Basic Paxos

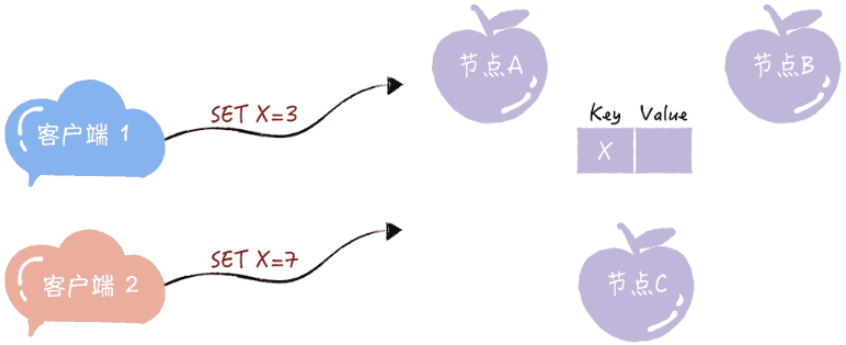
jask

2024-08-11

Paxos 算法

分布式共识算法的代表。

兰伯特提出的 Paxos 算法有两部分：
1.Basic Paxos 算法，描述的是多节点之间如何就某个值达成共识。
2.Multi Paxos 算法，描述的是执行多个 Basic Paxos 实例，就一系列值达成共识。



思考题：假设我们要实现一个分布式集群，这个集群是由节点 A、B、C 组成，提供只读 KV 存储服务。你应该知道，创建只读变量的时候，必须要对它进行赋值，而且这个值后续没办法修改。因此一个节点创建只读变量后就不能再修改它了，所以所有节点必须要先对只读变量的值达成共识，然后所有节点再一起创建这个只读变量。那么，当有多个客户端（比如客户端 1、2）访问这个系统，试图创建同一个只读变量（比如 X），客户端 1 试图创建值为 3 的 X，客户端 2 试图创建值为 7 的 X，这样要如何达成共识，实现各节点上 X 值的一致呢？

Basic Paxos 算法中的角色：提议者(Proposer)，接受者(Acceptor)，学习者(Learner)。

提议者（Proposer）：提议一个值，用于投票表决。为了方便演示，你可以把图 1 中的客户端 1 和 2 看作是提议者。但在绝大多数场景中，集群中收到客户端请求的节点，才是提议者（图 1 这个架构，是为了方便演示算法原理）。这样做的好处是，对业务代码没有入侵性，也就是说，我们不需要在业务代码中实现算法逻辑，就可以像使用数据库一样访问后端的数据。

接受者（Acceptor）：对每个提议的值进行投票，并存储接受的值，比如 A、B、C 三个节点。一般来说，集群中的所有节点都在扮演接受者的角色，参与共识协商，并接受和存储数据。

学习者（Learner）：被告知投票的结果，接受达成共识的值，存储保存，不参与投票的过程。一般来说，学习者是数据备份节点，比如“Master-Slave”模型中的 Slave，被动地接受数据，容灾备份。

前面不是说接收客户端请求的节点是提议者吗？这里怎么又是接受者呢？这是因为一个节点（或进程）可以身兼多个角色。想象一下，一个 3 节点的集群，1 个节点收到了请求，那么该节点将作为提议者发起二阶段提交，然后这个节点和另外 2 个节点一起作为接受者进行共识协商。

三种角色本质上是三种功能

提议者代表的是接入和协调功能，收到客户端请求后，发起二阶段提交，进行共识协商；

接受者代表投票协商和存储数据，对提议的值进行投票，并接受达成共识的值，存储保存；

学习者代表存储数据，不参与共识协商，只接受达成共识的值，存储保存。

如何达成共识

Basic Paxos 中，兰伯特使用提案代表一个提议，提案中除了提案编号还包含提议值。在此化简，只包含提案编号。

整个共识协商是分两个阶段进行的。

我们假设客户端 1 的提案编号为 1，客户端 2 的提案编号为 5，并假设节点 A、B 先收到来自客户端 1 的准备请求，节点 C 先收到来自客户端 2 的准备请求。### 准备阶段客户端 1,2 作为提议者，分别向所有接受者发送包含提案编号的准备请求：

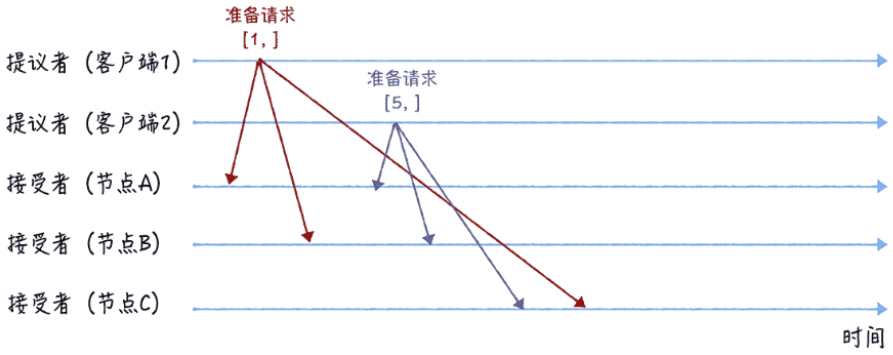


Figure 1: 准备请求

注意，在准备请求中不需要指定提议的值。

当节点 A、B 收到提案编号为 1 的准备请求，节点 C 收到提案编号为 5 的准备请求后：

由于之前没有通过任何提案，所以节点 A、B 将返回一个“尚无提案”的响应。也就是说节点 A 和 B 在告诉提议者，我之前没有通过任何提案呢，并承诺以后不再响应提案编号小于等于 1 的准备请求，不会通过编号小于 1 的提案。节点 C 也是如此，它将返回一个“尚无提案”的响应，并承诺以后不再响应提案编号小于等于 5 的准备请求，不会通过编号小于 5 的提案。

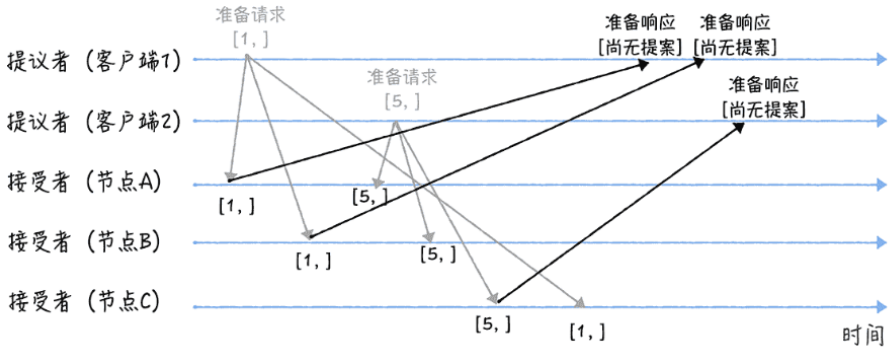
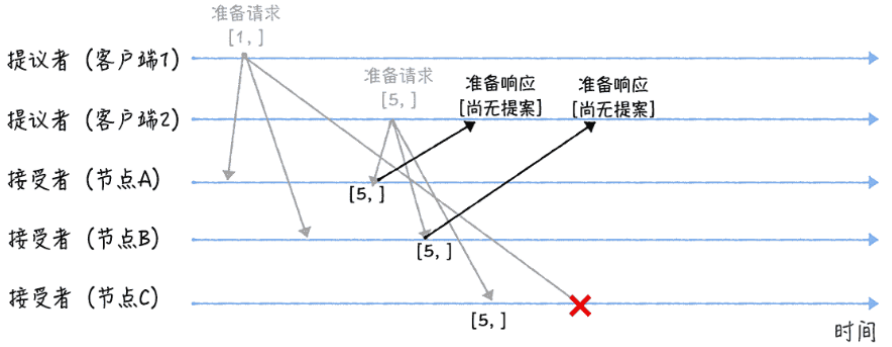


Figure 2: 准备响应

另外，当节点 A、B 收到提案编号为 5 的准备请求，和节点 C 收到提案编号为 1 的准备请求的时候，将进行这样的处理过程：



当节点 A、B 收到提案编号为 5 的准备请求的时候，因为提案编号 5 大于它们之前响应的准备请求的提案编号 1，而且两个节点都没有通过任何提案，所以它将返回一个“尚无提案”的响应，并承诺以后不再响应提案编号小于等于 5 的准备请求，不会通过编号小于 5 的提案。当节点 C 收到提案编号为 1 的准备请求的时候，由于提案编号 1 小于它之前响应的准备请求的提案编号 5，所以丢弃该准备请求，不做响应。

接受阶段

首先客户端 1,2 在收到大多数节点的准备响应之后，会分别发送接受请求。

当客户端 1 收到大多数的接受者（节点 A、B）的准备响应后，根据响应中提案编号最大的提案的值，设置接受请求中的值。因为该值在来自节点 A、B 的准备响应中都为空（也就是图 5 中的“尚无提案”），所以就把自己的提议值 3 作为提案的值，发送接受请求 [1, 3]。当客户端 2 收到大多数的接受者的准备响应后（节点 A、B 和节点 C），根据响应中提案编号最大的提案的值，来设置接受请求中的值。因为该值在来自节点 A、B、C 的准备响应中都为空（也就是图 5 和图 6 中的“尚无提案”），所以就把自己的提议值 7 作为提案的值，发送接受请求 [5, 7]。

当三个节点收到两个客户端的接受请求时，会这样处理：

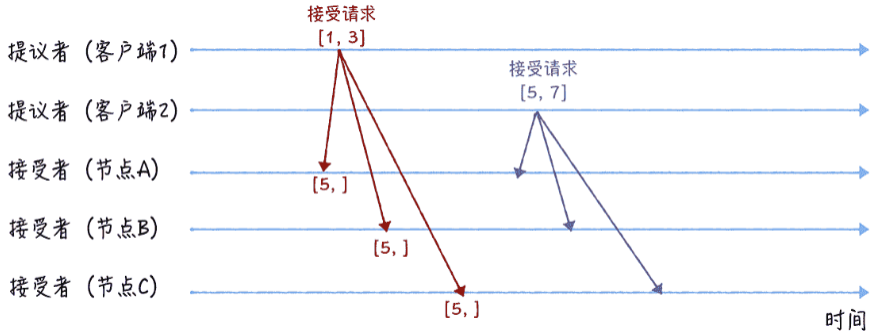
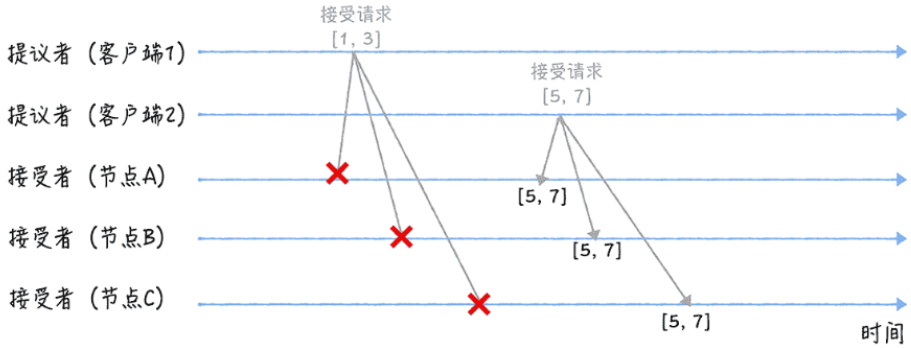


Figure 3： 接受阶段



当节点 A、B、C 收到接受请求 [1, 3] 的时候，由于提案的提案编号 1 小于三个节点承诺能通过的提案的最小提案编号 5，所以提案 [1, 3] 将被拒绝。当节点 A、B、C 收到接受请求 [5, 7] 的时候，由于提案的提案编号 5 不小于三个节点承诺能通过的提案的最小提案编号 5，所以就通过提案 [5, 7]，也就是接受了值 7，三个节点就 X 值为 7 达成了共识。

如果集群中有学习者，当接受者通过了一个提案时，就通知给所有的学习者。当学习者发现大多数的接受者都通过了某个提案，那么它也通过该提案，接受该提案的值。

小结

Basic Paxos 是通过二阶段提交的方式来达成共识的。二阶段提交是达成共识的常用方式，如果你需要设计新的共识算法的时候，也可以考虑这个方式。

除了共识，Basic Paxos 还实现了容错，在少于一半的节点出现故障时，集群也能工作。它不像分布式事务算法那样，必须要所有节点都同意后才提交操作，因为“所有节点都同意”这个原则，在出现节点故障的时候会导致整个集群不可用。也就是说，“大多数节点都同意”的原则，赋予了 Basic Paxos 容错的能力，让它能够容忍少于一半的节点的故障。

本质上而言，提案编号的大小代表着优先级，你可以这么理解，根据提案编号的大小，接受者保证三个承诺，具体来说：如果准备请求的提案编号，小于等于接受者已经响应的准备请求的提案编号，那么接受者将承诺不响应这个准备请求；如果接受请求中的提案的提案编号，小于接受者已经响应

的准备请求的提案编号，那么接受者将承诺不通过这个提案；如果接受者之前有通过提案，那么接受者将承诺，会在准备请求的响应中，包含已经通过的最大编号的提案信息。