Raft 分布式系统中的共识算法

为什么需要共识算法

顾名思义，共识算法的应用是为了维护各节点数据的一致性，换句话说，就是使集群里的节点存储的数据是一模一样的。

举个例子

在一个分布式集群中，client发起一个请求，在集群返回成功后，在client看来数据的变更是被集群保存下来了，即便是集群中的leader在返回之后发生了宕机，已经成功返回的请求的数据变更是不可丢失的。但是，如果leader没有将这条数据及时同步到其他的fellower，那新的leader就可能会发生数据丢失的问题。

拜占庭将军问题

拜占庭位于如今的土耳其的伊斯坦布尔，是东罗马帝国的首都。由于当时拜占庭罗马帝国国土辽阔，为了防御目的，因此每个军队都分隔很远，将军与将军之间只能靠信差传消息。在战争的时候，拜占庭军队内所有将军必需达成 **一致的共识**，决定是否有赢的机会才去攻打敌人的阵营。但是，在军队内有可能存有叛徒和敌军的间谍，左右将军们的决定又扰乱整体军队的秩序，在进行共识时，结果并不代表大多数人的意见。这时候，在已知有成员不可靠的情况下，其余忠诚的将军在不受叛徒或间谍的影响下如何达成一致的协议，拜占庭问题就此形成。拜占庭假设是对现实世界的模型化，由于硬件错误、网络拥塞或断开以及遭到恶意攻击，计算机和网络可能出现不可预料的行为。

上述问题的本质就是在集群中多数节点由于硬件错误、网络拥塞或断开以及遭到恶意攻击，计算机和网络可能出现不可预料的行为的情况下达成数据共识的问题，

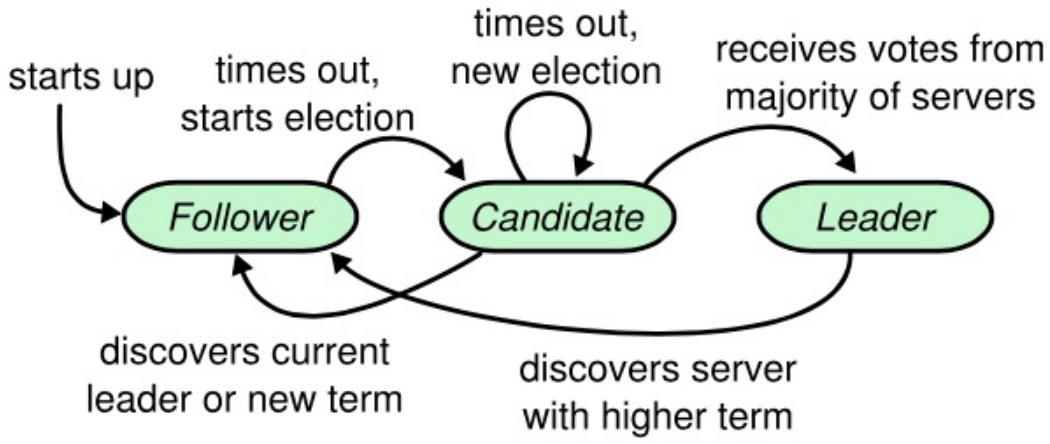
什么是 raft 算法

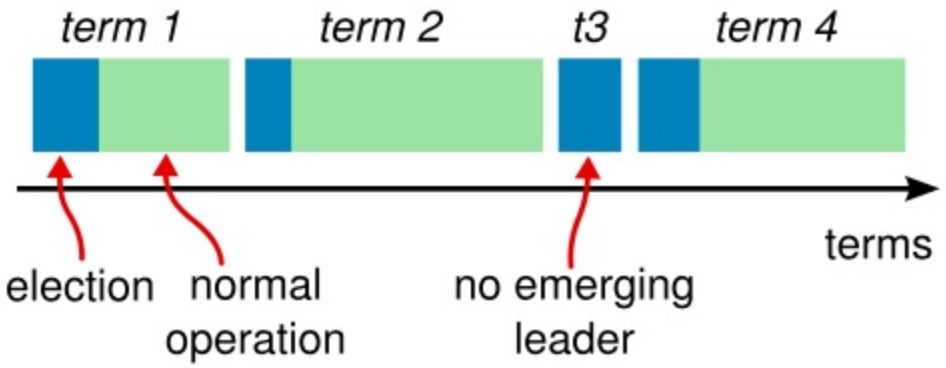
在一个由 Raft 协议的集群中有下面三类角色：

Leader（领袖）

Follower（群众）

Candidate（候选人）





如上图所示，时间被分为一个个的任期（term），每一个任期的开始都是领导人选举。在成功选举之后，一个领导人会在任期内管理整个集群。如果选举失败，该任期就会因为没有领带人而结束。

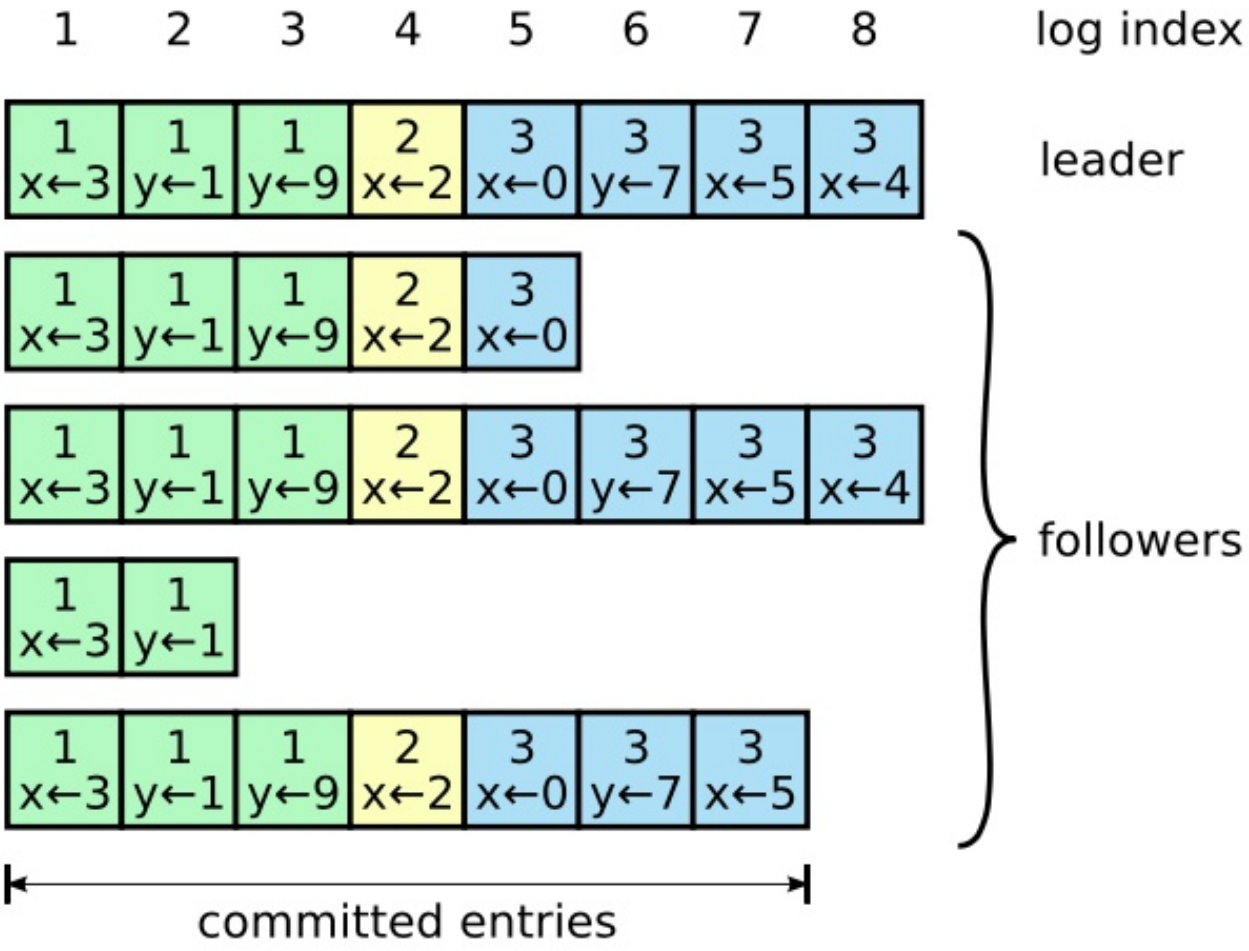
如上图所示，Raft 算法将时间划分成为任意不同长度的任期（term）。任期用连续的数字进行表示。每一个任期的开始都是一次选举（election），在一个term内，会有一个或多个候选人会试图成为领导人。如果一个候选人赢得了选举，它就会在该任期的剩余时间担任领导人。在某些情况下，选票会被瓜分，有可能没有选出领导人，那么，将会开始另一个任期，并且立刻开始下一次选举。Raft 算法保证在给定的一个任期最少要有一个领导人。

通过选出领导人，Raft 将一致性问题分解成为几个相对独立的子问题：今天我们仅考虑下面两个问题

- 领导人选取（Leader election）： 在一个领导人宕机之后必须要选取一个新的领导人
- 日志复制（Log replication）： 领导人必须从客户端接收日志然后复制到集群中的其他服务器，并且强制要求其他服务器的日志保持和自己相同

日志复制

日志格式：term + index + cmd + type

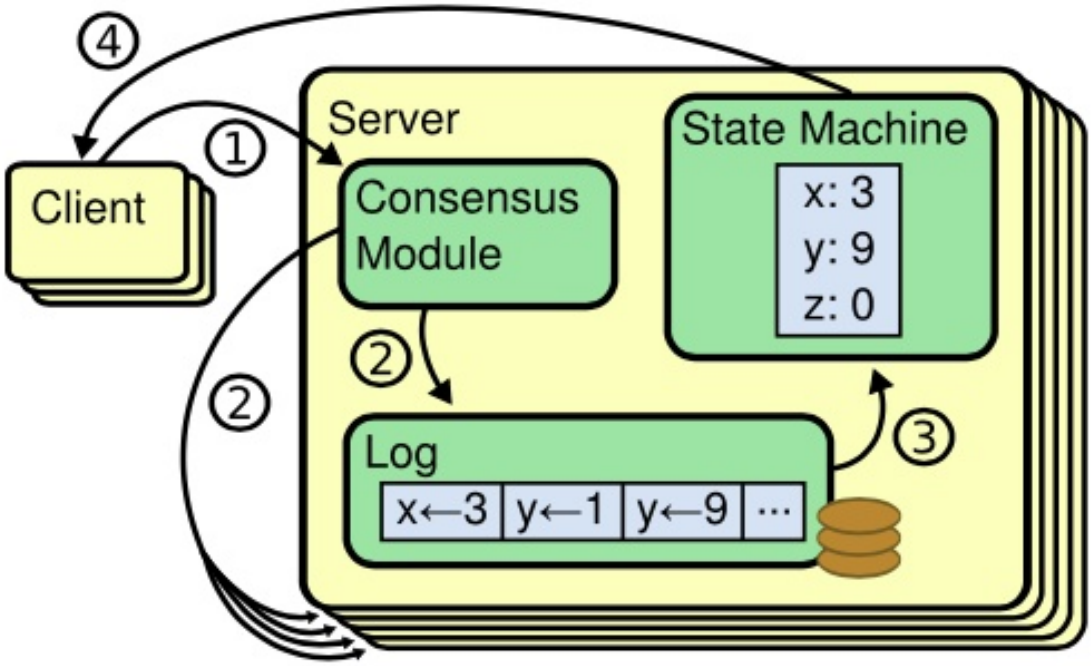


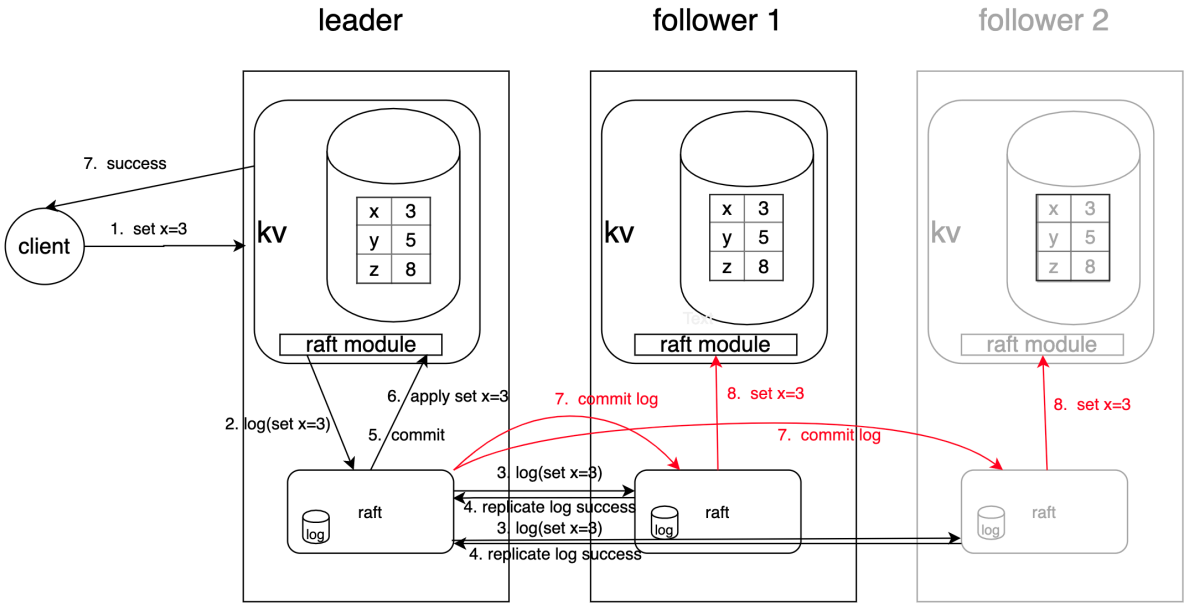
如上图所示：日志由有序编号的日志条目组成。每个日志条目包含它被创建时的任期号（每个方块中的数字），并且包含用于状态机执行的命令。如果一个条目能够被状态机安全执行，就被认为可以提交了。

日志就像 上图 所示那样组织的。每个日志条目存储着一条被状态机执行的命令和当这条日志条目被领导人接收时的任期号。日志条目中的任期号用来检测在不同服务器上日志的不一致性。

leader 决定什么时候将日志条目应用到状态机是安全的；这种条目被称为可被提交（commited）。Raft 保证可被提交（commited）的日志条目是持久化的并且最终会被所有可用的状态机执行。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。领导人日志中之前的条目都是可被提交的（commited），包括由之前的领导人创建的条目。leader 跟踪记录它所知道的被提交条目的最大索引值，并且这个索引值会包含在之后的 AppendEntries RPC 中（包括心跳 heartbeat 中），为的是让其他服务器都知道这条条目已经提交。一旦一个追随者知道了一个日志条目已经被提交，它会将该条目应用至本地的状态机（按照日志顺序）。

请求处理流程





注: 7. commit log 应该是在5. commit 之后，但是由于提交的timeout，会在7. success之后出现

可视化推导：http://thesecretlivesofdata.com/raft/

领导人选取

Raft 使用一种心跳机制（heartbeat）来触发领导人的选取。当服务器启动时，它们会初始化为追随者。一太服务器会一直保持追随者的状态只要它们能够收到来自领导人或者候选人的有效 RPC。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。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。

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。一个候选人会一直处于该状态，直到下列三种情形之一发生：

- 它赢得了选举；
- 另一台服务器赢得了选举；
- 一段时间后没有任何一台服务器赢得了选举

这些情形会在下面的章节中分别讨论。

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。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）。大多数原则使得在一个任期内最多有一个候选人能赢得选举。一旦有一个候选人赢得了选举，它就会成为领导人。然后它会像其他服务器发送心跳信息来建立自己的领导地位并且组织新的选举。

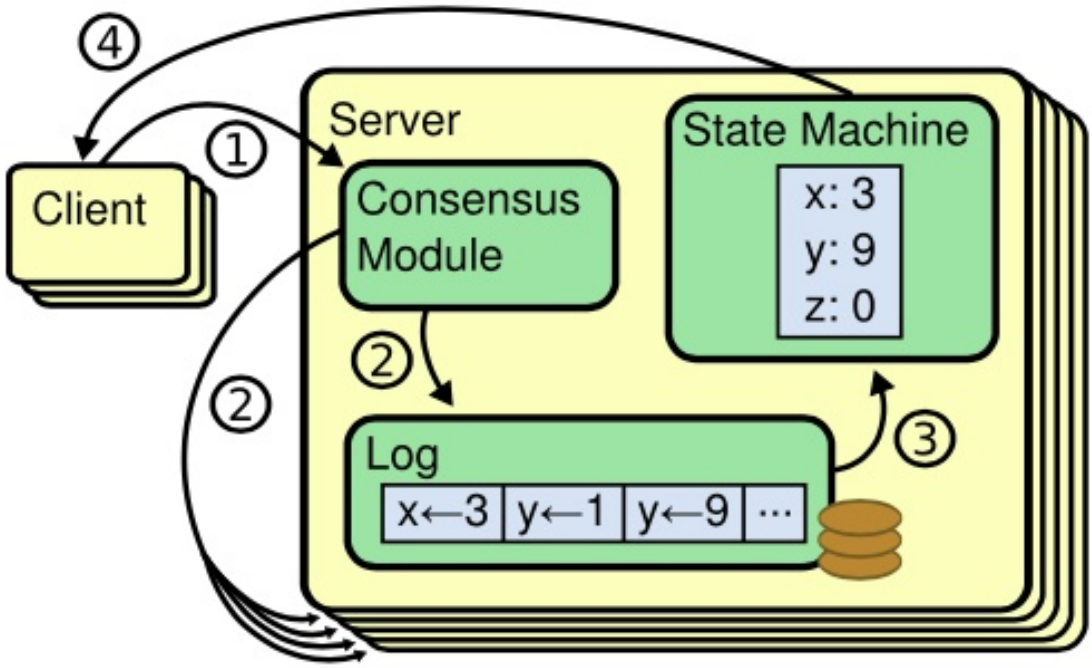
当一个候选人等待别人的选票时，它有可能会收到来自其他服务器发来的声明其为领导人的 AppendEntries RPC。如果这个领导人的任期（包含在它的 RPC 中）比当前候选人的当前任期要大，则候选人认为该领导人合法，并且转换自己的状态为追随者。如果在这个 RPC 中的任期小于候选人的当前任期，则候选人会拒绝此次 RPC， 继续保持候选人状态。

第三种情形是一个候选人既没有赢得选举也没有输掉选举：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。然而，如果没有其它手段来分配选票的话，这种情形可能会无限的重复下去。

Raft 使用随机的选举超时时间来确保第三种情形很少发生，并且能够快速解决。为了防止在一开始是选票就被瓜分，选举超时时间是在一个固定的间隔内随机选出来的（例如，150~300ms）。这种机制使得在大多数情况下只有一个服务器会率先超时，它会在其它服务器超时之前赢得选举并且向其它服务器发送心跳信息。同样的机制被用于选票一开始被瓜分的情况下。每一个候选人在开始一次选举的时候会重置一个随机的选举超时时间，在超时进行下一次选举之前一直等待。这能够减小在新的选举中一开始选票就被瓜分的可能性。9.3 节 展示了这种方法能够快速的选出一个领导人。

复制状态机

共识算法是在复制状态机的背景下提出来的。在这个方法中，在一组服务器的状态机产生同样的状态的副本因此即使有一些服务器崩溃了这组服务器也还能继续执行。复制状态机在分布式系统中被用于解决许多有关容错的问题。



一致性算法管理来自客户端状态命令的复制日志。状态机处理的日志中的命令的顺序都是一致的，因此会得到相同的执行结果。

如图上图所示，复制状态机是通过复制日志来实现的。每一台服务器保存着一份日志，日志中包含一系列的命令，状态机会按顺序执行这些命令。因为每一台计算机的状态机都是确定的，所以每个状态机的状态都是相同的，执行的命令是相同的，最后的执行结果也就是一样的了。

如何保证复制日志一致就是一致性算法的工作了。在一台服务器上，一致性模块接受客户端的命令并且把命令加入到它的日志中。它和其他服务器上的一致性模块进行通信来确保每一个日志最终包含相同序列的请求，即使有一些服务器宕机了。一旦这些命令被正确的复制了，每一个服务器的状态机都会按同样的顺序去执行它们，然后将结果返回给客户端。最终，这些服务器看起来就像一台可靠的状态机。

应用于实际系统的一致性算法一般有以下特性：

- 确保安全性（从来不会返回一个错误的结果），即使在所有的非拜占庭（Non-Byzantine）情况下，包括网络延迟、分区、丢包、冗余和乱序的情况下。
- 高可用性，只要集群中的大部分机器都能运行，可以互相通信并且可以和客户端通信，这个集群就可用。因此，一般来说，一个拥有 5 台机器的集群可以容忍其中的 2 台的失败（fail）。服务器停止工作了我们就认为它失败（fail）了，没准一会当它们拥有稳定的存储时就能从中恢复过来，重新加入到集群中。
- 不依赖时序保证一致性，时钟错误和极端情况下的消息延迟在最坏的情况下才会引起可用性问题。
- 通常情况下，一条命令能够尽可能快的在大多数节点对一轮远程调用作出相应时完成，一少部分慢的机器不会影响系统的整体性能。

参考文档

pincap：https://pingcap.com/blog-cn/linearizability-and-raft/#in-practice

论文：

中文：https://www.infoq.cn/article/raft-paper

英文：https://raft.github.io/raft.pdf

可视化算法：https://raft.github.io/

可视化推导：http://thesecretlivesofdata.com/raft/

mit 6.824 ： https://pdos.csail.mit.edu/6.824/schedule.html