

In Search of an Understandable Consensus Algorithm

USENIX ATC 14, Diego Ongaro and John Ousterhout

Xinyu Yang

Shanghai Jiao Tong University

June 2021

Table of Contents

① Introduction

② Raft Basics

③ Leader Election

④ Log replication

⑤ Safety

Table of Contents

① Introduction

② Raft Basics

③ Leader Election

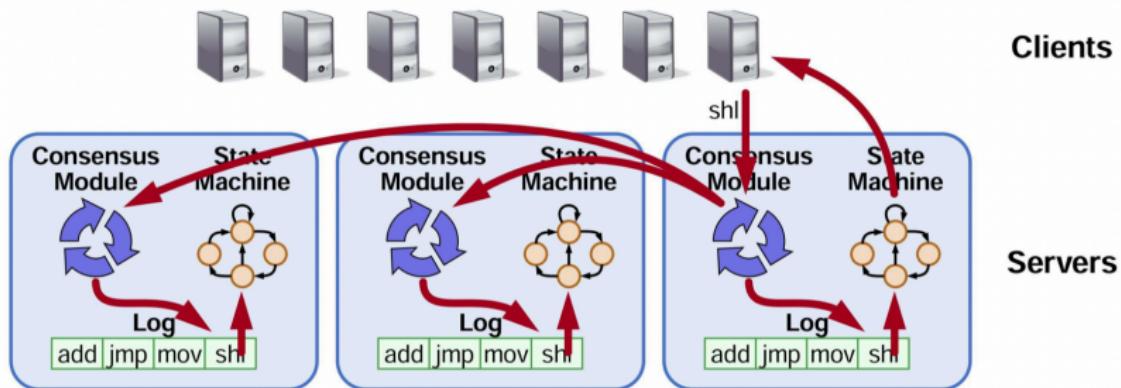
④ Log replication

⑤ Safety

Consensus Algorithm

定义：一致性算法

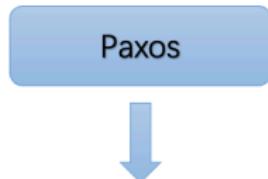
组织机器使其状态最终一致并允许局部失败的算法。



Raft: an understandable consensus algorithm

发展历史:

如何就一个提案达成一致

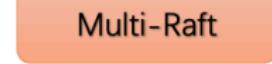


如何就一批提案达成一致



Raft

单个切片



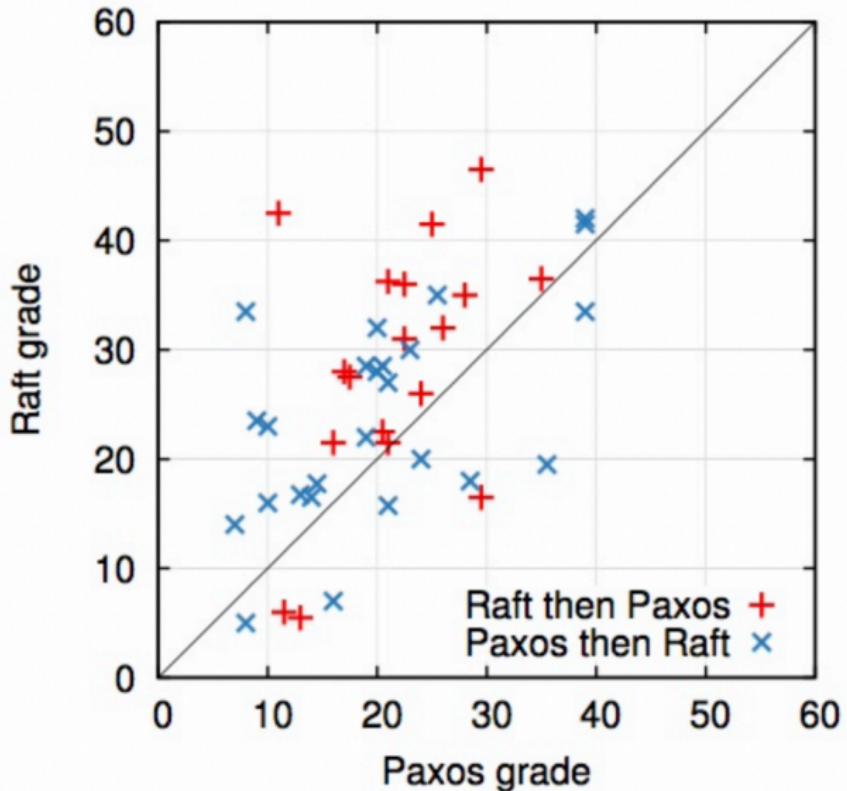
多个切片

Raft vs Multi-Paxos

Raft vs Multi-Paxos:

- Raft 协议强调日志的连续性，Multi-Paxos 则允许日志有空洞。
- Raft 领导选举有限制，必须包含最新、最全日志的节点才能被选为 leader. 而 Multi-Paxos 没有这个限制，日志不完备的节点也能成为 leader。

Understandability



Understandability

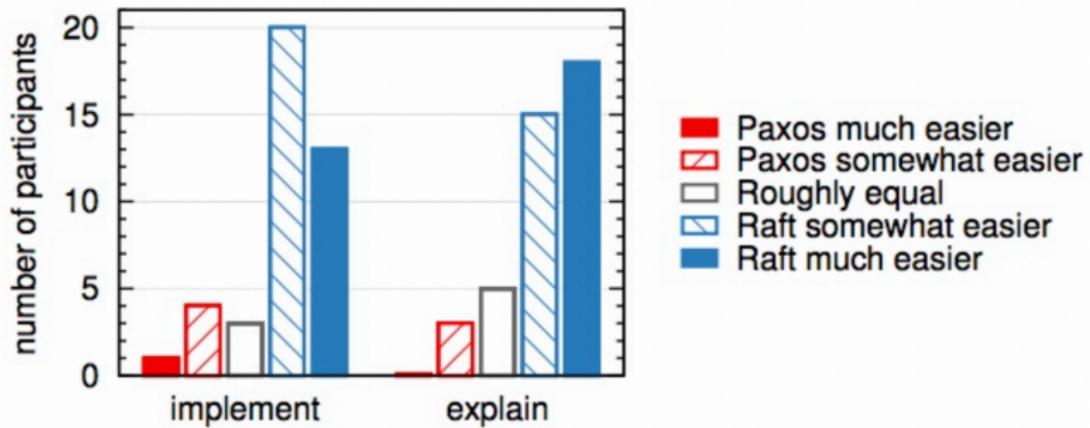


Table of Contents

① Introduction

② Raft Basics

③ Leader Election

④ Log replication

⑤ Safety

Replicated state machine

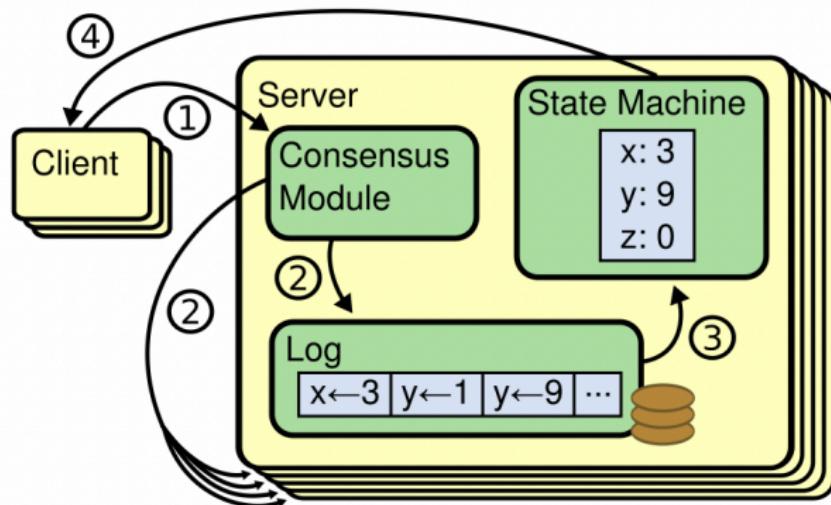


Figure 1: Replicated state machine architecture. The consensus algorithm manages a replicated log containing state machine commands from clients. The state machines process identical sequences of commands from the logs, so they produce the same outputs.

输入: 写入命令

输出: 所有节点最终处于相同的状态

约束:

- 安全性保证: 在非拜占庭错误情况下, 包括网络延迟、分区、丢包、冗余和乱序等错误都可以保证正确。
- 可用性: 集群中只要有大多数的机器可运行并且能够相互通信、和客户端通信, 就可以保证可用。
- 不依赖时序来保证一致性: 物理时钟错误或者极端的消息延迟只有在最坏情况下才会导致可用性问题。
- 快速响应: 通常情况下, 一条指令可以尽可能快的在集群中大多数节点响应一轮远程过程调用时完成。小部分比较慢的节点不会影响系统整体的性能。

A possible solution: Raft

Main idea: strong leader

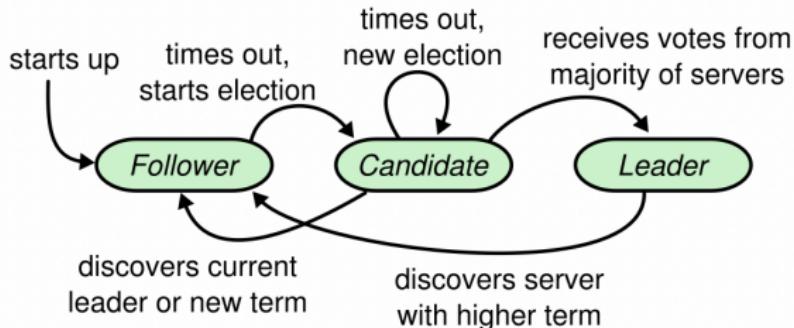
Raft implements consensus by first electing a distinguished leader, then giving the leader complete responsibility for managing the replicated log.

Decomposing into three independent subproblems:

- Leader election
- Log replication
- Safety

Some terminologies

Server states:



Term:

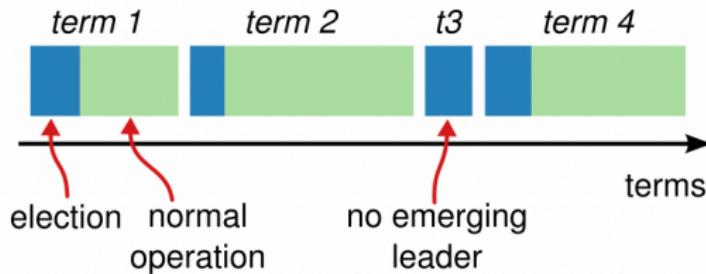


Table of Contents

① Introduction

② Raft Basics

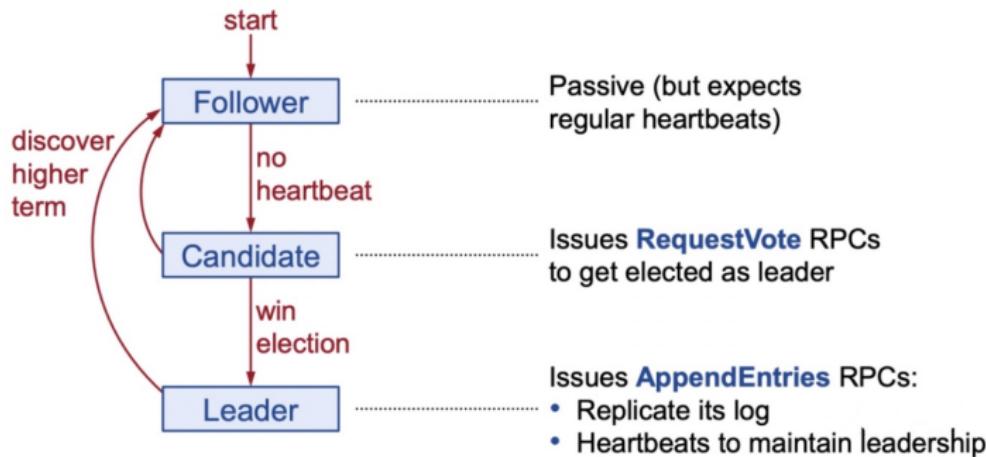
③ Leader Election

④ Log replication

⑤ Safety

When to start an election?

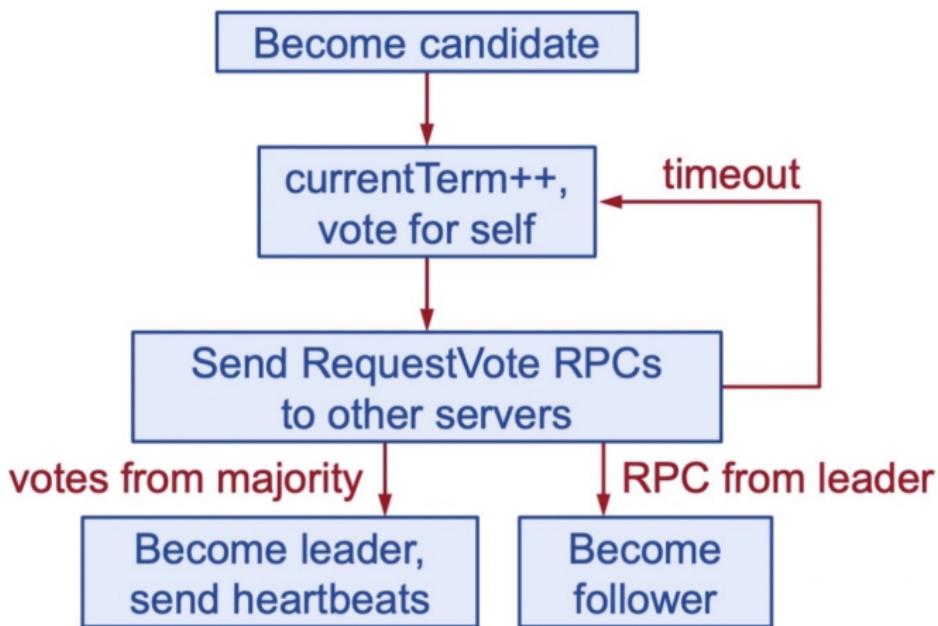
- Raft 集群启动之后开始第一次选举。
- Leader 出现故障，无法使用心跳机制维持自己的统治，导致选举超时机制触发，节点开始尝试新一轮的选举



心跳机制：领导者周期性的向所有跟随者发送心跳包（即不包含日志项内容的附加日志项 RPCs）来维持自己的权威。

Election process

When a node begins an election:



Some problems

- ① How to ensure that at most one candidate can win the election for a particular term?
- ② How to ensure that one node only vote once in a term?
- ③ What happens if no candidate obtains a majority?
- ④

Performance

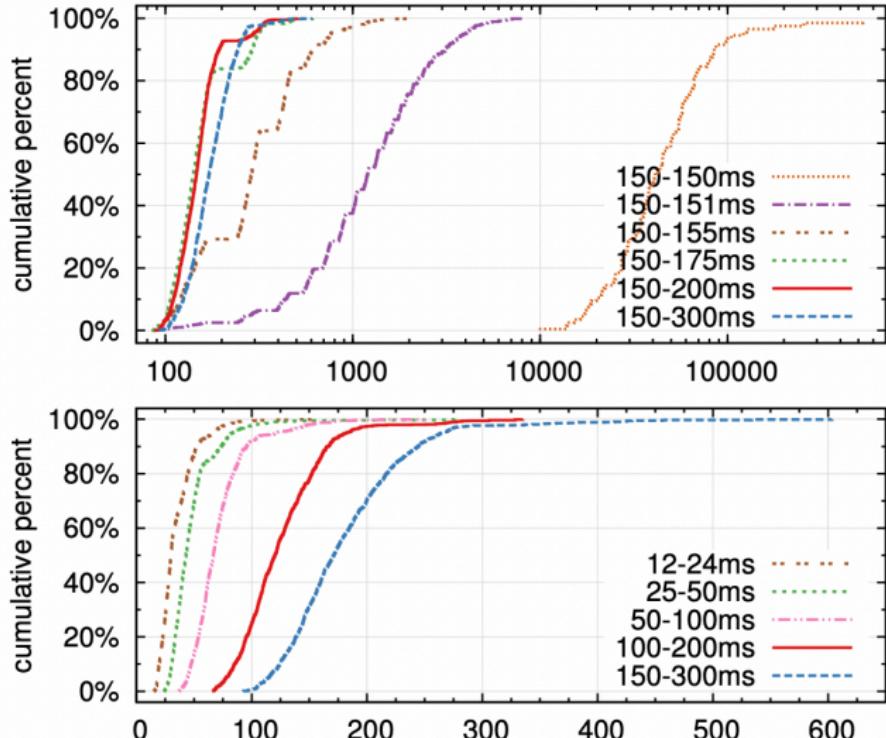


Table of Contents

① Introduction

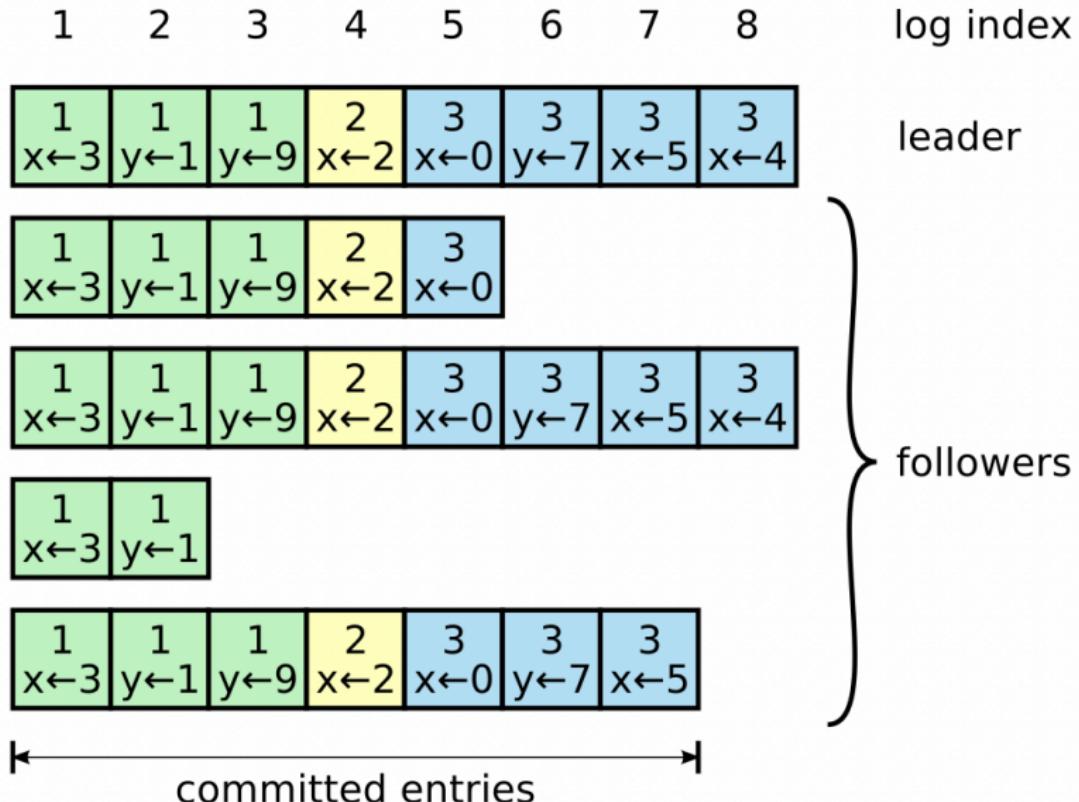
② Raft Basics

③ Leader Election

④ Log replication

⑤ Safety

Log structure



Log replication process

一些基本要求和定义：

日志必须持久化存储。一个节点必须先将记录安全写到磁盘，才能向系统中其他节点返回响应。

如果一条日志记录被存储在超过半数的节点上，我们认为该记录已提交 (committed)——这是 Raft 非常重要的特性！如果一条记录已提交，意味着状态机可以安全地执行该记录。

Log replication process

1. 客户端向 Leader 发送命令，希望该命令被所有状态机执行；
2. Leader 先将该命令追加到自己的日志中；
3. Leader 并行地向其它节点发送 AppendEntries RPC，等待响应；
4. 收到超过半数节点的响应，则认为新的日志记录是被提交的：
 - Leader 将命令传给自己的状态机，然后向客户端返回响应
 - 此外，一旦 Leader 知道一条记录被提交了，将在后续的 AppendEntries RPC 中通知已经提交记录的 Followers
 - Follower 将已提交的命令传给自己的状态机
5. 如果 Follower 宕机/超时：Leader 将反复尝试发送 RPC；

Table of Contents

① Introduction

② Raft Basics

③ Leader Election

④ Log replication

⑤ Safety

Election Safety: at most one leader can be elected in a given term. §5.2

Leader Append-Only: a leader never overwrites or deletes entries in its log; it only appends new entries. §5.3

Log Matching: if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index. §5.3

Leader Completeness: if a log entry is committed in a given term, then that entry will be present in the logs of the leaders for all higher-numbered terms. §5.4

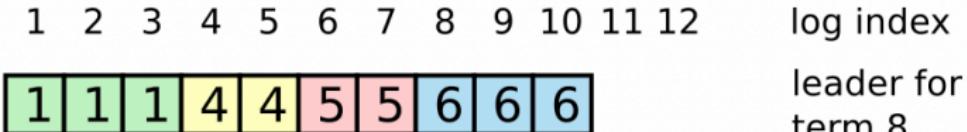
State Machine Safety: if a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index. §5.4.3

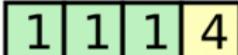
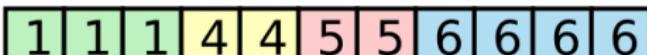
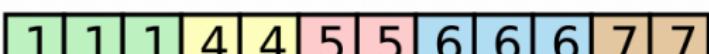
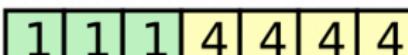
Log Matching

一致性检查机制：Raft 通过 AppendEntries RPC 来检测这两个属性。

- 对于每个 AppendEntries RPC 包含新日志记录之前那条记录的索引 (prevLogIndex) 和任期 (prevLogTerm)；
- Follower 检查自己的 index 和 term 是否与 prevLogIndex 和 prevLogTerm 匹配，匹配则接收该记录；否则拒绝；

Log Matching



- (a) 
- (b) 
- (c) 
- (d) 
- (e) 
- (f) 
- possible followers

Log Matching

新的 Leader 必须使 Follower 的日志与自己的日志保持一致，通过：

- 删除 Extraneous Entries；
- 补齐 Missing Entries；

实现方法：Leader 为每个 Follower 保存 nextIndex：

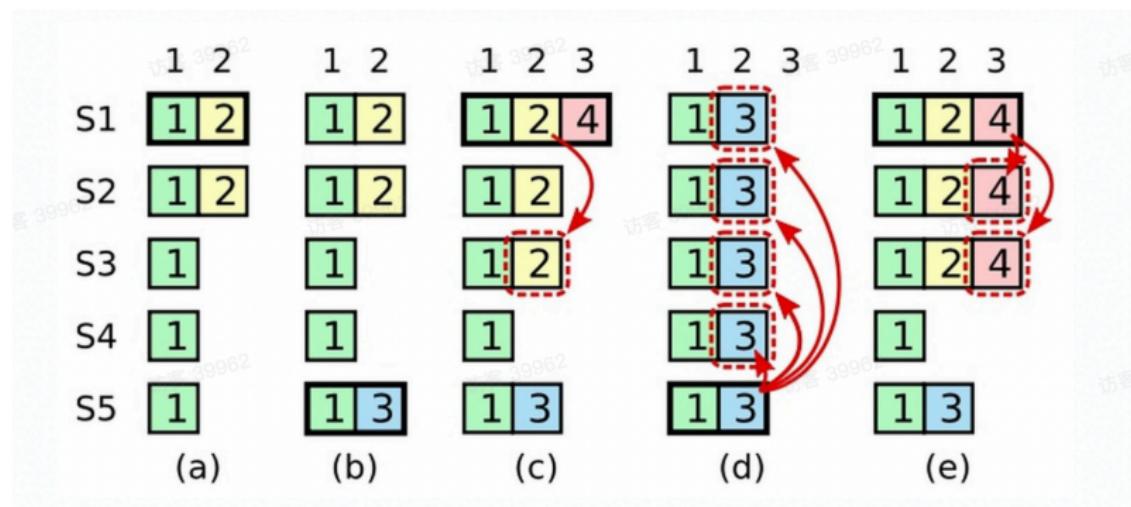
- 下一个要发送给 Follower 的日志索引；
- 初始化为： $1 + \text{Leader, 最后一条日志的索引}$ ；

Leader 通过 nextIndex 来修复日志。当 AppendEntries RPC 一致性检查失败，递减 nextIndex 并重试

Leader Completeness: Election restriction

选民只会投票给任期比自己大，最后一条日志比自己新（任期大于或者等于时索引更大）的候选人。

问题：



Leader Completeness: Election restriction

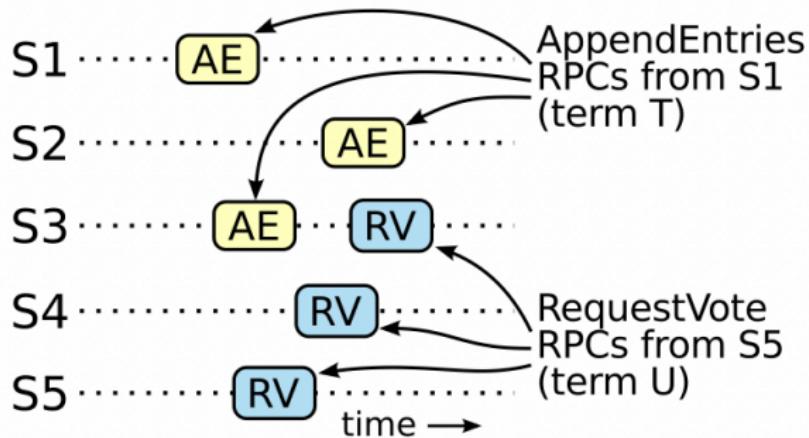
解决方法：Leader 提交一条日志时需满足以下两个条件：

- 日志必须存储在超过半数的节点上；
- Leader 必须看到：超过半数的节点上还必须存储着至少一条自己任期内的日志；

实现方法：新上任的领导者在接受客户端写入命令之前需要提交一个 no-op(空命令)，携带自己任期号的日志复制到大多数集群节点上才能真正的保证选举限制的成立。

Leader Completeness: Proof

反证法：假设任期 T 的领导人（领导人 T）在任期内提交了一条日志条目 L，但是这条日志条目没有被存储到未来某个任期的领导人的日志中。设大于 T 的最小任期 U 的领导人 U 没有这条日志条目。



Leader Completeness: Proof

1. 至少有一个投票给领导人 U 的节点中含有已经提交的那条日志 L ：
 - 领导人 T 复制 L 给集群中的大多数节点。
 - 领导人 U 从集群中的大多数节点赢得了选票。
2. 领导人 U 选举时存在以下三个事实：
 - 领导人 U 一定没有 L 。
 - 投票者必须在给领导人 U 投票之前先接受了 L 。
 - 投票者在给领导人 U 投票时依然保存有 L 。

3. 那么根据选举限制，节点只会将选票投给至少与自己一样新的节点，假设节点 C 是上文中提到的包含 L 的投票者，可推出以下矛盾：

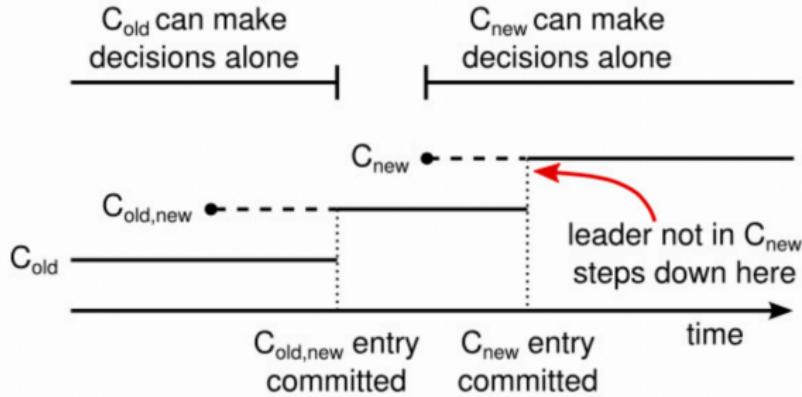
- 如果领导人 U 与节点 C 的最后一条日志的任期号一样大，根据选举限制，那么领导人 U 的日志至少和节点 C 一样长，所以领导人 U 的日志一定包含 C 的所有的日志，矛盾。
- 如果领导人 U 最后一条日志的任期号大于节点 C 最后一条日志的任期号，考虑 U 的最后一条日志对应的任期的领导 V，易知 V 的任期号比 T 大，则 V 一定包含 L（根据假设）。根据日志匹配，U 一定包含 L，矛盾。

由此领导人完全特性得证。

基于领导人完全特性和日记完全匹配特性，因为任意节点仅会将已提交日志按顺序应用于自身的状态机，更新 lastApplied 指针，因此所有节点的状态机都会最终顺序一致，即状态机安全性得证。

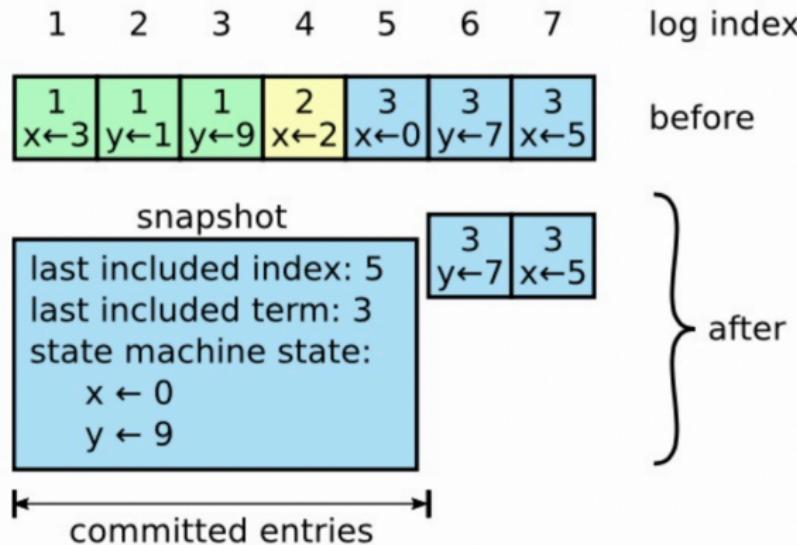
Supplement

- 跟随者和候选人崩溃: 基础的 raft 算法, 通过无限次幂等的附加复制 rpc 进行重试来解决。
- 时间和可用性:
广播时间 << 选举超时时间 << 平均故障间隔时间
- 集群成员变化: 共同一致机制



Supplement

4. 日志压缩: 快照技术



5. 客户端交互

- 客户端随机选择一个节点去访问, 如果是跟随者, 跟随者会把自己知道的领导者告知客户端。

Thanks for your attention!

In Search of an Understandable Consensus Algorithm

- Reference:
 - <https://www.usenix.org/system/files/conference/atc14/atc14-paper-ongaro.pdf>
 - https://github.com/maemual/raft-zh_cn
 - <http://thesecretlivesofdata.com/raft/>
 - <https://zhuanlan.zhihu.com/p/268571088>
 - <https://zhuanlan.zhihu.com/p/129316906>
- Presentation given by Xinyu Yang.