# raft-extended

## 中文翻译

<https://www.jianshu.com/p/2a2ba021f721>

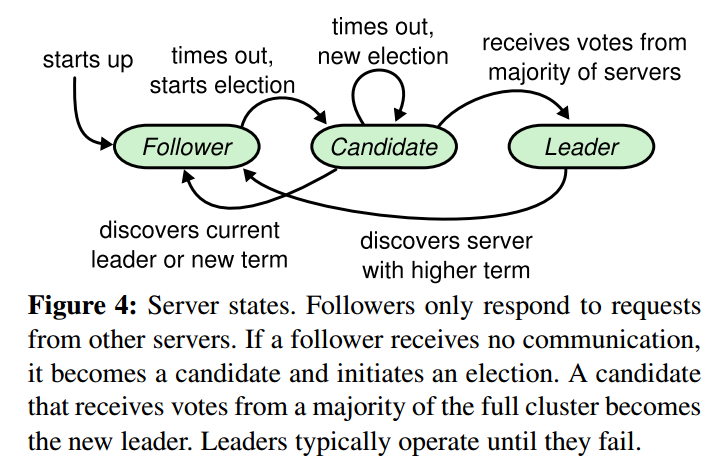
<http://blog.luoyuanhang.com/2017/02/02/raft-paper-in-zh-CN/>

<https://www.cnblogs.com/YaoDD/p/6172011.html>

<https://blog.csdn.net/hfty290/article/details/44281205>

# 状态图

## 状态图



# 三种状态

## 领导者（Leader）

## 追随者（Follower）

## 候选人（Candidate）

# 时间

## 广播时间（Broadcast Time）

一台服务器并行的向集群中的其他服务器发送 RPC 并且收到它们的响应的平均时间。

## 选举超时（election timeout）

### 周期性发送心跳时间

领导人会向所有追随者周期性发送心跳（heartbeat，不带有任何日志条目的 AppendEntries RPC）来保证它们的领导人地位。

### 心跳超时

如果一个追随者在一个周期内没有收到心跳信息，就叫做选举超时（election timeout）,然后它就会假定没有可用的领导人并且开始一次选举来选出一个新的领导人。

### 随机的选举超时

### 最小选举超时

当服务器在当前最小选举超时时间内收到一个 RequestVote RPC，它不会更新当前的任期号或者投出选票。

## 服务器平均故障时间（MTBF）

单个服务器发生故障的间隔时间的平均数。

## 时间要求

Broadcast Time << Election Timeout << MTBF

Broadcast Time应该比Election Timeout小一个数量级，为的是使领导人能够持续发送心跳信息来阻止追随者们开始选举。

# 数据维护

## 每个服务器需要维护的数据

### 稳定的数据

#### currentTerm

服务器最后知道的任期号（从0开始递增）。

#### votedFor

在当前任期内收到选票的候选人 id（如果没有就为 null）。

#### log[]

日志条目。

每个条目包含状态机要执行的命令和当时领导人的任期号。

### 不稳定的数据

#### commitIndex

已知的被提交的最大日志条目的索引值（从0开始递增）。

#### lastApplied

被状态机执行的最大日志条目的索引值（从0开始递增）。

## 领导者服务器需要维护的数据

### 不稳定的数据

#### nextIndex[]

对于每一个服务器，记录需要发给它的下一个日志条目的索引（初始化为领导人上一条日志的索引值+1）。

#### matchIndex[]

对于每一个服务器，记录已经复制到该服务器的日志的最高索引值（从0开始递增）。

# 三种RPC

## Append Entries RPC

### 由领导者发起

### 函数定义

（term，leaderId，prevLogIndex，prevLogTerm，entries[]，leaderCommit） ---> （term，success）

### 两种类型

#### 心跳（不带有任何日志条目）

#### 日志复制（带有一条或多条）

## Request Vote RPC

### 由候选人发起

### 函数定义

（term，candidateId，lastLogIndex，lastLogTerm，）---> （term，voteGranted）

## Install Snapshot RPC

### 由领导者发起

### 函数定义

（term，leaderId，lastIncludedIndex，lastIncludedTerm，offset，data[]，done）---> （term）

# 规则

## 所有服务器

### 如果commitIndex > lastApplied，lastApplied自增，将log[lastApplied]应用到状态机

### 如果 RPC 的请求或者响应中包含一个 term T 大于 currentTerm，则currentTerm赋值为 T，并切换状态为追随者（Follower）

## 追随者（followers）

### 响应来自候选人和领导人的 RPC

### 如果在超过选取领导人时间之前没有收到来自当前领导人的Append Entries RPC或者没有收到候选人的投票请求，则自己转换状态为候选人

## 候选人

### 当转变为候选人之后开始选举

#### currentTerm自增

#### 给自己投票

#### 重置选举计时器

#### 向其他服务器发送RequestVote RPC

### 如果收到了来自大多数服务器的投票：成为领导人

### 如果收到了来自新领导人的AppendEntries RPC（heartbeat）：转换状态为追随者

### 如果选举超时：开始新一轮的选举

## 领导人

### 一旦成为领导人：向其他所有服务器发送空的AppendEntries RPC（heartbeat）;在空闲时间重复发送以防止选举超时

### 如果收到来自客户端的请求：向本地日志增加条目，在该条目应用到状态机后响应客户端

### 对于一个追随者来说，如果上一次收到的日志索引大于将要收到的日志索引（nextIndex）：通过AppendEntries RPC将 nextIndex 之后的所有日志条目发送出去

#### 如果发送成功

将该追随者的 nextIndex和matchIndex更新

#### 如果由于日志不一致导致AppendEntries RPC失败

nextIndex递减并且重新发送

### 如果存在一个满足N > commitIndex和matchIndex[i] >= N并且log[N].term == currentTerm的 N，则将commitIndex赋值为 N

# 五大原则

## 选举安全原则（Election Safety）

一个任期（term）内最多允许有一个领导人被选上（5.2节）

## 领导人只增加原则（Leader Append-Only）

领导人永远不会覆盖或者删除自己的日志，它只会增加条目

## 日志匹配原则（Log Matching）

如果两个日志在相同的索引位置上的日志条目的任期号相同，那么我们就认为这个日志从头到这个索引位置之间的条目完全相同（5.3 节）

## 领导人完全原则（Leader Completeness)

如果一个日志条目在一个给定任期内被提交，那么这个条目一定会出现在所有任期号更大的领导人中

## 状态机安全原则（State Machine Safety）

如果一个服务器已经将给定索引位置的日志条目应用到状态机中，则所有其他服务器不会在该索引位置应用不同的条目（5.4.3节）

# 领导选举（Leader election）

## 什么时候会触发选举

选举超时。

## 投票规则

### 在一个任期内，一台服务器最多能给一个候选人投票，按照先到先服务原则（first-come-first-served）

### 使用投票的方式来阻止没有包含全部日志条目的服务器赢得选举

## 追随者

### 当服务器启动时，初始化为追随者

### 选举超时（election timeout）

如果一个追随者在一个周期内没有收到心跳信息，就叫做选举超时（election timeout）,然后它就会假定没有可用的领导人并且开始一次选举来选出一个新的领导人。

## 候选人

为了开始选举，一个追随者会自增它的当前任期并且转换状态为候选人。然后，它会给自己投票并且给集群中的其他服务器发送 RequestVote RPC。

### 它赢得了选举

一个候选人如果在一个任期内收到了来自集群中大多数服务器的投票就会赢得选举。

### 另一台服务器赢得了选举

当一个候选人等待别人的选票时，它有可能会收到来自其他服务器发来的声明其为领导人的 AppendEntries RPC。

如果这个领导人的任期（包含在它的 RPC 中）比当前候选人的当前任期要大，则候选人认为该领导人合法，并且转换自己的状态为追随者。

如果在这个 RPC 中的任期小于候选人的当前任期，则候选人会拒绝此次 RPC， 继续保持候选人状态。

### 一段时间后没有任何一台服务器赢得了选举

如果许多追随者在同一时刻都成为了候选人，选票会被分散，可能没有候选人能获得大多数的选票。

当这种情形发生时，每一个候选人都会超时，并且通过自增任期号和发起另一轮 RequestVote RPC 来开始新的选举。

# 日志复制

## 什么叫已经提交的

一旦被领导人创建的条目已经复制到了大多数的服务器上，这个条目就称为可被提交的。

## 一旦选出了领导人，它就开始接收客户端的请求

每一个客户端请求都包含一条需要被复制状态机（replicated state machine）执行的命令。

领导人把这条命令作为新的日志条目加入到它的日志中去，然后并行的向其他服务器发起 AppendEntries RPC ，要求其它服务器复制这个条目。当这个条目被安全的复制之后，领导人会将这个条目应用到它的状态机中并且会向客户端返回执行结果。

如果追随者崩溃了或者运行缓慢或者是网络丢包了，领导人会无限的重试 AppendEntries RPC（甚至在它向客户端响应之后）知道所有的追随者最终存储了所有的日志条目。

## 如何保证不同服务器上的一致性

### 如果在不同日志中的两个条目有着相同的索引和任期号，则它们所存储的命令是相同的

领导人在一个任期里在给定的一个日志索引位置最多创建一条日志条目，同时该条目在日志中的位置也从来不会改变。

### 如果在不同日志中的两个条目有着相同的索引和任期号，则它们之间的所有条目都是完全一样的

源于AppendEntries 的一个简单的一致性检查。

当发送一个 AppendEntries RPC 时，领导人会把新日志条目紧接着之前的条目的索引位置和任期号都包含在里面。

如果追随者没有在它的日志中找到相同索引和任期号的日志，它就会拒绝新的日志条目。

这个一致性检查就像一个归纳步骤：一开始空的日志的状态一定是满足日志匹配原则的，一致性检查保证了当日志添加时的日志匹配原则。因此，只要 AppendEntries 返回成功的时候，领导人就知道追随者们的日志和它的是一致的了。

## 领导人通过强制追随者们复制它的日志来处理日志的不一致

这就意味着，在追随者上的冲突日志会被领导者的日志覆盖。

# 安全性

## 为什么会有安全性问题

之前的章节中讨论了 Raft 算法是如何进行领导选取和复制日志的。

然而，到目前为止这个机制还不能保证每一个状态机能按照相同的顺序执行同样的指令。

例如，当领导人提交了若干日志条目的同时一个追随者可能宕机了，之后它又被选为了领导人然后用新的日志条目覆盖掉了旧的那些，最后，不同的状态机可能执行不同的命令序列。

## 选举限制

一个候选人为了赢得选举必须要和集群中的大多数进行通信，这就意味着每一条已经提交的日志条目最少在其中一台服务器上出现。

如果候选人的日志至少和大多数服务器上的日志一样新，那么它一定包含有全部的已经提交的日志条目。

RequestVote RPC 实现了这个限制：这个 RPC包括候选人的日志信息，如果它自己的日志比候选人的日志要新，那么它会拒绝候选人的投票请求。

## 如何判断日志一样新

Raft 通过比较日志中最后一个条目的索引和任期号来决定两个日志哪一个更新。如果两个日志的任期号不同，任期号大的更新；如果任期号相同，更长的日志更新。

# 动态改变集群成员

## 为什么会更改集群配置

### 替换掉那些崩溃的机器

### 更改复制级别

## 共同一致是旧的配置和新的配置的组合

### 日志条目被复制给集群中新、老配置的所有服务器。

### 新、老配置的服务器都能成为领导人。

### 需要分别在两种配置上获得大多数的支持才能达成一致（针对选举和提交）

## 如何实现更改配置

### 当一个领导人接收到一个改变配置 Cold 为 Cnew 的请求

### 领导人首先在它的日志中创建 Cold,new配置条目并且将它提交到Cold,new（使用旧配置的大部分服务器和使用新配置的大部分服务器）

### 一旦 Cold,new 被提交，这个时候，领导人创建一条关于 Cnew 配置的日志条目并复制给集群就是安全的了

## 三个问题

### 一开始的时候新的服务器可能没有任何日志条目

### 集群的领导人可能不是新配置的一员

### 移除不在 Cnew 中的服务器可能会扰乱集群

# 日志压缩

## 快照（snapshot）

### 两个问题

#### 服务器必须决定什么时候应该创建快照

#### 写入快照需要花费显著的一段时间，并且我们还不希望影响到正常操作