第 37 卷 第 21 期
Vol.37 No.21

计 算 机 工 程
Computer Engineering

2011 年 11 月
November 2011

·开发研究与设计技术· 文章编号：1000—3428(2011)21—0287—04 文献标识码：A 中图分类号：TP301.6

基于消息传递的 Paxos 算法研究

许子灿，吴荣泉

(中国电子科技集团公司第三十二研究所，上海 200233)

摘 要：针对分布式系统中的一致性问题，对基本 Paxos 算法中的 3 种角色进行分步骤阶段分析，提出 5 种行为优化改进措施，其中包括限制角色提案、引入随机机制、提前拦截消息、减少消息传递和增加角色行为等方法。实验结果表明，改进后的算法能降低通信负载，提高系统安全，从而使分布式系统具有高可用性及高一致性。

关键词：一致性；消息传递；Paxos 算法；投票选举；行为优化

Research on Paxos Algorithm Based on Messages Passing

XU Zi-can, WU Rong-quan

(The 32nd Research Institue of China Electronics Technology Group Corporation, Shanghai 200233, China)

【Abstract】Aiming at the consensus problem in distributed system, this paper analyzes the three parts in the basic Paxos algorithm step by step, and proposes five kinds of improvements to optimize the behavior, including the methods of limiting the role to propose, introducing the random mechanism, intercepting the message in advance, reducing messaging and increasing the behavior of the role. Experimental result proves that the improved Paxos algorithm reduces the communication load, improves the system security, and makes the distributed system usable and consistent.

【Key words】consensus; messages passing; Paxos algorithm; voting by ballot; behavior optimizing

DOI: 10.3969/j.issn.1000-3428.2011.21.098

1 概述

随着企业应用对计算能力需求的不断膨胀，传统的服务器模式已经无法满足需要。网络带宽的不断增长，使得通过网络访问非本地的计算服务日益成熟。采取云计算模式，使得所有计算服务集群于云计算数据中心已成为发展趋势。云计算是一种商业计算模型，它将计算任务分布在大量计算机构成的资源池上，使用户能够按需获取计算力、存储空间和信息服务。由于硬件和软件本身的不稳定性，如何能够保证服务的高可靠性已经成为云计算中越来越重要的一个问题。

对于由大规模廉价服务器群构成的云计算数据中心而言，分布式同步机制是开展一切上层应用的基础，是系统正确性和可靠性的基本保证。在一个分布式系统中，如果各节点的初始状态一致，每个节点都执行相同的操作序列，那么它们最后应能得到一个一致的状态。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个“一致性算法”以保证每个节点看到的指令一致。

Paxos 算法是由微软的 Leslie Lamport 提出的一种基于消息传递的分布式一致性算法[1]，其解决的问题是一个分布式系统如何就某个决议达成一致，可以应用在数据复制、命名服务、配置管理、权限设置和号码分配等场合。Google Chubby 和 Hadoop ZooKeeper 是云基础架构分布式同步机制的典型代表。Chubby Lock Service(分布式锁服务)利用 Paxos 算法为多个数据副本间基于锁机制进行同步提供了有效的支持，并将其用于 Bigtable(大表)，Hadoop 的 ZooKeeper 则是 Chubby 系统的开源实现。

本文描述并分析 Paxos 算法，通过对 3 种角色进行阶段分析和行为优化，提出了一种改进算法。

2 通信模型

为保持数据一致性，分布式系统的不同节点需要进行通信来达成共识，节点通信存在 2 种模型：消息传递(messages passing)和共享内存(shared memory)[2]。

(1)消息传递：通过 IPC 机制来交换字节流和“面向记录”的数据。通常，应用程序开发者需要定义这些信息的格式、内容和应用协议；参与者必须遵守这一协议以交换信息。在基于消息传递的并行计算环境中，用户必须通过显示地发送和接收消息来实现处理机间的数据交换。此时，每个进程都有自己独立的地址空间，进程之间的访问不能直接进行，必须通过显示的消息传递来完成。因此，并行处理开销比较大，主要适合于大粒度的并行计算，特别是由于消息传递的程序设计要求用户很好地分解问题、组织不同进程间数据交换，因此它特别适合于大规模可扩展性并行算法。

(2)共享内存：允许相同或不同主机上的多个进程访问、交换数据，就像数据位于每一个进程的本地地址空间一样。在网络应用程序中，如果数据必须被多个进程读取或处理，那么，较之“消息传递”，“共享内存”设施将是一种更有效的通信机制。在基于共享内存的并行计算环境中，并行应用的成员进程之间的数据共享是通过共享内存地址空间实现的。分布式共享内存(DSM)扩展了 OS 虚拟内存机制，可以通过全局/共享内存中的数据进行透明的进程间通信。

3 Paxos 算法

3.1 数据一致性问题

如图 1 所示，一般银行 ATM 系统的体系结构：$C=\{C_1, C_2,\cdots,C_n\}$代表 n 个客户端(C_i是取款机)，$S=\{S_1, S_2,\cdots,S_m\}$代表 m 个服务端(S_i 响应每个与之相连的客户端的命令请求，是 Paxos 算法运行的实体)。

作者简介：许子灿(1985－)，男，硕士研究生，主研方向：分布式计算，云计算；吴荣泉，研究员

收稿日期：2011-04-20 **E-mail：**xuzican@163.com

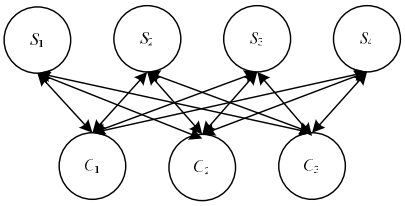

图 1 银行 ATM 结构

采用这样的体系结构的目的就是保证在若干个服务端出现故障的情况下，所有的客户端仍能不受影响地进行工作。由于 C 与 S 之间的通信是异步的，那么如果 C_1、C_2、C_3 分别发出如下命令 a、b、c，$operation(C_1)=a$，$operation(C_2)=b$，$operation(C_3)=c$，那么总共可能会产生 6 种合法命令序列，即(a, b, c), (a, c, b),…,(c, b, a)。如果 S 中的每一个服务端执行了不同的合法命令序列，将会导致整个系统的不一致性问题，所以 Paxos 算法的任务是保证操作一致性，即 S 中每一个处于正常工作的服务端都将执行一个相同的命令序列，例如(a, b, c)。

3.2 基于消息传递的算法流程

3.2.1 算法存在的基本前提

Paxos 算法是目前在云计算领域中真正得到有效使用的一致性算法，其使用范围存在如下的约束条件：

(1)算法执行的环境处在一个可靠的通信环境中，即在异步通信过程中，发送的数据可能会被丢失、延长、重复，但不会被篡改。

(2)算法运行的实体(服务端 S_i)不会出现拜占庭失效(Byzantine failures)。各个角色可工作在任意速度，允许停止和重启的错误。

3.2.2 节点类型和阶段描述

在 Lamport 提出的 Paxos 算法中节点被分成了 3 种类型：proposer，acceptor 和 learner[3]。其中，proposer 向 acceptor 提交提案(proposal)，提案中含有决议(value)；acceptor 审批提案；learner 获取并执行已经通过审批的提案中含有的决议。一个节点可以兼有多重类型。其中：

提案：一个有序对<编号，决议>；

决议：一个整数；

提案被接受：某一个 acceptor 向该提案的 proposer 发出 accepted 信息，并将该提案所包含的 value 加入自己的 V_A(list)；

决议被批准：含有该决议的提案被 acceptor 集合中的任意一个 majority 的所有成员接受。

在这种情况下，满足以下 3 个条件就可以保证数据的一致性[4]：(1)决议只有在被 proposer 提出后才能批准；(2)每次只批准一个决议；(3)只有决议确定被批准后 learner 才能获取这个决议。

Lamport 通过约束条件的不断加强，最后得到了一个可以实际运用到算法中的完整约束条件：如果一个编号为 n 的提案具有值 v，那么存在一个多数派，要么他们中没有人批准过编号小于 n 的任何提案，要么他们进行的最近一次批准具有值 v。为了保证决议的唯一性，acceptor 也要满足一个如下的约束条件：当且仅当 acceptor 没有收到编号大于 n 的请求时，acceptor 才批准编号为 n 的提案。在这些约束条件的基础上，可以将一个决议的通过分成 2 个阶段：

(1)准备阶段：proposer 选择一个提案，并将它的编号设为 n，然后将它发送给 acceptor 中的一个多数派。acceptor 收到后，如果提案的编号大于它已经回复的所有消息，则 acceptor 将自己上次的批准回复给 proposer，并不再批准小于 n 的提案。

(2)批准阶段：当 proposer 接收到 acceptor 中的这个多数派的回复后，就向回复请求的 acceptor 发送 accept 请求，在符合 acceptor 一方的约束条件下，acceptor 收到 accept 请求后即批准这个请求。

通过这 2 个阶段的消息传递和实例执行，即可保证数据和操作的一致性。

3.3 角色/阶段分析

表 1 为一个 paxos 实例的角色/阶段关系。

表 1 角色/阶段关系

角色	Phrase1	Phrase2	Phrase3
proposer(P)	竞争 Leader，向 A 发送 prepare	(1)接收 A 的 promise 或 reject (2)对于 reject，编号加 1，重新向 A 发送 prepare (3)对于 promise，选取一个 value 并发送 accept	(1)接收 A 的 accepted 或 nack (2)对于 nack，编号加 1，重新向 A 发送 prepare (3)对于 accepted，向所有 proposer 发送自身成为 leader 的消息，并向 learner 发送 value 值
acceptor(A)	(1)接收处理 P 的 prepare (2)回复 promise 或 reject	(1)接收处理 accept (2)回复 accepted 或 nack	无
learner(L)	无	无	接收广播，学习 value，执行任务

角色 proposer 按如下流程执行：

(1)向所有 acceptor 发送 prepare(N_A)请求。

(2)如果收到 reject(N_H)信息，那么重新发送 prepare(N_H+l)；如果收到 acceptor 集合任意一个 majority 的 promise(N_A, V_A)回复，那么如果所有的 V_A 均为空，proposer 自由选取一个 V_A'，回发 accept(N_A ,V_A')；否则回发 accept(N_A , V_i)。

(3)如果收到 nack(N_H)，回到步骤(1)过程，发送 prepare(N_H+l)；如果收到任意一个 majority 所有成员的 accepted 信息(表明选举完成)，向所有 proposer 发送自身成为 leader 的消息，并向 learner 发送 value 值。

其中，N_A 为该次提案的编号；N_H 为当前提案的最高编号；V_i 为 V_A 中提案编号最高的 value。

角色 acceptor 按如下流程执行：

(1)接收 prepare(N_A)，如果 N_A>N_H，那么回复 promise (N_A , V_A)，并置 N_ H=N_ A；否则回复 reject(N_H)。

(2)接收 accept(N_A, V_A)，如果 N_A<N_H，那么回复 nack(N_H)信息(暗示该 proposer 提完案后至少有一个其余的 proposer 广播了具有更高编号的提案)；否则设置 this.V_A＝V_A，并且回复 accepted 信息。

其中，promise (N_A , V_A)表示向 proposer 保证不再接受编号小于 N_H 的提案；accepted 表示向 proposer 发送决议被通过信息；V_A 为 acceptor 之前审批过的决议(可为空)；N_H 为 acceptor 之前接收提案的最高编号。

角色 learner 学习 value，并执行任务。

4 算法分析与改进

Paxos 算法采用了投票选举的形式，通过数学归纳的思想进一步保障了 majority 机制，保证了 $2F+1$ 的容错能力，即具备 $2F+1$ 个结点的系统最多允许 F 个结点同时出现故障[5]。但上述算法也存在一些局限，可以对算法中提到的 3 个角色

进行行为优化，作出如下改进：

(1)针对角色 proposer 的行为优化

优化 1：存在如下特殊情况，根据算法一个编号更大的提案会终止之前的提案过程，如果 2 个 proposer 在这种情况下都转而提出一个编号更大的提案，那么就可能陷入活锁。此时可以选举出一个 president，仅允许 president 提出提案，避免活锁现象的发生。

优化 2：在一个 proposer 被告之它的 proposal 编号不是当前最大的时候，一般是立即创建另一个具有更大编号的 proposal，并发给所有的 acceptor。此时可以引入一个随机等待机制，即在一个 proposer 的 proposal 被否决后，它随机等待一定时间，然后再创建一个具更大编号的 proposal 进行竞争，这样有利于降低出现循环竞争的可能性。

(2)针对角色 acceptor 的行为优化

优化 3：在一个 acceptor 收到更大编号的 proposal 时，可以向它已经 promise 过的 proposer 提早发出 nack 消息，这样可以降低通信负载。因为如果对方在发出 acccpt 消息之前收到 nack 消息，那么可以减少一次一定会被拒绝的 accept 消息的发送。即便对方已经发送了 accept 消息，那么也不会增加任何不必要的通信，因为 nack 消息可以被认为是回复 accept 的异步消息。

优化 4：acceptor 在发送 acceped 消息后，可以同时将这个通过的决议发送给 learner 的一个子集，然后由这个子集中的 learner 去通知所有其他的 learner。这样可以减少决议发布过程中的消息量，同时减少 proposer 的负担，提高 learner 的可靠性。

(3)针对角色 learner 的行为优化

优化 5：在一个 proposal 被通过后，learner 可以通过创建 proposer 角色并向 acceptor 发送 proposal 信息来获得当前最新的 proposal，从而不需要任何 acceptor 或 proposer 进行广播，既降低通信负载，又避免了安全性与通信负载之间的制约[6]。

5　实验结果与分析

5.1　仿真场景

本实验主要是在单台机器下，通过多个终端来仿真模拟多台机器，从而对 Paxos 算法进行性能分析。优化 1 和优化 2 从理论上优化了 Paxos 算法，减少了特殊情况出现的概率；优化 4 和优化 5 都是关于决议发布的；相对来说，优化 5 更巧妙，更具有实际可操作性。本文主要对基本 Paxos 算法和采用优化 1、优化 2、优化 3 和优化 5 后的 Paxos 算法进行性能比较。

本实验环境为 PC 机 Core2 Duo CPU，2.09 GHz，2 GB 内存，操作系统为 Windows XP，在 Visual Studio 2005 上用 C++实现了基本 Paxos 算法和改进的 Paxos 算法，并分别使用 3 台和 5 台终端做了 5 组实验。对于 3 台终端，本文仿真模拟了系统中 3 台机器正常运行的情况，也模拟了其中 1 台机器宕机、剩余 2 台机器正常运行的情况。对于 5 台终端，本文仿真模拟了系统中 5 台机器正常运行的情况，接着模拟了其中 1 台机器宕机、剩余 4 台机器正常运行的情况，最后模拟了其中 2 台机器宕机、剩余 3 台机器正常运行的情况。

5.2　算法评价标准

对于分布式一致性算法，主要考虑如下评价指标：

(1)容错性：在系统中机器宕机、数据包丢失和传送延迟等情况下，系统能够正常工作，算法正常执行，并能保证系统的分布式一致性。

(2)运行时间：反映一致性算法达成协议的速度，运行时间越少，系统越早达到一致。

(3)消息个数：反映系统内部通信负载大小。

(4)提交失败数：反映一致性算法在系统内部的运行效率，失败数越低，算法运行效率越高。

(5)平均每秒提交数：反映系统内部算法的运行速度。

5.3　算法性能分析

图 2 和图 3 分别是 3 台和 5 台终端时运行时间的仿真结果，图 4 和图 5 分别是 3 台和 5 台终端时消息个数的仿真结果，图 6 和图 7 分别是 3 台和 5 台终端时提交失败数的仿真结果，图 8 和图 9 分别是 3 台和 5 台终端时平均每秒提交数的仿真结果。其中，使用 3 台终端时，第 4 次后有 1 台终端宕机；使用 5 台终端时，第 3 次后有 1 台终端宕机，第 6 次后有 2 台终端宕机。

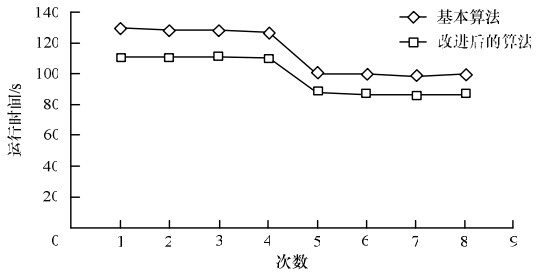

图 2　3 台终端时运行时间比较

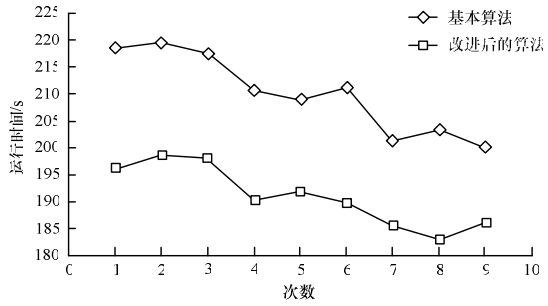

图 3　5 台终端时运行时间比较

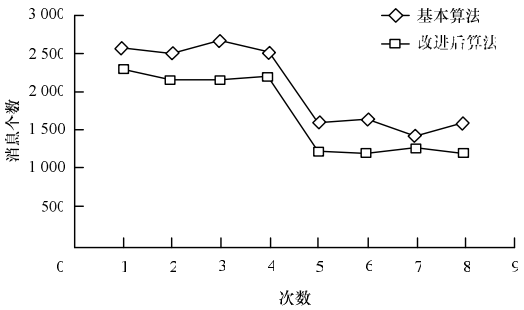

图 4　3 台终端时消息个数比较

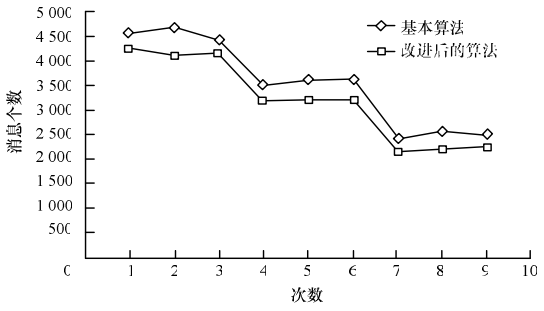

图 5　5 台终端时消息个数比较

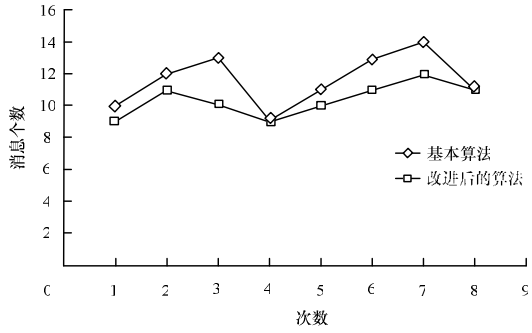

图 6 3 台终端时提交失败数比较

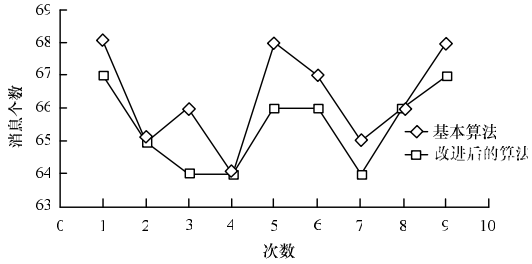

图 7 5 台终端时提交失败数比较

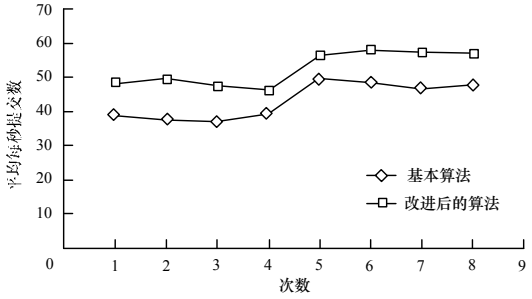

图 8 3 台终端时平均每秒提交数比较

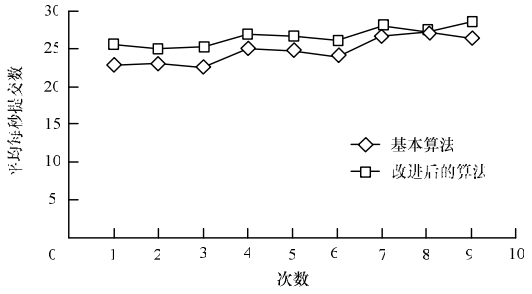

图 9 5 台终端时平均每秒提交数比较

从上述实验结果可以得出如下结论：

(1)本文实现的系统具有容错性，对于一个有 2*F*+1 个终端的系统，能够保证在终端宕机数小于 *F*+1 的情况下系统正常运行；基本算法和改进后的算法都能够实现各运行终端数据间的一致性。

(2)改进后的算法比基本算法达成协议的速度要快，节省了运行时间；消息个数减少，通信负载明显降低；提交失败数有一些减少，有轻微效果；由于通信负载的降低，改进后的算法平均每秒提交数略有提高。

(3)系统中终端数越少，达成协议的速度越快；在终端数量一定的系统中，宕掉的机器越多，出现提交失败的比例也越高，而整个系统运行速度也越慢。

6 结束语

由于 Paxos 算法需要高速稳定网络；一个 2*F*+1 个节点的网络中，需要 *F*+1 个节点完成事务才能成功；吞吐量低，不适合高请求量的场合，因此大部分分布式存储产品并不直接使用 Paxos 算法来同步数据[7]，这些约束前提也限定了本文研究的范围。

本文在对基本 Paxos 算法的角色进行阶段分析后，提出了一种基于角色行为优化的改进的 Paxos 算法。通过亲自开发实现 Paxos 算法，进行性能比较和分析，更加深入了解了分布式系统如何保持数据间的一致性和容错性。下一步工作是怎样在云计算环境中灵活运用改进的 Paxos 算法，有效地实现负载均衡。

参考文献

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

[2] Alexander C K, Hery M L. A Comparison of Message Passing and Shared Memory Architectures for Data Parallel Programs[J]. ACM SIGARCH Computer Architecture News, 1994, 22(2): 94-105.

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

[4] Lamport L. Fastpaxos[J]. Distributed Computing, 2006, 19(2): 79-103.

[5] Jim G, Lamport L. Consensus on Transaction Commit[J]. ACM Transactions on Database Systems, 2006, 31(1): 133-160.

[6] 唐西林, 杨智勇, 杨长海. 基于匿名消息广播的电子选举方案[J]. 计算机工程, 2009, 35(13): 137-138.

[7] Hagit A, Jennifer W. 分布式计算[M]. 骆志刚, 译. 北京: 电子工业出版社, 2008.

编辑 任吉慧

(上接第 286 页)

6 结束语

本文提出的 EAST 现场高速采集子系统能够更加准确地反映信号的真实情况，方便物理学家更好地进行科学研究。在正在进行的中科院等离子体所 2010 年 EAST 秋季实验中，该系统运行稳定，得到了物理学家的认可。

参考文献

[1] PXI-2020/2022 Simultaneous Sampling Card User's Manual[Z]. ADLINK Technology Inc., 2010.

[2] 孙 鑫, 余安萍. VC++深入详解[M]. 北京: 电子工业出版社, 2006.

[3] 刘 英, 罗家融, 李贵明, 等. EAST 数据采集控制系统[J]. 计算机工程, 2008, 34(14): 228-230.

[4] D2K-DASK Function Reference[Z]. ADLINK Technology Inc., 2010.

[5] 杨 飞, 肖炳甲, 刘连忠, 等. 基于MDSplus的EAST工程数据获取及显示[J]. 微计算机信息, 2010, 26(13): 20-22.

编辑 顾逸斐