# Raft实验报告

**黄潇颖 2020201622**

报告包括使用到的**数据结构**、**2A和2B部分的实现思路**、实验中遇到的**问题和解决方案**、**实验结果**四个部分。

## 1.数据结构

除了Raft struct中的identity和heartBeatStamp以外，其它参数都是按照论文figure2中来的。
其中，logs,nextIndex和matchIndex我选择了使用slice作为容器。

- 1.1 server structure

```
type Raft struct {
	mu        sync.Mutex          // Lock to protect shared access to this peer's s
	peers     []*labrpc.ClientEnd // RPC end points of all peers
	persister *Persister          // Object to hold this peer's persisted state
	me        int                 // this peer's index into peers[]
	dead      int32               // set by Kill()

	// Your data here (2A, 2B, 2C).
	// Look at the paper's Figure 2 for a description of what
	applyCh chan ApplyMsg

	// state a Raft server must maintain.
	currentTerm int
	votedFor    int //candidateId that received vote in current term (or null if no

	logs []*LogEntry

	//for all servers
	commitIndex int //index of highest log entry known to be committed (initialized
	lastApplied int //index of highest log entry applied to state machine (initiali

	//records of management (for leader)
	nextIndex  []int //for each server, index of the next log entry to send to that
	matchIndex []int //index of highest log entry applied to state machine (initial

	//extra:
	identity       int       //0:candidate;1:follower;2:leader
	heartBeatStamp time.Time //handle timeout issue
}
```

- RequestVote RPC arguments structure

```go
type RequestVoteArgs struct {
	// Your data here (2A, 2B).
	Term         int //candidate's term
	CandidateId  int //candidate requesting vote
	LastLogIndex int //index of candidate's last log entry
	LastLogTerm  int //term of candidate's last log entry
}

// RequestVote RPC reply structure.
// field names must start with capital letters!
type RequestVoteReply struct {
	// Your data here (2A).

	Term        int  //currentTerm, for candidate to update itself
	VoteGranted bool //true means candidate received vote

}
```

- Request of LogAppend RPC arguments structure

```go
type RequestLogAppendArgs struct {
	Term         int         //leader's term
	LeaderId     int         //so follower can redirect clients
	PrevLogIndex int         //index of log entry immediately preceding new ones
	PrevLogTerm  int         //term of prevLogIndex entry
	Entries      []*LogEntry //log entries to store (empty for heartbeat;may send mo
	LeaderCommit int         //leader's commitIndex

}

// Reply of LogAppend RPC arguments structure.
type RequestLogAppendReply struct {
	Term    int
	Success bool
}
```

## 2.实现思路

- **2.1 Vote**

  2.1.1 在 `ticker` 函数中，先设置**两个重要的超时时长**:

  1）**心跳超时** `HeartBeatTimeOut = 400 Millisecond` ,负责监测leader是否挂机，在检测时刻距离上一次收到领导者心跳超过这个时长即leader挂掉，本server的term自增1并掉用 `Raft.LaunchElection` 发起新一轮选举。

  2）**投票超时** `VoteTimeOut = 30 Millisecond` ，在 `Raft.LaunchElection` 中用于candidate等待其它server给自己投票，超时返回的选票一律不统计。

  实验给的ticker框架中已经为我们写好了检测心跳超时的循环，每两次检测间隔50-350ms的随机

时间。

2.1.2 **`Raft.LaunchElection` 重要实现细节**:

先声明一个变量 `var voteforme uint64` 来负责投票的计数，然后将选票箱并行地递给每一个server即 `go Raft.BallotBox(i, &voteforme, voteargs)` 。

**`Raft.BallotBox` 函数**的操作：发送vote rpc，若server投票，则voteforme原子加1，否则没有动作。

给server递BallotBox后，等待一段时间(VoteTimeOut)，然后直接读取选票 `ballots := int(atomic.LoadUint64(&voteforme))` 并检测其是否达到半数以上，若达到了，则执行：

初始化Raft中领导者维护的变量 `Raft.LeaderServerInit()` ，

更新identity为leader `Raft.TransformToLeader()` ，

发送心跳 `go rf.StartLeaderHeartBeat(serversNum)` 确保不会超时。

若未达到，则没有动作，等待下一次选举。

2.1.3 **RequestVote RPC handler的实现**:

按照论文里figure2的规则实现即可，值得说明的有两个细节。

- 添加了一个 `args.Term == rf.currentTerm` 的判断区域，若是自己的信息，直接投票返回；若不是，则说明有同时发起的vote，返回false并等待下一次vote。
- 论文中"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)"的实现：若candidate的last log term比该server的小，则不投票；若相等，则看log的长度，若candidate的不如该server的长，则不投票。(rpc_vote.go中83-86行)

- **2.2 Log Append**

  按照论文里figure2的规则实现，并在此基础上添加了三个细节。

  增添了一个判断区域：在args.term >= rf.currentTerm的情况下，如果发送rpc的不是自己，则需要如下操作：(rpc_logreplicate.go 72-79)

  - 转换identity为follower
  - 修改currentTerm
  - 当发送者的term和自己修改前的term相等，且自己是leader的情况时 `termin == args.Term && is_leader` ，说明集群出现了两个leader，则需要返回等待下一次选举。

  余下增添的细节在下文3.3和3.4中有提到。

- **2.3 Commit & Apply**

  follower的commitIndex的更新在 `Raft.RequestLogAppend` 中进行，主要依赖心跳来周期性对比leader发来的的commitIndex和更新。当然每一次log append也会检测，但这有滞后性，每一次log append都可能发来旧的commitIndex，若不与心跳绑定而与log append绑定会导致没有log append

时follower的commitIndex得不到更新。

在server最初启动时 `go rf.LogClientCheck()` ，其用途：

- 用于leader的commitIndex更新，论文中的"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)."在这里实现，没有特别的细节。
- 所有server将新的committed log应用到状态机(apply to state machine):
  若 `Raft.commitIndex > Raft.lastapply` ，将依次为所有index大于Raft.commitIndex的log创建对应的ApplyMsg并发送到applyCh中。

## 3.实现时遇到的问题及其思考与解决

3.1 什么时候加锁？加锁的粒度？

在做本实验时时常会遇到死锁和data race的问题，go提供的defer功能在官方来说是一个简洁美观且安全的操作，但在我看来它可能会让人偷懒然后成为死锁的罪魁祸首。加锁的粒度太大可能会影响性能，加锁的粒度太小可能会产生数据不一致。

3.2 选票统计的data race

一开始没有多写一个Raft.BallotBox函数，而是直接make了一个vote reply struct的slice，接着无论我怎么加锁都会data race。

```
⊗ (base) hxy@localhost raft % go test -run 2A -race
  Test (2A): initial election ...
  ==================
  WARNING: DATA RACE
  Read at 0x00c00001a3e8 by goroutine 21:
    6.5840/raft.(*Raft).LaunchElection()
        /Users/hxy/Desktop/distribution and cloud cal/lab-raft/6.5840/src/raft/ticker.go:85 +0x444
    6.5840/raft.(*Raft).ticker()
        /Users/hxy/Desktop/distribution and cloud cal/lab-raft/6.5840/src/raft/ticker.go:32 +0x1b3
    6.5840/raft.Make.func1()
        /Users/hxy/Desktop/distribution and cloud cal/lab-raft/6.5840/src/raft/raft.go:124 +0x39

  Previous write at 0x00c00001a3e8 by goroutine 30:
    reflect.Value.SetBool()
        /usr/local/go/src/reflect/value.go:2253 +0x71
    encoding/gob.decBool()
        /usr/local/go/src/encoding/gob/decode.go:238 +0x64
    encoding/gob.(*Decoder).decodeStruct()
        /usr/local/go/src/encoding/gob/decode.go:494 +0x2c3
    encoding/gob.(*Decoder).decodeValue()
        /usr/local/go/src/encoding/gob/decode.go:1249 +0x392
    encoding/gob.(*Decoder).DecodeValue()
        /usr/local/go/src/encoding/gob/decoder.go:229 +0x2ca
    encoding/gob.(*Decoder).Decode()
        /usr/local/go/src/encoding/gob/decoder.go:204 +0x2c4
    6.5840/labgob.(*LabDecoder).Decode()
        /Users/hxy/Desktop/distribution and cloud cal/lab-raft/6.5840/src/labgob/labgob.go:55 +0x7b
    6.5840/labrpc.(*ClientEnd).Call()
        /Users/hxy/Desktop/distribution and cloud cal/lab-raft/6.5840/src/labrpc/labrpc.go:119 +0x449
    6.5840/raft.(*Raft).SendRequestVote()
        /Users/hxy/Desktop/distribution and cloud cal/lab-raft/6.5840/src/raft/voterpc.go:131 +0xac
    6.5840/raft.(*Raft).LaunchElection.func1()
        /Users/hxy/Desktop/distribution and cloud cal/lab-raft/6.5840/src/raft/ticker.go:63 +0x64
```

曾经我想出了一个方案，为每一个server创建channel来进行对齐，只有 `SendRequestVote` 返回了才可

以从vote reply struct slice中读取reply。冲突确实消失了，但是阻塞会使得它无法在限定时间内选出leader,而且选举结果很不稳定。这让我意识到在这个场景下创建指针数组是很差劲的。
最后使用了atomic包来构造Raft.BallotBox，同时解决了冲突问题和选举时间限制问题。

3.3 `Test (2B): concurrent Start()s ...` 中某些log突然变短。
由于rpc的乱序到达，在log append中用下面的方法实现第3、4条规则在并发写入日志的情况下是不可靠的,可能旧的日志会把新的覆盖掉。

```
//3. If an existing entry conflicts with a new one
//(same index but different terms),delete the
//existing entry and all that follow it
rf.logs = rf.logs[:args.PrevLogIndex+1]
//4. Append any new entries not already in the log
rf.logs = append(rf.logs, args.Entries...)
```

最终的思路为在prevIndex之后的log逐个检测，直到找到第一个conflict的log才进行覆盖，对于当前term下携带旧日志的rpc被接收之后则不会发生覆盖动作。

3.4 `Test (2B): rejoin of partitioned leader ...` 中server重连之后会提交错误的log。以下图为例，经检测后发现102和103的index是相同的。

```
⊗ (base) hxy@localhost raft % go test -run 2B -race
  Test (2B): basic agreement ...
    ... Passed --   0.4  3   24    5754    3
  Test (2B): RPC byte count ...
    ... Passed --   1.4  3   66  116912   11
  Test (2B): test progressive failure of followers ...
    ... Passed --   4.6  3  209   39962    3
  Test (2B): test failure of leaders ...
    ... Passed --   5.0  3  221   38269    3
  Test (2B): agreement after follower reconnects ...
    ... Passed --   5.5  3  194   46526    8
  Test (2B): no agreement if too many followers disconnect ...
    ... Passed --   4.1  5  356   65278    3
  Test (2B): concurrent Start()s ...
    ... Passed --   1.0  3   41   10336    6
  Test (2B): rejoin of partitioned leader ...
  2023/05/07 23:51:09 0: log map[1:101]; server map[1:101 2:103]
  2023/05/07 23:51:09 0: log map[1:101]; server map[1:101 2:103]
  2023/05/07 23:51:09 apply error: commit index=2 server=0 102 != server=2 103
  exit status 1
  FAIL    6.5840/raft     24.617s
```

于是给log append rpc的第5条增添了对log[commint index]的term判断，当它等于current term才能commit.

```go
//5. If leaderCommit > commitIndex, set
	//commitIndex = min(leaderCommit, index of last new entry)
	if args.LeaderCommit > rf.commitIndex {
		lastlogi, _ := rf.GetLastLogInfo()
		if lastlogi >= args.LeaderCommit && rf.logs[args.LeaderCommit].Term ==
			rf.commitIndex = min(args.LeaderCommit, lastlogi)
		}

	}
```

3.5 对断联和从集群(peers)中移除的区别很模糊，打印状态之后发现断联了之后大家的peers没有减少，查看 `test_test.go` 和 `config.go` 发现断联只是让线程sleep。
曾经想靠监测rpc连接情况来判断集群中的server个数，然后用自己监测的可连接server的数值来判断是否超过半数，这样做对于vote没有影响，但是会出现commit错误log的情况。

## 4.实验结果

```
(base) hxy@localhost raft % go test -run 2A
Test (2A): initial election ...
  ... Passed --   3.1  3   96   23187    0
Test (2A): election after network failure ...
  ... Passed --   4.5  3  174   30714    0
Test (2A): multiple elections ...
  ... Passed --   5.8  7  755  136098    0
PASS
ok      6.5840/raft     13.643s
(base) hxy@localhost raft % go test -run 2B
Test (2B): basic agreement ...
  ... Passed --   0.7  3   27    6188    3
Test (2B): RPC byte count ...
  ... Passed --   1.2  3   64  116530   11
Test (2B): test progressive failure of followers ...
  ... Passed --   4.2  3  206   38527    3
Test (2B): test failure of leaders ...
  ... Passed --   5.0  3  209   35868    3
Test (2B): agreement after follower reconnects ...
  ... Passed --   5.3  3  198   47261    8
Test (2B): no agreement if too many followers disconnect ...
  ... Passed --   3.7  5  319   58624    3
Test (2B): concurrent Start()s ...
  ... Passed --   0.7  3   39    9708    6
Test (2B): rejoin of partitioned leader ...
  ... Passed --   2.3  3  123   24829    4
Test (2B): leader backs up quickly over incorrect follower logs ...
  ... Passed --  13.3  5 2093  946366  102
Test (2B): RPC counts aren't too high ...
  ... Passed --   2.1  3   93   23843   12
PASS
ok      6.5840/raft     38.733s
```