选举算法原理和实现

——Raft中基本的leader选举

- SY1706330 许崇杨
- SY1706414 王紫璇
- SY1706246 由伟希

目录

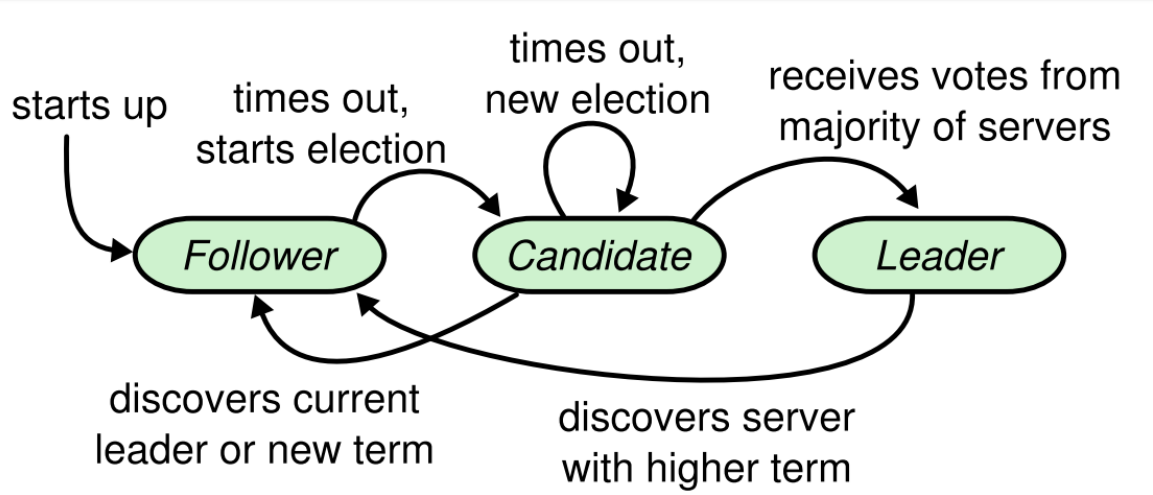
CONTENTS

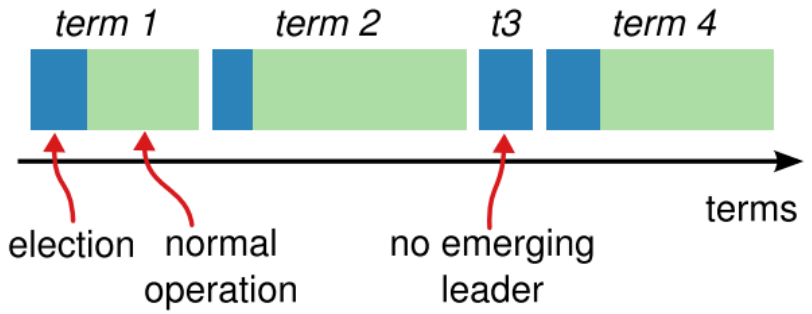
背景

- Leader选举
  - 分配一个进程作为协调者（协调分布到多个节点上的任务）的过程[1]
  - 某节点发起选举，选举出结果，各节点得知leader
- 关于Raft
  - 给出算法具体说明的共识算法
  - Leader 选举+日志复制
- Raft中的选举算法
  - 容忍(非拜占庭)错误
  - 测试方便：Raft实验框架[2]
    - 仿真的不可靠RPC、测试程序

[1]. https://en.wikipedia.org/wiki/Leader_election
[2]. http://nil.csail.mit.edu/6.824/2017/labs/lab-raft.html

原理





In Search of an Understandable Consensus Algorithm

如何容忍错误

非拜占庭错误模型

- 网络丢包、重复、乱序、时延
  - 丢包：RPC超时机制
  - 重复：“幂等的” RPC
  - 乱序、时延：逻辑时钟
- 网络分区(network-partition)、节点错误(fail-stop)
  - 有过半节点正常时能选举出leader(节点重新启动/连接正常后继续运行)
  - 否则，处于不可用状态，但不违背安全性

实现和测试

实现：状态机的实现示意

```
start:
switch(role){
    case follower:
        //do sth
    case candicate:
        //do sth
    case leader:
        //do sth
        for{
            select{
                case <-newTerm:
                    goto start
                case <-cond2:
                    //do sth
                default:
                    //do sth
            }
        }
}
```

测试：测试用例重复执行1000次

测试用例(3节点)	预期结果
1.初始选举	选举出leader
2.移走leader	选举出新leader
3.旧leader重新加入	不影响2中新leader
4.移除leader，和另一个节点	剩余节点中无法选举出leader (不存在大多数节点可用)
5.加入一个节点	选举出新leader (存在大多数节点可用)
6.加入最后一个节点	leader仍然存在

展示环节

根据日志重放协议执行过程

请大家批评指正