Distributed System Final Project Proposal
Understandablility of Raft versus Multi-Paxos

Yin-Hsun Huang (yhh303)

October 2018

1 Introduction

Raft[2] and Multi-Creed Paxos[3] are among the most famous distributed consensus algorithm. One of the contributions claimed in the Raft paper is that the Raft consensus algorithm is much easier to and implementation. One of the goals is to see if this is true. Another interest of this project is to measure the performance of these two algorithms under different failure sequence.

2 Approach

This project is an extension of Lab2. All algorithms are implemented with Go language.

2.1 Multi-Paxos Implementation

Multi-Creed Paxos are notoriously hard to comprehend. I implemented all of the Remote Procedure Calls(RPCs) in [3] in Go language and tested it on the task in Lab 2 [1], a key-value store. The following setting works:

- 2 to 4 replicas, 1 leader, 3 to 7 acceptors
- 2 replicas, 2 leaders, 3 acceptors

I observed a scenario that the client might not receive a result for some of the key-value function calls in the two leaders scenario. This might be because two leaders are preempted, deactivated, and then activated again. The next first $\langle$*purpose*$\rangle$ message will then trigger the execution. The client, in this case, needs to resend the command. This is acceptable in the setting of this paper.

	Line of Code	File Count
Raft	812	5
Paxos	1706	17

Table 1: Lines of code and file count of each project

2.2 Program Structure

Replicas, Acceptors, and Leaders are independent programs. They communicate through grpc protocols. Each message is implemented as an RPC defined in file kv.proto. Each program is following the structure in [1], where there is a main loop handling all messages from various channels.

2.3 Liveness

The pseudo-code in paper [3] does not include timeout mechanism. So I implemented my own based on the description in Section 3. Leaders send a ping to each other periodically, and if they did not find an active leader, they would spawn the scout routine.

2.4 Understandability

Implementing multi-Paxos is like writing raft multiple times. This might indicate that the program structure I am using is not optimal. The pseudo-code in[3] is harder for me to grasp compare to the raft one [2]. But each role in Paxos is separated cleanly, so there is fewer concurrency problem I have to deal with.

You can see from table1 that I wrote more than double of the code in my Paxos implementation. This is partially due to the test I wrote. These tests help minimizing the problem when every program are tested in the Kubernetes environment. Another reason that the codebase is large is that there is a log of code providing the same functionality in each program.

I like the raft much because of the uniformity of each replica. Nevertheless, breaking the protocol into smaller pieces does make the testing easier to do.

3 Validation

The correctness is tested using a client, which read a value from key ”hello” and add random integer from 1 to 10 and compare_and_set with key ”hello”, expected value being previous value, and value being the sum.

The correctness of my implementation can be tested by the following setting:

Listing 1: Correctness testing

```
#!/bin/bash
```

```
./launch-tool/launch.py boot 2 2 3
go run ./client/ [ip]:[rep0,port] [ip]:[rep1,port]
```

where ip is the cluster [ip] of minikube, [rep0,port] is the port of replica0. You might need to wait for 30 second before running the client for leaders to send and receive message from acceptors.

3.1 Random Shutdown using Kubernetes

For the setting in list 1, we can kill at most one replica, one leader and one acceptor. You can kill process with:

Listing 2: Kill pods

```
#!/bin/bash
./launch-tool/launch.py kill acc0
```

References

[1] Distributed systems 2018 - lab 2.

[2] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, 2014.

[3] Robbert Van Renesse and Deniz Altinbuken. Paxos made moderately complex. *ACM Computing Surveys (CSUR)*, 47(3):42, 2015.