Spring 2022, Raft Assignment

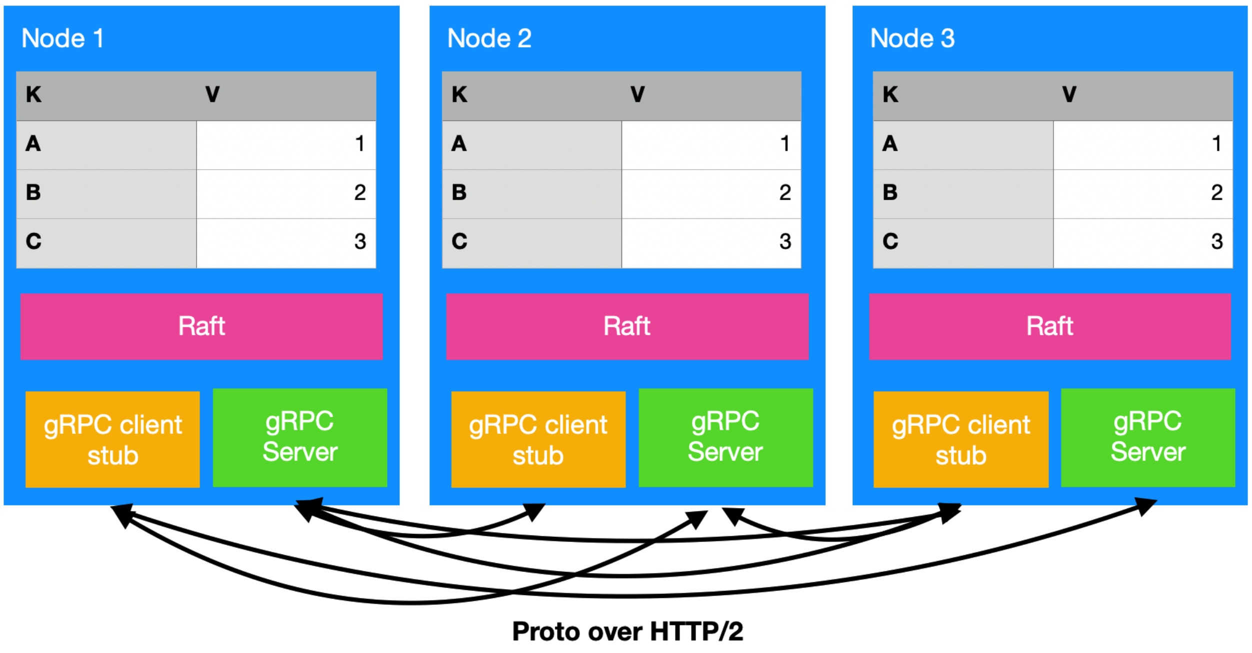
The Chinese University of Hong Kong

Prepared by Chris Liu

Graded by Chenxia Han

January 24, 2022

Contents



1 Introduction

In this assignment, you will build a **distributed replicated (key-value) map** using Raft [1]. Follow the Raft paper to implement and pass all the test cases in the grader. We thank Shimin Wang from Carnegie Mellon University when preparing this assignment.

For the details of Raft please refer to

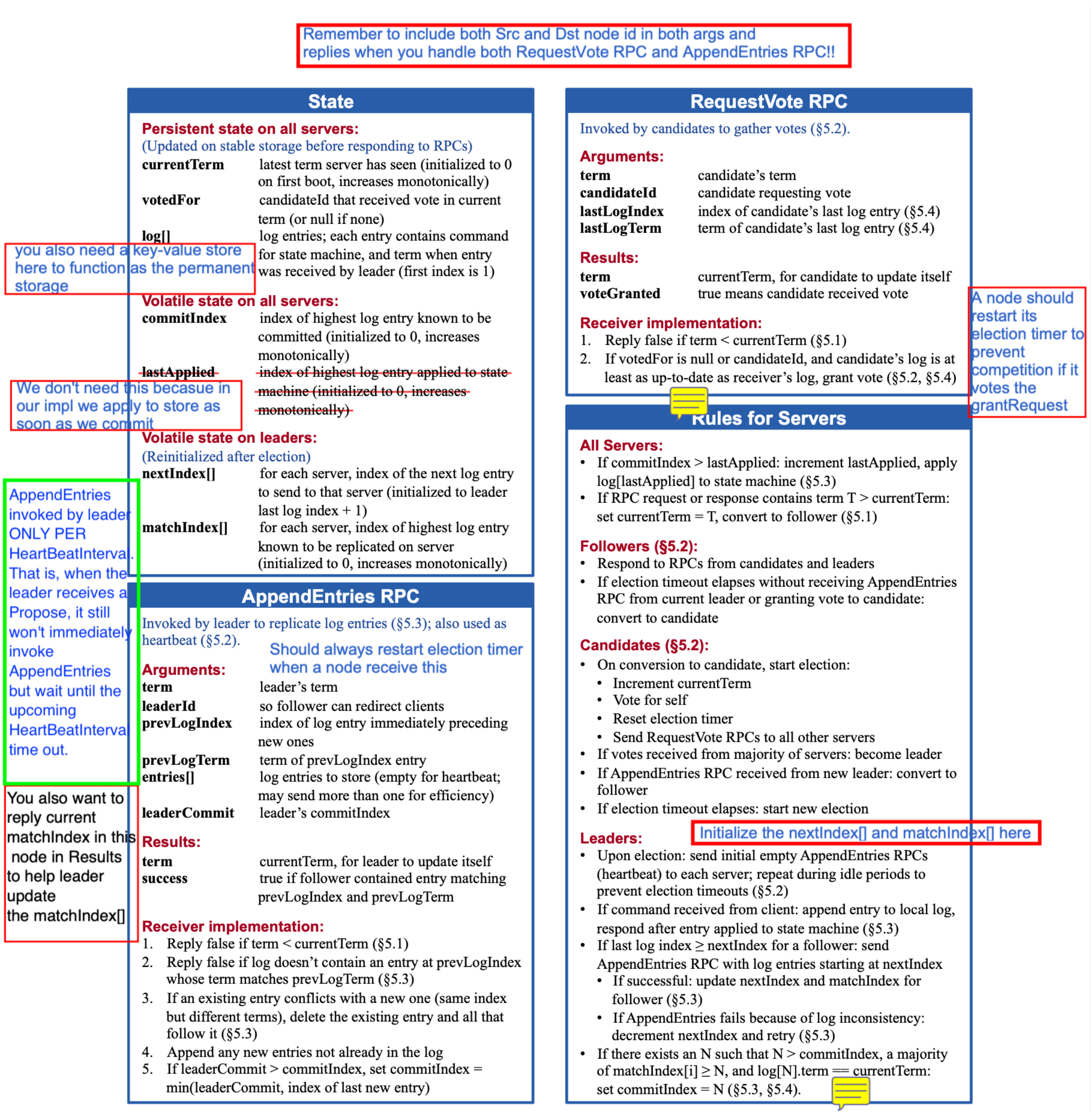
- The detail of Raft presented by their authors: `https://youtu.be/YbZ3zDzDnrw` and `https://youtu.be/vYp4LYbnnW8`
- The Raft extended paper annotated by Shimin Wang, which is included in the assignment package.
- The official web site of Raft, `http://raft.github.io`. It provides a good interactive demo with the Raft consensus protocol.

You do not need to handle all cases in the full Raft paper. Specifically,

- You don't need to randomize the election timeout. The election timeout value would be changed by the grader at run-time to simulate different situations.

- You don't need to implement log compaction and snapshotting.
- We don't need to handle cluster membership changes.
- No disk is considered in this implementation. Your key-value map is in memory. When you commit any log, you should directly apply that operation, committed will be treated equally as applied. Consequently, the `lastApplied` operation in the Raft paper is unnecessary to you.

Name	Description
`bin/`	Student-compiled binaries and grading tools binary.
`scripts/`	Grading shell scripts **(Don't touch)**.
`tests/`	Source code of grading tools. (**Don't touch**).
`yourCode/`	Your working directory
`yourCode/src/main/java/RaftRunner.java`	**For Java**. Write all your code in this file if you use Java (**Work on it**).
`yourCode/main.go`	**For Go**. Write all your code in this file if you use Go (**Work on it**).
`yourCode/compile.sh`	Script to compile your program (**Don't touch it**).
`raft.proto`	The gRPC proto file. (**Don't touch**. Just FYI only, since we have already generated the stubs for you.)
`raft(anotated).pdf`	The extended raft paper with some hints for the implementation (**Please read it**).



Remember to include both Src and Dst node id in both args and replies when you handle both RequestVote RPC and AppendEntries RPC!!

State

Persistent state on all servers:
(Updated on stable storage before responding to RPCs)

currentTerm	latest term server has seen (initialized to 0 on first boot, increases monotonically)
votedFor	candidateId that received vote in current term (or null if none)
log[]	log entries; each entry contains command for state machine, and term when entry was received by leader (first index is 1)

you also need a key-value store here to function as the permanent storage

Volatile state on all servers:

commitIndex	index of highest log entry known to be committed (initialized to 0, increases monotonically)
~~**lastApplied**~~	~~index of highest log entry applied to state machine (initialized to 0, increases monotonically)~~

We don't need this because in our impl we apply to store as soon as we commit

Volatile state on leaders:
(Reinitialized after election)

nextIndex[]	for each server, index of the next log entry to send to that server (initialized to leader last log index + 1)
matchIndex[]	for each server, index of highest log entry known to be replicated on server (initialized to 0, increases monotonically)

AppendEntries invoked by leader ONLY PER HeartBeatInterval. That is, when the leader receives a Propose, it still won't immediately invoke AppendEntries but wait until the upcoming HeartBeatInterval time out.

AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Should always restart election timer when a node receive this

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

You also want to reply current matchIndex in this node in Results to help leader update the matchIndex[]

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. Reply false if log doesn't contain an entry at prevLogIndex whose term matches prevLogTerm (§5.3)
3. If an existing entry conflicts with a new one (same index but different terms), delete the existing entry and all that follow it (§5.3)
4. Append any new entries not already in the log
5. If leaderCommit > commitIndex, set commitIndex = min(leaderCommit, index of last new entry)

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. If votedFor is null or candidateId, and candidate's log is at least as up-to-date as receiver's log, grant vote (§5.2, §5.4)

A node should restart its election timer to prevent competition if it votes the grantRequest

Rules for Servers

All Servers:

- If commitIndex > lastApplied: increment lastApplied, apply log[lastApplied] to state machine (§5.3)
- If RPC request or response contains term T > currentTerm: set currentTerm = T, convert to follower (§5.1)

Followers (§5.2):

- Respond to RPCs from candidates and leaders
- If election timeout elapses without receiving AppendEntries RPC from current leader or granting vote to candidate: convert to candidate

Candidates (§5.2):

- On conversion to candidate, start election:
  - Increment currentTerm
  - Vote for self
  - Reset election timer
  - Send RequestVote RPCs to all other servers
- If votes received from majority of servers: become leader
- If AppendEntries RPC received from new leader: convert to follower
- If election timeout elapses: start new election

Leaders: Initialize the nextIndex[] and matchIndex[] here

- Upon election: send initial empty AppendEntries RPCs (heartbeat) to each server; repeat during idle periods to prevent election timeouts (§5.2)
- If command received from client: append entry to local log, respond after entry applied to state machine (§5.3)
- If last log index ≥ nextIndex for a follower: send AppendEntries RPC with log entries starting at nextIndex
  - If successful: update nextIndex and matchIndex for follower (§5.3)
  - If AppendEntries fails because of log inconsistency: decrement nextIndex and retry (§5.3)
- If there exists an N such that N > commitIndex, a majority of matchIndex[i] ≥ N, and log[N].term == currentTerm: set commitIndex = N (§5.3, §5.4).

Figure 1: A condensed summary of the Raft consensus algorithm

2 Your Assignment

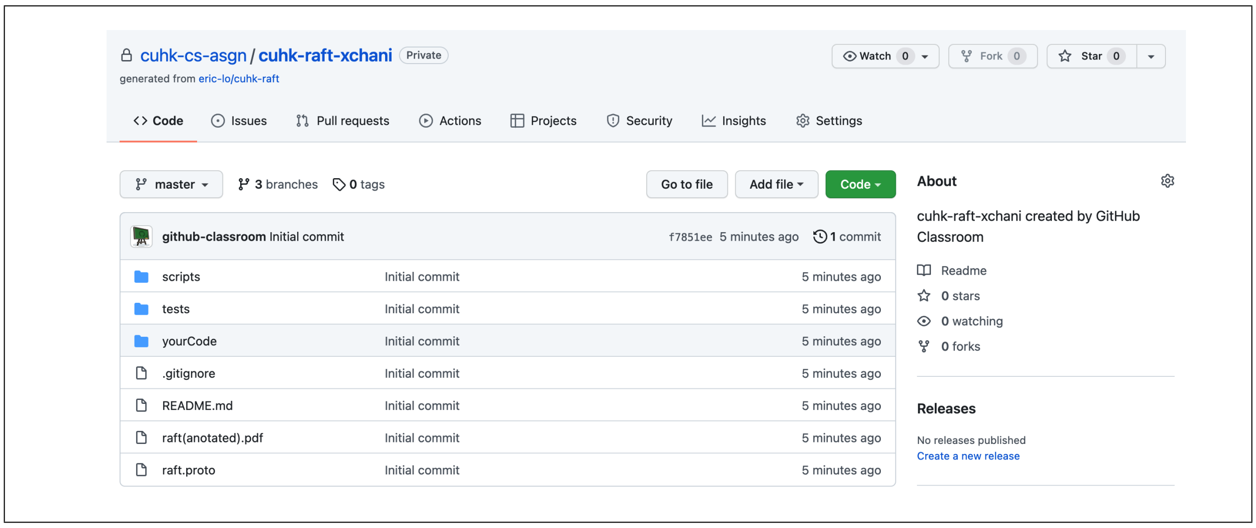
2.1 Submission: git classroom

We use **Github Classroom** `https://classroom.github.com` for assignment managment. Git is a predominant version control system, created by the inventor of Linux. Github is an online hosting of Git and it is now hosting thousands of open-source projects.

2.2 Opening an account and getting the assignment package

You are supposed to open a Github account using your CUHK email address (it is fine to use your own Github account if you already have) and follow the steps below to register an account and get the starter code.

1. Go to Github (`http://www.github.com`) to register an account using your CUHK email address.
2. Come here (`https://forms.gle/SyMD51sHJVjxXTih8`) to let us know your github account and your student ID.
3. Use your Github account to go here `https://classroom.github.com/a/Z33y1-9e` to clone the assignment starter package to be under your Github account.
4. If everything runs smoothly, your Github account will show a new repo named cuhk-raft-[your-Github-account-name], like below:



cuhk-cs-asgn / cuhk-raft-xchani

generated from eric-lo/cuhk-raft

github-classroom Initial commit f7851ee 5 minutes ago 1 commit

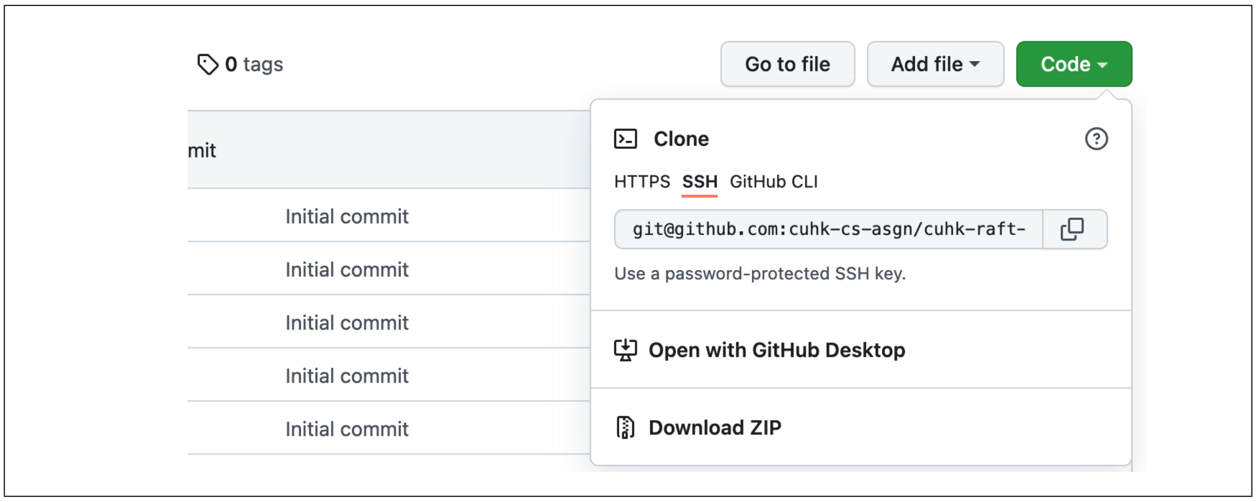
scripts	Initial commit	5 minutes ago
tests	Initial commit	5 minutes ago
yourCode	Initial commit	5 minutes ago
.gitignore	Initial commit	5 minutes ago
README.md	Initial commit	5 minutes ago
raft(anotated).pdf	Initial commit	5 minutes ago
raft.proto	Initial commit	5 minutes ago

About

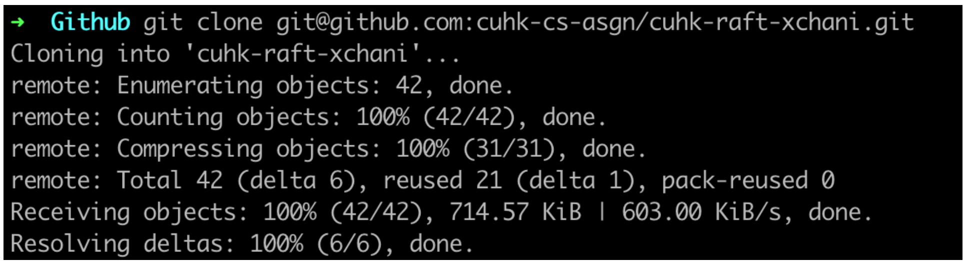
cuhk-raft-xchani created by GitHub Classroom

5. As GitHub shut down password authentication since Aug. 2021, you're required to setup a ssh key for connection following the GitHub Docs.

6. Look for the tab "`Code`" to get the URL of your assignment repo, like below:

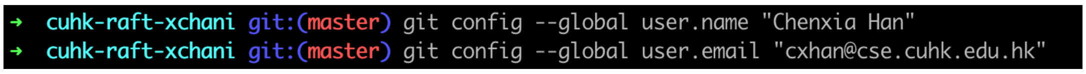


7. Open a terminal, change to a local directory of your desire, type "`git clone {Your Repo URL}`" and your username/password to get a local copy of the repo on your computer.



```
➜  Github git clone git@github.com:cuhk-cs-asgn/cuhk-raft-xchani.git
Cloning into 'cuhk-raft-xchani'...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 42 (delta 6), reused 21 (delta 1), pack-reused 0
Receiving objects: 100% (42/42), 714.57 KiB | 603.00 KiB/s, done.
Resolving deltas: 100% (6/6), done.
```

8. Finally, we should set the user name and email such that the git system recognizes you. You should do this once only. To begin with, type the followings in the terminal and **replace the name and email** with your own Git account.



```
➜  cuhk-raft-xchani git:(master) git config --global user.name "Chenxia Han"
➜  cuhk-raft-xchani git:(master) git config --global user.email "cxhan@cse.cuhk.edu.hk"
```

2.3 Get your starter code based on your preferred language

First clone the project from your repository. If you use Java, please execute:

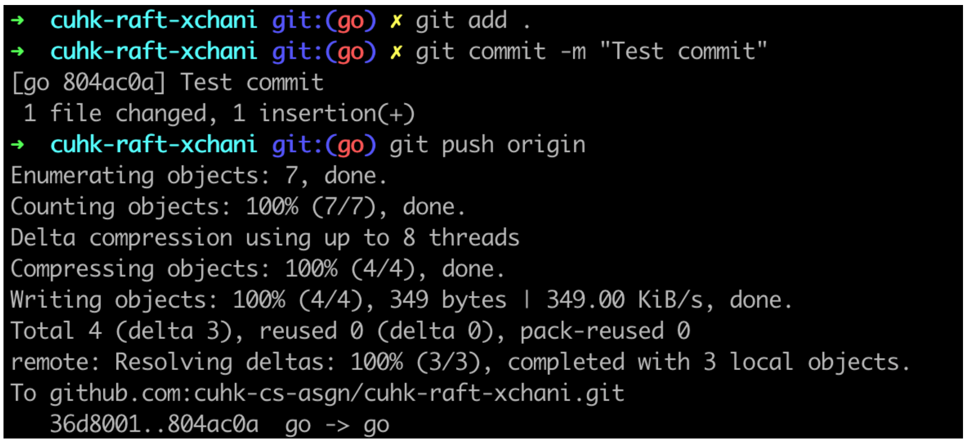
```
git checkout java
```

If you use Go, please execute:

```
git checkout go
```

2.4 Submission

Git is actually a complicated version control system (VCS) that has a lot of features. Briefly, it installs a local VCS on your computer and you can do any versioning locally (e.g., committing some changes, reverting some changes). If necessary, you can "push" your local changes from your local repo to an online repo (Github). You may Google the powerful usage of Git online. Here, we focus on how you can commit your finished assignment to your local repo and then push to the online private repo. We will fetch your assignment from your online private repo for grading. The process is essentially the "assignment submission" process:



```
➜  cuhk-raft-xchani git:(go) ✗ git add .
➜  cuhk-raft-xchani git:(go) ✗ git commit -m "Test commit"
[go 804ac0a] Test commit
 1 file changed, 1 insertion(+)
➜  cuhk-raft-xchani git:(go) git push origin
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 349 bytes | 349.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:cuhk-cs-asgn/cuhk-raft-xchani.git
   36d8001..804ac0a  go -> go
```

Assuming you have done some edits on the source files, the command `git add .` states your interest of adding everything under the current directory "." to your local Git repo. Next, the command `git commit` tries to commit your changes to your local repo. The parameter `-m "comments"` is your comment about this commit. Lastly, the command `git push origin` tries to synchronize your local repo with your online repo.

For Git beginners, make sure you use a web browser to login your Github online and check if your source codes have been pushed there successfully.

Warning: DON'T change the file names, otherwise you get 0 marks

2.5 Your job

For Java, write your program in '**src/main/java/RaftRunner.java**'. For Go, write your program in '**main.go**'.

- Follow the "State" box in Figure 1 to implement `RaftNode` class (Java) / `raftNode` struct (Go). You will need to add a lot more than what Figure 1 tells you such as adding a `Map` to be the in-memory key-value map, adding a `server-state` to store whether this node is in "Follower", "Candidate", or "Leader" state, etc.

- Function `NewRaftNode`: After creating an instance of `raftNode`, its initial state should be a "Follower", a node shall create a new thread to kick off a new leader election if it has not heard from the leader within a period of `electionTimeout`. To kick off the leader election, invoke the others' `RequestVote` by RPC **concurrently**. If a remote node doesn't respond to your `RequestVote`, don't resend `RequestVote`, wait for the next election.

- When a node becomes a "leader", invoke others' `AppendEntries` RPC concurrently based on the heartbeat interval.

  Follow Figure 1's "AppendEntries" box to implement it. Once again, if a remote node doesn't respond to your call, don't resend, wait for the next heartbeat interval.

- RPC `Propose`: Our grader will call this RPC to `put` or `delete` to update the distributed replicated map.

  Log this operation using `LogEntry` and append to the local log of the leader. `Propose` return only after the the log is committed (i.e., replicated to the majority of other nodes).

- RPC `GetValue`: Implement this to return a value given a key.

- RPCs `SetElectionTimeout` and `SetHeartBeatInterval`: invoked by the grader to set up different test cases. On invoked, these functions (re)-set the election timeout and heartbeat interval based on the input values.

After you have finished the assignment, compile it first to check whether there is any syntax error. You shall first enter into `yourCode` and execute:

```
bash compile.sh
```

A successful compilation would generate the `bin/raftrunner` file.

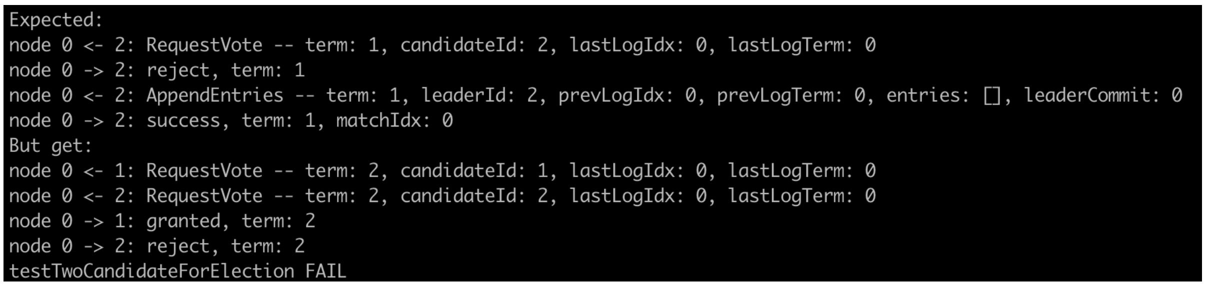
3 Run our grader

After your program can be successfully compiled, run `scripts/rafttest.sh` to compile the test framework and check whether you can pass the given test cases. The grader will launch 5 Raft instances as 5 processes.[1] There are 10 test cases: 6 to test the leader election, 4 to test the log replication. For each test case, it may

- Set different election timeouts and heartbeat interval on each node (process)
- Delay or drop the messages between the nodes
- Invoke `Propose` RPC to update the distributed replicated map
- Invoke `GetValue` RPC to get value by a key and match it with an expected value
- Match the messages between different nodes with the expected messages
- Note: there is a running time limit for each test case

All tests run about 200 $\sim$ 300 seconds. If a test case fails, the reason and the expected messages (if any) will be printed out. For example:

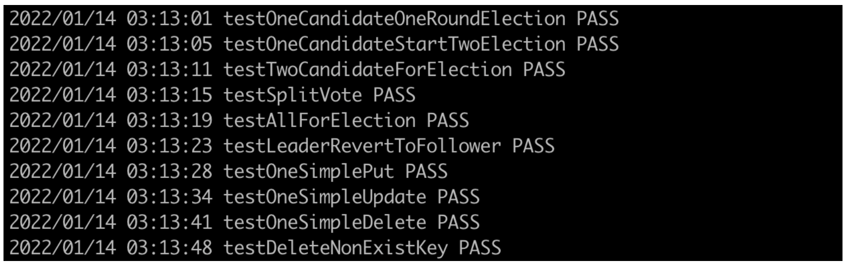
[1]It will also launch 5 proxies for the raft instances. The proxies are used by the grader to delay the messages (to simulate various test cases), etc. You don't need to understand them.



```
Expected:
node 0 <- 2: RequestVote -- term: 1, candidateId: 2, lastLogIdx: 0, lastLogTerm: 0
node 0 -> 2: reject, term: 1
node 0 <- 2: AppendEntries -- term: 1, leaderId: 2, prevLogIdx: 0, prevLogTerm: 0, entries: [], leaderCommit: 0
node 0 -> 2: success, term: 1, matchIdx: 0
But get:
node 0 <- 1: RequestVote -- term: 2, candidateId: 1, lastLogIdx: 0, lastLogTerm: 0
node 0 <- 2: RequestVote -- term: 2, candidateId: 2, lastLogIdx: 0, lastLogTerm: 0
node 0 -> 1: granted, term: 2
node 0 -> 2: reject, term: 2
testTwoCandidateForElection FAIL
```

In the above, the message "`node 0 <- 2: RequestVote -- term: 1, ...`" means node 2 requests a vote from node 0, and the one who sends the message (node 2) is in term 1. Similarly, the message "`node 0 -> 2: reject, term: 1`" means node 0 replies node 2 with a reject, and the one who sends the message (node 0) is in term 1.

In the end of the test, you will see a summary. If you see something like below, congratulations.



```
2022/01/14 03:13:01 testOneCandidateOneRoundElection PASS
2022/01/14 03:13:05 testOneCandidateStartTwoElection PASS
2022/01/14 03:13:11 testTwoCandidateForElection PASS
2022/01/14 03:13:15 testSplitVote PASS
2022/01/14 03:13:19 testAllForElection PASS
2022/01/14 03:13:23 testLeaderRevertToFollower PASS
2022/01/14 03:13:28 testOneSimplePut PASS
2022/01/14 03:13:34 testOneSimpleUpdate PASS
2022/01/14 03:13:41 testOneSimpleDelete PASS
2022/01/14 03:13:48 testDeleteNonExistKey PASS
```

We also provide a script to test for a specific test case. For example:

```
bash ../scripts/rafttest_single.sh \
    testOneCandidateOneRoundElection
```

Remember to run `rafttest.sh` at least once before `rafttest_single.sh`, otherwise the grader is not compiled and can't be found.

4 Grading

1. The TA will fetch and grade your latest version of **master** branch in your repository as of the deadline. **Remember to commit and push before the deadline!**

2. 10 test cases in total.

3. Passing one test case will earn you 10 marks.
4. Part A is leader election part of Raft. It covers the first six test cases. Part B is the log replication part, which covers the last four test cases.

5 Remarks and hints

- Don't deadline fighting! We give you such a long deadline for a reason – this assignment might first take you some weeks to start passing the first test case. Some more weeks to pass some more but may take even more weeks to pass only one more (and some previous passed test cases might fail after you pass a new one! i.e., regression). The above cycle might repeat! Simply put, start early.
- In our experience, sometimes you may spend hours to debug just 1 line of code (e.g., change its position). Yes, it is normal (and painful). But we all ran through it.
- No other package is allowed to use except `sync` if you use Go.
- Work on leader election first (to pass the first 6 test cases). By that time, you may ignore those log replication related entities like `commitIndex` and work back on those when you start working on log replication.
- Implement `SetElectionTimeout` and `SetHeartBeatInterval` first. They are not built-in Raft RPCs but used by the grader to simulate different test cases.
- Don't use Go's `time.Timer` or `time.Ticker`, which are difficult to use correctly.
- The parameters `heartBeatInterval` and `electionTimeout` in the `NewRaftNode` function are the default values for heartbeat interval and election timeout.
- For every node, set `lastLogIndex` = 0 and `lastLogTerm` = 0 initially.
- When winning an election, immediately announce winner by `AppendEntries`(empty log entry), don't wait for next heart beat.
- On winning the election, the leader creates 1 thread per follower and reuses those threads until the leader dies.

- Remember to update `votedFor` after you vote for a candidate.
- Set 100ms connection timeout when contacting a follower as the grader would sometimes emulate messages dropped.
- The log is implemented as an array. But to follow the Raft paper, our test regards the array begins from index **1** (i.e., log[1] is the first element), not from **0** (i.e., no log[0]).
- In this version of Raft, every follower shall also answer to `GetValue` from its own key-value store, not redirecting that request to the leader.
- If a `Propose` is sent to a follower node, `Status_WrongNode` shall be replied to tell the requester who is the current leader and the requester shall resend the request to the current leader.
- For the delete operation, you have to check whether the key exists in the leader's key-value store. The timing is very tricky: you have to wait for the majority of followers replied (i.e., the log entry of that operation is committed) before you really delete the key from the key-value store.
- There is certain randomness in distributed systems. So, if you pass a test case, make sure you repeat the test many times because sometimes you may just pass it by accident.
- Run `rafttest.sh` frequently to make sure you won't break previous passed test cases when working on a new one.
- It's possible that the grader doesn't kill the processes completely. If the grader reports that some ports are occupied by other processes, type "`ps aux | grep raft`" to check and "`kill -9 $(pgrep -f raft)`" to kill those processes.
- Check `tests/rafttest/leaderElectionTest.go` and `logReplicationTest.go` to figure out the test cases if necessary.
- When an RPC is called by another process, gRPC would auto launch a new thread to execute that RPC function.
- Ignore those `XXX_` entries in the RPC.

6 Change Log

1.0	this document

7 Questions

If you have doubts about the assignment, you are encouraged to ask questions on Piazza **using the corresponding tag**. Please focus on knowledge. Unhealthy questions/comments that focus on scores and grades are not encouraged.

If you find any (possible) bugs, send private questions on Piazza to us instead — otherwise that may cause unnecessary panic among the class if that is not a real bug.

8 Academic Honesty

We follow the University guide on academic honesty against any plagiarism.

9 General Notes

- If you use your own environment except for the environment specified by specification and got any question, test it on our specified environment before you ask.
- The TA reserves the right to adjust your scores for any request that requires their extra manual effort.
- Unless specified, you should work and fill your code in designated areas. There are cases that the modification outside the designated area leads to misbehavior of the auto grader. Proceed with caution and look back the changes if your output is different from what is shown in the grader.
- While we have already tried our best to prepare this assignment, we reserve all the rights to update the specification and the grading scheme. If there are any mistakes/bugs which are on ours, the TA will step in and do manual grading to ensure you get what you deserve. Please respect each other.

- When grading, the TAs will execute the grader program to run the whole test suite (which consists of all test cases). The TAs won't grade each individual test case separately.

- **(Frequently Asked)** [No hidden test case] We are not intended to run any secret/extra test cases that deliberately break your assignment. That is, WYSIWYG — your final score shall be generally indicated by what the grader reports when it runs on our the environment specified by the specification. However, we do reserve the right to run some additional test cases to avoid any mis-conduct (e.g., hard-coding the results), and/or invite you to explain the source code to the teaching assistants and adjust the scores accordingly.

- We welcome discussions among classmates. But don't share your assignment with the others in any means. For example, don't put your source code in any public venue (e.g, public repo, your homepage, Facebook). We handle plagiarism strictly. On submitting this assignment, you are agreed that your code's copyright belongs to the Chinese University of Hong Kong. Unless with our written approval, you **must not** release your source code and this specification now and **forever**.

- Google is your friend. We encourage you use Google for help to do the assignment. However, if you happen to find any source codes related to this assignment, you still cannot copy it but use your own way to implement it. You need to put down your list of source code references as comments in the top of your source code.

- **(Frequently Asked)** [Late Policy] TAs will only grade the latest version submitted before the deadline, no late submission is allowed. It is your responsibility to make sure you added, committed, and pushed the final version before the deadline. You are required to check whether your final version is really in Github Classroom.

References

[1] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In 2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14), pages 305–319, 2014.