基于 Paxos 的分布式一致性算法应用研究

刘克礼，张文盛

（安徽开放大学，合肥 230022）

摘要：针对当前大多数 Paxos 实现和分布式业务处理实现之间存在耦合度高、关系复杂、不易分析等缺点，设计一种轻量级 Paxos 系统。在分析分布式一致性实践中存在问题的基础上，提出选主和成员变更算法，制定了保序准则，并给出两种优化措施。实验结果表明，该设计能够满足进一步研究和应用的需求。

关键词：Paxos；分布式一致性；选主；成员变更

中图分类号：TP302.8　　**文献标识码**：A　　**文章编号**：2097-0625(2022)01-0091-06

一、引言

Paxos 是解决分布式一致性的核心算法[1]。Tushar Chandra 在 *Paxos made live*[2] 中提出 Paxos 从理论到实现之间有较大的鸿沟，实现一个 Paxos 往往需要对 Paxos 的经典理论做一些扩展[3]。目前针对 Paxos 的研究，除了降低理解难度，使用形式化的方法证明外[4]，更多的研究是进行优化，以提高性能和便于实践中应用。这方面已经有很多应用的实例，如 Google 的 Chubby[5]，Tencent 的 PhxPaxos[6]，Alibaba 的 X-Paxos 等。作为应用研究成果的 MultiPaxos，引入了实例和选主[7]。Raft 也是一个分布一致性算法，其优点是易实现，对 Paxos 的应用有借鉴意义[8]。Lampson[9] 分析了 Paxos 算法在工程领域实践中遇到的问题和经验教训。Tecent[10] 详细地研究了实现 PhxPaxos 中存在的问题，并给出解决方法。现有的 Paxos 应用研究中存在选主和成员变更较复杂，难以实现的问题，本文对此进行梳理，并提出两条优化措施。

二、Paxos 的实现要点及优化

（一）选主和配置变更

1. Paxos 中节点间的通信

Paxos 应用的一个重要设计是将 Proposer、Acceptor、Learner、SM(State Machine，状态机)融合在一个节点中，通常是同一个进程，其中 Proposer、Acceptor 和 Learner 使用线程实现。这样做的好处是所有节点都是对等和对称的，可以降低复杂度。此外 Proposer、Acceptor 和 Learner 之间可以共享数据，能够执行一些优化措施，提高系统性能。为了保证提案编号唯一，所有节点都要按顺序进行编号，确保编号唯一。

Paxos 是针对异步通信环境而设计的，在算法实现过程中存在通信协议的选择。UDP 协议属于典型的异步通信，支持多播和广播，可以减少发送的消息数量，适合 Paxos 应用。TCP 协议能够消除异步消息的各种问题，如通过超时重传解决丢失和延迟，通过序号解决乱序和重复的问题。使用 TCP 协议降低了编程难度，例如 ZooKeeper[11] 就是使用了 TCP 协议。

2. Paxos 中的实例

Paxos 算法只能确定一个值，但在实际应用中，需要确定多个值，并且这些值是有序的，形成值序列。各个节点都维持一个状态机，状态机的初始状态一致。各个节点将相同值序列输入本地状态机，最终得到的状态机仍然一致，从而实现一致性和高可用性。多次执行 Paxos 算法，就可以确定多个值。

相同节点的集合，可以先执行一次 Paxos 算法，直到值 V_1 确定，保存该值和相关状态，然后再初始化 Paxos 算法，重新执行，直到值 V_2 确定，依次进行

收稿日期：2021-09-16

基金项目：安徽省高校自然科学研究重点项目“基于计算机视觉的在线学业情绪分析及学习倦怠预测研究”(项目编号：KJ2021A1255)

作者简介：刘克礼(1976－)，男，安徽东至人，讲师，硕士。研究方向：计算机网络。

下去。每个 Paxos 算法的执行被称为一个实例(Instance),并被编号,这种方法称为 MultiPaxos。由于后面的选主和配置变更也要用到 Paxos,为了方便区分,记确定应用相关值的 Paxos 为 $MP(V)$,实例 i 确定的值记为 $MP(V,i)$,执行 $MP(V)$时产生的消息称为应用消息。实例和值按实例号顺序组织,一般称为日志。

各个实例之间的执行关系有两种:顺序和乱序。现有实例号 i 和 j,顺序是指当 $i<j$ 时,只有实例 i 执行完毕(V_i 被确定)才能执行实例 j。乱序是指实例 i 和实例 j 可以乱序执行,V_j 可以先于 V_i 确定。顺序是最安全的执行方法,但是效率不高。乱序效率高,但是复杂,只有判断 V_j 不依赖于 V_i($i<j$)时才能提前执行。

3. Paxos 中的选主过程

Paxos 算法因为 Proposer 竞争导致确定值可能要很长时间,甚至存在活锁问题,一般通过选主(Leader)来解决。在所有 Proposer 中选出一个 Leader,其他 Proposer 的请求都发给 Leader,由 Leader 发起提案。Leader 消除了锁竞争,可以迅速确定值,其缺点就是可能存在单点故障。

Leader 也是一种共识,为了达成共识,选主也使用 Paxos 算法。当 Leader 因为某种原因失效后,需要重新选主。记用于选主的 MultiPaxos 为 $MP(L)$,执行 $MP(L)$产生的消息称为选主消息。选主状态机很简单,只有一个变量,就是当前主(curLeader)。$MP(L)$在第二阶段选择提案值的时候,如果没有值可选,可将自己作为提案中的值。一开始所有的节点都无主,所有的节点都开始执行选主过程(假设实例号为 1),若最终节点 a 胜出,那么实例 1 的值就是 a,记为 $MP(L,1)=a$,并设置到各个节点 curLeader,此后其他节点将请求转发给节点 a。产生主的实例号一般称为代或任期(term),当主失效后,任期增加。$MP(L)$形成的日志称为选主日志。假设当前任期是 l,有主的 $MP(V,i)$扩展记为 $MP(V,i,l)$。在应用消息中携带任期,节点可检测主是否切换,如果切换,那么先要执行选主日志同步操作。

选主过程如图 1 所示。

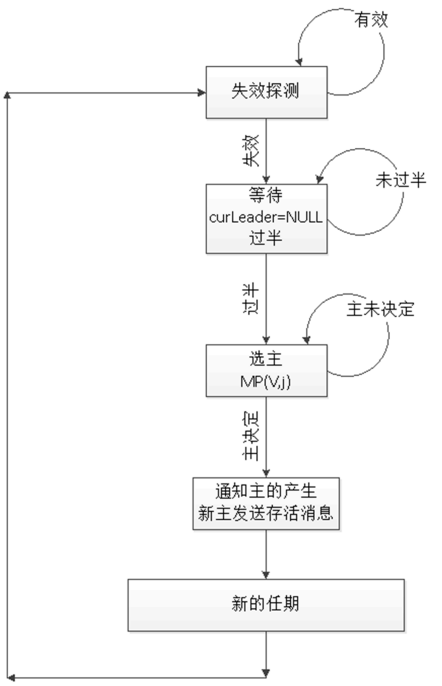

图 1　选主过程

为了维持 a 不失效,a 需要定期向其他节点发送存活消息,其他节点也可以向节点 a 定期发送探测消息,此外其他节点向 a 转发请求时也可以看成附带的探测。当超过一定时间没有收到存活消息,或发送几次探测消息没有响应,或转发几次相同的请求都没有响应时,在这三种情况下都可以认定 a 失效,并设置变量 validCurLeader 为 $NULL$。

认定 a 失效的节点 b 开始执行定时探测,向其他节点发送消息,询问其当前的 validCurLeader,其他节点返回当前实例号和 validCurLeader。如果当前实例号相同,且过半节点(包括 b)的 validCurLeader 为 $NULL$,则 b 开始执行 $MP(L,2)$。等待过半节点的 validCurLeader 为 $NULL$ 实质是一个同步的过程,是触发选主的必要条件。

如果 a 真的宕机了,那么其他节点将几乎同时认定 a 失效,并几乎同时启动 $MP(L,2)$的执行。发现选主成功的节点(例如 c)会通知其他节点当前主是谁(例如 d),d 在确认自己当选主之后,开始发送存活消息,而其他节点在同步选主日志后将 curLeader 和 validCurLeader 都设置为 d。

认定 a 失效并不等于 a 真的失效,例如分区导致节点 b 在一个少数分区 C 中,而该分区不包括 a。当 C 重新加入主分区,b 经过几轮探测发现,当前任期相同,且过半节点的 validCurLeader 仍然是 a,则重新设置本地 validCurLeader$=a$。如果任期不同,那么要先执行同步操作。

当节点 b 认定主 a 失效后(validCurLeader = $NULL$)，将不再转发应用消息给 a，也不再处理 a 的任何应用消息，直到新的主产生。

4. 成员和配置变更

运行 Paxos 算法需要知道系统的配置，至少包括各个节点的信息(IP 和端口)，记为集合 CC。各个节点上的配置必须一致，且默认情况下配置固定不变。但在实际应用中，为了应对多变的环境和需求，要求系统是可配置的和可伸缩的。

配置的变更需要所有节点参与。如果只有部分节点参与，将会导致配置不一致，从而各个节点看到的节点数 n 是不一致的，这样会影响 Quorum 的计算，从而使得 Paxos 算法执行产生混乱。配置也是一种共识，因此配置的变更也适用 Paxos 算法，记为 $MP(C,i)$，配置的状态机就是集合 CC。为了解决执行 $MP(C,i)$后可能有部分节点不知配置已变更的问题，在节点向 Leader 发送消息时，携带本地当前配置的实例号(也称为版本)。如果 Leader 的当前配置版本大于收到消息中的配置版本，立即返回错误，指示节点先同步配置。配置变更一般都是管理员手动进行，管理员在确认所有节点都正常运行后，才会执行变更，因此 $MP(C,i)$都能执行成功。

配置的变更有很多种组合：增加一个节点，删除一个节点，增加多个节点，删除多个节点，同时增加和删除多个节点。只考虑删除一个节点(DeleteNode)和增加一个节点(Add Node)两种基本操作，其他组合都可以分解成这两种基本操作。

$MP(C,i)$执行完成并应用到状态机 CC 后，如果是 DeleteNode，则判断节点值是否是自己，如果是的话，则退出；如果节点值是主节点，则各个节点将 validCurLeader 设置为 $NULL$，并立即启动选主过程，不必等待超时；如果是 AddNode，Leader 会主动通知新节点成功加入，此后新节点可以参与后续的各种消息处理。

5. 三个 MultiPaxos 的关系

上述三个一致性协议的执行对比如表 1 所示。

表 1　三个 MultiPaxos 比较

项目	用途	参数说明	值/命令	状态机	新实例触发条件	有主否
$MP(V,i,l,c)$	应用	V 代表应用、i 实例编号、l 当前任期、c 当前配置版本	应用相关	应用相关	客户端请求	有
$MP(L,i,c)$	选主	L 代表选主、i 代表任期、c 当前配置版本	SetCurLeader x	curLeader	经检测过半 validCurLeader 为 $NULL$	无
$MP(C,i,l)$	配置变更	C 代表配置、i 当前配置版本、l 当前任期	AddNode x DeleteNode x	CC	管理员发出命令	有

选主和配置变更，都会影响业务处理。配置变更可能会触发选主，而选主不会影响配置。只有选主是通过竞争确定值，执行过程不确定；而配置变更和业务处理都是在有主的情况下进行的，可以迅速确定值。当无主时，配置和业务处理暂停执行，直到新主选出，因此三个 MultiPaxos 只能交替运行。

(二)日志和数据同步

1. Paxos 中的日志和快照

在执行 Paxos 协议过程中，Acceptor 在 accept 阶段要持久化[mp,ap,av]，记为 plog，其中 mp 是收到的最大提案号，ap 是接受提案的编号，av 是接受提案的值。在 MultiPaxos 中的 Leader 在值确定后，要将实例号和值写入本地日志，记为 mlog。然后 Leader 将值输入状态机，并把日志确定标志发给其他节点，其他节点也执行同样的动作，即写日志和输入状态机。由于每个节点都具有相同的结构，因此可以将 plog 和 mlog 合并，组成(i,s,mp,ap,av)，记为 mplog，其中 i 是实例号，s 是值的状态。

节点重启后，从头执行 mplog 可恢复状态机。在日志量很少的情况下，该方法有效。但在日志量很大的情况下，一方面日志要占用大量磁盘空间，另一方面恢复状态机也需要更长的时间，在恢复期间节点不可用，这降低了系统可用性。为了减少日志量和加速状态机的重建，引入快照技术，定期导出状态机的

当前状态，并删除之前的日志。节点重启后，直接从最新快照恢复状态机，然后执行增量日志，可以快速恢复服务。

导出快照时状态机不能执行任何改动，在数据量很大的情况下，可能需要较长时间。节点对外服务是基于状态机的，状态机的冻结降低了节点的可用性。一种称为 COW（Copy On Write，写时复制）的技术解决了该问题。在准备生成快照时，父进程节点先克隆一个子进程，该子进程共享父进程的内存，父进程可以读写状态机，而子进程对状态机只读不写。父进程写状态机时操作系统会自动 COW 新的内存页，保证了父进程正常运行。子进程的状态机始终是克隆时的状态，可以从容导出快照，并在完成后退出。

2. 节点间数据同步

新节点加入集群时，状态机为空，需要从其他节点同步状态，这与节点重启时遇到的问题类似。一种同步方法是从其他节点学习所有日志，从头执行，在日志很少的情况下这种方法是有效的。在日志数据多的情况下，一方面传输日志要占用大量带宽和时间，另一方面执行日志也会需要较长的时间，此时若业务请求频繁，就可能会出现故障。另一种同步方法是从其他节点导入最新快照，再同步增量日志。

在极端情况下，同步耗时较长且新日志增长迅速，此时采用并行操作能够让新节点迅速加入集群并提供服务。新节点加入集群的操作如下：一方面参与一致性协议的执行并记录新日志（开始日志编号记为 y），但不输入状态机，另一方面通过旧日志或快照同步状态（截止日志编号记为 x）。当状态机还原到 x 时，由于 $x \leqslant y$，新节点需要学习其他节点 (x, y) 间的日志，然后将 x 之后的日志输入状态机，直至最新日志执行完毕，此时新节点才算成功加入集群。为了降低 Leader 的压力，选择同步日志节点时将 Leader 排在最后。

3. 读数据和读写保序

集群中的写操作需要通过一致性协议完成，读操作只要读取状态机中的数据。读有两种方式，一种是通过本地节点的状态机提供，称为本地读，另一种是读取 Leader 的状态机，称为全局读。本地读的优点是速度快，其缺点是不能保证读到最新数据。全局读的优点是可以读到最新的数据，其缺点是比较费时，增加了 Leader 的压力，且在集群发生分区时无法完成。

如果两个操作之间有因果关系，为了保证结果的正确性，需要对操作进行串化，或者是保序。如果两个操作之间没有因果关系，则可以乱序执行。操作通常只有两种方式：读和写。其中写操作是纯粹写，不包含自增类的写。两个操作共有三种组合：读读，读写，写写。两个操作的目标之间有三种关系：不相交，部分相交，相同。如果两个操作不相交，则可以乱序执行。通常很难判断操作是否相交，这种情况下，读写和写写都要保序，读读不需要保序。

当某个节点收到读操作后，为了保证因果关系，确保能读到最新的数据，需要将该节点之前的写操作全部提交到日志中，写操作写入状态机后，才允许执行读操作。

同一个节点上的写操作按接收的顺序保序提交，普通节点再保序提交到 Leader，Leader 保序执行一致性协议，日志按顺序执行。

（三）性能优化措施

Leader 可以让提案迅速通过，最少需要 2RTT（Round Trip Time）。通过分析 Paxos 的两阶段（Prepare 和 Accept），可以发现 Prepare 的目的是抢占锁和选择值，Accept 的目的是验证锁并提交值。因为 Leader 的存在，锁竞争已经成为小概率事件，所以 Prepare 可以省略，直接跳转至 Accept 阶段。省略 Prepare 后，$MP(V)$ 只需要 1RTT 就可完成，系统性能可以提高 1 倍。

省略 Prepare 需要满足条件，即不能违反 Paxos 值不变性约束。分析如下：在 Leader 发出 promise 消息时，此时值还未确定，假设该 promise 的提案编号 P 是 1，而每个实例的初始 P 是 0，在 (0,1) 之间没有其他提案，则 promise 安全。在新旧 Leader 交替时，存在如下情况，即旧 Leader 发送提案 $[1, V1]$ 被某个节点接收后，因某种原因旧 Leader 宕机，假设此时新 Leader 生成 $[2, V2]$ 提案，如果 $V1$ 不等于 $V2$，那么在新 Leader 完成 Accept 后，不变性约束就被破坏。为了限制上一个 Leader 造成的影响，将任期信息附加到 P 上，新的提案编号为 $(term, P)$。如果 promise 消息中的任期小于当前任期，提案将被拒绝。

通常情况下 $MP(V)$ 一次可确定一个值，也可以同时确定多个值，即同时确定编号分别是 $i, i+1, i+2, \ldots, i+n$ 的实例，这样的批量确定可以使系统性能得到提升。当然也不能无限增加同时确定的实例

个数，受网络传输数据包大小限制，当数据包超过最大传输单元时就会发生分片和重组，由此导致的开销和不确定性会抵消性能的优化。批量确定只能串行操作，在前一个批量确定完成后，才能执行下一个批量的确定，这样就不会产生大量日志空洞。

三、实验结果与分析

在所有的一致性实现当中，一致性 KV（Key Value，键值对）数据库是最简单的。根据上述分析，使用 Python 实现一致性 KV 数据库，并进行测试。实验环境采用 3 个节点的虚拟机，配置为 4CPU AMD Opteron（tm） Processor 6376，4G 内存，50G 硬盘，1G 网卡。测试项目包括 2RTT 吞吐量，1RTT 吞吐量，批量确定吞吐量。结果如表 2 所示，单位为 RPS（Request Per Second）。

表 2　一致性 KV 数据库吞吐量测试　　Request Per Second

长度 KV	测试项目				
	2RTT(A)	1RTT(B)	批量确定(C)(MTU=1 500)	B/A	C/B
$K=10,V=10$	870	1 513	112 110	1.74	74.10
$K=10,V=100$	713	1 400	18 201	1.96	13.00
$K=10,V=1000$	537	1 074	1 589	2.00	1.48
$K=100,V=10$	743	1 436	19 581	1.93	13.64
$K=100,V=100$	663	1 306	9 705	1.97	7.43
$K=100,V=1000$	513	1 016	1 375	1.98	1.35
$K=1000,V=10$	543	1 096	1 637	2.02	1.49
$K=1000,V=100$	520	1 041	1 430	2.00	1.37
$K=1000,V=1000$	497	1 003	993	2.02	0.99

表 2 中 $K=10$ 指 Key 的长度是 10 字节，$V=10$ 指值的长度是 10 字节，余类推。A 指代 2RTT 列数据，B 指代 1RTT 列数据，C 指代批量确定列数据。B/A 指 B 列除以 A 列的值，显示将 2RTT 优化为 1RTT 后系统吞吐量提升效果。C/B 指 C 列除以 B 列的值，显示将 1RTT 优化为批量确定后系统吞吐量提升效果。根据表 2 可知，2RRT 比 1RTT 吞吐量大将近 1 倍，而批量确定的吞吐量比 1RTT 又大若干倍，优化效果明显。

四、小结

详细分析 Paxos 应用过程中可能遇到的问题，讨论了常规的通信、实例、日志、同步等，就选主和成员变更提出了轻量级的算法，将选主、配置和应用三个一致性协议进行有机组合，互相配合，确保系统稳定、可靠和易用。为了保证结果的正确性，研究分析了读写保序问题，并给出一般准则。最后讨论了性能优化措施，将 2RTT 缩减到 1RTT，再到批量确定。结果表明，该设计能够满足进一步研究和应用的需求。

参考文献：

[1] LAMPORT L. The Part-time Parliament[J]. ACM Transactions on Computer Systems，1998，16(2)：133-169.

[2] CHANDRA T，GRIESEMER R，REDSTONE J. Paxos Made Live-An Engineering Perspective[C]//Proc of the 26th Annual ACM Sympon Principles of Distributed Computing(PODC'07). New York：ACM，2007：398-407.

[3] 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.

[4] 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.

[5] BURROWS M. The Chubby Lock Service for Loosely-coupled Distributed Systems[C]//Proc of the 7th USENIX Sympon Operating System Design and Implementation (OSDI'06). Berkeley，CA：USENIX Association，2006：335-350.

[6] Tencent. PhxPaxos：A State-synchronization Lib Based on Paxos Protocol[CP/OL]. (2017-01-05)[2021-08-30]. https://

github.com/Tencent/phxpaxos.

[7] LAMPORT L. Paxos Made Simple[J]. ACM SIGACT News,2001,32(4):18-25.

[8] ONGARO D,OUSTERHOUT J. In Search of an Understandable Consensus Algorithm[C]//Proceedings of the ATC14, USENIX Annual Technical Conference,2014:305-319.

[9] LAMPSON B W. How to Build a Highly Available System Using Consensus[C]//International Workshop on Distributed Algorithms. Springer-Verlag,1996:1-17.

[10] Tencent. 微信自研生产级 paxos 类库 PhxPaxos 实现原理介绍[EB/OL]. (2016-06-22)[2021-08-30]. https://mp.weixin.qq.com/s?__biz=MzI4NDMyNTU2Mw==&mid=2247483695&idx=1&sn=91ea422913fc62579e020e941d1d059e#rd.

[11] REED B,JUNQUEIRA F P. A Simple Totally Ordered Broadcast Protocol[C]//Large-Scale Distributed Systems and Middleware,2008:1-6.

On the Application of Distributed Consistency Algorithm Based on Paxos

LIU Keli, ZHANG Wensheng

(Anhui Open University, Hefei 230022, China)

Abstract: In view of the shortcomings between most current Paxos implementations and distributed business processing implementations, such as high coupling, complex relationships, and difficult analysis, a lightweight Paxos system is designed. On the basis of analyzing the problems in the practice of distributed consistency, the algorithm for the selection of the principal and the change of members is proposed, the rules for order preservation are formulated, and two optimization measures are given. Experimental results show that the design can meet the needs of further research and application.

Keywords: Paxos; distributed consistency; leader selection; member change

[责任编辑　李潜生]

(上接第 66 页)

Neutralization and Elegant Legitimism: the Pursuit of Literature in the Early Tang Dynasty in the Construction of Cultural Community

SU Liguo

(School of Literature and Journalism, Gansu University of Political Science and Law, Lanzhou 730070, China)

Abstract: Under the influence of the ideal of new literature, neutralization and elegant legitimism became the focus of tortuous pursuit in the construction of cultural community in the early Tang Dynasty. Literature has a unique function and bearing in the construction of cultural community, which is mainly embodied in adhering to the elegant and upright way, eulogizing the upward spirit, publicizing the prosperity of the country and creating a beautiful humanistic environment. So the pursuit of neutrality and elegance in the early Tang Dynasty can undoubtedly enhance the cultural self-confidence and sense of responsibility of both sides of literary acceptance, ensure the good inheritance of culture and academia, and lay a solid humanistic foundation for the cultural community.

Keywords: literature in the early Tang Dynasty; neutralization and elegant legitimism; cultural community

[责任编辑　夏强]