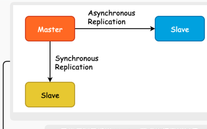
Basic Paxos

前言

Paxos 算法是一种基于消息传递并且在分布式系统中具有高度容错的共识（consensus）算法，不能将其认为是 “一致性（consistency）算法”，“一致性” 和 “共识” 并不是同一个概念

一致性和共识

一致性



在一般的主从复制模型中，可以分为异步复制、同步复制以及半同步复制

复制

1. 异步复制是指当 Master 节点执行完毕以后，将数据通过另外一个线程/进程发送给 Slave 节点。因此 Master 节点的数据和 Slave 节点的数据并不是实时保持一致的，通常 Slave 会落后 Master 节点。因此，异步复制能够保证的是弱一致性，或者说最终一致性
2. 同步复制是指当 Master 节点在执行实际的数据操作之前，先将其发送给所有的 Slave 节点，只有所有的 Slave 节点全部响应成功后 Master 才可以继续执行。也就是说，当我们写一条数据时，实际上是同步地向所有节点中写入。因此，同步复制能够保证数据的强一致性，只要写入成功，那么任何时刻在任意节点所查询到的数据都是一样的
3. 半同步复制则是异步复制和同步复制的组合体，半同步复制要求至少成功地向一个 Slave 节点同步写入，剩余节点进行异步写入。这么做的好处就在于集群中至少存在一个 Slave 节点和 Master 节点保持强一致性，当现有 Master 节点宕机后，该 Slave 节点可进行无损地故障转移（Failover）

所以，一致性问题是指，对于同一个数据的多个副本之间，如何保持其对外表现的数据一致性

共识

共识不同于一致性，并没有 “强共识” 或者是 “弱共识” 一说，只要集群中的节点达成共识，那么它们的行为就会保持一致

仍然以复制为例，假设集群中存在一个节点的异步复制，两个节点的同步复制

当 Master 节点宕机以后，集群需要决策出新的 Master 节点，此时 Slave-1 和 Slave-2 由于是同步节点，所以最有可能成为主节点。如果 Slave-1 和 Slave-2 就选主这一问题没有达成共识，都认为自己将会是下一届的 Master 节点的话，那么就出现集群中出现 2 个 Master 节点的情况，即脑裂发生

如果 Slave-1 和 Slave-2 就选主这一问题达成了共识，不管最终的结果是 Slave-1 成为新的 Master 节点，还是 Slave-2 成为了新的 Master 节点，Slave-1 和 Slave-2 将会坚定不移地按照达成共识的结果进行执行

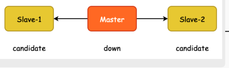
Paxos 和 Raft 协议能够正确的选主原因就在这里，因为它们是共识算法，而不是一致性算法

所以，共识问题是指，多个节点共同协商出某一个问题的结果以后，使得所有节点对这个结果达成一致

最后解释一下高度容错，这里的容错并不包含拜占庭容错，也就是集群中的节点不会对消息进行篡改，消息只会出现高延迟、丢失以及重复问题，也就是说，Paxos 算法是非拜占庭容错算法

问题的提出

paxos-simple 论文中使用 State Machine 来描述集群对某一个值达成共识，个人感觉会有些晦涩，因此这里我使用选主这一个更具象的话题来描述 Basic Paxos 算法的原理和过程

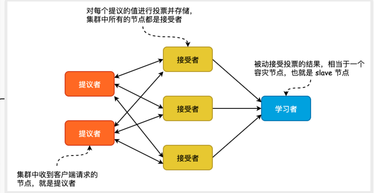


我们假设 Slave-1 的节点编号为 108，Slave-2 的节点编号为 208

假设集群中节点复制状态如上图所示，在某一时刻，Master 节点发生宕机。集群需要在 Slave-1 和 Slave-2 中重新选择出一个 Master 节点。那么，Slave-1 和 Slave-2 要如何才能达成共识，最终只有一个新的 Master 节点呢？

Paxos 算法中的角色

正如同主从复制模型中有 Master 节点和 Slave 节点一样，Paxos 算法同样为不同功能的节点定义了不同的角色，它们分别是提议者（proposer）、接受者（acceptor）和学习者（learner），它们之间的关系如下图所示



需要注意的是，每一个节点可以身兼数职，也就是一个节点既可以是提议者，也可以是接受者

对角色的说明

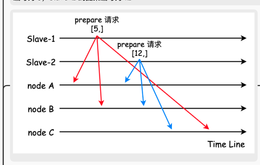
- 提议者 —— 如上图所示，接收客户端请求的那个节点其实就是提议者，提议者用于向集群中其它节点提供一个值，用于投票表决。提议者可以认为是 “入口点”，流程从这里开始发起
- 接受者 —— 对每个提议的值进行投票，并存储接受的值。集群中所有的节点都是接受者，包括提议者
- 学习者 —— 用于存储最终达成共识的结果，并不参与投票过程，可以大致地认为学习者是整个集群中的一个容灾备份节点

达成共识的过程

Paxos 协商算法是分两阶段进行的，在协商过程中有两个非常重要的因子，一个是唯一的提案编号，我们使用 number 来表示；另一个则是此次的提议值，我们使用 value 来表示，那么 (number, value) 表示在编号为 number 的提案中，提议的值为 value

准备（prepare）阶段

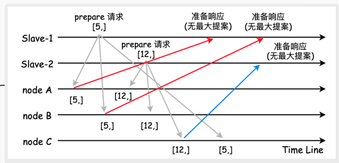
首先，Slave-1 和 Slave-2 作为客户端，向其它接受者发送提案编号和提议值。假设 Slave-1 的提案编号为 5，Slave-2 的提案编号为 12



prepare 阶段并不需要携带提议值，只需要携带提案编号即可

当节点 A、B、C 收到 Slave-1 和 Slave-2 的 prepare 请求以后，会根据提案编号做出不同的处理，如下方过程所示

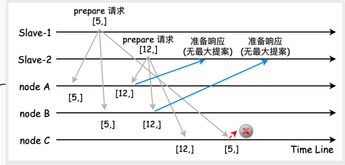
节点收到的第一次 prepare 请求



当节点 A 和节点 B 收到 Slave-1 提案编号为 5 的 prepare 请求时，由于之前没有任何的提案提交，所以返回 “无最大提案” 的响应。并且承诺接下来不会响应任何提案编号小于等于 5 的 prepare 请求，同时也不会提交编号小于等于 5 的提案

当节点 C 收到 Slave-2 提案编号为 12 的 prepare 请求时，由于之前没有任何的提案提交，所以返回 “无最大提案” 的响应。并且承诺接下来不会响应任何提案编号小于等于 12 的 prepare 请求，同时也不会提交编号小于等于 12 的提案

节点收到的第二次 prepare 请求



由于节点 A 和节点 B 尚无提案，并且 Slave-2 的提案编号 12 大于此前的提案编号 5，所以节点 A 和节点 C 能够正常响应 Slave-2

但是，当节点 C 收到 Slave-1 编号为 5 的提案编号时，由于节点 C 已经向 Slave-2 做出了承诺，不再响应提案编号小于等于 12 的 prepare 请求，因此无正常响应

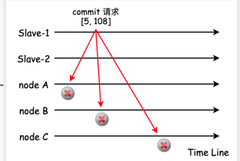
提交（commit）阶段

我们可以看到，Slave-1 和 Slave-2 在经过了 prepare 请求阶段以后，均得到了 2 个正常响应，也就是得到集群中超过半数节点的正确响应。因此，Slave-1 和 Slave-2 均可进行提交动作

我们先来看 Slave-1 所执行的提交动作

Slave-1 在收到多数接受者的正确 prepare 请求响应以后，会根据响应中的最大提案编号来设置提交请求中的提交提案值

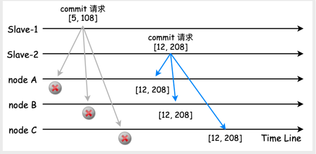
这是什么意思呢？在我们这个例子中因为此前未有提交的提案，所以 Slave-1 可以直接使用 prepare 请求中的提案编号，也就是 5 来作为提交阶段的提案编号。如果此前有提交的提案，那么 prepare 请求时将会响应最大的提案编号，假设为 10，那么此时 Slave-1 在提交时就必须使用大于 10 的编号来作为提交阶段的提案编号



没错，节点 A、B、C 这 3 个接受者均会拒绝 Slave-1 的提交请求，因为它们已经向 Slave-2 做出了承诺，不接收提案编号小于等于 12 的提交

再来看 Slave-2 所执行的提交动作

Slave-2 在收到多数接受者的正确 prepare 请求响应以后，会根据响应中的最大提案编号来设置提交请求中的提交提案值



当节点 A、B、C 这 3 个接受者接收到 Slave-2 发送的 [12, 208] 的提交请求时，满足之前的承诺，所以均会通过这个提案，也就是接受了值 208。3 个节点就 Master 节点为 208 达成一致

如果此时集群中有学习者的话，当学习者的发现大多数的接受者都通过了某一个提案，那么它就会将这个提案持久化，保存下来

最终，节点 A、B、C 就 Master 节点为 208，也就是我们最开始所说的 Slave-2 为 Master 节点达成了共识，此时就可以执行切主操作了

实际上，真正的 Paxos 选主过程并不是我们上面所描述的那样，其过程要更为复杂一些，但是，使用选主这一个例子能够让我们更好的理解 Paxos 共识算法的运转流程，要比干巴巴的设置一个值更贴切一些

Basic Paxos 算法最为关键的地方就在于两阶段提交以及做出的承诺，这是维持共识的根本，并且，Basic Paxos 只会批准一个 value，也就是说，当 value 被提交以后，它将变为只读，假设 Slave-1 又发起了一个 [18, 108] 的 prepare 请求以及 commit 请求，那么最终节点 A、B、C 的值为 [18, 208]，而不是 [18, 108]

总结

1. 两阶段提交 —— Basic Paxos 是基于两阶段提交所实现的，两阶段提交也是分布式系统中达成共识的常见方式，例如 MySQL 的分布式事务实现
2. 容错性保证 —— 只要集群中大多数节点能够正常工作，那么 Basic Paxos 算法就可以正常运行，具有一半的节点容错能力
3. 承诺的保证
   - 若 prepare 请求的提案编号小于等于接受者已经响应准备请求的提案编号，那么接受者将不会正常响应这个准备请求
   - 若接受者此前有提案通过，那么承诺将已经通过的最大提案编号和信息（值）返回给准备请求的响应中
   - 若 commit 请求的提案编号小于等于接受者已经响应准备请求的提案编号，那么接受者将承诺不会通过这个提案