计算机科学与探索
Journal of Frontiers of Computer Science and Technology
1673-9418/2019/13(05)-0866-09
doi: 10.3778/j.issn.1673-9418.1804062

MEPaxos:低延迟的共识算法*

赵守月[1,2],葛洪伟[1,2+]
1. 轻工过程先进控制教育部重点实验室(江南大学),江苏 无锡 214122
2. 江南大学 物联网工程学院,江苏 无锡 214122
+ 通讯作者 E-mail: ghw8601@163.com

摘 要:共识问题作为分布式计算中最重要的基本问题之一,被广泛应用在状态机复制、原子广播、领导者选举等领域。解决共识问题的算法通常存在单领导者性能瓶颈、响应延迟受命令冲突的影响等问题。针对这些问题,在非拜占庭故障下的异步分布式系统中,提出了一种低延迟的共识算法MEPaxos(modified Egalitarian Paxos)。首先,提出了系统平均延迟的计算方法;然后,引入超时机制对二阶段提交算法进行改进;接着,根据系统平均延迟计算结果,利用改进的二阶段提交算法自动选择平均延迟较小的算法模式执行;最后,在亚马逊弹性计算云(elastic compute cloud,EC2)平台上将此算法与当前共识算法进行实验对比分析,结果表明,MEPaxos算法下,系统延迟性能得到了提升。

关键词:分布式计算;共识算法;低延迟;Paxos
文献标志码:A **中图分类号:**TP301.6

赵守月, 葛洪伟. MEPaxos:低延迟的共识算法[J]. 计算机科学与探索, 2019, 13(5): 866-874.
ZHAO S Y, GE H W. MEPaxos: consensus algorithm for low latency[J]. Journal of Frontiers of Computer Science and Technology, 2019, 13(5): 866-874.

MEPaxos: Consensus Algorithm for Low Latency*

ZHAO Shouyue[1,2], GE Hongwei[1,2+]
1. Ministry of Education Key Laboratory of Advanced Process Control for Light Industry (Jiangnan University), Wuxi, Jiangsu 214122, China
2. School of Internet of Things Engineering, Jiangnan University, Wuxi, Jiangsu 214122, China

Abstract: As one of the most important basic problems in distributed computing, the consensus problem is widely used in the fields of state machine replication, atomic broadcast, leader election and so on. The algorithm for solving the consensus problem usually has the problems of single leader performance bottlenecks, response latency affected

* The National Natural Science Foundation of China under Grant No. 61305017 (国家自然科学基金); the Research Innovation Program for College Graduates of Jiangsu Province under Grant Nos. KYLX16_0781, KYLX16_0782 (江苏省普通高校研究生科研创新计划项目).
Received 2018-04-26, Accepted 2018-06-11.
CNKI网络出版: 2018-06-26, http://kns.cnki.net/KCMS/detail/11.5602.TP.20180622.1318.010.html

by command conflicts. In response to these problems, a low-latency consensus algorithm MEPaxos (modified Egalitarian Paxos) is proposed in asynchronous distributed systems under non-Byzantine failures. First, the method of calculating the system average latency is proposed. Then, a timeout mechanism is introduced to improve the two-phase commit algorithm. Later, based on the system average latency calculation result, an improved two-phase commit algorithm is used to automatically select an algorithm mode with a smaller average latency. Finally, this algorithm is compared with the current consensus algorithm experimentally on Amazon EC2 (elastic compute cloud) platform. The results show that the system latency performance has been improved using the MEPaxos algorithm.

Key words: distributed computing; consensus algorithm; low latency; Paxos

1 引言

分布式计算中的一个基本问题是在故障存在的情况下，依然能实现整体系统的可靠性[1]。这通常需要分布式系统中各个节点（副本）对处理过程中某一数值达成一致，即取得共识。共识问题是许多商业分布式系统[2-6]的核心。然而现实世界中，由于各种故障（进程故障、通信链路故障等）的存在，解决共识问题十分困难[7]。

FLP不可能定理[8]指出：在存在故障（即使只有一个进程故障）的异步系统中，不存在用于解决共识问题的确定性算法。这说明，解决共识问题的算法必须在安全性（safety）和灵活性（liveness）之间取舍。其中，确保安全性算法中影响最深远的是Lamport提出的Multi-Paxos（multi-decree Paxos）算法[9-10]。

Multi-Paxos致力于解决异步分布式系统[11]中非拜占庭故障[12]的共识问题。Multi-Paxos依赖单个领导者处理并发客户端发送的命令，但在广域网环境下的分布式系统中，单领导者的设置给客户端和系统交互带来了更大的延迟（和领导者不在同一局域网的客户端需要更多的时间和领导者通信）。同时，领导者易成为整个系统的性能瓶颈。

针对上述单领导者的缺陷，许多Multi-Paxos算法变种被提出。Fast Paxos[13]允许客户端将命令发送给所有副本，但在并发命令的情况下，会产生冲突（collision），造成延迟性能显著下降。Generalized Paxos[14]可以按任意顺序执行可交换的命令，但对于不可交换的命令，延迟性能显著下降。Mencius[15]通过每个副本轮流当领导者的方式均衡负载，但延迟性能易受到单个较慢副本和客户端负载不均衡的影响。S-Paxos（scalable Paxos）[16]中副本对客户端发送的命令批处理之后发送给领导者，减轻了领导者的压力，但在广域网环境下，领导者依然是性能瓶颈。EPaxos（egalitarian Paxos）[17]允许客户端将命令发送至任意副本（通常是距客户端最近的副本），但延迟性能易受命令冲突的影响。

以上算法在某种程度上较Multi-Paxos降低了客户端感知到的延迟，但在负载不均衡、命令冲突增大等情况下，延迟性能会变差。本文基于低延迟的设计目标，结合EPaxos和Multi-Paxos，提出了共识算法MEPaxos（modified egalitarian Paxos）。MEPaxos综合考虑客户端的负载情况、并发客户端的命令冲突情况以及网络的实时情况提出了系统平均延迟的计算方法；接着引入超时机制对二阶段提交算法进行改进；最后根据系统平均延迟计算公式，利用改进的二阶段提交算法自动进行算法转换，以获取最优的延迟性能。

2 相关工作

2.1 Multi-Paxos算法

Multi-Paxos将客户端发送的命令复制到$2F+1$（F为能忍受的最大故障数）个副本来确保安全性，即：在$F+1$个（称为法定人数）副本无故障的情况下，Multi-Paxos能确保安全性。算法具体过程如图1所示。客户端将命令C发送给单个领导者，领导者和所有副本（称之为接受者）进行一轮信息交流（图1中Prepare阶段，每个领导者只执行一轮Prepare消息），

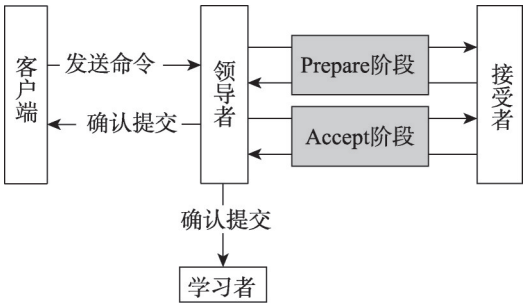

Fig.1 Multi-Paxos process

图1 Multi-Paxos处理过程

确保接受者不再响应之前的命令。若收到法定人数接受者的回复，领导者和接受者再进行一轮信息交流（图1中Accept阶段），请求接受者复制 C。若领导者收到法定人数接受者的回复，确认 C 被成功提交，发送确认消息给客户端和所有副本（称之为学习者），学习者本地提交 C。

2.2 EPaxos算法

EPaxos是近年业内最认可，广域网环境下延迟性能综合最好的Multi-Paxos算法变种。和Multi-Paxos类似，EPaxos将客户端命令复制到 $2F+1$ 个副本上确保安全性。算法具体过程如图2所示。通常情况下，客户端将命令 C 发送给最近的副本（称该副本为 C 的领导者）。C 的领导者和所有副本进行一轮消息交流（图2中Fast path阶段）。期间，C 的领导者将 C 和本地与 C 相关的命令集合一起发送，副本回复时包含本地与 C 相关的命令集合。若 C 的领导者收到 $F+\left\lfloor\frac{F+1}{2}\right\rfloor$ 个（称为Fast path阶段法定人数）相同的回复，发送确认消息给客户端和所有副本，所有副本本地提交 C。否则，C 的领导者和所有副本再进行一轮消息交流（图2中Slow path阶段）。若 C 的领导者收到 $F+1$ 个（称为Slow path阶段法定人数）副本的回复，发送确认消息给客户端和所有副本，所有副本本地提交命令 C。

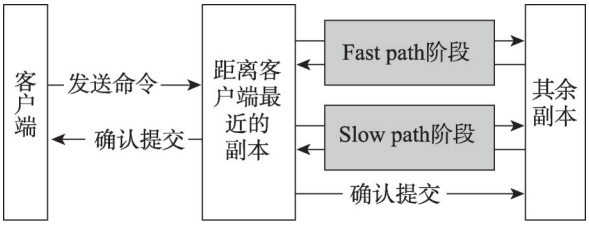

Fig.2 EPaxos process

图2 EPaxos处理过程

2.3 Multi-Paxos和EPaxos对比分析

由于Multi-Paxos中每个领导者只执行一轮Prepare消息，可忽略不计，故客户端发送命令到收到回复大概需要经过4次消息交流（图1中发送命令，Accept阶段，确认提交）；对于EPaxos，若Fast path阶段领导者收到法定人数副本相同的回复，则客户端发送命令到收到回复只需经过4次消息交流（图2中发送命令，Fast path阶段，确认提交）；否则，还需进行一轮Slow path，需要6次信息交流。

EPaxos客户端将命令发送给最近的副本，而Multi-Paxos客户端将命令发送给指定的领导者，因此，EPaxos在4次消息交流提交命令的情况下，延迟性能优于Multi-Paxos。而EPaxos在6次消息交流提交命令的情况下，延迟性能通常劣于Multi-Paxos。

3 MEPaxos算法

MEPaxos适用于非拜占庭故障下的异步分布式系统，是一种低延迟的共识算法。观察到EPaxos存在需要多执行一轮Slow path才可提交命令，延迟增加的情况，MEPaxos将EPaxos与Multi-Paxos结合。首先，MEPaxos提出了系统平均延迟的计算方法，并对二阶段提交算法进行改进。接着，每隔时间段 t，计算EPaxos和Multi-Paxos系统平均延迟。根据计算结果，利用改进的二阶段提交算法自动转换到系统平均延迟较小的算法模式，以获得最优的延迟性能。

3.1 命令冲突

为了方便描述EPaxos需要执行一轮Slow path才可提交命令，延迟增加的情况，本文提出命令冲突的概念。

定义（命令冲突）如果 q 个相关命令（command interference）[17] $a_1,a_2,\cdots,a_q$ 同时被提出，且EPaxos在处理命令 $a_i(i\in[1,q])$ 时，受到其余相关命令 $a_{k1},a_{k2},\cdots,a_{kn}(k1,k2,\cdots,kn\in[1,q])$ 的影响，多执行一轮Slow path才可提交，那么说命令 a_i 和命令 $a_{k1},a_{k2},\cdots,a_{kn}$ 是冲突的。

由命令冲突的定义和EPaxos处理步骤[17]可知，EPaxos中，并发客户端向同一副本提交相关命令，命令之间不会产生冲突。只有不同副本处理相关命令时，命令之间才可能产生冲突。不同副本处的并发客户端提出具有相关性的命令越多，命令冲突也就越多。本文通过系统平均延迟的计算，利用转换算法自动对不同命令冲突下的情况做出反应，以取得更优的延迟性能。

3.2 系统平均延迟

MEPaxos设计的主要目标是最小化客户端感知到的延迟（本文所指的延迟均为提交延迟[17]），给客户端带来更好的用户体验。本节考虑整个系统响应客户端的平均延迟，提出系统平均延迟的计算方法。

在MEPaxos中，共有 N 个副本（ $N=2F+1$ ，其中 F 是能容忍的副本最大故障数）。用 R_i 表示第 i 个副本（$i\in[1,N]$）；t_{ri} 表示从 R_r 到 R_i 所需时间；副本 t_{ci} 表示消息从客户端到 R_i 所需的时间。由于客户端和最近的副本进行交流，因此和同一副本进行交流的客户端到该副本所需的时间大致相同，这里不进行区分。

系统平均延迟的计算，主要考虑三个因素：

（1）算法运行过程中客户端的负载情况，可用算法运行过程中副本处理客户端提交的命令数表示。

（2）算法运行过程中并发客户端的命令冲突情况，可用副本作为领导者执行Slow path的命令数表示。

（3）算法运行过程中网络的实时情况，可用客户端与副本以及副本与副本之间消息传输所需时间表示。

故EPaxos模式下，系统平均延迟 $ELat$ 可表示为：

$$ELat=\frac{\sum_{i=1}^{N}[(T_i-SC_i)\times(2t_{ci}+2k\min(i))+SC_i\times(2t_{ci}+2k\min(i)+2p\min(i))]}{\sum_{i=1}^{N}T_i} \quad (1)$$

其中，T_i 表示 R_i 处理客户端提交的命令总数；SC_i 表示 R_i 作为领导者执行Slow path的命令数；$k\min(i)$ 表示对于 $r\in[1,N]$，第 k 小的 t_{ri}，其中 k 表示Fast path中的法定人数；$p\min(i)$ 表示对于 $r\in[1,N]$，第 p 小的 t_{ri}，其中 p 表示Slow path中的法定人数。以上数据每隔时间段 t 统计得出。

Multi-Paxos模式下，系统平均延迟 $MLat$ 可表示为：

$$MLat=\frac{\sum_{i=1}^{N}[T_i\times(2t_{ci}+2t_{il}+2p\min(l))]+2p\min(l)}{\sum_{i=1}^{N}T_i} \quad (2)$$

其中，T_i 表示 R_i 处理客户端提交的命令总数；t_{il} 表示从副本 R_i 到领导者 R_l 的时间；$p\min(l)$ 表示对于 $r\in[1,N]$，第 p 小的 t_{rl}（EPaxos中Slow path的法定人数和Multi-Paxos中法定人数相同）。以上数据每隔时间段 t 统计得出。

3.3 转换算法

MEPaxos在二阶段提交算法的基础上引入超时机制，提出了EPaxos和Multi-Paxos之间的转换算法，具体步骤如下：

（1）算法转换的发起者 RI 给各个副本 $R_i(i\in[1,N])$ 发送更改算法的通知并设置超时时间 TO。

（2）副本 R_i 确认收到通知，发送确认消息以及本地信息 LM_i 给 RI，进入转化状态（副本在转化状态时不会分配新实例，即：对于收到的客户端命令，放入缓存，在非转化状态时再进行处理），同样设置超时时间 TO。

（3）若 RI 在 TO 到达之前收到所有副本的确认消息以及本地信息 LM_i，则对 LM_i 进行统计，统计结果记为 $CLM=\{LM_i|i\in[1,N]\}$，发送 CLM 以及算法转换的执行命令给所有副本；否则，发送算法转换的取消命令给所有副本。

（4）若副本 R_i 在 TO 到达之前收到 CLM 以及算法转换的执行命令，根据 CLM 执行相关命令 $RCOM$。之后跳出转化状态，进入要转换的算法模式。否则，副本 R_i 跳出转化状态，返回之前算法模式（副本在一种算法模式下，不会响应另一种算法模式下的任何消息）。

其中，LM_i 和 $RCOM$ 根据MEPaxos转换模式的不同，也有所不同，具体如表1所示。

3.4 MEPaxos算法步骤

MEPaxos详细步骤如下：

（1）MEPaxos进入EPaxos算法模式，处理客户端

Table 1 Different LM_i and $RCOM$ under different change modes

表1 LM_i 和 $RCOM$ 在不同转换模式下的选取

转换模式	EPaxos转换到Multi-Paxos	Multi-Paxos转换到EPaxos
LM_i	R_i 开始处理的最大实例 I_i	若 R_i 为领导者，为 R_i 开始处理的最大实例 I_i；否则为空
$RCOM$	执行各个副本 $R_r(r\in[1,N])$ 所有提交的实例（最大提交实例为 I_r）	执行领导者所有提交的实例（最大提交实例为 I_i）

提交的命令。

（2）每隔时间段 t，对式（1）所需数据进行统计，计算出 $ELat$，并估计 $MLat$。估计方法如下：

（2.1）假设 $R_i(i\in[1,N])$ 为领导者，根据式（2）计算出系统平均延迟 $RMLat_i$；

（2.2）取 $MLat=\min(\{RMLat_i|i\in[1,N]\})$，领导者为此时的 R_i。

（3）若 $MLat<ELat$，执行转换算法，MEPaxos进入Multi-Paxos算法模式处理客户端提交的命令，转至步骤（4）执行；否则，转至步骤（2）继续执行。

（4）每隔时间段 t，对式（2）所需数据进行统计，计算出 $MLat$，并估计 $ELat$。估计方法如下：

（4.1）统计时间段 t 内 $R_i(i\in[1,N])$ 处理客户端提交的命令中对关键字 K_m（$m\in[1,M]$，M 表示关键字总数）操作的命令数 CK_{im}；

（4.2）则 R_i 处理的对关键字为 K_m 操作的命令中执行Slow path的命令数 SC_{im} 为：

$$SC_{im}=\begin{cases}\min(CK_{im},\sum_{r\in[1,N],r\neq i}CK_{rm}),\ CK_{im}=\max(\{CK_{im}|i\in[1,N]\})\\ CK_{im},\ CK_{im}\neq\max(\{CK_{im}|i\in[1,N]\})\end{cases}$$

（4.3）时间段 t 内 R_i 执行Slow path的命令总数 $SC_i=\sum_m SC_{im}$；

（4.4）根据统计的数据以及 SC_i，按照式（1）算出 $ELat$。

（5）若 $ELat\leqslant MLat$，执行转换算法，MEPaxos进入EPaxos算法模式处理客户端提交的命令，转至步骤（2）执行；否则，转至步骤（4）继续执行。

3.5 MEPaxos性能分析

由MEPaxos算法步骤可知，为实现最优化延迟性能的目标，MEPaxos主要进行了以下改进：

（1）算法运行过程中进行系统平均延迟的计算。

（2）根据改进（1）计算结果，利用转换算法转换至系统平均延迟较小的算法模式。

在共识算法实际应用中，副本之间通常需要定期发送心跳信息进行交流[18]。计算系统平均延迟所需数据可附带在心跳信息中一起发送，所需计算资源对整个系统来说可忽略不计，因此改进（1）只需少量可忽略不计的计算资源。

转换算法的引入，使MEPaxos可转换至系统平均延迟较小的算法模式。虽增加了算法的处理过程，但转换后系统的延迟性能更优。

MEPaxos做出了增加少量可忽略不计的计算资源以及算法处理过程的权衡，以实现更优的系统延迟性能。

4 MEPaxos算法保证及其证明

和Multi-Paxos、EPaxos类似，MEPaxos提供以下算法保证：非平凡性（nontriviality）、一致性（consistency）以及进展保证（progress guarantee）。本章将对三种算法保证进行详细介绍并证明MEPaxos可提供这三种算法保证。

4.1 变量说明

在MEPaxos，定义如下表示：T 表示MEPaxos算法从开始运行到结束运行之间的时间；$Mode_{ts}$ 表示任意时刻 ts，MEPaxos算法所处模式；$Mode_{bef}$ 表示MEPaxos算法转换前的模式；CM表示算法转换模式；EM表示EPaxos模式；MM表示Multi-Paxos模式；$Command$ 表示客户端提交命令的集合；$CT(Command_i)$ 为判定命令 $Command_i$ 是否被提交的函数，若是，为true，否则为false；Mo_i 表示命令 $Command_i$ 以模式 Mo_i 被提交；$Nontri(Mode_{ts})$ 为判定模式 $Mode_{ts}$ 是否满足非平凡性的函数，若是，为true，否则为false；$Consis(Mode_{ts})$ 为判定模式 $Mode_{ts}$ 是否满足一致性的函数，若是，为true，否则为false；$Process(Mode_{ts})$ 为判定模式 $Mode_{ts}$ 是否满足进程保证

的函数，若是，为true，否则为false。

4.2 算法保证描述及证明

根据MEPaxos算法步骤，以下公式成立：

$$Mode_{bef} \in \{EM, MM\} \tag{3}$$

$$Mode_{ts} \in \{CM, EM, MM\}, ts \in T \tag{4}$$

$$(Mode_{ts} = CM) \wedge (\exists Command_i \in Command, CT(Command_i) = true) \rightarrow Mo_i = Mode_{bef} \tag{5}$$

根据文献[9]和文献[17]，以下公式成立：

$$Nontri(EM) = true \tag{6}$$

$$Nontri(MM) = true \tag{7}$$

$$Consis(EM) = true \tag{8}$$

$$Consis(MM) = true \tag{9}$$

$$Process(EM) = true \tag{10}$$

$$Process(MM) = true \tag{11}$$

下面将详细介绍三种算法保证并在以上公式的基础上，证明MEPaxos算法非平凡性、一致性以及进展保证。

非平凡性 只有客户端提出的命令才能被提交。

证明 要证明非平凡性，只需证明以下公式成立：

$$Nontri(Mode_{ts}) = true, ts \in T \tag{12}$$

由式(4)、式(6)、式(7)知，要证明式(12)成立，只需证明：

$$Nontri(CM) = true \tag{13}$$

由式(3)、式(5)、式(6)、式(7)知，式(13)成立。因此，非平凡性满足。 □

一致性 对于同一实例(instance)，最多只有一个命令被提交，即对于同一实例，如果副本 R_p 和副本 R_q 分别提交了命令 C_p 和 C_q，则 C_p 和 C_q 相同。

证明 要证明一致性，只需证明以下公式成立：

$$Consis(Mode_{ts}) = true, ts \in T \tag{14}$$

由式(4)、式(8)、式(9)知，要证明式(14)成立，只需证明：

$$Consis(CM) = true \tag{15}$$

由式(3)、式(5)、式(8)、式(9)知，式(15)成立。因此，一致性满足。 □

进程保证 在大多数副本没有故障并且在接收者超时之前消息能送达的情况下，客户端的命令最终会被非故障副本提交。

证明 要证明进程保证，只需证明以下公式成立：

$$Process(Mode_{ts}) = true, ts \in T \tag{16}$$

由式(4)、式(10)、式(11)知，要证明式(16)成立，只需证明：

$$Process(CM) = true \tag{17}$$

在CM期间，若存在少数副本故障且在接收者超时之前消息能送达的情况下，根据转换算法步骤，MEPaxos会跳出转化状态，返回 $Mode_{bef}$，根据式(3)、式(10)、式(11)知，此时式(17)成立。

在CM期间，若无副本故障且在接收者超时之前消息能送达的情况下，根据转换算法步骤，MEPaxos会进入EM或MM模式，根据式(10)、式(11)知，此时式(17)成立。

故式(17)成立。因此，进程保证满足。 □

5 实验与分析

5.1 实验环境与参数设置

本文实验运行在亚马逊弹性计算云(elastic compute cloud，EC2)平台，采用实例配置如下：1个2.5 GHz的vCPU，内存为1 GB，存储空间为8 GB，网络性能低到中等，64位ubuntu16.04操作系统。采用go语言(1.6.2版本)实现MEPaxos，并将其与EPaxos和Multi-Paxos进行对比分析。

实验采用节俭模式[17]：在节俭模式下，副本将消息发送给法定人数副本，而不是全部副本。

参数设置：

(1)时间段 t。t 越大，计算EPaxos和Multi-Paxos系统平均延迟所耗费的副本计算能力以及网络带宽等资源越少，算法的稳定性越好，但算法的灵敏度越低。在实际生产实践中，t 的选取可根据可使用资源的多少，客户端命令冲突变化情况以及对算法的灵敏度需求决定。本文实验中 t 设置为15 s。

(2)超时时间 TO。TO 越小，算法转换所需的最大时间越小，客户端等待的时间越少，但算法转换失败的概率越大。在实际生产实践中，TO 的选取可根据客户端负载情况，所能容忍的算法转换时间以及所需的算法转换成功率确定。本文实验中 TO 设置为1 s。

5.2 均衡负载下的延迟

本节在副本数为3和5，客户端负载均衡的情况下，分别比较MEPaxos、EPaxos和Multi-Paxos的延迟性能。当副本数为3时，将3个副本部署在加利福尼亚北部（California，CA），弗吉尼亚北部（Virginia，VA）和爱尔兰（Ireland，IE）；当副本数为5时，另外两个副本部署在俄勒冈（Oregon，OR）和东京（Tokyo，TKY）。Multi-Paxos的领导者一直为CA处的副本。在每个副本实例上同时也设置了10个客户端，它们同时发送相同数量的命令（客户端发送命令时，收到前一条命令的回复之后才会发送后一条命令）。由于命令冲突只在不同副本处理相关命令时才会发生，因此副本实例上客户端的数量对体现3种共识算法延迟性能差异影响不大，这里设置为10个客户端。图3和图4分别记录了各副本处的客户端从发送命令到收到回复感知到的中位数和99%ile延迟（算法后的百分比表示具有相关性的命令所占的百分比）。

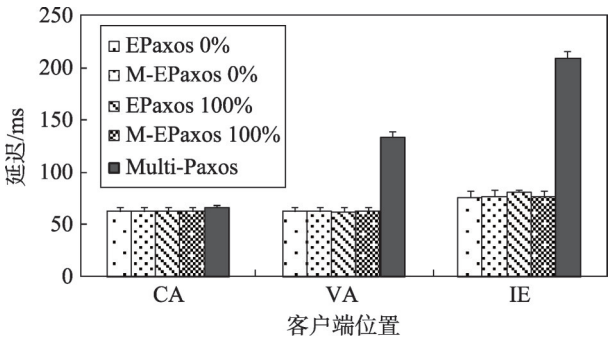

Fig.3 Median commit latency (99%ile indicated by lines atop bars) perceived by clients at each replica when *N* is 3

图3 *N*=3时各副本处客户端感知到的延迟的中位数（顶部的线表示99%ile）

副本数为3时，无论不同副本处并发客户端发送的相关命令所占百分比为多少，EPaxos模式下，相关命令都不会产生冲突（节俭模式导致，具体原因详见参考文献[17]）。此时，EPaxos的延迟性能要优于Multi-Paxos延迟性能。根据系统平均延迟计算结果，MEPaxos一直执行EPaxos模式。因此图3中各副本处，EPaxos延迟性能不受相关命令所占百分比的影响，优于Multi-Paxos延迟性能。MEPaxos客户端感知到的延迟和EPaxos客户端感知到的延迟大致相同，小于Multi-Paxos客户端感知到的延迟。

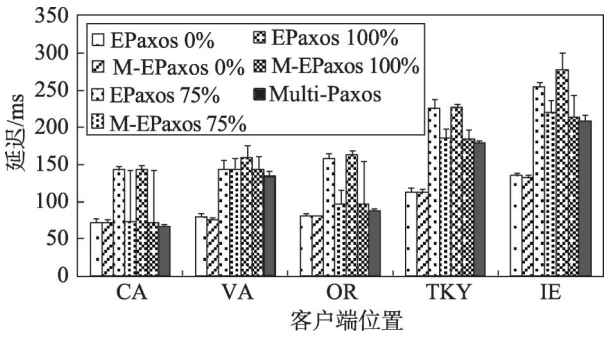

Fig.4 Median commit latency (99%ile indicated by lines atop bars) perceived by clients at each replica when *N* is 5

图4 *N*=5时各副本处客户端感知到的延迟的中位数（顶部的线表示99%ile）

副本数大于3时，随着不同副本处并发客户端发送的相关命令所占百分比的增大，EPaxos模式下，命令冲突数也增加，延迟性能变差。而Multi-Paxos延迟性能不受并发客户端相关命令百分比的影响。此时MEPaxos根据 *ELat* 和 *MLat* 的计算结果，利用转换算法自动地转换至系统平均延迟较小的算法模式。故在图4各副本处，EPaxos随着相关命令所占百分比的增大，延迟性能变差，Multi-Paxos不受相关命令所占百分比的影响。当相关命令所占百分比为0时，MEPaxos客户端感知到的延迟和EPaxos客户端感知到的延迟大致相同，小于Multi-Paxos客户端感知到的延迟；当相关命令所占百分比为75%和100%时，MEPaxos客户端感知到的延迟和Multi-Paxos客户端感知到的延迟大致相同，小于EPaxos客户端感知到的延迟。副本数大于5的情况和副本数为5的情况类似，这里不再赘述。

由以上实验结果可以看出，在不同副本处并发客户端发送的相关命令所占百分比增大时，EPaxos算法延迟性能显著下降。而MEPaxos能根据系统平均延迟，自动转换到系统平均延迟较小的算法模式，延迟性能要优于EPaxos。

5.3 不均衡负载下的延迟

为了测试在客户端负载不均衡情况下MEPaxos的性能，在5处CA、VA、IE、OR和TKY分别部署了1

个副本;Multi-Paxos的领导者依然为CA处的副本;副本数为3时,MEPaxos一直执行EPaxos算法模式,不进行算法转换,这里不再赘述。在OR和IE处的副本实例上同时设置了10个客户端,它们同时发送相同数量的命令(客户端发送命令时,收到前一条命令的回复之后才会发送后一条命令)。图5记录了OR和IE处的客户端从发送命令到收到回复感知到的中位数和99%ile延迟(算法后的百分比表示具有相关性的命令所占的百分比)。

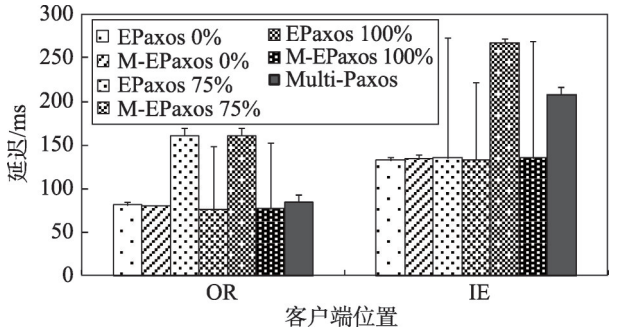

Fig.5 Median commit latency (99%ile indicated by lines atop bars) perceived by clients at OR and IE when *N* is 5

图5 *N*=5时OR和IE处客户端感知到的延迟的中位数(顶部的线表示99%ile)

如图5,当只在OR和IE处的实例部署客户端时,EPaxos随着不同副本处并发客户端相关命令百分比的增大,命令冲突数增加,延迟性能变差。Multi-Paxos延迟性能不受并发客户端相关命令百分比的影响。在相关命令所占百分比为0时,EPaxos延迟性能优于Multi-Paxos,MEPaxos一直执行EPaxos模式,和EPaxos延迟性能大致相同。在相关命令所占百分比为75%和100%时,MEPaxos根据 *ELat* 和 *MLat* 的计算结果,自动转换至系统平均延迟较小的算法模式。在转换至Multi-Paxos算法模式时,根据MEPaxos算法的步骤(2),会选择使系统平均延迟最小的副本(OR处副本)担任领导者,故此时,MEPaxos客户端感知到的延迟要小于Multi-Paxos客户端感知到的延迟。副本数大于5的情况和副本数为5的情况类似,这里不再赘述。

以上实验在客户端负载不均衡时,进行EPaxos、Multi-Paxos和MEPaxos算法的对比。实验结果表明,在客户端负载不均衡时,MEPaxos始终能选择系统平均延迟最小的算法模式,确保了算法的系统平均延迟最优。同时,转换至Multi-Paxos算法模式时,能自动选择最适合当前客户端负载情况的领导者,延迟性能要优于Multi-Paxos。

6 结束语

本文基于低延迟的设计目标,提出了共识算法MEPaxos。MEPaxos综合考虑算法运行过程中客户端的负载情况、并发客户端的命令冲突情况、网络的实时情况,提出了系统平均延迟的计算方法,并引入超时机制对二阶段提交算法进行改进。MEPaxos可根据系统平均延迟计算结果,利用改进的二阶段提交算法自动选择平均延迟较小的算法模式。实验表明,MEPaxos比当前算法具有更好的延迟性能。由于客户端环境的多样性,未来的研究重点将放在进一步有效加强MEPaxos算法在各种客户端环境下的稳定性,并将其应用到实际生产实践中。

References:

[1] Kleppmann M. Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems[M]. Sebastopol: O'Reilly Media Press, 2017.

[2] Burrows M. The chubby lock service for loosely-coupled distributed systems[C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation, Seattle, Nov 6-8, 2006. Berkeley: USENIX Association, 2006: 335-350.

[3] Ghemawat S, Gobioff H, Leung S T. The Google file system [C]//Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, Oct 19-22, 2003. New York: ACM, 2003: 29-43.

[4] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems, 2013, 31(3): 8.

[5] Hunt P, Konar M, Junqueira F P, et al. ZooKeepe: wait-free coordination for Internet-scale systems[C]//Proceedings of the USENIX Annual Technical Conference, Boston, Jun 23-25, 2010. Berkeley: USENIX Association, 2010: 9.

[6] DeCandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store[J]. ACM SIGOPS Operating Systems Review, 2007, 41(6): 205-220.

[7] Kendall S C, Waldo J, Wollrath A, et al. A note on distributed computing[J]. IEEE Micro, 1997, 1222: 49-64.

[8] Fischer M J, Lynch N A, Paterson M S. Impossibility of distributed consensus with one faulty process[J]. Journal of the ACM, 1985, 32(2): 374-382.

[9] Lamport L. The part-time parliament[J]. ACM Transactions on Computer Systems, 1998, 16(2): 133-169.

[10] Lamport L. Paxos made simple[J]. ACM Sigact News, 2001, 32(4): 18-25.

[11] Van Renesse R, Altinbuken D. Paxos made moderately complex[J]. ACM Computing Surveys, 2015, 47(3): 42.

[12] Lamport L, Shostak R, Pease M. The Byzantine generals problem[J]. ACM Transactions on Programming Languages and Systems, 1982, 4(3): 382-401.

[13] Lamport L. Fast paxos[J]. Distributed Computing, 2006, 19 (2): 79-103.

[14] Lamport L B. Generalized paxos: 7698465[P].2010-04-13.

[15] Mao Y, Junqueira F P, Marzullo K. Mencius: building efficient replicated state machines for WANs[C]//Proceedings of the USENIX Symposium on Operating Systems Design and Implementation, San Diego, Dec 7-11, 2008. Berkeley: USENIX Association, 2008: 369-384.

[16] Biely M, Milosevic Z, Santos N, et al. S-paxos: offloading the leader for high throughput state machine replication[C]// Proceedings of the 31st Symposium on Reliable Distributed Systems, Irvine, Oct 8-11, 2012. Washington: IEEE Computer Society, 2012: 111-120.

[17] Moraru I, Andersen D G, Kaminsky M. There is more consensus in egalitarian parliaments[C]//Proceedings of the ACM SIGOPS 24th Symposium on Operating Systems Principles, Farmington, Nov 3-6, 2013. New York: ACM, 2013: 358-372.

[18] Ongaro D, Ousterhout J K. In search of an understandable consensus algorithm[C]//Proceedings of the USENIX Annual Technical Conference, Philadelphia, Jun 19-20, 2014. Berkeley: USENIX Association, 2014: 305-319.

ZHAO Shouyue was born in 1995. She is an M.S. candidate at Jiangnan University, and the member of CCF. Her research interests include distributed computing and data mining.

赵守月(1995—),女,安徽六安人,江南大学硕士研究生,CCF学生会员,主要研究领域为分布式计算,数据挖掘。

GE Hongwei was born in 1967. He received the Ph.D. degree in control engineering from Jiangnan University in 2008. Now he is a professor and Ph.D. supervisor at School of Internet of Things Engineering, Jiangnan University. His research interests include artificial intelligence and pattern recognition, machine learning, image processing and analysis, etc.

葛洪伟(1967—),男,江苏无锡人,2008年于江南大学信息学院获得博士学位,现为江南大学物联网工程学院教授、博士生导师,主要研究领域为人工智能和模式识别,机器学习,图像处理与分析等。