Implement a multi-instance Paxos algorithm to come to a consensus on the sequence of input values that the service receives in a consistent and reliable way. That is, the i^{th} instance of Paxos determines the i^{th} input or command received by the service.
- From a client perspective, it can send the service request to any of the nodes that comprise the Paxos group. The node that receives the request tries to pass a Paxos instance using the received value. If it fails to pass it with the current instance, it tries to pass as the consensus value for a subsequent instance.
- Run a copy of the underlying state machine on each of the nodes that belongs to the Paxos group. Each state machine executes the i^{th} command once it is stable and after it has executed all previous commands. A command is stable if it is the chosen value for that instance of Paxos. Once the command is stable and the state machine has processed it, a response is sent back to the client.

You can organize your code in whatever mechanism that you think is appropriate. For example, you could have a single entity that serves all three roles of proposer, acceptor, and listener. The implementation of Paxos should be robust to message loss and node failures and should be able to make progress as long as a majority of nodes are online. Note that a Paxos node could be offline for a period of time, but when it comes back online, it can help constitute a majority for passing Paxos commands. (This is the traditional fail-stop-recover model that Paxos supports.)

3 Lock Service

For those of you who choose not to build on assignments 1 or 2, you can demonstrate the correctness of your Paxos implementation by developing a simple lock server as the replicated state machine. The lock server should have the following functionality:

- It manages the lock status for a number of locks.
- It supports a blocking *lock(x)* operation that returns when x is assigned to the requesting client. If x is currently assigned to a different client, the request is queued at the state machine. A reply is sent only when the lock becomes available and is assigned to the requesting client.
- It supports a *unlock(x)* operation that frees up the lock.

We would like you to focus primarily on the Paxos implementation, so you can make any simplifying assumptions that you feel appropriate with respect to the actual lock service exported by the replicated RSM. For instance, you can assume the following:

- the identity of the Paxos group members can be hardcoded at the client,
- the client to server communications are reliable, so you do not have to worry about issues such as atmost-once or atleast-once semantics for the delivery of client requests or responses.

4 Deliverables

4.1 Code

Students will need to provide the source code that implements the above requirements. The source code should be well-documented in order to ease grading and prevent misunderstandings about functionality. The groups will meet with the TA or instructor to demonstrate the code and explain how it works.

4.2 Write-up

In addition to the source code, a major component of your grade will be based on a project write-up, in which your group should explain the protocol and any relevant implementation details. You should include, at a minimum, the following:

- a detailed description of your group's implementation
- any assumptions you made
- how to use your implementation
- any outstanding issues
- anything else that you feel is important to discuss (e.g. quirks, interesting behavior, performance characteristics, etc)