ID2203 Distributed Systems, Advanced CourseProject Preliminary Report

Chen, Zidi

February 2021

1 Basic Introduction

This project is to design, implement and test a partitioned and distributed in-memory key-value store. After the basic infrastructure is done, the operations of PUT, GET and CAS (compare and swap) should also be designed and implemented.

2 Current Progress

The design of the project architecture which will be shown in the following sections, including the design of system architecture and test. In addition, the environment of developing the project was also done. Since my computer's system is windows, I have installed a ubuntu virtual operating system on the virtual box. OpenJDK, sbt and IntelliJ IDEA were set up for the project development.

3 System Architecture

Model assumption

I assume that our model is partially synchronous, so it is a fail-noisy model. In this case, eventually perfect failure detector with perfect link is suitable.

multi-Paxos with leader

To solve the problem of replication, I consider that multi-Paxos with leader should be a suitable solution. This is the most powerful algorithm to handle replication that I know so far in the course. Compared with read-impose write-consult majority algorithm with multiple writers, Paxos ensures the validity, uniform agreement, integrity and termination. To improve the efficiency, I think that multi-Paxos with leader should be a good choice compared with the leaderless one.

To select the leader, I want to use gossip leader election which should be implemented in Assignment 6.

Partition

As figure 1 shows, in the project, I would like to have one server node as the leader in the multi-paxos algorithm. All the other nodes (servers) just connect to the leader. When the client puts a new key-value pair, the pair should be distributed to majority of server nodes through leader. When the client does the CAS operation, the result should also be updated in all the nodes. So actually I just planned one partition here. If there it is unreasonable or uncomfortable while implementing, I should modify it.

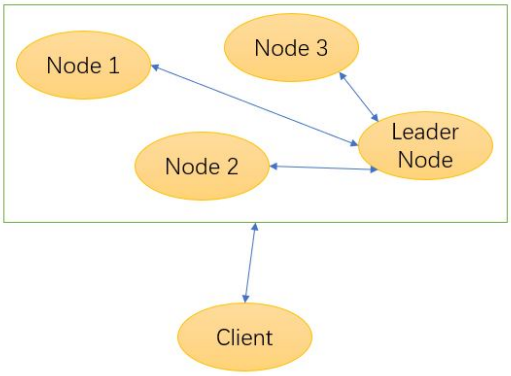

Figure 1: System partition

4 Tests

The tests should include GET, PUT and CAS operation, as well as replication and linearizability.

GET

1. get an exist key
2. get an non-exist key

PUT

1. put an non-exist key with value (add)
2. put an exist key with value (update)

CAS

1. test with a non-exist key
2. test with a correct key but non-correct old value
3. test with correct key and correct old value

replication
Let us start three nodes.

1. kill the leader node and do the put/get operation (two node left) in order to see if the system works fine with a new leader
2. restart a new node (now here are three nodes)
3. kill a node who might not be the leader (two nodes left) to see if the system works fine with one leader and one normal node
4. kill another node (one node left) to see the system should fail since there is no quorum

linearizability LIN
LIN was proven to be compositional in the lecture. So if LIN holds for one register, the system should be LIN. LIN(T) requires that there exists legal history S such that: 1. S is equivalent to T. 2. If o1 <T o2 then it must also be that o1 <S o2. It should be shown that S preserves the order of non-overlapping operations in T. So the test is designed as follows.

1. PUT, then CAS
2. PUT, then GET
3. PUT, then PUT
4. GET, then PUT
5. GET, then CAS
6. GET, then GET
7. CAS, then GET
8. CAS, then PUT
9. CAS, then CAS

Here I consider that CAS and PUT are write operations, GET is the read operation. The test cases should include the situations of read-read, read-write, write-read and write-write.

5 Conclusion

In conclusion, I plan to use eventually perfect failure detector with perfect link, multi-paxos algorithm in the project. For the partition, the replication degree is n (n nodes in total) and the I just have one partition. This might be fine for a small group of nodes, but it should not be suitable for larger number of nodes.