ID2203 –Distributed Systems, Advanced Course Project

Course Teacher:

Paris Carbone

Course Assistant:

Max Meldrum, Harald Ng

Xiaoying Sun

xsu@kth.se

2021-03-26

1. Abstract

The whole implementation originated from the final Sequence Paxos algorithm provided by our session “leader-based sequence Paxos”. The algorithm works in the asynchronous model but requires BallotLeaderElection which works in the partially synchronous model.

2. Infrastructure

The core need of this project can be transformed to guarantee the **safety** and **liveness** properties at the same time.

To guarantee the safety properties of the Sequence Consensus algorithm, the following assumptions should behold

1. FIFO perfect links
2. An eventual leader election that guarantees for any indication (response) event <Leader, L, n>, in which (l, n) is unique.

As for liveness, the leader election should satisfy,

3. If p is elected by <Leader, p, n>, then any previous leader q <Leader, q, m>. m < n should hold.
4. A leader should be admitted by a majority of processes for “a sufficient time”.

Perfect FIFO Link	The first assumption is easily met by using **Network** in Kompics is used as PerfectP2PLink with TCP, which also has FIFO property needed in Sequence Paxos.
Ballot Leader Election	**BLE2 Eventual Agreement** and **BLE3 Monotonic Unique Ballot** perfectly suit the liveness assumptions.
Sequence Paxos	The most important section, which guarantees that all the servers in the same group execute the same sequence of operations.
Eventually Perfect Failure Detector	EPFD sends heartbeats only to the replicas.
Best Effort Broadcast	Guarantees message delivery, used in perfect FIFO links.

3. KV-Store

KVService uses a simple Map to store (key, value) pairs and 3 simple operations named **GET**(read), **PUT**(write), and **CAS**(compare-and-swap).

Different from the project template, the header address is also added as one parameter of basic Operation, hence there is no need to repackage the operation message when using Sequence Consensus ports.

```
trait Operation extends KompicsEvent {
 def id: UUID;
 def key: String;
 def src: NetAddress
}
```

If a node receives a request to perform one of the three operations, it would forward it to the current leader where the request would be implemented and force all other nodes to learn the latest results. The fundamental solution is based on the Sequence Paxos algorithm.

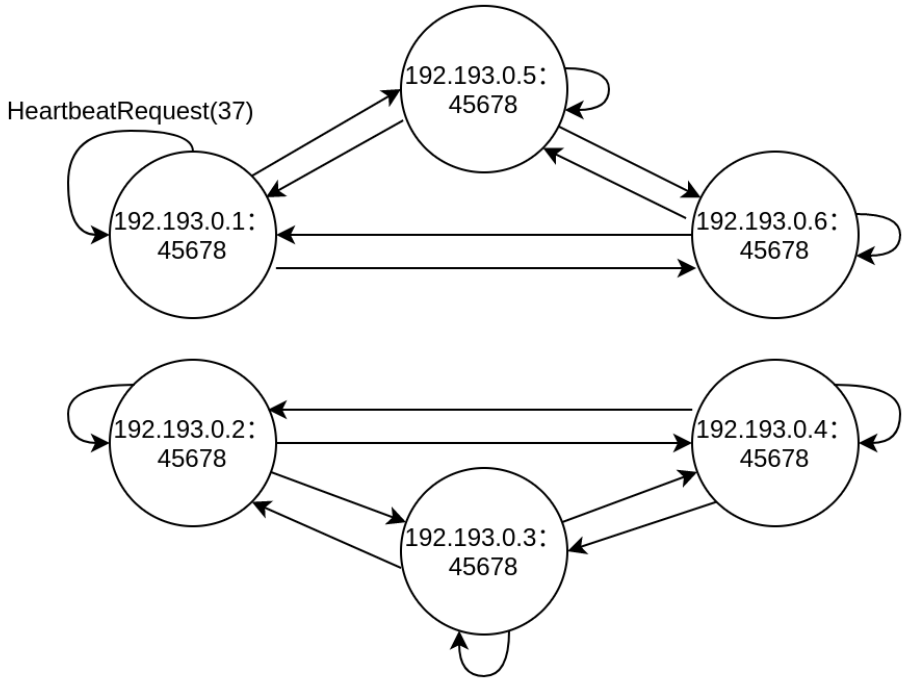
4. Consensus

Known as the ultimate issue of distributed systems, this part did take the most time.

Since this project is a partial-synchronous system, it is impossible to achieve consensus by relying on a perfect failure detector or an eventually perfect failure detector. At this point, each node only cared about its replication group, and EFPD sent a periodic heartbeat message only to its replicas.

We have 4 types of heartbeat message in total,

- **{HeartbeatReq, HeartbeatResp}**: BallotLeaderElection needs heartbeats to know when is the time to elect a new leader. In this way, the leader sends HeartbeatReq to check the highest ballot at the beginning of every new round, and every alive node responds to the leader by HeartbeatResp including its own highest ballot.
- **{HeartbeatRequest, HeartbeatReply}:** all nodes periodically suspect whether its replicas are still alive. Hence EPFD sends HeartbeatRequest to consult if the nodes in its suspected queue, and once HeartbeatReply is received, it will move this replica to the restored queue.



5. Test

The simulation and testing part implemented **6** servers(192.193.0.1:45678 - 192.193.0.6:45678) and the number of nodes in each group was random.

5.1 Get default value test

Scenario: This scenario is merely designed to test if KVService works well. The expected result should be all GET operations read the initial value successfully.

Result: Every key during (1, 10) exists, and GET returns the correct value (value=key + 10).

5.2 Linerizable test

Scenario: First use PUT to rewrite the default value as *key*, and If this operation works well, the refValue in CAS should equal to *key* instead of *(key+10)*. then if *(key%2 == 1)*, CAS will multiply its value by 10.

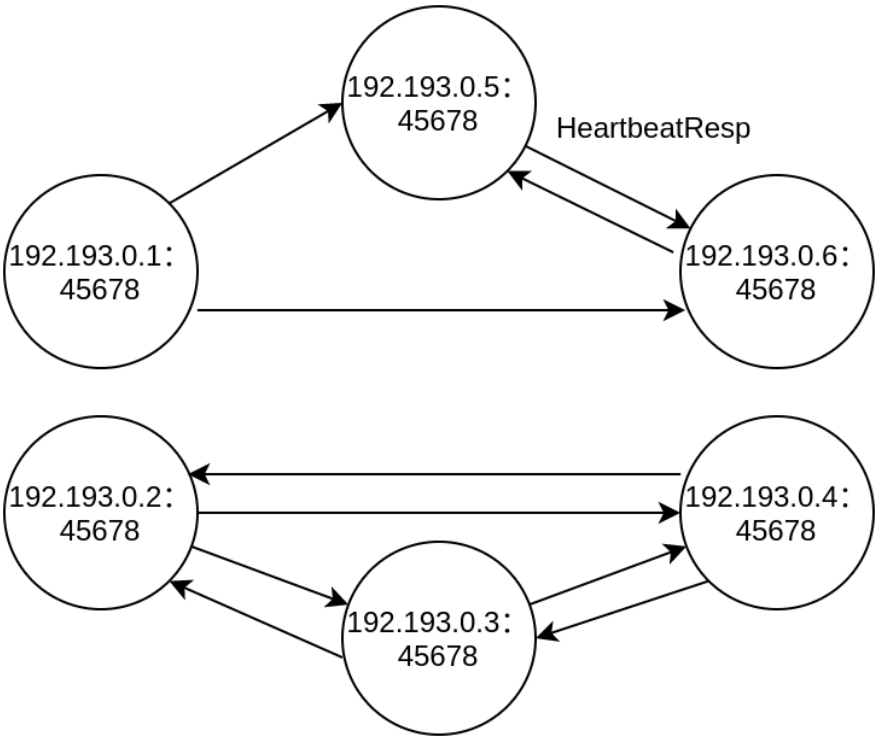
Result: If the key is even, the value equals to $|key|$; while if the key is odd, the value equals to $|key*10|$.

5.3 Work with one failed node test

Scenario: Send a specific Kill command to the first server(192.193.0.1) and kill one node. As long as the quorum is still alive, the server should still work as a reliable state machine and respond to the HeartbeatRequest in time.

Result:

```
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.1:45678),NetAddress(/192.193.0.6
:45678),TCP),HeartbeatResp(109,125166483)) from:NetAddress(/192.193.0.1:45678)
to:NetAddress(/192.193.0.6:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.5:45678),NetAddress(/192.193.0.6
:45678),TCP),HeartbeatResp(109,2008111646)) from:NetAddress(/192.193.0.5:45678)
to:NetAddress(/192.193.0.6:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.4:45678),NetAddress(/192.193.0.3
:45678),TCP),HeartbeatResp(109,3694090500)) from:NetAddress(/192.193.0.4:45678)
to:NetAddress(/192.193.0.3:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.2:45678),NetAddress(/192.193.0.3
:45678),TCP),HeartbeatResp(109,2628336166)) from:NetAddress(/192.193.0.2:45678)
to:NetAddress(/192.193.0.3:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.3:45678),NetAddress(/192.193.0.4
:45678),TCP),HeartbeatResp(109,3061380003)) from:NetAddress(/192.193.0.3:45678)
to:NetAddress(/192.193.0.4:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.6:45678),NetAddress(/192.193.0.5
:45678),TCP),HeartbeatResp(109,2721947693)) from:NetAddress(/192.193.0.6:45678)
to:NetAddress(/192.193.0.5:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.1:45678),NetAddress(/192.193.0.5
:45678),TCP),HeartbeatResp(109,125166483)) from:NetAddress(/192.193.0.1:45678)
to:NetAddress(/192.193.0.5:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.4:45678),NetAddress(/192.193.0.2
:45678),TCP),HeartbeatResp(109,3694090500)) from:NetAddress(/192.193.0.4:45678)
to:NetAddress(/192.193.0.2:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.2:45678),NetAddress(/192.193.0.4
:45678),TCP),HeartbeatResp(109,2628336166)) from:NetAddress(/192.193.0.2:45678)
to:NetAddress(/192.193.0.4:45678)
01/01 00:01:54 TRACE[ScalaTest-run-running-OpsTest] s.s.k.s.c.i.P2pSimulator -
delivered network
msg:NetMessage(NetHeader(NetAddress(/192.193.0.3:45678),NetAddress(/192.193.0.2
:45678),TCP),HeartbeatResp(109,3061380003)) from:NetAddress(/192.193.0.3:45678)
to:NetAddress(/192.193.0.2:45678)
```


192.193.0.5：
45678
HeartbeatResp
192.193.0.1：
45678
192.193.0.6：
45678
192.193.0.2：
45678
192.193.0.4：
45678
192.193.0.3：
45678