

Paxos 算法的研究及改进

刘克礼¹, 张文盛²

(1. 安徽开放大学 信息与建筑工程学院, 安徽 合肥 230022;
2. 安徽开放大学 信息技术与网络管理中心, 安徽 合肥 230022)

摘要:为进一步提高 Paxos 算法的执行效率和可用性,采用数学集合的方法,从锁的角度分析算法,深入研究值的确定和持久化问题,引入不变性保证和提案锁模型,通过将两者有机结合,形成简洁和有效的分布式一致性算法。

关键词:Paxos 算法;多数决;不变性;提案锁

中图分类号:TP302.7

文献标识码:A

文章编号:2097-0625(2024)02-0084-08

一、引言

一致性是分布式计算领域的一个重要研究方向,一致性要解决的是如何在多个计算节点中保持数据的一致性,少数节点宕机不影响整体稳定性,从而提高系统的可用性^[1]。这个看似简单的需求,却很难解决。经过不懈的努力,已有很多理论和算法或协议,如 CAP 理论^[2]、2 阶段提交(2PC)、Paxos 算法^[3]、ZAB 协议、VR 协议^[4]、Raft 协议等。相关的实现和应用也不断涌现,例如 ZooKeeper 使用 ZAB^[5],Etd使用 Raft^[6]等,其中 Paxos 算法最为经典。

Paxos 算法包括两个阶段(Preprare 和 Propose),每个阶段往返交互 1 次(Round Trip, RT),总共 4 句话^[7]。虽然算法表面很简洁,其实背后蕴藏着深刻的内涵,不深挖就难以理解 Paxos 算法为什么是正确的。如 Raft 协议的作者就认为 Paxos 难以理解,才提出了 Raft 协议。对 Paxos 正确性的研究层出不穷,有各种形式的推导和证明^[8-9]。由于算法被认为是难以理解,所以将其实现并证明实现和算法是一致的,变得更加困难。本文在介绍算法原理基础上进一步研究算法的实现机制。

二、一致性算法概述

一致性算法典型的应用是在一个分布式数据库

系统中,若各节点的初始状态一致,每个节点执行相同的操作序列,最后均得到一致的状态。为保证每个节点执行相同的命令序列,需要在每一条指令上执行一致性算法以保证每个节点执行的指令一致。原理如图 1 所示。

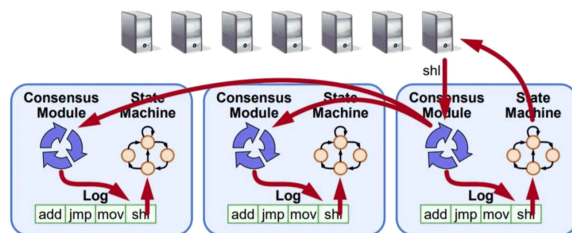


图 1 副本状态机

引入状态机(State Machine),对多个初始状态相同的状态机执行相同操作序列,能得到多个相同的状态机^[10]。相同操作序列通过分布式一致性算法(Consensus Algorithm)确定,并写入各个节点的本地磁盘,称为日志(Log)。在状态机崩溃后,通过重放日志,可以重建状态机。多个节点间能保持数据一致性的节点称为副本(Replica),这种机制也称为副本状态机(Replica State Machine)。在副本之间保证数据的一致性分布系统的一个核心问题,这个问题归根为如何确定一个日志项(Log Entry),且日志项

收稿日期:2024-03-21

基金项目:安徽省高等学校质量工程项目“网络团队合作,在线任务驱动”课程教学模式探索与实践——以《数据库运维》为例”(项目编号:2022jxgl016)

作者简介:刘克礼(1976—),男,安徽东至人,副教授,硕士。研究方向:数据挖掘。

不可更改。日志项主要存储值,因此分布式一致性算法解决的是如何就某个日志项(值)达成一致。

节点通信存在两种模型:共享内存(Shared Memory)和消息传递(Messages Passing)^[11]。分布式一致性算法通常基于消息传递,消息传递的最大特点是异步性,即消息会丢失、延迟、重复和乱序。节点通信的异步性是导致分布式一致性问题难以解决的主要原因。

一致性算法中的参与者在承担的功能上并不对称,通常被分成多个角色。不同的一致性算法在角色种类和命名上有所差别,本文采用 Paxos 算法中的分类和命名,相关角色如图 2 所示。

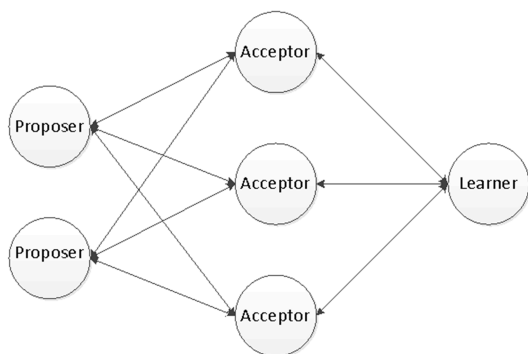


图 2 Paxos 算法中的角色

Paxos 中传递的主要消息称为提案(Proposal),提案包括编号和值。Proposer 称为提案发起者,Acceptor 是提案接受者,Learner 是学习者。Proposer 向所有 Acceptor 发送提案,要求他们接受提案。提案一旦被多数 Acceptor 接受,则认为提案中的值被选定。Learner 不参与值的选定过程,只从 Acceptor 中学习这个选定的值。

三、Paxos 算法的执行过程

Paxos 算法分为两个阶段,具体如下:

1. 阶段一(Prepare)

(1)Proposer 选择一个提案编号 N ,然后向半数以上的 Acceptor 发送编号为 N 的 Prepare 请求;

(2)如果 Acceptor 收到一个编号为 N 的 Prepare 请求,且 N 大于该 Acceptor 已响应过的所有 Prepare 请求的编号,Acceptor 将已经接收过的编号最大的提案作为响应反馈给 Proposer,同时该 Acceptor 承诺不再接受任何编号小于 N 的提案。

2. 阶段二(Accept)

(1)如果 Proposer 收到半数以上 Acceptor 对其

发出的编号为 N 的 Prepare 请求响应,那么它就会发送一个针对 $[N, V]$ 提案的 Accept 请求给半数以上的 Acceptor。 V 为收到的响应中编号最大提案的 Value,如果响应中不包含任何提案,那么 V 就由 Proposer 自己决定;

(2)如果 Acceptor 收到一个针对编号为 N 提案的 Accept 请求,只要该 Acceptor 没有对编号大于 N 的 Prepare 请求做出过响应,它就接受该提案。

Paxos 算法类型较多,原始的 Paxos 算法一般称为 Basic Paxos,执行过程如图 3 所示,算法的伪代码描述如图 4 所示。

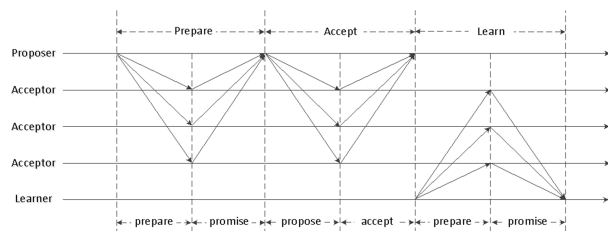


图 3 Paxos 算法执行流程

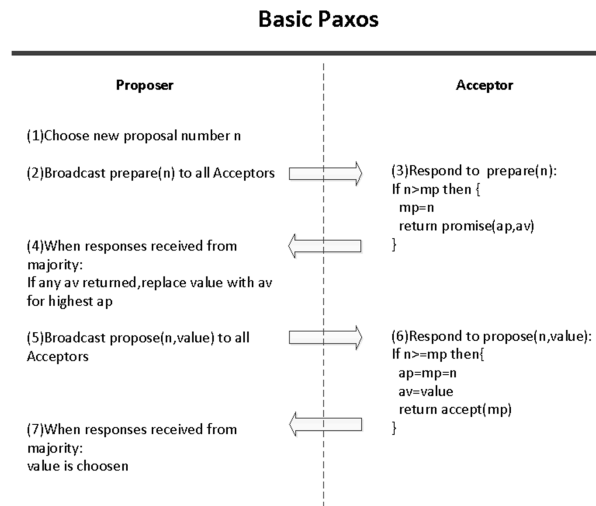


图 4 Paxos 的伪代码描述

四、算法分析及改进

(一)法定人数

单个 Acceptor 可以确定一个值,可能会存在单点故障,可用性低。多个 Acceptor 确定一个值之后,即使只剩一个节点而其他节点都宕机,也能提供服务,可用性较高。CAP 理论指出分区无法避免,当发生分区时,多个分区如何继续提供服务? 如果不加以约束,都能提供服务,那么分区之间数据的一致性无法保证,从而出现脑裂(Split Brain)现象,即存在不确定性。

把 Acceptor 组成的全体看作集合,给出如下推论:

推论 1:设有 n 个 Acceptor,分别编号为 A_1, A_2, \dots, A_n ,他们组成集合 $A, A_i \in A$;

推论 2:超过半数的临界值称为法定人数(Quorum),记为 $q, q = n$ 整除 2 后再加 1;

推论 3:记 A 所有的子集组成集合 B ;

推论 4:记 $Q = \{b \in B \mid |b| \geq q\}$, Q 是法定人数及以上节点组成的集合,也称多数派;

推论 5:集合 A 的划分中,只有一个多数派,记为 $C, C \in Q$;

推论 6:服务分区和集合划分等价;

推论 7: $x \in Q, y \in Q, z = x \cap y \Rightarrow |z| > 0$,即多数派必相交;

推论 8:如果 A 中存在一个多数派,其值都相同,称为值确定,该值记为 V ;

推论 9:当值确定后,任何多数派必包含 V ,这样的多数派记为 C_v ;

推论 10:在 C_v 中选择一个值,并且保证选择的都是 V ,这种保证称为不变性保证,记为 G 。

举例计算 q ,结果如表 1 所示。

表 1 法定人数举例

n	q	容忍故障节点数($n-q$)
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3
8	5	3

根据表 1,节点个数 n 为偶数时,节点数比“ $n-1$ ”的系统多 1 个,但是容忍故障节点数并没有增加。可见 n 为奇数时,系统可用性比“ $n+1$ ”更优。构建系统时,通常 n 选奇数。“ $2F+1$ ”节点可容忍 F 个节点故障。

根据推论 5 和推论 6,解决脑裂的方法是允许节点数过半的分区 C 继续提供服务,其他分区将挂起,等待人工干预或是在网络恢复并完成数据同步后,重

新加入 C 。

(二)提案

Paxos 中不同的 Proposer 会发起不同的提案,相同的 Proposer 每执行一次协议也会发起不同的提案。提案由编号和值组成,记为 $[p, v]$,其中 p 是提案编号且唯一, v 是提案值。在 Proposer 个数 n 确定的情况下,可以采用“ $n * r + i$ ”的方式生成 p ,其中 r 是执行协议的轮数, i 是 Proposer 的序号,这样生成的 p 单增且不相同。

(三)两阶段

Paxos 通过两个阶段确定值,即 Prepare 准备阶段和 Accept 提交阶段。

Prepare 阶段是查询和选定值的阶段,Proposer 查询所有 Acceptor 当前保存的值,从反馈的结果中选择要提交的值。Prepare 阶段要发送 Prepare(p)和 Promise($mp, [ap, av]$)两个消息。Acceptor 记录最大提案编号 mp 和接受的提案 $[ap, av]$ 。

不是所有的 Acceptor 都会返回查询结果,有三种情况:

(1)如果 $p < mp$,说明提案过期,拒绝返回查询结果;

(2)如果 $P = mp$,说明提案重复,已经 Promise,为了防止消息重复,不做回复;

(3)如果 $p > mp$,提案有效,更新 $mp = p$,返回 Promise($mp, [ap, av]$)消息。

由于异步通信的特点,在 Promise 返回的 mp 和 p 相等的情况下,记发送 Promise 消息的 Acceptor 组成的集合为 A_{p1} ,被 Proposer 收到 Promise 消息的 Acceptor 组成的集合为 A_{p2} 。如果 $|A_{p1}| \geq q$,记为 C_{p1} ;如果 $|A_{p2}| \geq q$,记为 C_{p2} 。显然 $A_{p2} \subseteq A_{p1}, C_{p2} \subseteq C_{p1}$ 。

如果在 C_{p2} 中返回的值都为空,说明值还未选定,此时,Proposer 可以自行选择一个值并在 Accept 阶段提交。如果 C_{p2} 中有返回值,那么就选择其中 p 最大的那个提案,重新提交。

Paxos 在 Prepare 中利用了推论 10 保证确定的值确定性。无论是因为发生分区或节点性能不均匀或消息延迟,只要有超过 50% 的节点 A_{p2} 有反馈,那么 C_{p2} 中要么包含 V ,要么 V 还未选定。如果 $V \in C_{p2}$,根据推论 10,不需要等待更多的节点反馈,即可继续往下执行,因此推论 10 加快了协议的执行速度。

如果 V 未选定, 那么可在 C_{p2} 中任选一值, 为了和 V 的不变性保持一致, 也选择最大 ap 的提案值。

Accept 阶段是提交和接收提案的过程, 包括 Proposer(p, v) 和 Accept(mp) 两个消息。为了不产生混淆, 约定: Accept 指代阶段, Accept 指代消息。Acceptor 比较 p 和 mp , 有三种情况:

(1) 如果 $p < mp$, 说明有更大的提案被 Promise, 本次 Propose 请求无效, 拒绝;

(2) 如果 $p = mp$, 说明 Promise 有效, 接收该提案, 更新 $ap = p, av = v$, 返回 Accept;

(3) 如果 $p > mp$, 因为 p 被多数 Acceptor 发起 Promise, 而本节点由于某种原因未收到对应的 Prepare, 但是不妨碍本节点接收该提案, 更新 $mp = p$,

$Ap = p, av = v$, 返回 Accept。

由于异步通信的特点, 在 Accept 返回的 mp 和 p 相等的情况下, 记发送 Accept 消息的 Acceptor 组成的集合为 A_{a1} , 被 Proposer 收到 Accept 消息的 Acceptor 组成的集合为 A_{a2} 。如果 $|A_{a1}| \geq q$, 记为 C_{a1} ; 如果 $|A_{a2}| \geq q$, 记为 C_{a2} 。显然 $A_{a2} \subseteq A_{a1}, C_{a2} \subseteq C_{a1}$ 。

需要注意的是, 由于情况(3)的存在, 使得 C_{p2} 和 C_{a2} 不一定相同, 即 Promise 的集合和 Accept 的集合可以不一样。当 Accept 过半时, 说明值已选定, 否则跳回 Prepare 重新开始。

(四) 值的状态

以 $n=5$ 为例考察值的确定过程, 因为消息传递的异步性, 存在几种典型类型, 如表 2 所示。

表 2 Paxos 中值的类型举例

类型	A_1	A_2	A_3	A_4	A_5
1	[0, Null]	[0, Null]	[0, Null]	[0, Null]	[0, Null]
2	[1, V_1]	[0, Null]	[0, Null]	[0, Null]	[0, Null]
3	[1, V_1]	[2, V_2]	[0, Null]	[0, Null]	[0, Null]
4	[1, V_1]	[2, V_2]	[3, V_3]	[0, Null]	[0, Null]
5	[1, V_1]	[2, V_2]	[3, V_3]	[*, *]	[*, *]
6	[1, V_1]	[2, V_2]	[4, V_3]	[4, V_3]	[4, V_3]
7	[5, V_3]	[5, V_3]	[5, V_3]	[5, V_3]	[5, V_3]

表 2 中, 方括号中值指代 $[ap, av]$, Null 指未接受任何值, * 指任意情况;

类型 1: 是最初始情况, 所有的 Acceptor 都未接受提案;

类型 2: 只有一个 Acceptor(A_1) 接受提案。这种情况发生在 Prepare 成功后 Accept 时只有 A_1 收到提案;

类型 3: 有两个 Acceptor 接受提案, 且提案不同。这种情况发生 1 型基础上, 在 Prepare 成功后 Accept 时只有 A_2 收到提案;

类型 4: 有三个 Acceptor 接受提案, 且提案不同。这种情况发生 2 型基础上, 在 Prepare 成功后 Accept 时只有 A_3 收到提案;

类型 5: 当达到类型 4 时, 由于 Proposer 会选择 C 中最大提案, 那么提案只会在 $[1, V_1]$ 、 $[2, V_2]$ 、 $[3, V_3]$ 中选出, 即值只可能是 V_1, V_2, V_3 。因此最终的提案值, 将在 V_1, V_2, V_3 中选出, 直到某个值在 A

中竞争超过半数, 将进入类型 6;

类型 6: 假设接受 V_3 的 Acceptor 已超过半数, 协议剩下的内容就是同步其他值不是 V_3 的 Acceptor;

类型 7: 所有 Acceptor 接受 V_3 。

可见值的确定过程, 包括几个状态: 初始状态, 值都为 Null。随机状态, 值可随机选择。Q 状态, 已经有 q 个 Acceptor 接受提案, 此后值只能在这些提案中选择; 确定状态, 值 V 已确定, 超过半数的节点接受同一个值 V ; 同步状态, 将 V 同步到其他 Acceptor; 一致状态, 所有的 Acceptor 都接受相同的值 V 。

从初始状态到确定状态, 值的确定是一个随机和竞争的过程, 无法保证某个提案的值一定被选定。状态从上到下是递进关系且不可逆, 也不完全严格按顺序出现, 可以跳过中间其他状态, 直接从初始状态进入确定状态。当值进入确定状态后, 任何 C 都将包

含 V, 且包含 V 的提案号最大(不变性保证 G), 这样确保 V 不会发生更改。

(五) 分布式锁

对共享资源操作需要加锁。锁也是一个协议, 需要大家共同遵守, 才能保证操作的安全性。锁分悲观锁和乐观锁两种, 操作通常包括三个阶段: 抢占锁、持有锁和释放锁。悲观锁认为锁竞争很激烈, 对锁的使用持悲观态度, 需要对所有操作先加锁, 悲观锁不允许抢占; 乐观锁是认为锁竞争不激烈, 对锁的使用持乐观态度, 先不加锁, 在发现冲突时再加锁, 乐观锁允许抢占。

Paxos 算法中只要过半节点 C 确定一个值 V, 系统的值就可确定, 过半节点 C 组成的集合就是共享资源, 对 C 中的值进行整体修改, 就要加锁。Prepare 阶段是查询的过程, 也是加锁的过程; Accept 阶段是提交的过程, 也是验证锁的过程。Paxos 算法是以消息传递为基础的, 消息传递会存在乱序、丢失、重复、延迟等问题, Proposer 持有强制锁后, 如果 Proposer 崩溃了, 或者释放锁的消息始终无法送达 Acceptor, 那么就会发生死锁, 因此不能使用强制锁, 分布式系统只能使用可抢占锁。

分布式系统有三种加锁模型: 集中模型(图 5)、版本模型(图 6)和提案模型(图 7)。下面在 Client/Server 架构中只有单个 Server 节点为例说明三种模型的工作原理。

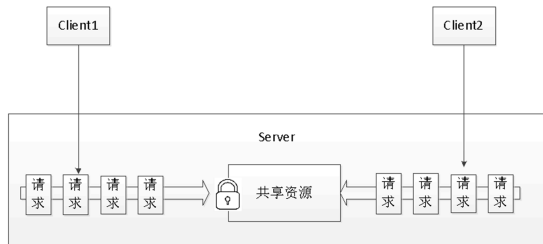


图 5 集中模型

集中模型中, Server 持有共享资源, Client 向 Server 发送请求, 修改该共享资源。并行的请求经过网络排序, 最终进入一个或多个请求队列, 一个队列对应一个处理线程。Server 按顺序处理请求, 对共享资源进行操作。该模型的特点是: Server 持有锁和操纵锁; 按 FIFO 的方式提供服务, 保持一定的公平性; 集中模型属于悲观锁, 读写操作都要加锁, 但是由于队列的存在, 减少了由于锁竞争导致的服务失败。

版本模型有两个操作阶段: 查询和更新。Client

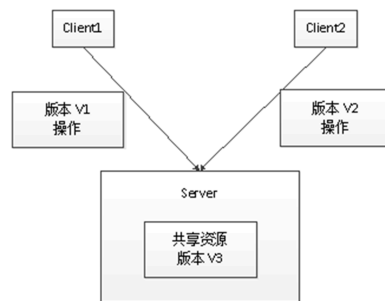


图 6 版本模型

向 Server 查询值, 服务器返回值和版本号。Client 对值进行计算, 然后向 Server 发送更新请求, 包括新值和版本号。服务器收到更新请求后, 先比对版本号, 如果一致则更新, 版本号增加 1, 否则拒绝。更新结果最后反馈给 Client。版本模型中的版本号发挥了锁的作用, Client 可以公开获得锁, 并对锁的使用有了一定的自主性, 读不需要竞争锁, 写要竞争锁, Client 负责由此引入的不一致性。版本模型属于主从一致性数据模型。当版本不更新时, Client 持有的值是 Server 的一个一致副本, 可以自由使用。当版本更新后, Client 持有的值就失效了, 需要重新获取和同步。查询和更新可以独立进行, 没有顺序限制。版本模型用的锁属于乐观锁, 可以被抢占, 适用于读多写少场景。

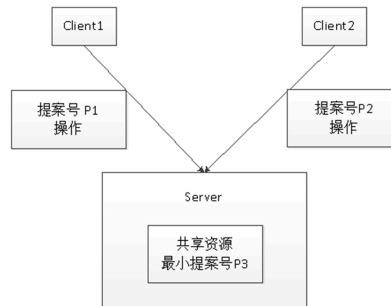


图 7 提案模型

提案模型是从 Paxos 协议提炼出来的。提案模型同样有两个阶段: 查询和更新。

1. 查询阶段

步骤 1: Client 向 Server 发起值查询, 并携带一个唯一提案号 p。这里 p 还承担申请锁的功能;

步骤 2: Server 收到查询后, 比较 p 和本地保存的最大提案号 mp, 如果 p 比 mp 大, 认为锁申请有效, 则更新 mp, 将锁分配给 p, 并返回 (mp, [ap, av]), 其中 [ap, av] 是 Server 接受过的提案, ap 是接受的提案号, av 是接受的值;

步骤 3: Client 比较返回的提案号是否一致, 如果不同, 说明锁申请失败, 则重新发起提案, 新提案号要大于 mp 且唯一。

2. 更新阶段

步骤 1: Client 经过计算后, 发送更新请求, 携带 p 和新值;

步骤 2: Server 收到更新请求, 比较提案号, 如果 p 等于 mp, 检验锁有效, 则允许更新, 否则拒绝更新, 最后返回更新结果给 Client;

步骤 3: Client 检查返回结果, 如果失败, 从头开始。

提案模型中, Client 获得了锁的更多控制权, Client 在查询阶段申请锁, 然后更新时 Server 会验证

锁的有效性。这种模型读不需要竞争锁, 写要竞争锁, 查询和更新需要严格保证顺序。提案模型的锁属于乐观锁, 可以被抢占, 适用于读多写少场景。

集中模型、版本模型和提案模型这三种加锁模型的复杂度是递增的。集中模型和版本模型无法保证一致性, 因为 Client 的请求会以乱序的形式到达各 Server, 各 Server 会依乱序执行, 导致各 Server 上的值不一致, 或者无法趋于一致。而提案模型可使得各节点的值趋于一致, 并最终决定值。

(六) 再议值的确定

值同步可能出现多种情况, 以 $n=5$ 为例考察同步过程, 如表 3 所示。

表 3 值同步

类型	A_1	A_2	A_3	A_4	A_5
1	1, [1, V]	1, [1, V]	1, [1, V]	0, [0, Null]	0, [0, Null]
2	2, [1, V]	2, [2, V]	2, [1, V]	0, [0, Null]	0, [0, Null]
3	3, [1, V]	3, [2, V]	3, [3, V]	0, [0, Null]	0, [0, Null]
4	3, [1, V]	4, [2, V]	4, [3, V]	4, [4, V]	0, [0, Null]
5	3, [1, V]	4, [2, V]	5, [3, V]	5, [4, V]	5, [5, V]
6	3, [1, V]	4, [2, V]	5, [3, V]	6, [4, V]	5, [5, V]
7	3, [1, V]	4, [2, V]	5, [3, V]	6, [4, V]	7, [5, V]

表 3 中, 值的形式是 mp, [ap, av]。

类型 1: 值刚确定;

类型 2: 只有一个 Acceptor(A_2)接受提案。这种情况发生在类型 1 基础上, Prepare 成功后 Accept 时只有 A_2 收到提案;

类型 3: 只有一个 Acceptor(A_3)接受提案。这种情况发生在类型 2 基础上, Prepare 成功后 Accept 时只有 A_3 收到提案。类型 4 和类型 5 的出现同类型 3;

类型 6: 在类型 5 的基础上, 只有 A_4 收到 Prepare(6)消息;

类型 7: 在类型 6 的基础上, 只有 A_5 收到 Prepare(7)消息;

可见即使值相同, 但是 mp 可以完全不同, ap 可以完全不同。

q 个节点的值 V 相同, 根据不变性保证, 当值 V 已确定, 后面 Proposer 选择的值必定是 V, 因此 $V=$

V_2 。因此值确定 V 后, 后面协议执行中, 所有的提案值都是 V。

Acceptor 无法知道值是否已确定, 只有 Proposer 在收到 Accept 后, 才能判断值是否确定。Proposer 在收到 Promise 消息时也可以判断值是否已确定, 如果 C_{p2} 中有过半节点值相同, 则值已确定, 此时如果没有不同值, 可以不需要往下执行, 减少不必要的同步动作。

Paxos 协议的两阶段中拒绝情况默认不发消息。为了减少 Proposer 超时等待时间, 使 Proposer 快速跟上轮数, 加快协议的执行速度和提高效率, 通常都会增加对应的拒绝消息, Promose_failed(mp) 和 Accept_failed(mp)。如何处理拒绝消息有 2 种策略:

策略 1: 优先处理, 只要 Proposer 收到 Promose_failed, 立即重新 Prepare; 只要 Proposer 收到 Accept_failed, 立即重新 Prepare;

策略 2: 滞后处理, 如果 C_{p2} 存在, 忽略 Promose_

failed 消息,继续往下执行,否则按策略 1 处理;如果 C_{a2} 存在则值已确定,忽略 Accept_failed 消息,返回否则按策略 1 处理。

Learner 学习确定值的方法是只执行 Prepare 阶段。为了不影响 mp, Learner 总是发出 Prepare(0),同时修改 Promote_failed 返回(mp,[ap,av]),如果 A_{p2} 中有过半节点值相同,则值已确定。

(七)持久化

分布式协议执行过程中,节点宕机重启现象不可避免,通常将重要数据写盘持久化,确保重启后恢复状态,保证协议继续正确执行。具体到 Paxos 协议执行过程中,如果部分节点或者全部节点宕机重启,要保证确定的值不受影响。Acceptor 需记录两次值,一次是回复 Promise 之前,记录 mp;另一次是回复 Accept 之前,要记录(mp,[ap,av])。

当值确定后,如果 C_{a2} 中某些节点崩溃重启,此时内存中[ap,av]丢失,那么值状态可能从确定状态回退到之前的状态,造成确定值丢失,这显然是不允许的。因此节点接受提案[ap,av]时必须写盘,并在节点重启后还原[ap,av],这样可保证确定的值不会改变。

五、改进效果分析

从前面的分析和推导过程可以发现,分布式一致性算法要解决的是多个 Proposer 如何对一个 Acceptor 集合整体 A 进行操作。由于异步性、宕机和分区的存在,使得一次性完成这种操作变得很困

难,只有不断尝试,才可能最终完成操作。为了提高可用性和解决歧义,选择 A 中的任何一个多数派 C ($C \subseteq A$)代表 A,对这个多数派完成的操作,即可认为对 A 完成相关操作。这包括两个方面:

一方面在值 V 确定之前,是一个随机选择和竞争的过程,最终哪个备选值被选定存在不确定性,但是遵从不变性保证(选择最大提案号的值)会缩短该进程。值确定 V 后,根据多数派 C 必相交推论, $V \in C$ 。按照不变性保证,V 必将被 A 全体接受。

另一方面在将 C 设置成同一个值的要求下,C 是共享资源。根据多数派 C 必相交推论,同时只有一个 Proposer 才能获得 QL,QL 用于保护 C。QL 可以被抢占,当 Proposer 持有的 QL 一直未被抢占(有效),它的提案才有可能成功提交,从而值被确定。

因此 Paxos 的算法可以理解为多数决,即多数派决定,在此基础上,引入不变性保证和提案锁模型,通过将两者有机结合,形成简洁和有效的分布式一致性算法。

六、总结

本文首先阐述分布式一致性算法要解决的问题,描述 Paxos 算法的内容和执行流程。然后按照循序渐进的思路,以集合定义和推论为基础,详细讨论提案、两阶段、值状态、提案锁、值的确定、持久化等 Paxos 算法,从不变性保证、值状态、分布式锁等新角度,挖掘算法核心思想,指出 Paxos 算法是在多数决机制下将不变性保证和提案锁模型进行有机结合的产物。

参考文献:

- [1] 王江,章明星,武永卫,等.类 Paxos 共识算法研究进展[J].计算机研究与发展,2019,56(4):692-707.
- [2] GILBERT S,LYNCH N A. Perspectives on the CAP Theorem[J]. Computer,2012,45(2):30-36.
- [3] LAMPORT L. The Part-time Parliament[J]. Acm Transactions on Computer Systems,1998,16(2):133-169.
- [4] OKI B M. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems[C]// the Seventh Annual ACM Symposium. ACM, 1988.
- [5] 刘芬,王芳,田昊.基于 Zookeeper 的分布式锁服务及性能优化[J].计算机研究与发展,2014(S1):229-234.
- [6] 周佳威. Kubernetes 跨集群管理的设计与实现[D]. 杭州:浙江大学,2017.
- [7] LAMPORT L. Paxos Made Simple[J]. ACM SIGACT News,2001,32(4):18-25.
- [8] 李亚男,邓玉欣,刘静.基于 Coq 的 Paxos 形式化建模与验证[J].软件学报,2020,31(8):2362-2374.
- [9] 易星辰,魏恒峰,黄宇,等. PaxosStore 中共识协议 TPaxos 的推导、规约与精化[J].软件学报,2020,31(8):2336-2361.
- [10] 肖茜文.基于副本状态机的分布式一致性算法研究[D].武汉:武汉大学,2018.
- [11] 许子灿,吴荣泉.基于消息传递的 Paxos 算法研究[J].计算机工程,2011,37(21):287-290.

Research and Improvement of Paxos Algorithm

LIU Keli¹, ZHANG Wensheng²

(1. School of Information and Architectural Engineering, Anhui Open University, Hefei 230022, China;

2. Information Technology and Network Management Center,

Anhui Open University, Hefei 230022, China)

Abstract: To further improve the execution efficiency and usability of the Paxos algorithm, a mathematical set approach is adopted to analyze the algorithm from the perspective of locks. In-depth research is conducted on the problem of value determination and persistence, and invariance guarantee and proposal lock models are introduced. By organically combining the two, a concise and effective distributed consistency algorithm is formed.

Keywords: Paxos algorithm; majority decision; invariance; proposal lock

[责任编辑 许炎]

(上接第 28 页)

Evaluation and Analysis of Efficiency of Chinese Commercial Banks Under the Background of High-quality Development:

Empirical Research Based on DEA-Malmquist Model

LU Zhou¹, HAN Changlin²

(1. School of Economics and Management, Anhui Open University, Hefei 230022, China;

2. School of Information and Architectural Engineering, Anhui Open University, Hefei 230022, China)

Abstract: As a social financing medium, improving the efficiency of commercial banks will not only deepen the reform of the financial system, but also contribute to high-quality economic development. The DEA-Malmquist model is used to study the efficiency of 114 commercial banks in China from 2012 to 2022, and the conclusions are as follows: from the static point of view, the efficiency of banks decreases and the scale efficiency needs optimizing; from a dynamic perspective, technological progress is the key to the growth of bank total factor productivity; the pure technical efficiency of state-owned large commercial banks, urban commercial banks and rural commercial banks is higher than the scale efficiency, while the joint-stock commercial banks are the opposite. Finally, suggestions are put forward from the perspectives of optimizing scale efficiency, promoting technological progress and implementing differentiated management.

Keywords: high-quality development; commercial banks; efficiency evaluation; DEA-Malmquist model

[责任编辑 王七萍]