raft lab实验文档

raft lab实验介绍

在raft lab中我们要实现raft中的选举机制，实验需要你手动实现appendEntry函数，即当节点收到心跳之后，需要做出一系列逻辑，其中包括如果心跳包的term太小，我们可以选择拒绝该心跳包。如果接受心跳包，follower需要更新自己的计时器，follower需要通过维持自己的计时器来保持状态，如果一段时间内没有收到心跳，会变成candidate开始选举。

raft lab实验工具

开发工具：在raft lab中我们选择开发语言为golang语言，在实验开始前确保你已经正确安装了Go语言开发环境。推荐采用goland作为开发工具，同时，我们可以采用教育邮箱免费申请JetBrains全家桶。

Go语言学习资料：为了让同学们开始一个go语言的学习，我们给大家推荐以下资料，首先是go指南：https://tour.go-zh.org/。其次是https://www.runoob.com/go/go-tutorial.html。

tips:建议大家好好学习go中的mutex和channel，go语言为并发而成。它的并发思路可以很好的启发大家对并发的理解。

结果测试：测试工具我们采用sh脚本，确保你已经在WSL/Ubuntu/Mac环境下。sh脚本运行教程：

```
run.sh times threads [test name]
```

run.sh是我们的脚本的名字，times是次数，threads是并发线程数，test name一共有两个**TestInitialElection2A和TestReElection2A**，在TestInitialElection2A中我们只会在网络良好的情况下的leader选举，在TestReElection2A下我们会测试在网络出现中断的情况下，选举是否还可以正常的进行。

比如说: `run.sh 1000 10 2A`

我们通过1000次10个线程来跑我们的测试，2A表示我们同时测试两个test，确保你可以在本地的情况下跑至少1000次达到bug free，我们在评分的时候通过多次测试来评分，而不仅仅只测试1次。

```
Done 988/1000; 988 ok, 0 failed
Done 989/1000; 989 ok, 0 failed
Done 990/1000; 990 ok, 0 failed
Done 991/1000; 991 ok, 0 failed
Done 992/1000; 992 ok, 0 failed
Done 993/1000; 993 ok, 0 failed
Done 994/1000; 994 ok, 0 failed
Done 995/1000; 995 ok, 0 failed
Done 996/1000; 996 ok, 0 failed
Done 997/1000; 997 ok, 0 failed
Done 998/1000; 998 ok, 0 failed
Done 999/1000; 999 ok, 0 failed
Done 1000/1000; 1000 ok, 0 failed
```

在测试之后我们运行以下命令来删除刚刚产生的文件。

`rm -rf test-*`

`rm -rf tester`

rm是linux系统下的删除命令，-r代表递归删除，-f代表强力删除，记住，千万不要误删自己的文件。*的作用是匹配任何名字，比如说test-1和test-2。如果你删除`rm -rf *`将会删除你的所有文件，千万不要这么做！！！

实验：你需要实现一个appendEntry函数

我们需要实现的代码在src/raft/raft.go中，sendAppendEntries是一个自己实现的rpc调用，这个函数可以让主机调用，发送appendEntries请求给其他的主机，一共有3个参数，分别是server, args, reply, server是发送的主机的id号，args是rpc的参数，reply是rpc的回复。

RPC是远程服务调用的意思，他的目的是让远程函数调用像本地调用一样简单。比如说sendAppendEntries其实是一个远程通信，但通过rpc的方法，我们就像是在调用本地函数一样。

当我们调用sendAppendEntries之后，server就会接受一个心跳，我们可以通过AppendEntries对心跳进行处理。当接受到心跳的时候，主机会刷新自己的定时器，当长时间未收到心跳的时候，主机会从follower变成candidate开始进行选举。

以下是我们的实现逻辑，同学可以参考我们的实现逻辑来实现代码。

```
// 如果大家不想参加挑战，请下载普通版本，如果想要参加挑战，请下载挑战版本.
if rf.term > sender.term:
  1.拒绝心跳 并且发送自己的term
  2.发送者接受到比自己心跳大的reply之后，意识到自己不是最新的，会停止选举，退化成follower
  return

3.如果没有拒绝心跳，我们首先发送一个成功到messageCh中，主机通过从meesageCh接受信息来刷新自己的定时器

if rf.term < sender.term:
  4.如果term小于sender的term，我们首先更新自己的term，并且更新自己的状态为follower

5.接下来我们开始判断args中的log是不是空的，如果是空的，我们同意该心跳.
6.我们可以通过DPrintf来发送日志来帮助我们debug，如果我们想要发送日志到终端或者是在日志文件中，我们需要把util.go中的debug置为1.
```

挑战：你需要额外实现一个requestVote函数(挑战不计入总成绩)

```
// 如果同学想要实现挑战，首先请下载挑战版本.
if rf.term > sender.term:
  1.拒绝投票 并且发送自己的term
  2.sender接收到term比自己大的reply之后会退化为follower
  return
if rf.term < sender.term:
  3.更新自己的term，并且退化为follower
4.接下来我们要判断在这轮中我们是否已经开始了选举，rf.votedFor代表在本轮中是否已经投票，-1代表没有投票，在voteFor的时候，我们可能需要加锁来实现race问题。
5.如果没有投票，我们会在这轮中同意投票，并且更新自己的状态.
```

最后对大家的建议

1. 希望同学们在闲暇时间可以维护一下我们的lab，我们的lab还有很多的不足，比如说在注释方面，希望各种同学可以完善一下我们的实验，我会在github对改进实验的同学做出致谢。
2. mit6.824是一门分布式系统的课程，mit已经在2020年将最新的视频和作业资源发放了出来，很建议大家去写lab，6.824一共有4个lab，lab2和lab3实现了完整的raft算法，但是难度很大，希望大家去挑战自己!
3. raft论文https://pdos.csail.mit.edu/6.824/papers/raft-extended.pdf，我们可以参考raft论文来对代码进行实现。