Raft 复习

jask

2024-08-26

共识算法

一般来说，要投入使用的共识算法要用有的特性：

在非拜占庭条件下保证共识的一致性。非拜占庭条件就是可信的网络条件，即与你通信的节点的信息都是真实的，不存在欺骗。

在多数节点存活时，保持可用性。“多数”永远指的是配置文件中所有节点的多数，而不是存活节点的多数。多数等同于超过半数的节点，多数这个概念概念很重要，贯穿 Raft 算法的多个步骤。

不依赖于绝对时间。理解这点要明白共识算法是要应对节点出现故障的情况，在这样的环境中网络报文也很可能会受到干扰从而延迟，如果完全依靠于绝对时间，会带来问题，Raft 用自定的 Term（任期）作为逻辑时钟来代替绝对时间。

在多数节点一致后就返回结果，而不会受到个别慢节点的影响。这点与第二点联合理解，只要“大多数节点同意该操作”就代表整个集群同意该操作。对于 raft 来说，”操作“是储存到日志 log 中，一个操作就是 log 中的一个 entry。

日志、log、entry:

日志 log: raft 保存的外部命令是以日志保存

entry: 日志有很多，可以看成一个连续的数组，而其中的的一个称为 entry

日志:

首先掌握日志的概念，Raft 算法可以让多个节点的上层状态机保持一致的关键是让各个节点的日志保持一致，日志中保存客户端发送来的命令，上层的状态机根据日志执行命令，那么日志一致，自然上层的状态机就是一致的。

所以 raft 的目的就是保证各个节点的日志是相同的。

任期:

Raft 将 Term 作为内部的逻辑时钟，使用 Term 的对比来比较日志、身份、心跳的新旧而不是用绝对时间。Term 与 Leader 的身份绑定，即某个节点是 Leader 更严谨一点的说法是集群某个 Term 的 Leader。Term 用连续的数字进行表示。Term 会在 follower 发起选举（成为 Candidate 从而试图成为 Leader ）的时候加 1，对于一次选举可能存在两种结果：

1. 胜利当选：胜利的条件是超过半数的节点认为当前 Candidate 有资格成为 Leader，即超过半数的节点给当前 Candidate 投了选票。

2. 失败：如果没有任何 Candidate（一个 Term 的 Leader 只有一位，但是如果多个节点同时发起选举，那么某个 Term

的 Candidate 可能有多位）获得超半数的选票，那么选举超时之后又会开始另一个 Term（Term 递增）的选举。

Raft 怎么保证一个 Term 只有一个 Leader?

因为 Candidate 变成 Leader 的条件是获得超过半数选票，一个节点在一个 Term 内只有一个选票（投给了一个节点就不能再投递给另一个节点），因此不可能有两个节点同时获得超过半数的选票。

发生故障时，一个节点无法知道当前最新的 Term 是多少，在故障恢复后，节点就可以通过其他节点发送过来的心跳中的 Term 信息查明一些过期信息。

当发现自己的 Term 小于其他节点的 Term 时，这意味着“自己已经过期”，不同身份的节点的处理方式有所不同：

leader、Candidate：退回follower并更新term到较大的那个Term

follower：更新Term信息到较大的那个Term

这里解释一下为什么自己的 Term 小于其他节点的 Term 时 leader、Candidate 会退回 follower 而不是延续身份，因为通过 Term 信息知道自己过期，意味着自己可能发生了网络隔离等故障，那么在此期间整个 Raft 集群可能已经有了新的 leader、提交了新的日志，此时自己的日志是有缺失的，如果不退回 follower，那么可能会导致整个集群的日志缺失，不符合安全性。

相反，如果发现自己的 Term 大于其他节点的 Term，那么就会忽略这个消息中携带的其他信息。

安全性：

Election Safety ：每个 term 最多只会有一个 leader；集群同时最多只会有一个可以读写的 leader。

很自然的问题是：节点之间通过网络通信，其他节点（follower）如何知道 leader 出现故障？follower 知道 leader 出现故障后如何选举出 leader？符合什么条件的节点可以成为 leader？

节点之间通过网络通信，其他节点（follower）如何知道 leader 出现故障？

leader 会定时向集群中剩下的节点（follower）发送 AppendEntry（作为心跳，hearbeat ）以通知自己仍然存活。

可以推知，如果 follower 在一段时间内没有接收 leader 发送的 AppendEntry，那么 follower 就会认为当前的 leader 出现故障，从而发起选举。

这里“follower 在一段时间内没有接收 leader 发送的 AppendEntry”，在实现上可以用一个定时器和一个标志位来实现，每到定时时间检查这期间有无 AppendEntry 即可。

AppendEntry 具体来说有两种主要的作用和一个附带的作用：

主要作用：

心跳

携带日志 entry 及其辅助信息，以控制日志的同步和日志向状态机提交

附带的作用：

通告 leader 的 index 和 term 等关键信息以便 follower 对比确认 follower 自己或者 leader 是否过期

follower 知道 leader 出现故障后如何选举出 leader？

参考【节点身份：follower、Candidate、Leader】一节中的描述，follower 认为 leader 故障后只能通过：term 增加，变成 candidate，向其他节点发起 RequestVoteRPC 申请其他 follower 的选票，过一段时间之后会发生如下情况：

赢得选举，马上成为leader （此时term已经增加了）

发现有符合要求的 leader，自己马上变成 follower 了，这个符合要求包括：leader 的 term≥ 自己的 term

一轮选举结束，无人变成leader，那么循环这个过程，即：term增加，变成candidate，。。。。。

赢得选举的条件前面也有过提及，即获得一半以上的选票。

如果在选举过程中没有“一半以上”选票的限制，会发生什么？会有什么问题？

如果在 Raft 协议中取消了对“一半以上选票”（即多数票）的限制，允许某个节点在没有获得多数票的情况下当选为领导者（Leader），这会导致以下严重问题：

脑裂（Split Brain）问题： 可能出现多个节点认为自己是领导者的情况，因为没有获得多数票的要求，不同节点可能在同一任期（term）内自认为是领导者。这会导致集群出现多个领导者，破坏了系统的一致性。

数据不一致： 如果多个领导者同时进行日志复制操作，集群中的不同节点会接收到不同的日志条目，从而导致数据不一致。在这种情况下，无法保证线性一致性（linearizability），集群可能返回错误或冲突的数据。

无法保证安全性： Raft 的安全性依赖于多数节点对领导者的认可。只有在多数节点同意的情况下，日志条目才能被认为是已提交的。如果取消了多数票的限制，领导者可能会在无法保证安全性的情况下提交日志，导致不安全的状态更新。

因此，获得“一半以上选票”的限制是 Raft 协议中用来确保集群一致性和安全性的重要机制。

Raft 节点的数量要求是奇数，为什么有这个要求？

Raft 节点数量要求是奇数，主要是为了最大化集群的容错性和可用性。

多数投票机制： Raft 需要多数节点投票（超过一半）来选举领导者或者提交日志条目。例如，在一个 5 个节点的集群中，至少需要 3 个节点同意才能做出决策。如果节点数量是奇数，选举过程中将不会出现平票的情况，从而减少了无法选出领导者的可能性。

最大化容错性： 奇数节点可以确保系统在失去少数节点后仍然能够正常运行。例如，在 5 个节点的集群中，即使有 2 个节点宕机，仍然可以通过剩下的 3 个节点形成多数票并继续运作。如果节点数量是偶数，损失 1 个节点时，剩下的节点数量可能会导致平票，从而导致无法选出领导者，系统停止运作。

如果发现一个领导者（Leader），但其 term 小于自己，会发生什么？

如果一个 Raft 节点发现当前的领导者的 term（任期号）小于自己的 term，这意味着该领导者是过时的。在这种情况下，会发生以下几件事情：

拒绝领导者的指令： 节点将拒绝任何来自这个旧领导者的请求（如日志复制请求或心跳）。节点会认为这个领导者不再有效，因为它的任期号已经落后于自己的任期号。

触发重新选举： 发现领导者的任期落后后，该节点会向集群中的其他节点发出投票请求，试图通过新的选举成为领导者。由于它的 term 更新过，因此更有可能成为新的领导者。

降级领导者： 旧的领导者会因为其他节点拒绝其请求而意识到自己的任期号落后，从而自动降级为跟随者（Follower），并更新自己的 term 以与其他节点保持一致。

这种机制可以防止旧的领导者继续控制集群，从而确保 Raft 协议的安全性和一致性

符合什么条件的节点可以成为 leader？

这一点也成为“选举限制”，有限制的目的是为了保证选举出的 leader 一定包含了整个集群中目前已 committed 的所有日志。

当 candidate 发送 RequestVoteRPC 时，会带上最后一个 entry 的信息。所有的节点收到该请求后，都会比对自己的日志，如果发现自己的日志更新一些，则会拒绝投票给该 candidate，即自己的日志必须要“不旧于”该 candidate。

判断日志老旧的方法 raft 论文中用了一段来说明，这里说一下如何判断日志的老旧：

需要比较两个东西：最新日志 entry 的 term 和对应的 index。index 即日志 entry 在整个日志的索引。

if 两个节点最新日志 entry 的 term 不同 term 大的日志更新 else 最新日志 entry 的 index 大的更新 end

这样的限制可以保证：成为 leader 的节点，其日志已经是多数节点中最完备的，即包含了整个集群的所有 committed entries。