分布式系统一致性的Raft算法

by jiangplus

为什么要用分布式系统?

- 任何一台机器的能力都是有限的
- 任何一台机器的寿命都是有限的

分布式系统的挑战

- 节点故障：任何节点都会故障，甚至导致数据的不可逆损失
- 通讯异常：物理网络和通讯硬件总是存在不可用的风险
- 三态：除了成功和失败，还有超时的状态，无法知道当前请求是否被成功处理
- 网络分区：

  由于网络异常状况的发生，导致部分节点之间网络延迟不断增大或断开

  导致网络中只有局部节点能够正常通讯，产生的网络分裂

- 一致性: 不同数据副本之间，在进行一系列操作后，仍然保持一致的状态
- 可用性: 用户对系统的请求，总能在一个有限的时间内返回结果
- 分区容错性: 系统即使遇到分区故障，依然能够提供满足有效的服务

为了协调不同机器状态的一致性，需要协调数据的同步，*使系统处于一致的状态*

CAP 定理

2000 年 7 月，UCB 的 Eric Brewer 教授在 ACM PODC 会议上提出了 CAP 猜想，被在两年后被 MIT 的教授形式化证明。

一个系统不可能同时满足一致性（Consistency），可用性（Availability）和分区容错性（Partition Tolerance），最大只能取其二。

Paxos 算法

Leslie Lamport 1990 年提出的基于消息传递并且具有分区容错特性的算法，即使在机器故障和网络异常下，依然能够达成系统的一致性。他把算法写出一个考古学的故事，声称发现了一个叫 Paxos 的古希腊小岛上“兼职议会”的运作机制，可以用在一致性协议，论文题目是《The Part-Time Parliament》。因为审稿人看不懂，所以论文被拒了。直到 1998 年才被正式接受。

Paxos 是一种 CP 算法，并被形式证明。

Raft

Raft 是 Paxos 的一个以可理解性为目标的变种。基于复制状态机（Replicated State Machine）。

复制状态机通过复制日志来实现：

- 日志：每台机器保存一份日志，日志来自于客户端的请求，包含一系列的命令
- 状态机：状态机会按顺序执行命令
- 一致性模型：分布式环境下，保证多机的日志是一致的，继而状态机也是一致的


feed

一致性算法作用于一致性模型，一般有以下特性：

- safety：在非拜占庭问题下（网络延时，网络分区，丢包，重复发包以及包乱序等），结果是正确的
- availability：在半数以上机器能正常工作时，则系统可用
- timing-independent：不依赖于时钟来保证日志一致性，错误的时钟以及极端的消息时延最多会造成可用性问题

Raft 节点的三种角色

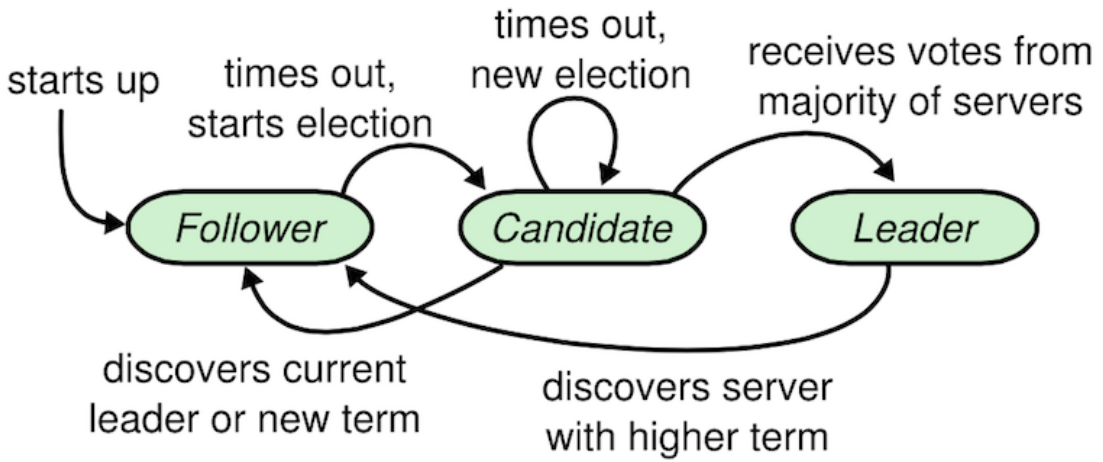
- 领导者
- 候选人
- 追随者

追随者只响应其他服务器的请求。

如果追随者没有收到任何消息，它会成为一个候选人并且开始一次选举。

收到大多数服务器投票的候选人会成为新的领导人。

领导人在它们宕机之前会一直保持领导人的状态。


starts up
times out,
starts election
times out,
new election
receives votes from
majority of servers
Follower
Candidate
Leader
discovers current
leader or new term
discovers server
with higher term

任期(Term)

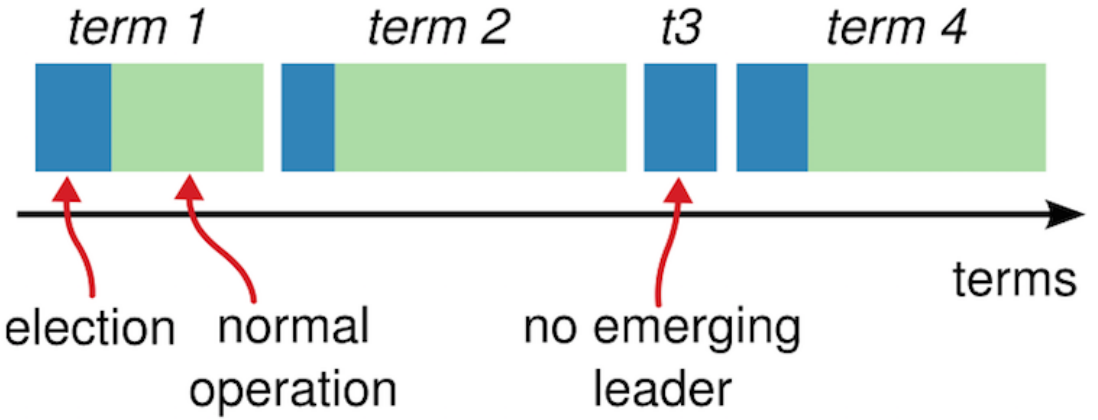
Raft 算法将时间划分成为任意不同长度的任期。用连续的数字进行表示。

每一个任期的开始都是一次选举（election），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多只有一个领导人。


term 1
term 2
t3
term 4
terms
election
normal operation
no emerging leader

RPC

Raft 算法中服务器节点之间通信使用远程过程调用（RPC）进行操作。

RPC有三种：

- RequestVote RPC: 候选人在选举期间发起
- AppendEntries RPC: 领导人发起的一种心跳机制，复制日志也在该命令中完成
- InstallSnapshot RPC: 领导者使用该RPC来发送快照给太落后的追随者。

超时

- BroadcastTime 领导者的心跳超时时间
- Election Timeout 追随者设置的候选超时时间
- MTBT 指的是单个服务器发生故障的间隔时间的平均数

BroadcastTime << ElectionTimeout << MTBF

领导人选举

触发条件:

一般情况下，追随者接到领导者的心跳时，把ElectionTimeout清零，不会触发；

领导者故障，追随者的ElectionTimeout超时发生时，会变成候选者，触发领导人选取；

领导人选举

参加选举过程：

追随者自增当前任期，转换为Candidate，对自己投票，并发起RequestVote RPC，
产生三种结果：

- 获得超过半数服务器的投票，赢得选举，成为领导者；
- 另一台服务器赢得选举，并接收到对应的心跳，成为追随者；
- 选举超时，没有任何一台服务器赢得选举，自增当前任期，重新发起选举；

领导人选举

选举的特性

- 服务器在一个任期内，最多能给一个候选人投票，采用先到先服务原则；
- 候选者等待投票时，可能会接收到来自其它声明为领导人的的AppendEntries RPC。

  如果该领导人的任期（RPC中有）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，并转换成追随者；否则拒绝

- 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，超时时间是随机的，避免不断重复这种情况

日志复制

接受命令的过程：

- 领导者接受客户端请求；
- 领导者把指令追加到日志；
- 发送AppendEntries RPC到追随者；
- 领导者收到大多数追随者的确认后，领导者Commit该日志，把日志在状态机中回放，并返回结果给客户端；

日志复制

提交过程：

- 在下一个心跳阶段，领导者再次发送AppendEntries RPC给追随者，日志已经commited；
- 追随者收到Commited日志后，将日志在状态机中回放。

日志压缩

日志会随着系统的不断运行会无限制的增长，这会给存储带来压力

几乎所有的分布式系统都采用快照的方式进行日志压缩，做完快照之后快照会在稳定持久存储中保存，而快照之前的日志和快照就可以丢弃掉。

安全性

- 领导者追加日志（Append-Only)

日志永远只有一个流向：从领导者到追随者； 领导者永远不会覆盖已经存在的日志条目，日志只能不断追加；

安全性

- 选举限制：投票阻止没有全部日志条目的服务器赢得选举

如果投票者的日志比候选人的新，拒绝投票请求；

这意味着要赢得选举，候选者的日志至少和大多数服务器的日志一样新，那么它一定包含全部的已经提交的日志条目。

安全性

- 永远不提交任期之前的日志条目（只提交任期内的日志条目）

在Raft算法中，当一个日志被安全的复制到绝大多数的机器上面，即 AppendEntries RPC 在绝大多数服务器正确返回了，

那么这个日志就是被提交了，然后领导者会更新commit index。