

Assignment2. Raft Leader Election

MF1733071, 严德美, 1312480794@qq.com

2017 年 11 月 27 日

1 简述分析与设计

本次任务是实现一个Raft [1]算法中的一个Leader Election。在某一时刻, server处于以下状态中的一个,leader、follower和candidate,正常情况下, n个server中, 有一个leader,n-1个followers,所有的server开始状态都是follower, raft中有两个超时机制, 一个是election timeout,另一个是heartbeat timeout,在本次实现中, election timeout设置为400ms~500ms中的一个随机值, heartbeat timeout设置为100ms,FOLLOWER,CANDIDATE,LEADER分别设置为常量值0,1,2, 每个server都有一个election timeout, 当follower的election timeout超时, 将开始新一轮的election,server将从follower状态转换成candidate,raft使用随机election timeout机制去确保在大多数情况下只有一个server超时, 同时保证了一个超时server能获得大多数vote, 成为candidate的server先给自己投票, 当前Term加一, 并且广播发送request vote信息给其他server nodes, 收到request vote的server node, 将request vote中的Term和自己的当前Term比较并查看自己的votedFor是否为-1(-1代表server node还没给candidate投过票), 如果满足大于自己的Term并且votedFor为-1则给candidate投票, 并重置自己的election timeout, 否则忽略, 并进行相应的操作。如果candidate收到的投票数(包括它自己)大于节点数的一半, 则成为leader, 成为这一时期的管理者, 给follower发送心跳包, 直到该节点失效或者和其他节点失去连接。当follower收到投票信号或者心跳信号时, 重置election timeout。leader每隔一段heartbeat timeout发送一个心跳包给follower。相应的数据结构如下:

```
1  const (
2      FOLLOWER = iota
3      CANDIDATE
4      LEADER
5
6      HEARTBEAT_TIMEOUT = 100 // 心跳间隔
7      MIN_ELECTION_TIMEOUT = 400
8      MAX_ELECTION_TIMEOUT = 500
9
10 )
```

```

1 //
2 // A Go object implementing a single Raft peer.
3 //
4 type Raft struct {
5     mu          sync.Mutex
6     peers       []*labrpc.ClientEnd
7     persister   *Persister
8     me          int // index into peers[]
9
10    // Your data here.
11    // Look at the paper's Figure 2 for a description of
12    // what state a Raft server must maintain.
13    votedFor int //给谁投票
14    voteAcquired int //获得的票数
15    state int32 //当前状态
16    currentTerm int32 //当前任期
17    electionTimer *time.Timer //选举时间间隔
18    voteCh chan struct{} //成功投票的信号
19    appendCh chan struct{} //成功更新log的信号
20 }

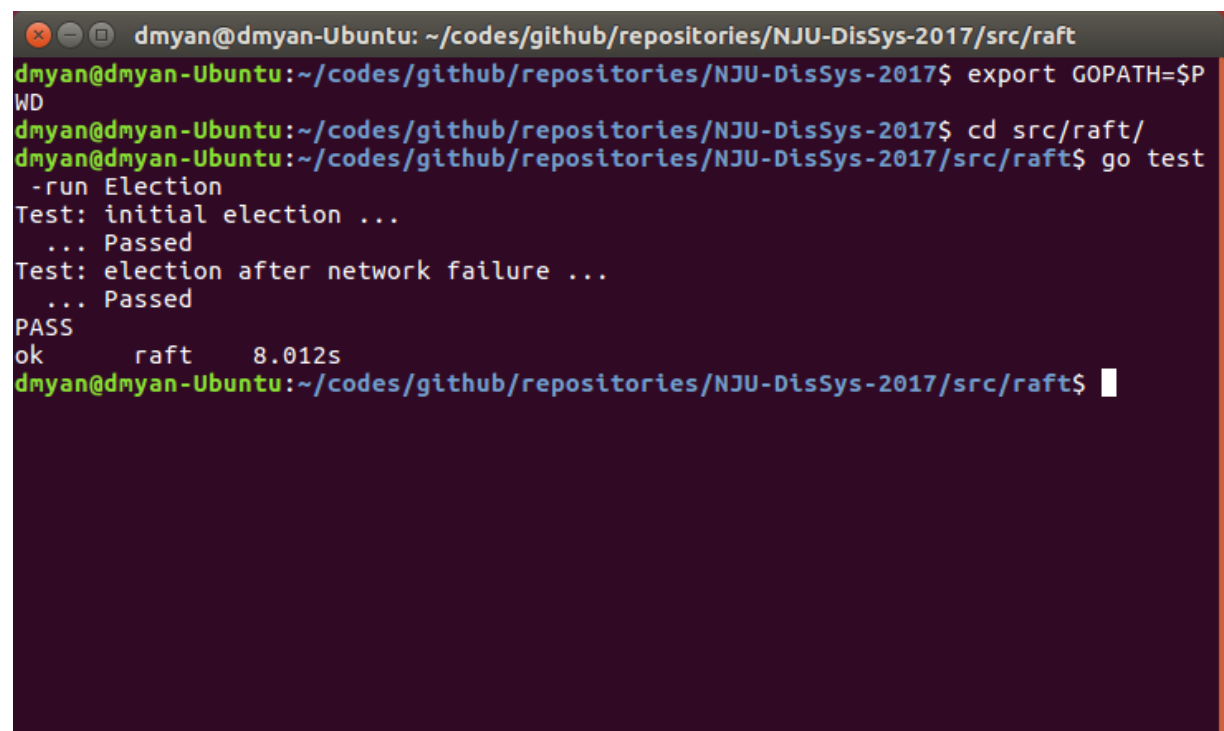
```

```

1 //
2 // example RequestVote RPC arguments structure.
3 //
4 type RequestVoteArgs struct {
5     // Your data here.
6     Term int32 //candidate term
7     CandidateId int //candidate requesting vote
8 }
9
10 //
11 // example RequestVote RPC reply structure.
12 //
13 type RequestVoteReply struct {
14     // Your data here.
15     Term int32 //回复者所处时期
16     IsVote bool //是否投票
17 }

```

2 实现演示

A terminal window with a dark background and light-colored text. The window title is 'dmyan@dmyan-Ubuntu: ~/codes/github/repositories/NJU-DisSys-2017/src/raft'. The terminal shows a series of commands and their outputs: 'export GOPATH=\$PWD' is executed; 'cd src/raft/' changes the directory; 'go test -run Election' runs tests, showing 'Test: initial election ...' and 'Test: election after network failure ...' both passing; 'PASS' is printed; 'ok raft 8.012s' shows the test completed successfully. The prompt returns to 'dmyan@dmyan-Ubuntu:~/codes/github/repositories/NJU-DisSys-2017/src/raft\$'.

```
dmyan@dmyan-Ubuntu: ~/codes/github/repositories/NJU-DisSys-2017/src/raft
dmyan@dmyan-Ubuntu:~/codes/github/repositories/NJU-DisSys-2017$ export GOPATH=$PWD
dmyan@dmyan-Ubuntu:~/codes/github/repositories/NJU-DisSys-2017$ cd src/raft/
dmyan@dmyan-Ubuntu:~/codes/github/repositories/NJU-DisSys-2017/src/raft$ go test
-run Election
Test: initial election ...
... Passed
Test: election after network failure ...
... Passed
PASS
ok      raft      8.012s
dmyan@dmyan-Ubuntu:~/codes/github/repositories/NJU-DisSys-2017/src/raft$
```

图 1: Raft Leader Election

3 总结

通过本次实验，对分布式系统一致性的问题有了更多的理解，阅读论文和把它实现出来还是不一样的，实现中才发现有很多细节需要考虑和处理，实现Leader Election加深了对论文中作者描述的算法理解，也对go语言更加熟悉了，对Assignment3的理解和实现有很大的帮助。

参考文献

- [1] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, 2014.