Quorum Report

1. Implementation

1) Entity.
    a) Data: key, value
    b) Node: id, address (HOST:PORT)
    c) Clock: node, clock (vector clock)
    d) Log: clock (map), data
    e) Heartbeat: node(from), node(leader), operation, data, clocks, logs
2) Service.
    a) read (Data) returns (Data).
       Send key return value
    b) add_update (Data) returns (Heartbeat).
       Send Data and increase its own clock, then broadcast to all replicas
    c) send_heartbeats (Heartbeat) returns (Heartbeat)
       Send "INIT", "SYNC-LOG", "UPDATE", "READ", "RE_ELECTION", "IWON", "REDIREACT-TO-LEADER", "COMMIT" operations
    d) Send_quorum_request (Heartbeat) returns (Heartbeat)
       Check if data can be committed or read
    e) leaderElection (Heartbeat) returns (Heartbeat)
       Elect leader
3) Work flow.

START -> assign timestamp as id -> send heartbeat with "INIT" operation to all replicas

If no others is alive, make itself the Leader

If some replicas are alive, ask who is the Leader and synchronize the store by comparing the difference between local logs and Leader's logs. Using Dequeu to store logs since it is efficient reading from last. The Leader will send back the logs that vector clock less than the vector clock of the log sent from the new alive replica. (This is how I handle the crash, when a node is back online, it will synchronize all changes from Leader by comparing the logs, and tell every node it is available)

-> save all available replicas

READ -> send heartbeat with "READ" operation to Nr replicas and check if the result is the same.

I don't implement this totally randomly, since it may be in an infinite loop. So I calculate all the combinations of Nr replicas from all the replicas, then try one by one. If there are no consistent results, then cannot read.

UPDATE
-> if conflict!!! -> send “SYNC_LOG” then do followings
-> if it is Leader
-NO-> redirect to Leader
-YES-> follow below
-> send heartbeat with “UPDATE” operation to all replicas
if it is ready to update, respond heartbeat with “READY” operation
-> if at least Nw respond “READY”
-No-> Update fail
-Yes->commit change, update clock and store log -> send heartbeat with “COMMIT” operation -> replicas are received “COMMIT”, commit change, update clock and store log, respond “COMMIT_FINISH” -> Leader update clock when receive the “COMMIT_FINISH”

RE_ELECTION: trigger by “READ”, “UPDATE”, “INIT” heartbeat
-> if Leader is unresponsive -> send heartbeat with “RE_ELECTION” operation to all replicas -> the replica has the largest id (timestamp) will broadcast the “IWON” heartbeat, other will send “OK”-> once any replica receive “IWON”, the new Leader is elected.

4) Test on Docker. Details in test1/2/3.docker.log.txt
   a) Single replica available, try to update (test1)

```
docker build -t quorum_t1 -f test1.Dockerfile .
docker run -d -p 7000:7000 --name quorum_t1  quorum_t1
```

7000 is up -> send INIT heartbeat (from, clock) to 7001 and 7002 -> timeout

quorum_t1 quorum_t1
EXITED (0)

```
------localhost:7000's TERMINAL------
---HEARTBEAT TO localhost:7001---
from {
  id: 1649003016649
  addr: "localhost:7000"
}
operation: "INIT"
clocks {
  node {
    addr: "localhost:7000"
  }
}
clocks {
  node {
    addr: "localhost:7001"
  }
}
clocks {
  node {
    addr: "localhost:7002"
  }
}

---INITIAL HEARTBEAT ERROR---
UNAVAILABLE: io exception
--- 2.483s ---
```

quorum_t1 quorum_t1
EXITED (0)

```
---HEARTBEAT TO localhost:7002---
from {
  id: 1649003016649
  addr: "localhost:7000"
}
operation: "INIT"
clocks {
  node {
    addr: "localhost:7000"
  }
}
clocks {
  node {
    addr: "localhost:7001"
  }
}
clocks {
  node {
    addr: "localhost:7002"
  }
}

---INITIAL HEARTBEAT ERROR---
UNAVAILABLE: io exception
--- 0.051s ---
```

Current no other available replicas, so the leader is localhost 7000
Update request broadcast to 7001 and 7002 -> timeout -> cannot commit

quorum_t1 quorum_t1
EXITED (0)

```
---AVAILABLE REPLICAS---
{}
LEADER:
id: 1649003016649
addr: "localhost:7000"

Starting server on port 7000
Server started!
---UPDATE REQUEST---
key: "testData"
val: "10"

---HEARTBEAT TO localhost:7001---
from {
  id: 1649003016649
  addr: "localhost:7000"
}
operation: "UPDATE"
data {
  key: "testData"
  val: "10"
}

---localhost:7001 NOT READY TO UPDATE---
UNAVAILABLE: io exception
--- 0.027s ---
```

quorum_t1 quorum_t1
EXITED (0)
LOGS IN

```
operation: "UPDATE"
data {
  key: "testData"
  val: "10"
}

---localhost:7001 NOT READY TO UPDATE---
UNAVAILABLE: io exception
--- 0.027s ---

---HEARTBEAT TO localhost:7002---
from {
  id: 1649003016649
  addr: "localhost:7000"
}
operation: "UPDATE"
data {
  key: "testData"
  val: "10"
}

---localhost:7002 NOT READY TO UPDATE---
UNAVAILABLE: io exception
--- 0.033s ---

---NOT READY TO COMMIT---
Update fail
localhost:7000 current clock{localhost:7000=0, localhost:7001=0, localhost:7002=0}

Update Fail: Less than 2 nodes are alive
```

b) Two replicas are available (7000 is leader, 7001 is alive, 7002 is crashed), commit successfully (test2)

```
docker build -t quorum_t2 -f test2.Dockerfile .
docker run -d -p 8000:7000 --name quorum_t2  quorum_t2
```

update data on 7000 flow:
if 7000 is the leader ?
-No-> redirect to leader
-Yes-> send quorum to all replicas asking if ready to commit
If at least Nw replicas respond “READY” ?
-No-> update fail
-Yes-> commit change, update clock and store in logs ->send “COMMIT” to all replicas

quorum_t2 quorum_t2
EXITED (0)

send update request to 7001

```
---HEARTBEAT TO localhost:7001---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "UPDATE"
data {
  key: "testData"
  val: "10"
}
```

7001: receive update request

```
---RECEIVED QUORUM REQUEST---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "UPDATE"
data {
  key: "testData"
  val: "10"
}
```

send “ready” back to 7000

```
---localhost:7001 READY TO UPDATE---
--- 0.456s ---

---HEARTBEAT TO localhost:7002---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
```

quorum_t2 quorum_t2
EXITED (0)

```
}

---RECEIVED QUORUM REQUEST---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "UPDATE"
data {
  key: "testData"
  val: "10"
}

---localhost:7001 READY TO UPDATE---
--- 0.456s ---
```

7002: connect error not ready to update

```
---HEARTBEAT TO localhost:7002---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "UPDATE"
data {
  key: "testData"
  val: "10"
}

---localhost:7002 NOT READY TO UPDATE---
UNAVAILABLE: io exception
--- 0.043s ---
```

quorum_t2 quorum_t2
EXITED (0)

7000 and 7001 are ready to commit

7000 coomit update and write in local log with current clock

```
---READY TO COMMIT---
---LOCAL LOGS---
[clocks {
  key: "localhost:7000"
  value {
    node {
      addr: "localhost:7000"
    }
    clock: 1
  }
}
clocks {
  key: "localhost:7001"
  value {
    node {
      addr: "localhost:7001"
    }
  }
}
clocks {
  key: "localhost:7002"
  value {
    node {
      addr: "localhost:7002"
    }
  }
}
data {
  key: "testData"
  val: "10"
}
```

quorum_t2 quorum_t2
EXITED (0)

send 7001 “commit” message

```
---HEARTBEAT TO localhost:7001---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "COMMIT"
data {
  key: "testData"
  val: "10"
}
clocks {
  node {
    addr: "localhost:7000"
  }
  clock: 1
}
clocks {
  node {
    addr: "localhost:7001"
  }
}
clocks {
  node {
    addr: "localhost:7002"
  }
}
```

quorum_t2 quorum_t2
EXITED (0)

7001: recieve “commit” message

```
---RECEIVED QUORUM REQUEST---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "COMMIT"
data {
  key: "testData"
  val: "10"
}
clocks {
  node {
    addr: "localhost:7000"
  }
  clock: 1
}
clocks {
  node {
    addr: "localhost:7001"
  }
}
clocks {
  node {
    addr: "localhost:7002"
  }
}
```

quorum_t2 quorum_t2
EXITED (0)

commit change, update local clock and store log

```
---CURRENT CLOCKS---
{localhost:7000=1, localhost:7001=1, localhost:7002=0}
---LOCAL LOGS---
[clocks {
  key: "localhost:7000"
  value {
    node {
      addr: "localhost:7000"
    }
    clock: 1
  }
}
clocks {
  key: "localhost:7001"
  value {
    node {
      addr: "localhost:7001"
    }
    clock: 1
  }
}
clocks {
  key: "localhost:7002"
  value {
    node {
      addr: "localhost:7002"
    }
  }
}
data {
  key: "testData"
  val: "10"
}
]
```

quorum_t2 quorum_t2
EXITED (0)

```
---COMMIT RESPONSE---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "COMMIT_FINISH"
data {
  key: "testData"
  val: "10"
}
```

commit finish

```
clocks {
  node {
    addr: "localhost:7000"
  }
  clock: 1
}
clocks {
  node {
    addr: "localhost:7001"
  }
  clock: 1
}
clocks {
  node {
    addr: "localhost:7002"
  }
}

---CURRENT CLOCKS---
{localhost:7000=2, localhost:7001=1, localhost:7002=0}
--- 0.187s ---
```

7000: recieve “commit_finish”, update clock

quorum_t2 quorum_t2
EXITED (0)

```
{localhost:7000=2, localhost:7001=1, localho
--- 0.187s ---
```

fail to connect to 7002

```
---HEARTBEAT TO localhost:7002---
from {
  id: 1649010293047
  addr: "localhost:7000"
}
operation: "COMMIT"
data {
  key: "testData"
  val: "10"
}
clocks {
  node {
    addr: "localhost:7000"
  }
  clock: 1
}
clocks {
  node {
    addr: "localhost:7001"
  }
}
clocks {
  node {
    addr: "localhost:7002"
  }
}
```

quorum_t2 quorum_t2
EXITED (0)
LOGS INS

```
  }
}
```

update suceessfully 7000 and 7001 can commit

```
---localhost:7002 COMMIT FAIL---
UNAVAILABLE: io exception
--- 0.033s ---

update successfully
localhost:7000 current clock{localhost:7000=2, localhost:7001=1, localhost:7002=0}

Update successfully, no conflict
```

c) Read (test2)
-> try all potential combination of Nr of all replicas
First try (7000, 7002)

quorum_t2 quorum_t2
EXITED (0)

```
---READ REQUEST---
---LOCAL STORE---
testData:10
---ASK REMOTE STORE---
1 TIME(S)
TRY 2NODES: [localhost:7000, localhost:7002]
---HEARTBEAT TO localhost:7000---
from {
  id: 1649015237579
  addr: "localhost:7001"
}
operation: "READ"
data {
  key: "testData"
}

---RECEIVED QUORUM REQUEST---
from {
  id: 1649015237579
  addr: "localhost:7001"
}
operation: "READ"
data {
  key: "testData"
}

--- 0.05s ---
```

read request, local value is 10

first try 7000 and 7002

7000 recieve request, respond value

quorum_t2 quorum_t2
EXITED (0)

```
1 TIME(S)
TRY 2NODES: [localhost:7000, localhost:7002]
---HEARTBEAT TO localhost:7000---
from {
  id: 1649015237579
  addr: "localhost:7001"
}
operation: "READ"
data {
  key: "testData"
}

---RECEIVED QUORUM REQUEST---
from {
  id: 1649015237579
  addr: "localhost:7001"
}
operation: "READ"
data {
  key: "testData"
}

--- 0.05s ---

---HEARTBEAT TO localhost:7002---
from {
  id: 1649015237579
  addr: "localhost:7001"
}
operation: "READ"
data {
  key: "testData"
}

---READ ERROR---
UNAVAILABLE: io exception
```

7002 is not response

Second try (7000, 7001)

quorum_t2 quorum_t2
EXITED (0)

```
2 TIME(S)
TRY 2NODES: [localhost:7000, localhost:7001]
---HEARTBEAT TO localhost:7000---
from {
  id: 1649015237579
  addr: "localhost:7001"
}
operation: "READ"
data {
  key: "testData"
}

---RECEIVED QUORUM REQUEST---
from {
  id: 1649015237579
  addr: "localhost:7001"
}
operation: "READ"
data {
  key: "testData"
}

--- 0.037s ---

10
```

second try

only send 7000, since 7001 is itself

value is consistent

d) Failure handle 7002 is crashed, when it back online (test2)

quorum_t2 quorum_t2
EXITED (0)

```
---LOCAL LOGS---
[clocks {
  key: "localhost:7000"
  value {
    node {
      addr: "localhost:7000"
    }
    clock: 2
  }
}
clocks {
  key: "localhost:7001"
  value {
    node {
      addr: "localhost:7001"
    }
  }
}
clocks {
  key: "localhost:7002"
  value {
    node {
      addr: "localhost:7002"
    }
    clock: 1
  }
}
data {
  key: "testData"
  val: "10"
}
```

send sync to Leader with the last log in 7002

quorum_t2 quorum_t2
EXITED (0)

```
---SYNC FROM LEADER---
---STORE---
{testData=10}
---CLOCK---
{localhost:7000=2, localhost:7001=0, localhost:7002=1}
---LOGS---
[clocks {
  key: "localhost:7000"
  value {
    node {
      addr: "localhost:7000"
    }
    clock: 2
  }
}
clocks {
  key: "localhost:7001"
  value {
    node {
      addr: "localhost:7001"
    }
  }
}
clocks {
  key: "localhost:7002"
  value {
    node {
      addr: "localhost:7002"
    }
    clock: 1
  }
}
data {
  key: "testData"
  val: "10"
```

send logs that their clock of 7000 greater than the clock from 7002

data was commited in 7000 and 7001

quorum_t2 quorum_t2
EXITED (0)

```
Starting server on port 7002
Server started!
---READ REQUEST---
---LOCAL STORE---
testData:10
---ASK REMOTE STORE---
1 TIME(S)
TRY 2NODES: [localhost:7001, localhost:7000]
---HEARTBEAT TO localhost:7001---
from {
  id: 1649015249548
  addr: "localhost:7002"
}
operation: "READ"
data {
  key: "testData"
}

---RECEIVED QUORUM REQUEST---
from {
  id: 1649015249548
  addr: "localhost:7002"
}
operation: "READ"
data {
  key: "testData"
}

--- 0.039s ---

---HEARTBEAT TO localhost:7000---
from {
  id: 1649015249548
  addr: "localhost:7002"
}
```

read request to 7002

quorum_t2 quorum_t2
EXITED (0)

```
  id: 1649015249548
  addr: "localhost:7002"
}
operation: "READ"
data {
  key: "testData"
}

--- 0.039s ---

---HEARTBEAT TO localhost:7000---
from {
  id: 1649015249548
  addr: "localhost:7002"
}
operation: "READ"
data {
  key: "testData"
}

---RECEIVED QUORUM REQUEST---
from {
  id: 1649015249548
  addr: "localhost:7002"
}
operation: "READ"
data {
  key: "testData"
}

--- 0.046s ---


10
```

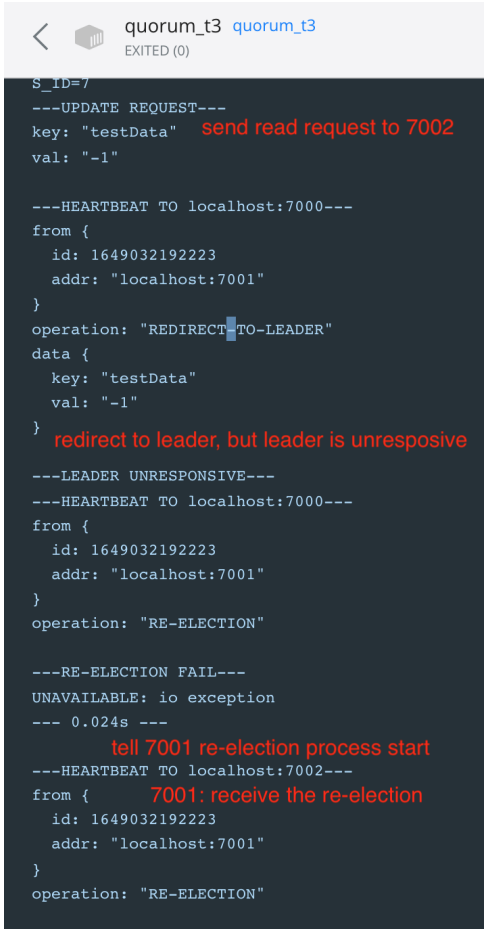
get correct result

If the SYNC-LOG message is lost, it will synchronize again when you request read or update to 7002. Especially, during update, it will send update requests again after asking synchronization from Leader. The workflow and logs are the same as above.

e) Re-election (test3)

```
docker build -t quorum_t3 -f test3.Dockerfile .
docker run -d -p 8000:7000 --name quorum_t3  quorum_t3
```

Start three server, 7000, 7001, 7002
Then 7000 is crashed, send update request to 7002
Re-elect leader then try to update



2. Improvement
   1. Leader should send a heartbeat every fixed period of time rather than only when an event happens. So when it handles read or update

operations, it doesn't need any re-election process that causes latency and increases the chance of losing messages.

2. Store, clocks and logs all should be stored in a local file. So that node can easily restore from the state it was crashed.