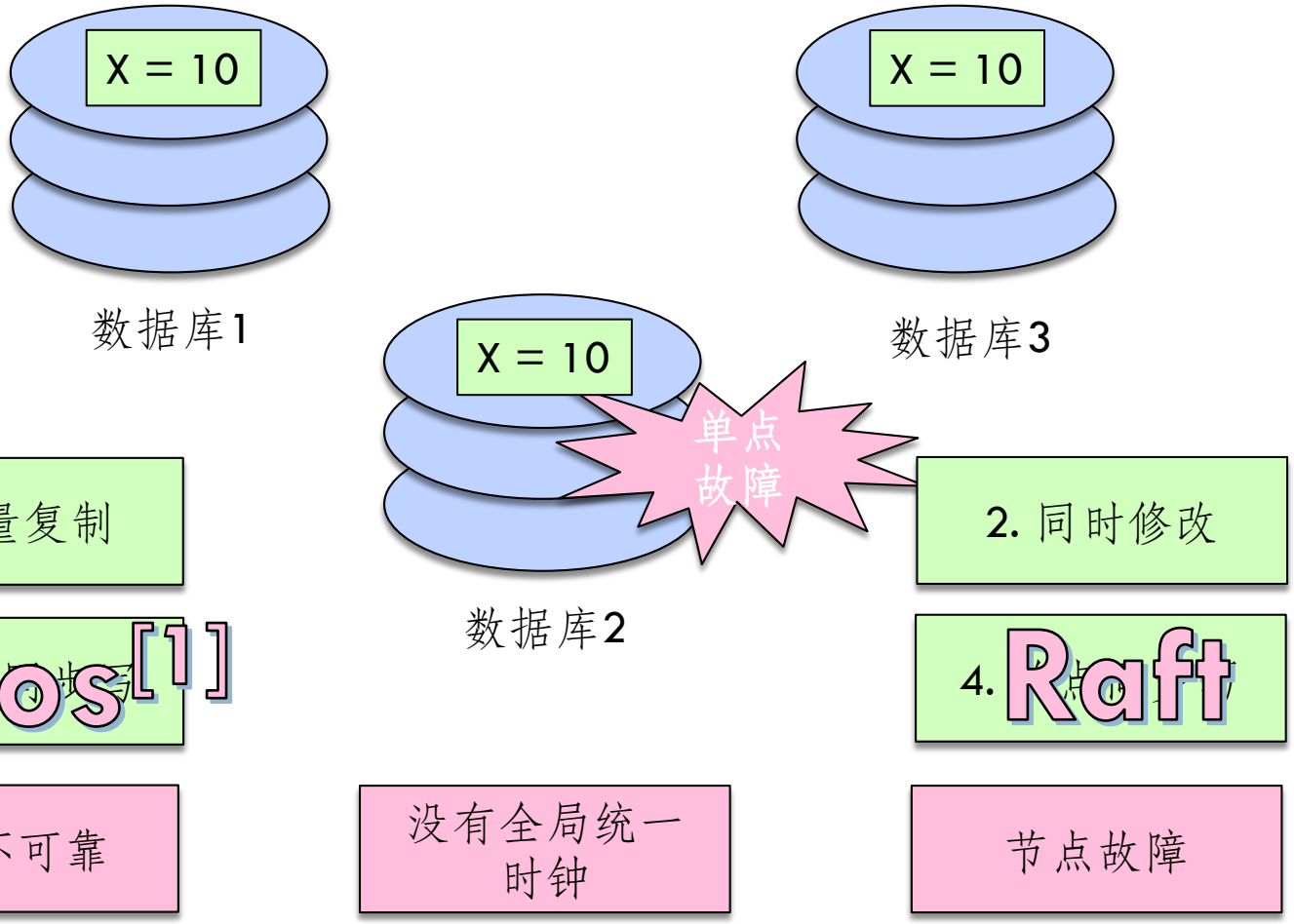
分布式共识算法

Raft[1]

[1] Ongaro, Diego, and John Ousterhout. "In search of an understandable consensus algorithm." 2014 USENIX Annual Technical Conference (Usenix ATC 14). 2014.

孙佳丽

引入：多份数据如何保持一致？



[1] Lamport, Leslie. "The part-time parliament." ACM Transactions on Computer Systems (TOCS) 16.2 (1998): 133-169.

概述

- **共识：**
  - 允许一组机器作为一致的工作组，能容忍故障，具有高可用性。
- **共识算法的保证：**
  - 在网络延迟、分区、数据丢失、重复和乱序时，返回正确的结果。
  - 不依赖于时间来确保日志的一致性。
  - 任何少数服务器故障，还可以正常工作。在2F + 1个节点的集群中，只要有F + 1个节点能够正常工作，则可以服务。
- **Paxos算法的缺点：**
  - 难以理解，状态空间复杂。
  - 没有构建现实系统的统一基础。
- **Raft算法的目标：**
  - **可理解性**：拆分算法+缩减状态空间的大小。
  - **安全性**：共识算法的保证。
- **Raft算法的应用：**
  - 分布式 KV 系统，etcd
  - 微服务基础设施，consul

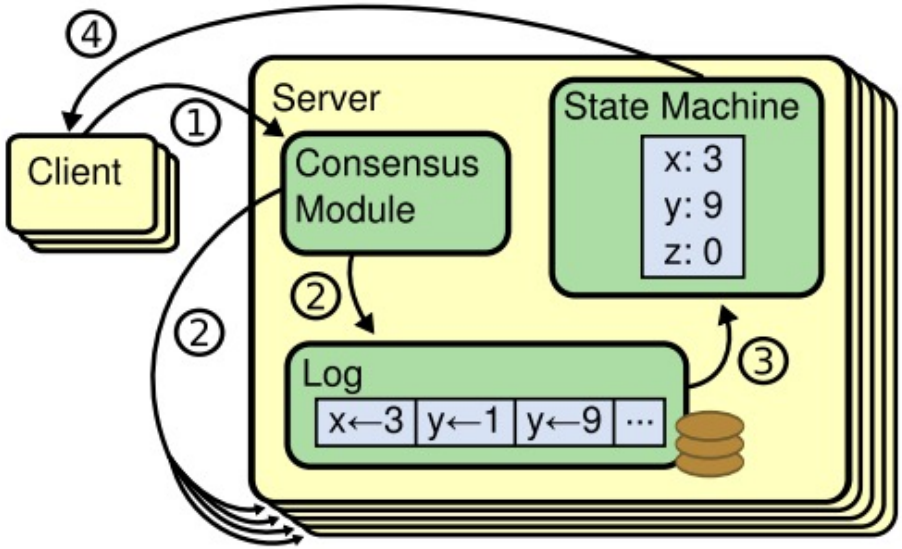
Raft介绍目录

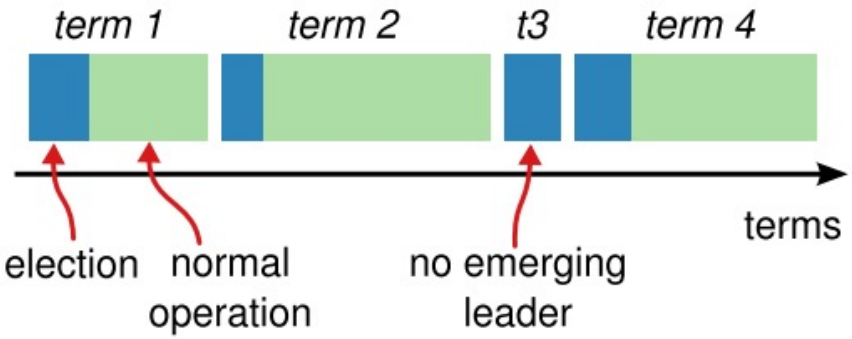
- 基本知识
- Leader选举
- 日志复制
- 安全性
- 线性一致性
- “进阶”资料

基本知识

复制状态机：

- 相同的初状态 + 相同的输入 = 相同的结束状态



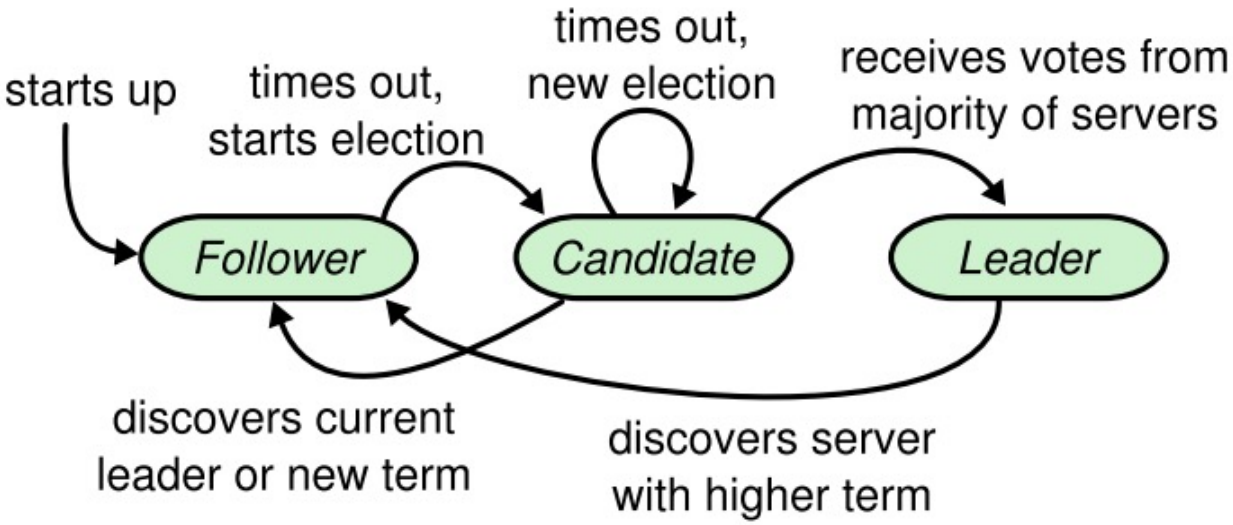


任期：

- 时间按任期划分，通过任期识别过时消息等。
- 每个任期从Leader选举开始，选举结束后才正常处理客户端请求。
- 存在某些选票被瓜分的情况，没有Leader产生，开始下一个任期。

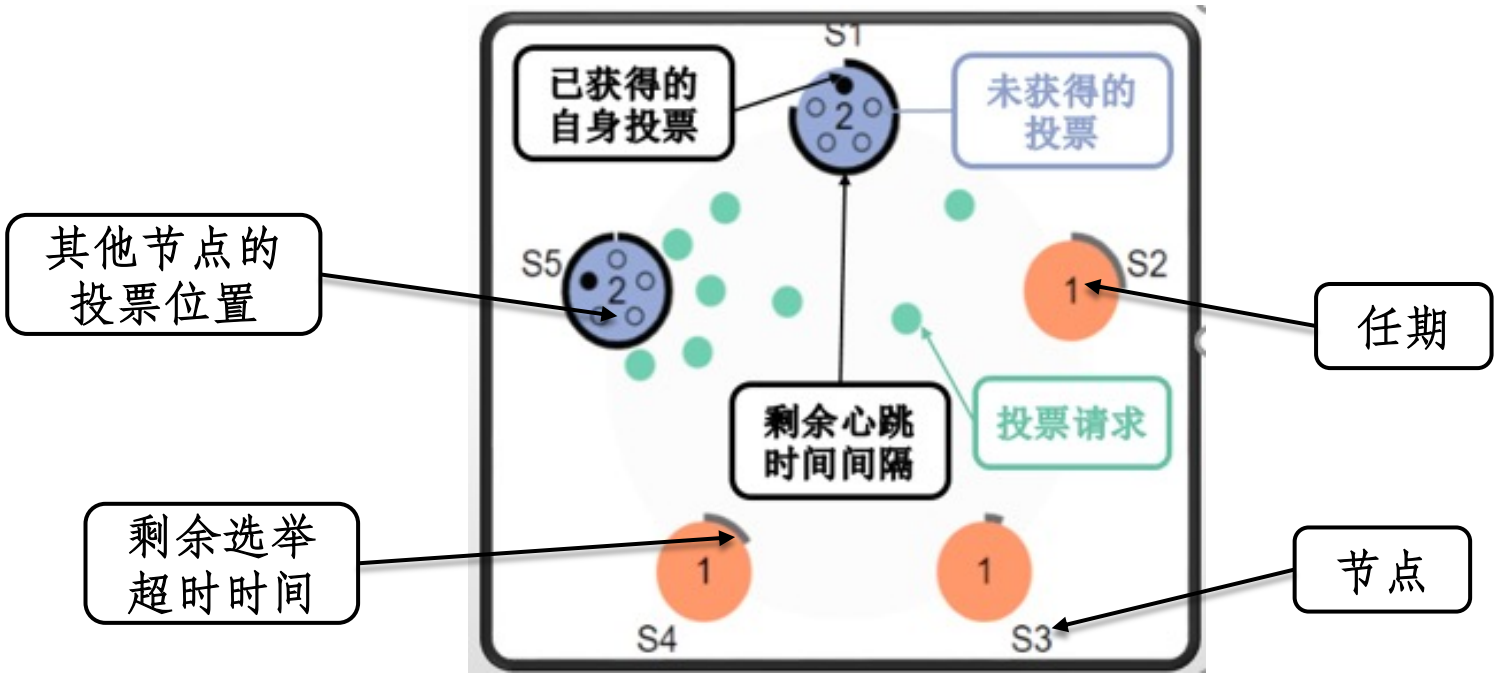
基本知识

- 三种角色：
  - **Leader**：响应客户端的请求，同步数据。
  - **Candidate**：Leader选举时的状态，获得多数选票可担任Leader。
  - **Follower**：初始启动状态，接收日志同步请求并响应。
- 三种远程过程调用（RPC）：
  - RequestVote RPC：用于Candidate 收集选票
  - AppendEntries RPC：Leader日志复制或发送心跳（不含日志项）
  - InstallSnapshot RPC：Leader通过此RPC发送快照给Follower



Leader选举

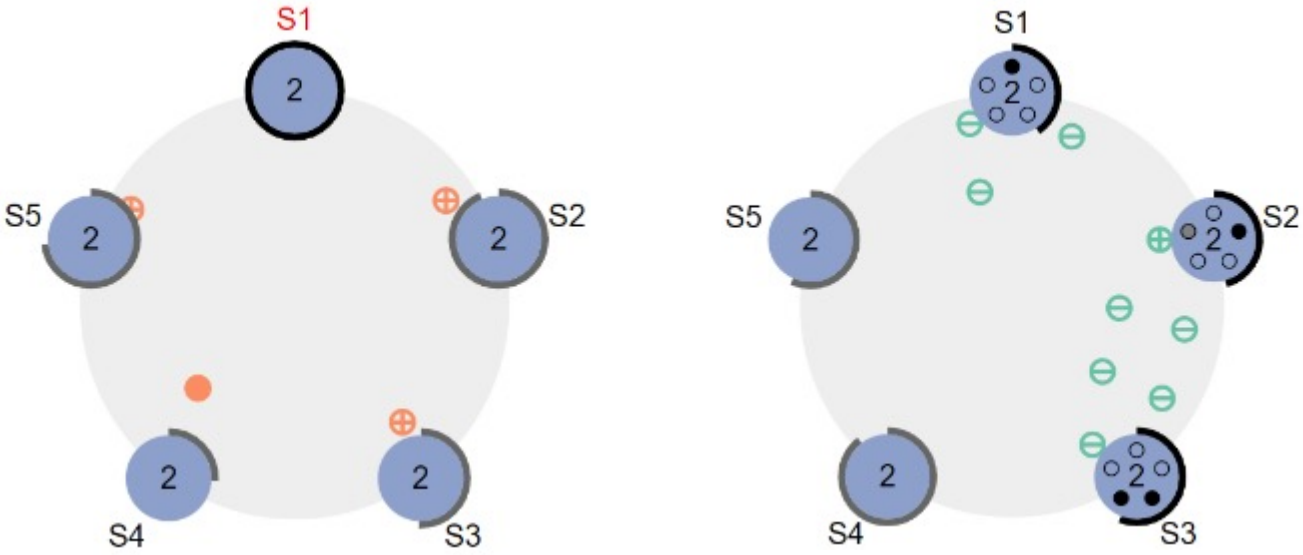
- 触发条件：
  - 一个Follower在**随机的选举超时时间（election timeout）**内没有收到来自Leader的心跳，就发起选举，选出新的Leader 。



[1] 动画展示： https://raft.github.io/

Leader选举

- 选主的过程：
  - 1. Follower选举超时后，增加当前任期，并转换为Candidate状态。
  - 2. 给自己投票，并给其他节点发送RequestVote RPC，请求投票。
  - 3. 统计其他节点的投票结果：
    - 3.1 赢得大多数选票，成为Leader。
    - 3.2 被告知有其它节点成为了Leader，转换为Follower状态。
    - 3.3 没有节点获得大多数选票（可能出现多个Candidate瓜分选票的情况），等待超时重新选举。



Leader选举

- 重要保证：
  - 随机的选举超时时间（150~300ms），减少3.3中瓜分选票的情况。
  - 每个任期内最多只有一个Leader，任何节点收到之前任期的消息都不会影响现在的状态。
  - 选票先到先得，只要**请求消息满足要求（任期+日志）**则投票。
  - 大多数选票原则，这也要求每个节点在每个任期内有且只有一张选票。

RequestVote RPC

Invoked by candidates to gather votes (§5.2).

Arguments:

term	candidate's term
candidateId	candidate requesting vote
lastLogIndex	index of candidate's last log entry (§5.4)
lastLogTerm	term of candidate's last log entry (§5.4)

Results:

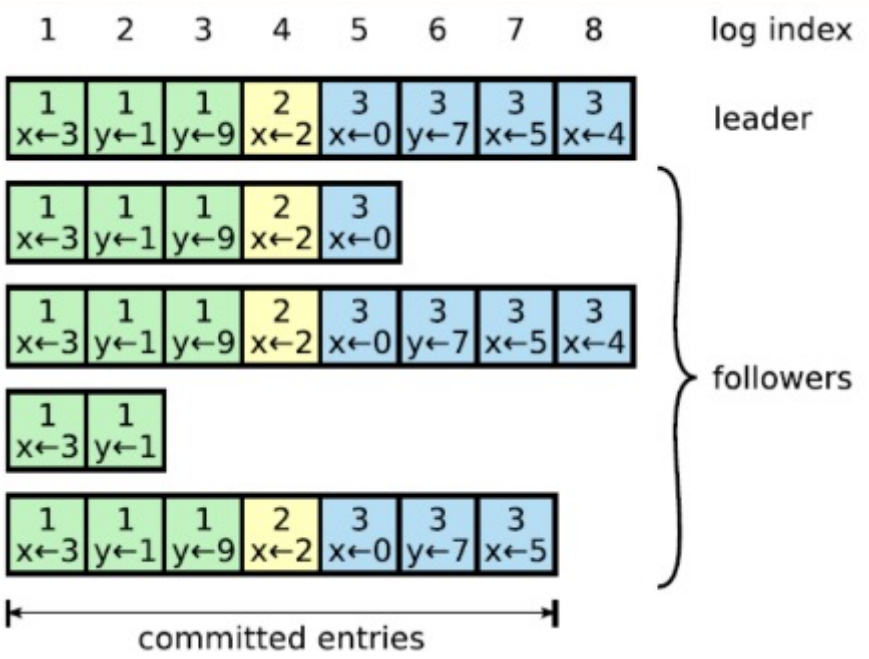
term	currentTerm, for candidate to update itself
voteGranted	true means candidate received vote

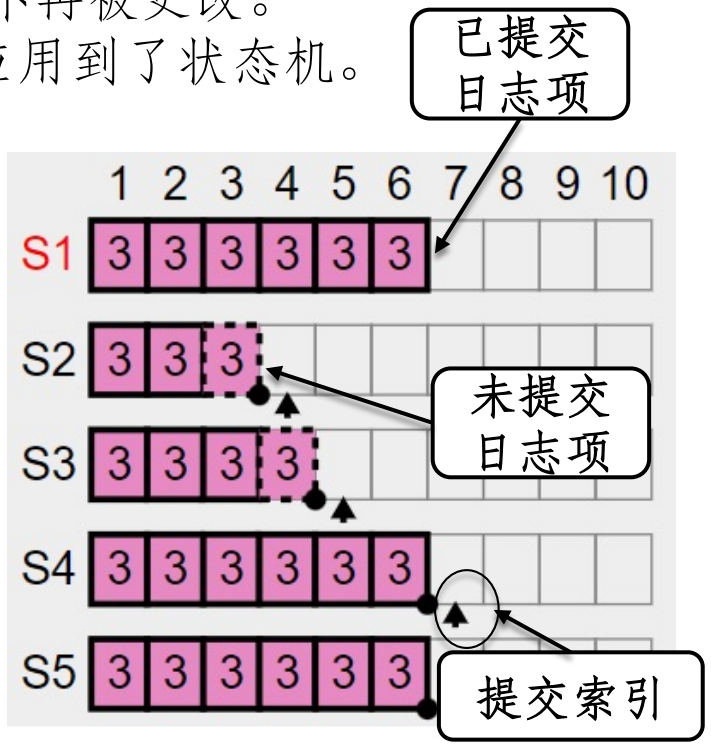
Receiver implementation:

1. Reply false if term < currentTerm (§5.1)
2. 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)

日志复制

- 日志复制：
  - 复制(replication)：某个日志项被写入到Follower的日志中。
  - 提交(commit)：如果当前任期内的日志项被多数节点写入，则可以变为提交状态。此状态下日志项不再被更改。
  - 应用(apply)：将已经提交的日志项应用到了状态机。

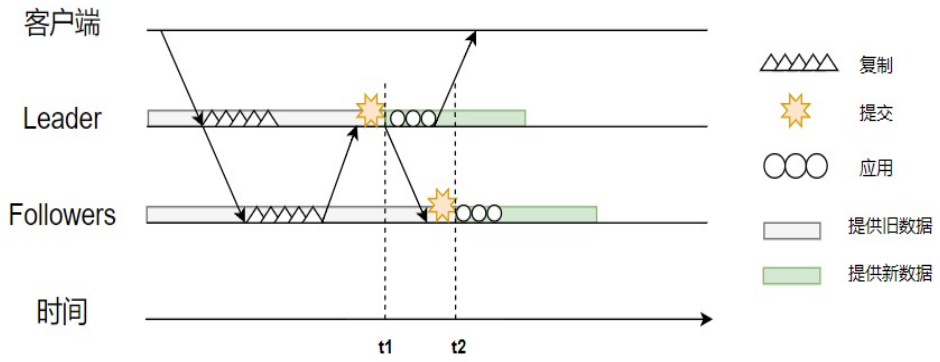




日志复制

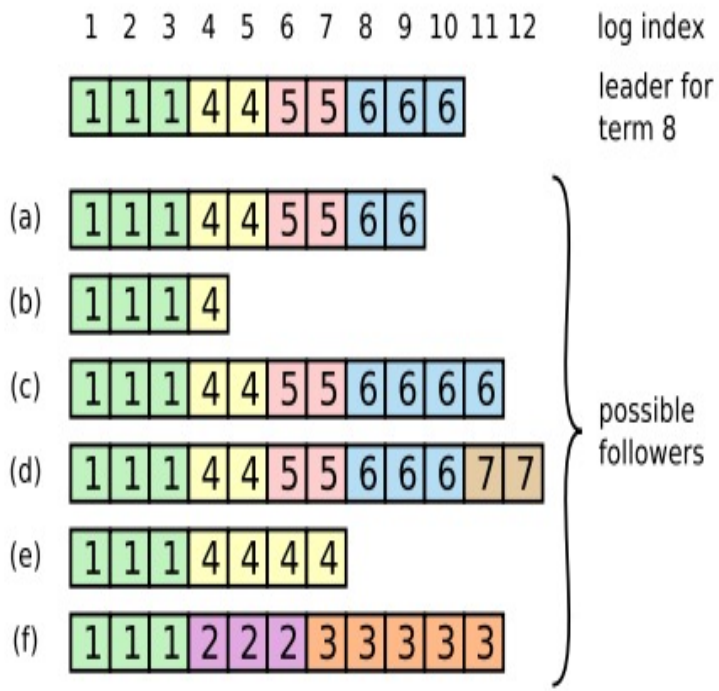
- 复制过程：
  - 1. 客户端将包含一条指令的请求发送到Leader上。
  - 2. Leader把这条指令作为日志项附加到本地的日志中，并发送AppendEntries RPC给其他服务器，复制日志项。
  - 3. Follower返回复制结果给Leader。
  - 4. 当Leader认为这个日志项已经被多数节点复制，那么在提交此日志项并将这条日志项应用到状态机后，返回给客户端。

Raft算法流程



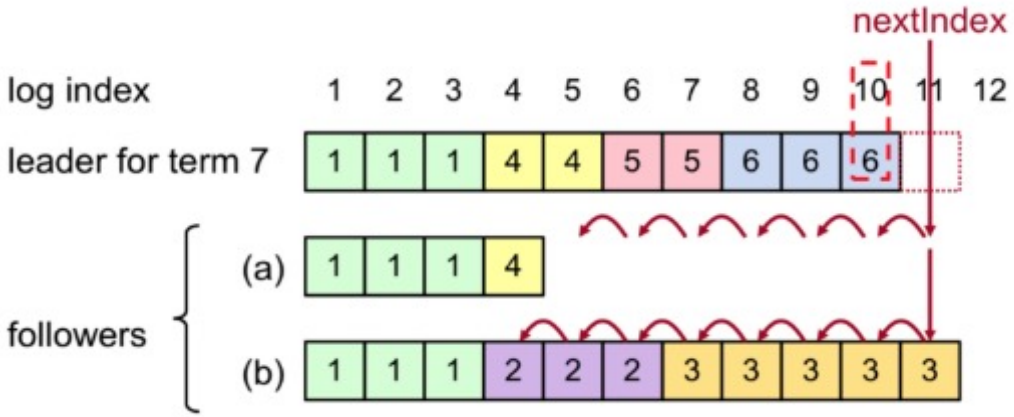
日志复制

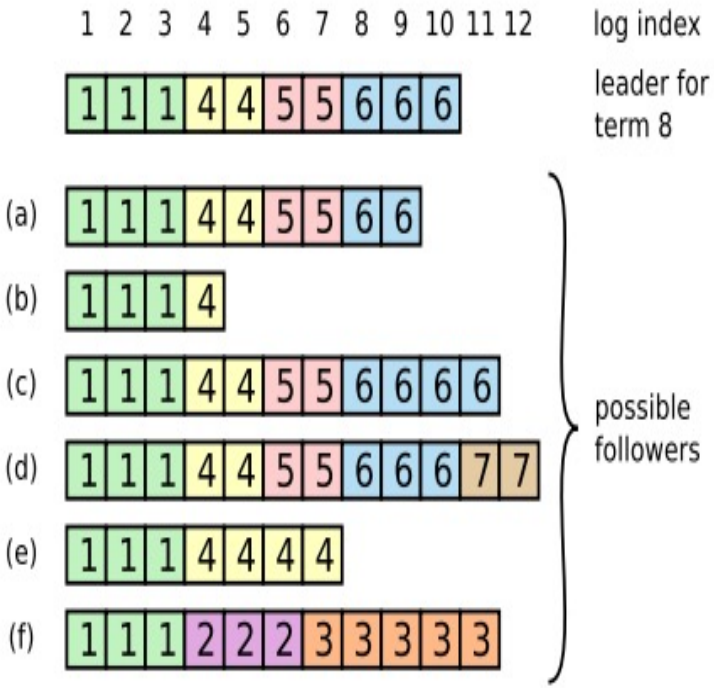
- 日志复制的保证：
  - 如果不同日志中的两个条目有着相同的索引和任期号，则它们所存储的命令是相同的。
  - 如果不同日志中的两个条目有着相同的索引和任期号，则它们之前的所有条目都是完全一样的。
- 可能出现的不一致场景：
  - a ~ b：日志项缺失
    - a在收到(6, 9)后宕机
    - b在收到(4,4)后宕机
  - c ~ d：日志项多余
    - c收到了(6,11)后，Leader宕机
    - d收到(7,11)(7,12)后，Leader宕机
  - e ~ f：日志项不匹配
    - e收到(4,6)(4,7)后宕机
    - F多收到任期2，3的日志项



日志复制

- 如何处理这些不一致问题：
  - 在发送AppendEntries RPC时，Leader会在消息中包含之前日志项的索引和任期号。如果Follower没发现对应的日志项，就拒绝。
  - Leader发现拒绝接收的消息，就会向前逐个查找日志项，并发送AppendEntries RPC，直到Follower最终找到与Leader对应的相同位置，并用Leader的日志项覆盖自身不匹配的日志项。
  - Leader 不会覆盖或删除自己的日志。
  - 优化：按任期逐个向前搜索。



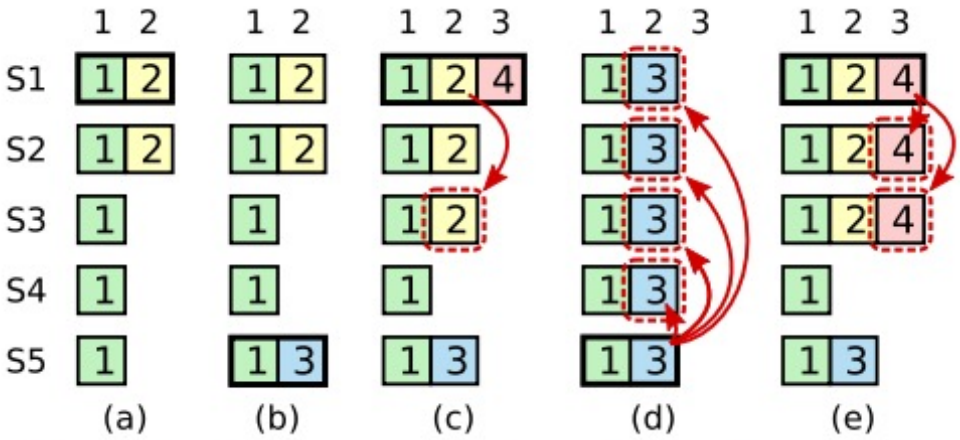


安全性

性质	描述	问题	解决
选举安全原则（Election Safety）	一个任期内最多允许有一个Leader	Split Vote	随机选举时间，多次选举
Leader只追加原则（Leader Append-Only）	Leader永远不会覆盖或者删除自己的日志，它只会增加日志项	日志不一致	强制Follower与其一致
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，那么就认为这个日志项索引位置之前的日志完全相同	日志不一致	日志复制，一致性检验
Leader完全原则（Leader Completeness)	如果一个日志项在一个给定任期内被提交，那么这个日志项一定会出现在所有任期号更大的Leader中	无法判断某个entry是否被提交	选举限制+推迟提交
状态机安全原则（State Machine Safety）	如果一个节点已经将给定索引位置的日志项应用到状态机中，则所有其他节点不会在该索引位置应用不同的日志项	反证法	

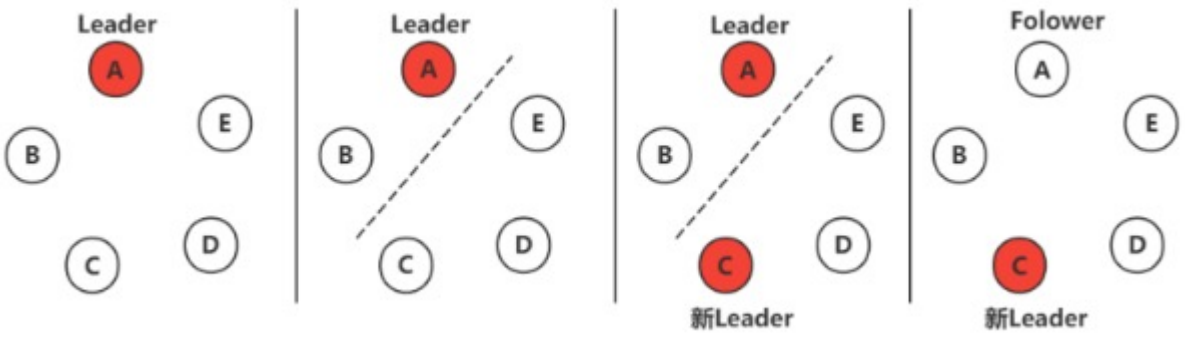
安全性

- 违背Leader完全原则：
  - 新的Leader不能通过某条日志项被保存到大多数的服务器上来断定它是否是已提交状态。
  - 解决方案：不通过计算复制的数目来提交之前任期的日志项。只有Leader当前任期的日志项才能通过计算数目来进行提交。
- 时间与可用性：
  - 广播时间 《 选举超时时间 《 单节点故障的平均时间
  - 第一个《：保证Leader能够通过发送心跳来阻止Follower开始新的选举。
  - 第二个《：保证系统的稳定运行。



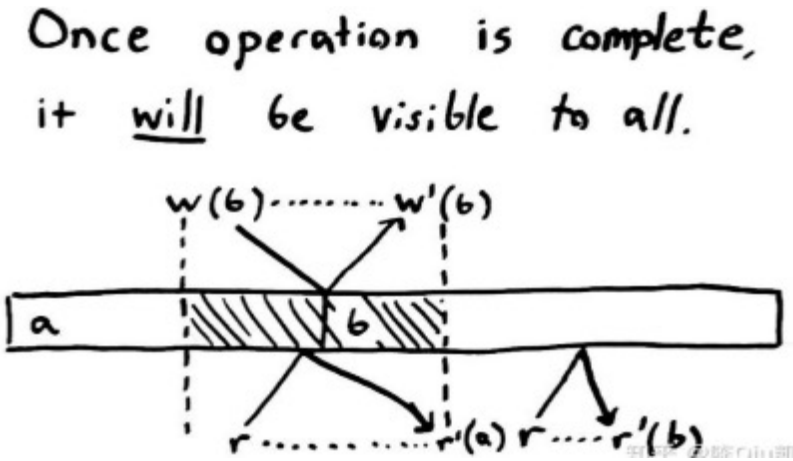
安全性

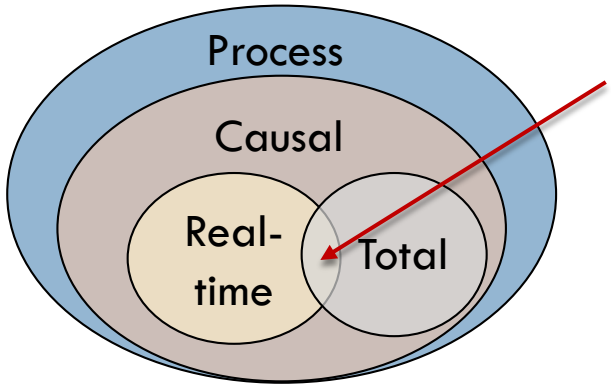
- 宕机异常
  - Leader宕机：其余节点超时重新选举，选出新的Leader。当主节点重新进入集群，接收心跳信息成为Follower。
  - Candidate宕机：集群没有出现故障问题，对于失效后恢复的从节点，接收来自Leader的心跳信息，恢复数据，保证同步。
- 网络分区
  - 问题：某一节点再次加入集群，不断增大的任期会打断系统执行。
  - 解决方案：Pre-Vote方案就像是一次Leader选举，不过不改变节点状态。如果能够获得多数选票，则可以增加任期，发起一轮选举。



强一致性

- **线性一致性**的定义
  - 数据像只有一个副本一样，在各节点上写生效顺序相同，写写先后顺序一致，读到最新的写。
- 如何基于 Raft 实现线性一致?
  - **Raft != 线性一致**。Raft的视线中，如果出现网络分区等情况，有可能读到旧时的数据。不满足先行一致性。
  - 读写操作都经过Leader并且通过日志应用到复制状态机以获得正确的结果。
  - 缺点：读操作不改变状态机的状态，但还要写日志，开销大。





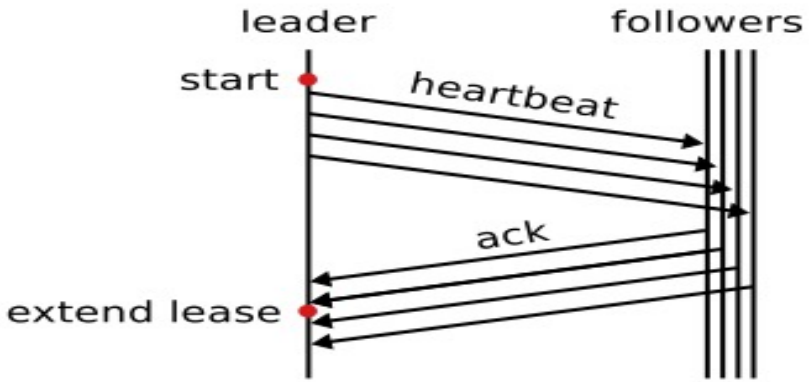
强一致性

- **Read Index：**
  - **无法确定当前的Leader就是全局唯一的Leader，因此需要额外一轮确定Leader的通信。**
  - **Leader Read**：当 Leader 接收到读请求的时候，需要通过一轮心跳RTT来确保当前自己仍然是Leader，并记录当前的commit index 为 read index，然后等到apply ≥ read index再提供读服务。
  - **Follower Read**：Follower 收到读请求，给 Leader 发送一个获取 read index的命令，仍然走一遍之前的流程，然后将read index返回给Follower，Follower等到apply≥ read index，返回客户端。
  - 优点：省掉了同步 log 的开销，能够大幅提升读的吞吐，一定程度上降低读的时延。

强一致性

Lease Read

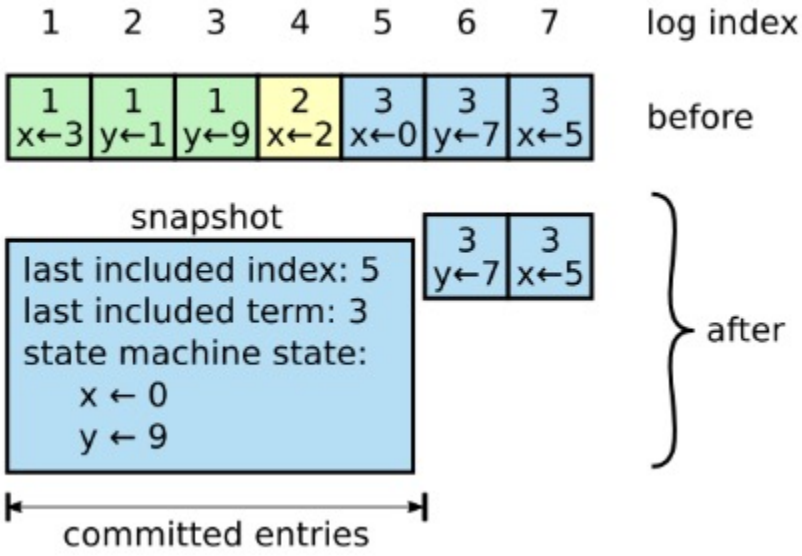
- 已经确定了当前的Leader就是全局唯一的Leader 。
- **Leader Read**：当 Leader 接收到读请求的时候，直接提供读服务。
- **Follower Read**：Follower 收到读请求，给 Leader 发送一个获取 read index的命令，Follower等到apply≥ read index，返回结果给客户端。
- 优点：与 Read Index 相比，Lease Read 进一步省去了网络交互开销，因此更能显著降低读的时延。
- 缺点：由于使用的是每个服务器各自的时钟，存在着时钟漂移现象，如果误差过大，则可线性化的保证就丢失了。



Lease = start + election timeout/ clock drift bound.

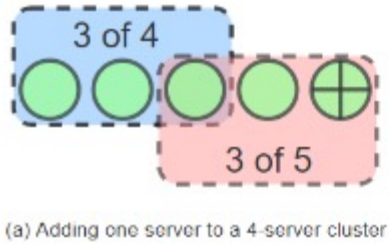
“进阶”资料

- 日志压缩：
  - 日志无限增长会影响性能。
  - 所有的节点都可以对已经提交的日志记录进行snapshot。
  - 如果某个Follower的日志落后太多，发送的log entry被丢弃，或者新添加了节点，Leader会使用InstalledSnapshot RPC发送snapshot。
  - 当日志达到某个固定的大小时做一次snapshot。
  - 可以通过使用写时复制（copy-on-write）技术避免snapshot过程影响正常的日志同步。

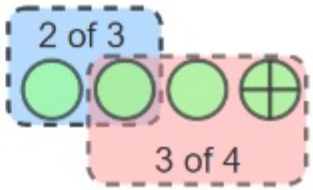


“进阶”资料

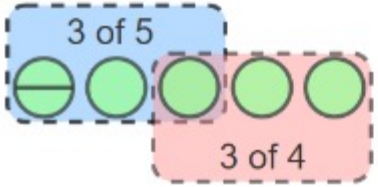
- 成员变更（自动更改配置）：
  - 单节点成员变更：
    - 成员变更限制每次只能增加或删除一个成员。
    - 存在的问题：两个更改彼此不重叠，可能导致脑裂现象。
    - 解决方案：在Leader提交当前任期的日志项之前不能开始下一次的配置，使用no-op方法。
  - 联合共识（两阶段成员变更）：
    - 出于安全性的考虑，在变更的过程中需要达成双多数派。



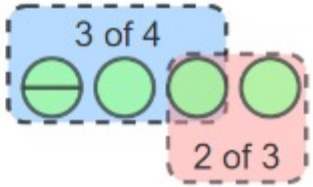
(a) Adding one server to a 4-server cluster



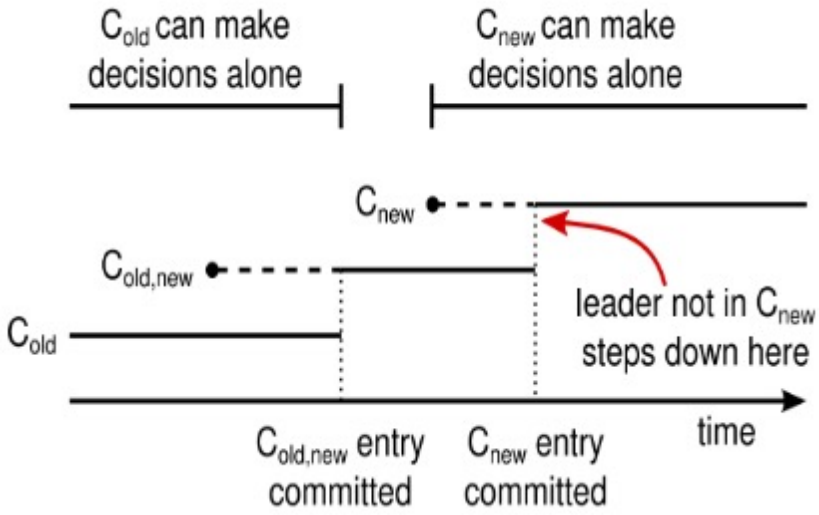
(b) Adding one server to a 3-server cluster



(c) Removing one server from a 5-server cluster

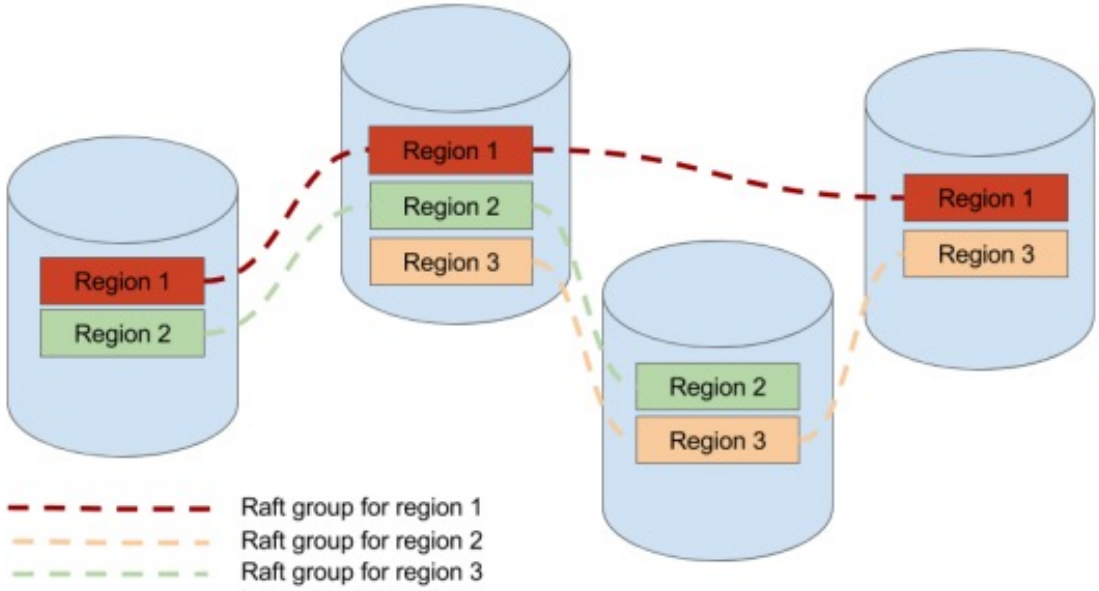


(d) Removing one server from a 4-server cluster



“进阶”资料

- Raft存在的问题：读写请求都由Leader处理，性能瓶颈。
- Multi-Raft：
  - 数据按照一定的方式切片，每一个切片的数据都有自己的副本，这些副本之间的数据使用Raft来保证数据的一致性，在全局来看整个系统中同时存在多个Raft-Group。



“进阶”资料

- 参考资料：
  - 动画Raft Understandable Distributed Consensus：http://thesecretlivesofdata.com/raft/
  - Raft详细描述动画： https://raft.github.io/
  - Raft算法详解： https://zhuanlan.zhihu.com/p/32052223
  - Raft一致性算法： https://lotabout.me/2019/Raft-Consensus-Algorithm/
  - Elasticell-Multi-Raft实现 https://zhuanlan.zhihu.com/p/33047950
  - TIDB 架构及分布式协议Paxos和Raft对比 https://blog.51cto.com/liuminkun/2377029
  - 基于 Raft 构建弹性伸缩的存储系统 https://pingcap.com/blog-cn/building-distributed-db-with-raft/
  - 什么是multi-Raft https://www.pianshen.com/article/12371692155/
  - 单服务器成员资格更改 https://groups.google.com/g/raft-dev/c/t4xj6dJTP6E
  - CONSENSUS: BRIDGING THEORY AND PRACTICE https://web.stanford.edu/~ouster/cgi-bin/papers/OngaroPhD.pdf
  - Scaling Raft https://www.cockroachlabs.com/blog/scaling-raft/
  - 深度解析Raft https://juejin.cn/post/6907151199141625870#heading-14