Assignment #3

Team members: Kaixuan Gao, Baojia Zhang, Yuxiao Guo.

How to compile/build source code

We used NetBeans IDE to development the project. Please use NetBeans to compile source code.

How to deploy and test our system

We used Docker to test our system. And we used TCP discovery to track node IP address. Just like assignment #2, firstly we need to create a m4.large docker machine and open 5 port from 8080 to 8084.

And then edit the *runserver.sh*

```
#!/bin/bash
#TCP Server -centralized node directory
java -jar GenericNode.jar ts 4410
#TCP Server -KV store
java -jar GenericNode.jar ts 1234 <nodedirectory IP> 4410 <node amount>
```

Please ***note*** that we use an additional parameter to tell the system how many node will be added in it. For instance, if you want a 3-nodes system, then replace this parameter with 3.

Then build and run nodedirectory cantainer and find its virtual IP address and re-edit the runserver.sh. Then build and run kvstore container. When you launch kvsotre container, please use command like this:

```
$docker run -p 8080:1234 -d --rm kvstore
$docker run -p 8081:1234 -d --rm kvstore
$docker run -p 8082:1234 -d --rm kvstore
```

Wait for ~30 seconds to let the system finish election. To find out which node is leader, please use command:

```
$docker logs <container ID>
```

If you see content like below, then it is a follower

```
Refresh timer!
Keep being follower
Keep being follower
Keep being follower
Keep being follower
```

If you see content like below, it is the leader

```
start vote: Sun Dec 17 21:16:02 PST 2017
pool-2-thread-1 Request vote from: rmi://10.0.0.155:1100/RequestVote
pool-2-thread-1 End.
Finished all vote requests!
election canceled: false
Leader has been elected: 10.0.0.115
pool-3-thread-1 Send heartbeat to:
rmi://10.0.0.155:1099/AppendEntries
```

Attention: Due to code bugs, sometimes there will be more than one leader in the system, in this case, please keep one leader and kill the others, and then relaunch other containers that have been killed.

Now that we know which one is leader, we can re-edit ***bigtest_tc.sh***, replace the IP with m4.large IPV4 address and port number with 808x which leads to leader. And then we can test.

For some reason we do not know, sometimes the redirection to leader function does not work in Docker. **But when we use 3 computers to test the system in a local area network, the redirection function actually works.** (So we suggest that testing the system locally would be better and easier to see what happens...)

Now kill one of the ***follower*** containers and relaunch it, after ~ 40 second, all data has been brought back, you can check use ***store*** command to that follower. But if you kill ***leader***, the system will re-elect the leader and maybe elect more than one leader, like we discuss above, in that case, keep one leader and kill the others and relaunch them.

Attention: Everytime when you want to change the node amount of the system, like expanding 3 nodes to 5 nodes, please re-edit the ***runserver.sh*** and change node amount. And relaunch the nodedirectory cantainter.

What to test

Our system works well on log replication and commit part, and we can make sure when node comes offline and then comes back, all data will be brought back, though the system lacks rubuestness in election part as it sometimes will have more than one leader.

All nine features are achieved:

F1. Randomized election timeout when leader doesn't respond (basically how RAFT starts up)

When the system first comes online, every node will publish its IP to node directory server, and then block until it gets specified amount of nodes IP address from node directory. And then the node sleeps for a random time, begin as a follower and start listening to heartbeat, we develped a recursive ***setTimer()*** function to produce random election timeout. You can check it in ***Follower.java*** class.

```
    public void setTimer() {
        Timer electionTimer = new Timer();
        TimerTask electionTask = new TimerTask() {
            int prevHb = appendEntriesImpl.heartbeatCount;
            @Override
            public void run() {
                if (appendEntriesImpl.heartbeatCount == prevHb) {
                    hasHeartbeat = false;
                    System.out.println("Start election!");
                }
                else {
                    setTimer();
                    System.out.println("Refresh timer!");
                }
            }
        };
        //Random set timer
        int randomTimeout = new Random().nextInt(Main.maxELectionTimeout) %
                (Main.maxELectionTimeout - Main.minELectionTimeout + 1) +
Main.minELectionTimeout;
        electionTimer.schedule(electionTask, randomTimeout);
    }
```

F2. Leader election within the system

We achieved this feature, but sometimes election will leads to more than one leader. But in most case we can make sure there will be only one leader.

F3. Leader heartbeat to followers

& F8. Use of multiple threads for AppendEntries.

We combine heartbeat function and appendEntries function, and set a heartbeat interval as 300 milliseconds. We use a ***thread pool*** to operate ***AppendEntries()*** to multiple nodes. You can check it in ***Leader.java*** class.

```
ExecutorService executor = Executors.newFixedThreadPool(serverNum);
        for (int i = 0; i < serverNum; i++) {
             String ip = Main.servers.get(i).ip;
             nextIndex.put(ip, Main.logs.size());
             matchIndex.put(ip, 0);
             if (!ip.equals(leaderId)) {
                 Runnable worker = new AppendEntriesThread(ip);
                 executor.execute(worker);
             }
         }
```

F4. Log replication across multiple nodes

We achieved this feature. You can check this in ***Follower.java*** class and ***Leader.java*** class.

F5. State machine commits (logs applied to KV stores for put and del)

We achieved this feature. You can check this in ***Follower.java*** class, ***Leader.java*** class and ***TcpServerThread.java*** class. After all logs concurrent operation, we commits the logs by two functions:

```
    public static void leaderCommit(String[] strings) {
        if (strings.length > 3 && strings[3].equals("put")) {
            sout.println(Main.kvstore.put(strings[4], strings[5]));
        }
        else if (strings.length > 3 && strings[3].equals("del"))
            sout.println(Main.kvstore.del(strings[4]));
        synchronized (flag) {
            flag.notify();
        }
    }

    public static void followerCommit(String[] strings) {
        if (strings.length > 3 && strings[3].equals("put"))
            Main.kvstore.put(strings[4], strings[5]);
        else if (strings.length > 3 && strings[3].equals("del"))
            Main.kvstore.del(strings[4]);
    }
```

F6. What type of membership tracking is used, how to configure it.

And we used TCP discovery to track node IP address. For more information, check **How to deploy and test our system** above.

F7. Use of multiple threads for RequestVote (optional)

We use a ***thread pool*** to operate ***RequesVote()*** to multiple nodes. You can check it in ***Candidate.java*** class.

F9. Sending put and del commands to leader node

Actually we did more than that, we also achieved redirection of command from follower to leader. You can test this function locally if it does not work on AWS...