Harp: 面向跨空间域的分布式事务优化算法

庄琪钰[1,2]，李彤[1,2]，卢卫[1,2]，杜小勇[1,2]
1. 数据工程与知识工程教育部重点实验室，北京 100872；
2. 中国人民大学信息学院，北京 100872

摘要
近数据计算范式驱动了银行、券商在全国或全球范围内建设多个数据中心。在传统的业务模式中，事务聚焦单个数据中心的数据访问。随着业务模式的变化，跨数据中心的分布式事务成为常态，例如，银行账户之间的转账、游戏账户之间的装备交换等，而这些账户的数据存储在不同区域的数据中心上。分布式事务处理需要两阶段提交协议来保证各参与节点子事务提交的原子性。在跨空间域场景下，节点之间的网络时延更长且存在差异性，传统的事务处理技术需要拓展，以保证系统能够提供较高的吞吐量。在分析了跨域事务存在的问题和优化空间后，提出了一种新的分布式事务处理算法Harp。Harp在保证可串行化隔离级别的前提下，根据网络时延的差异，将部分子事务延迟执行，减少了事务的锁争用时长，提升系统并发度和吞吐量。实验表明，在YCSB负载下，Harp的性能比传统算法提升了1.39倍。

关键词
跨空间域分布式事务；网络差异；事务调度；锁争用

中图分类号：TP315 文献标志码：A doi: 10.11959/j.issn.2096-0271.2023043

Harp: optimization algorithm for cross-domain distributed transactions

ZHUANG Qiyu[1,2], LI Tong[1,2], LU Wei[1,2], DU Xiaoyong[1,2]
1. Key Laboratory of Data Engineering and Knowledge Engineering, Beijing 100872, China
2. School of Information, Renmin University of China, Beijing 100872, China

Abstract
The paradigm of near-data computing has driven banks and securities firms to build multiple data centers globally or nationally. In the traditional business model, transactions focused on accessing data within a single data center. With the changing business model, distributed transactions across data centers have become common, such as transferring money between bank accounts or exchanging equipment between game accounts, with data stored in different data centers in different regions. Distributed transaction processing requires the two-phase commit protocol to ensure the

atomicity of the sub-transactions submitted by each participating node. In processing cross-domain transactions, traditional transaction processing technology needs to be expanded to ensure that the system can provide higher throughput due to the longer and more varied network latency between nodes. After analyzing the problems and optimizing space for cross-domain distributed transactions, this paper proposes a new distributed transaction processing algorithm called Harp. Harp delays the execution of some sub-transactions based on the difference in network latency while ensuring serializable isolation level, reducing the duration of transaction lock contention, and improving system concurrency and throughput. Experiments show that Harp improves the performance by 1.39 times compared with the traditional algorithm under YCSB workload.

Key words
cross-domain distributed transaction, network difference, transaction scheduling, lock contention

0 引言

2022年年初，国家完成了全国一体化大数据中心体系总体布局设计，正式启动“东数西算”工程，在京津冀、长三角、粤港澳等八大区域部署国家算力枢纽节点，建设全国一体化算力网络。数据管理正在从面向或限定于单空间域（单一数据中心）的孤立服务发展到跨空间域（跨数据中心）的协同服务阶段[1]。分布式数据库因其高可扩展、高可用以及低成本等特点，在大规模数据的管理和分析中有着广泛应用[2-3]。为了确保近数据计算，银行、电商、云计算厂商等往往会建设多个数据中心，例如，亚马逊建设有38个数据中心，苹果建设有11个数据中心（6个在美国，2个在丹麦，3个在中国）。通过这种方式，可以将用户的数据存储在最近的数据中心，当用户发起事务的读写操作仅局限于其个人数据时，近数据计算可以保证该事务具备较高的性能。然而，跨域交易打破了近距离计算的假设。跨域交易指的是一个事务涉及处在两个不同区域的用户发生的交易。例如，一个位于广东的用户需要转账10 000元给一个位于北京的用户，由于两个用户的数据存放在不同的数据中心，需要对这两个用户之间的数据进行协调。

在处理跨空间域分布式事务时，数据库的并发控制将会遇到新的挑战。两阶段封锁（two-phase lock，2PL）协议[4]是分布式数据库并发控制的经典协议，影响该协议性能的重要指标是锁争用时长（contention span）[5]。锁争用时长指的是从事务在执行阶段获取数据项上的锁到事务释放锁的这段时间。在网络时延的影响下，分布式事务的锁争用时长将远大于单机事务，这会限制系统的并发度并且增加系统的回滚率。在跨空间域场景下，分布式事务可能会涉及不同数据中心的多个节点，它们之间的网络时延更长，同时差异性更大。分布式事务的执行通常需要等待最“慢”的子事务（子事务所在参与者节点和协调者节点间的网络时延较大）执行完成后才可以被提交，这会导致锁争用时长进一步变长，给跨空间域分布式事务处理带来了较大的挑战。

为了在保证正确性前提下，最小化跨空间域分布式事务的锁争用时长，本文提出了新的事务处理算法Harp。Harp遵循两阶段封锁协议，因此可以保证事务的正确性。此外，Harp充分考虑了网络差异性，根据实时评估的网络时延调整子事务的执行时刻，推迟“快”子事务的执行时刻（子事务所在参与者节点和协调者节点间的网

络时延较小），从而减少“快”子事务因等待“慢”子事务产生的不必要的锁争用，提高锁资源的利用率，提升系统并发度和吞吐量。实验表明，在某些场景下，Harp的吞吐量能达到传统算法的2.39倍，同时回滚率下降了32%。

1 从单空间域分布式事务到跨空间域分布式事务

1.1 单空间域分布式事务

事务是用户定义的一组数据库操作组成的序列，是数据库管理系统中的最小执行单元[6]。分布式事务是指在事务中包含对不同节点上的数据项的操作，当服务端收到一个分布式事务时，通常将其交由一个协调者来负责。

在执行阶段，协调者将事务拆分成多个子事务并分发给各个参与者，参与者根据并发控制算法执行事务。例如，如果采用2PL作为并发控制协议，参与者将在执行阶段对数据项进行加锁，这适用于高冲突场景，但会降低系统的并发度，并且需要考虑死锁问题。如果采用乐观并发控制（optimistic concurrency control，OCC）协议[7]，参与者在执行阶段不需要对数据项加锁，在提交前对数据项进行验证，这适用于低冲突场景。在执行阶段结束后，分布式数据库通常采用两阶段提交（two-phase commit，2PC）协议[8]来保证数据库的一致性。2PC协议将事务提交分成了两个阶段。在准备阶段中，协调者收集各参与者的执行状态，并根据各子事务的执行结果确定事务的最终状态。如果所有参与者都可以提交事务，那么协调者将事务状态设置为“提交”（commit）并通知各参与者；任何一个参与者无法提交，协调者都会将事务状态设置为“回滚”（rollback），同时通知所有的参与者将子事务回滚。

接下来，用实例更直观地介绍数据中心内分布式事务的执行方式。例如，事务T_1可以被划分成3个子事务T_{11}~T_{13}，分别访问分区N_1~N_3，同时N_1作为T_1的协调者。如果使用2PL协议和2PC协议，那么事务的执行情况如**图1**所示，T_1的子事务T_{12}和T_{13}，在执行阶段经过1个往返时延（round trip time，RTT）后执行完毕。又经过了1个RTT的准备阶段，协调者获得了各子事务的执行结果。在提交阶段，协调者根据各子事务的执行结果确定事务的最终状态（提交/回滚），将事务的最终状态发给参与者。参与者在收到结束信息后释放锁。因为$N_1 \rightarrow N_2$和$N_1 \rightarrow N_3$的网络差异较小，所以T_{12}和T_{13}的时间节点基本是同步的。在**图1**中，⟼ 表示数据项上的锁争用时长，在这段争用时间内，其他想对这个数据项进行操作并且锁的类型冲突的事务，都会被阻塞或者回滚。

1.2 跨空间域分布式事务

近年来，“两地三中心”“三地五中心”等概念被提出，这意味着数据库将会处理越来越多的跨空间域分布式事务。**图2**给出了“三地五中心”的一种示例，在纽约和北京分别部署了两个数据中心，在伦敦部署了一个数据中心，这样的架构设计满足了大规模数据增长的需求，同时保证了更高的可用性。

在分布式事务处理中，需要引入多轮的网络调度，包括两阶段提交、副本同步等。从数据中心内到跨数据中心的转变，地域间的距离决定了跨空间域网络传输具有较高的基础时延。在数据中心内部可以利用基于RDMA的高速网络技术（InfiniBand (IB)或RoCE v2）[9-10]实现稳

定低时延，而广域网数据的端到端传输和介质共享特性为数据传输时延带来了不确定性。如**图3**所示，在跨空间域网络中，通信时延通常是10 ms或更高，而数据中心内网络时延通常是微秒级，IB专用网络甚至可以达到十微秒级。相比数据中心内部，跨空间域节点之间的网络时延不可忽略，因此，跨空间域节点间的网络传输将成为制约数据库系统性能的瓶颈[1]。

考虑到跨空间域网络通信时延远大于数据中心内部，而分布式事务需要等待所有的子事务完成后才能确定事务最终的状态，那么跨空间域分布式事务相比于数据中心内的分布式事务，时延将进一步变长。在跨空间域场景下，**图1**中实例的执行情况将发生改变。如**图4**所示，假设$N_1 \rightarrow N_3$的RTT远大于$N_1 \rightarrow N_2$的RTT，在子事务T_{12}执行结束后，需要等待T_{13}返回结果后才能确定T_1的最终状态。而T_{13}的执行结果需要经过较长网络时延才能被N_1知晓，这给T_{11}和T_{12}引入了不必要的锁争用时长。在高争用场景下，被阻塞或者回滚的事务也会随之增多。

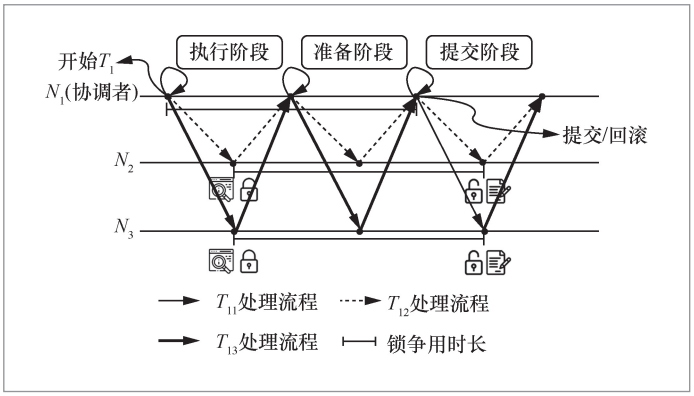


图 1 单空间域分布式事务执行流程

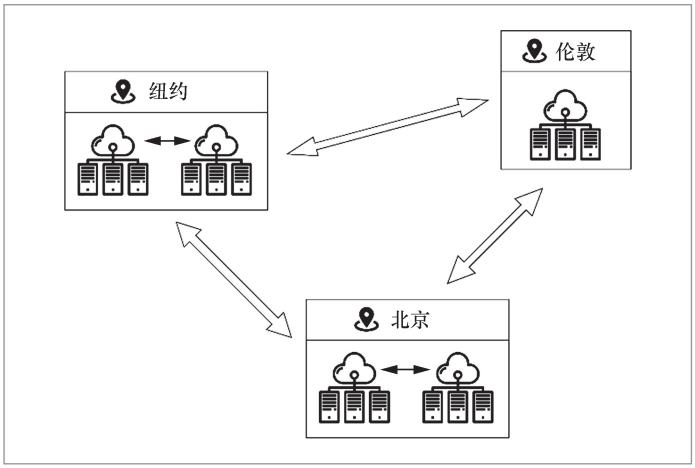


图 2 “三地五中心”示意图

2 跨空间域分布式事务的挑战

第1节以单事务场景为例，初步讨论了跨空间域场景下，网络差异对分布式事务的影响。接下来，将问题扩展到更通用的多事务场景，分析跨空间域分布式事务的问题和优化空间。

2.1 问题描述

分布式数据库将数据划分成不同的分区，为了保证近数据计算，用户数据通常被部署在距离用户较近的数据中心中。在第1节中，笔者介绍了分布式事务需要2PC

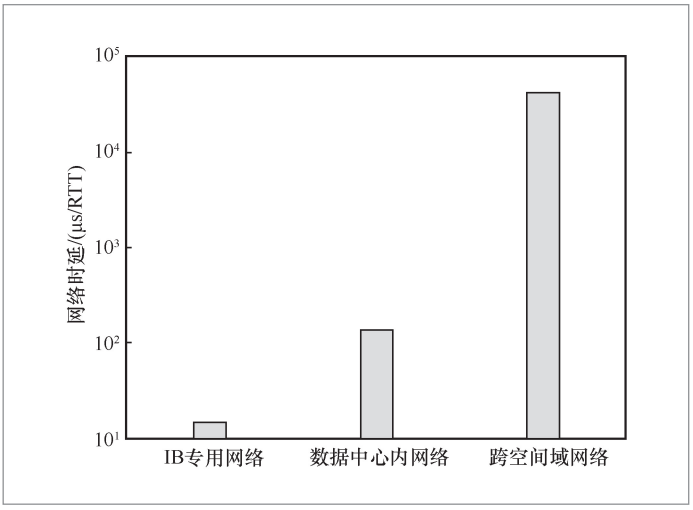


图 3 不同场景下网络时延

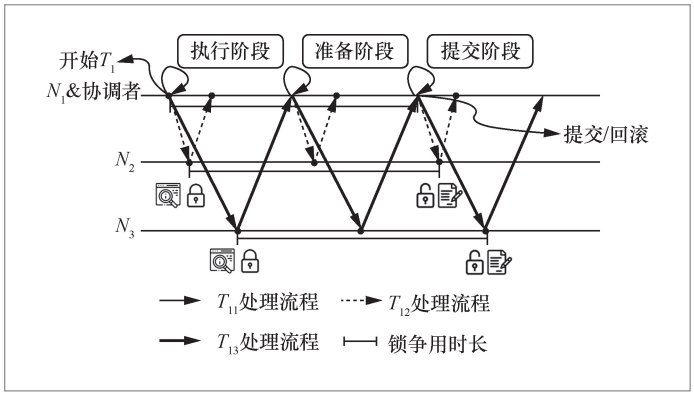


图 4 跨数据中心分布式事务执行流程

协议来保证事务的原子性和隔离性，在Early-Prepare[11]方法中，参与者节点在执行阶段后直接进入准备阶段，而无须等待协调者的通知消息。因此，锁争用时长可以从**图3**和**图4**中的2轮网络通信简化为1轮网络通信，即事务在执行阶段对数据项加锁并完成原先准备阶段的工作，在经过1轮网络通信并收到协调者的事务状态信息后，对数据项解锁并提交（回滚）事务。本文将在这种执行模型下对跨空间域分布式事务进行分析和优化。接下来，通过一个实例来更清晰地描述跨空间域分布式事务处理面临的问题。如**图5**所示，有3个数据库节点N_1、N_2和N_3，其中N_1与N_2间的网络时延为10 ms，N_1与N_3间的网络时延为30 ms，N_2与N_3间的网络时延为25 ms。数据项x和y位于N_1，数据项z位于N_2，数据项u和v位于N_3。

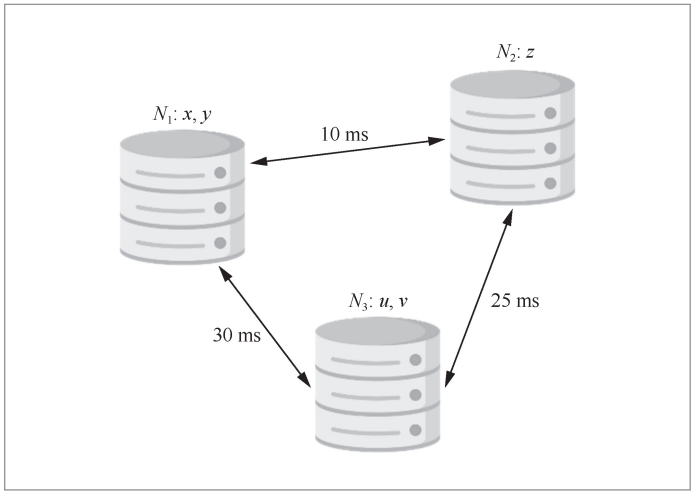


图 5 网络时延和数据分布

N_1作为协调者来处理来自客户端的4个事务请求，其中T_1涉及N_1~N_3 3个节点，其操作为$R_1(x)W_1(y)W_1(z)R_1(v)W_1(u)$；$T_2$和$T_3$涉及$N_1$和$N_2$两个节点，其中$T_2$的操作序列为$R_2(x)W_2(v)W_2(z)$；$T_3$的操作序列为$R_3(x)R_3(z)$；$T_4$为单机事务，其操作为$W_4(x)$。

如**图6**所示，数据库遵循2PL协议和2PC协议对上述实例进行调度。协调者按照操作涉及节点将T_1划分成子事务T_{11}~T_{12}，并将它们发送给对应参与者（消息1和消息2）。在执行T_2~T_4时，同样会将它们拆分成多个子事务并发送给参与者（为了图片更清晰，在图中没有画出T_2~T_4分发子事务的消息）。接下来，在执行T_{11}时，事务分别获得了数据项x和y上的共享锁和排他锁。同样地，T_{12}和T_{13}在执行时获取对应的锁，并将执行结果返回协调者（消息3和消息4）。接下来，在执行T_2~T_4时会发现，需要获取的数据项上的锁已经被T_1占有，并且锁的类型冲突，那么T_2~T_4会被T_1阻塞，直到T_1将锁释放。

经过60 ms后，协调者收到了T_{13}返回的执行结果（消息4），根据子事务执行结果确定T_1的最终状态，并向各参与者节点发送（消息5）。参与者N_2在10 ms后收到消息，在修改了T_1的事务状态后释放其获得的锁，并从锁的等待队列中取出T_{22}继续执行。在T_{22}执行完毕后，向N_1发送T_{22}的执行结果（消息6）。N_1在收到T_2所有子事务的执行结果后，可以确定T_2的最终状态并通知各参与者（消息7）。T_3和T_4的执行逻辑类似，可以看到，利用2PL协议和2PC协议进行事务调度时，T_1~T_4的执行时间为100 ms。根据锁争用时长的定义，T_1~T_4各数据项上锁争用时长之和分别为300 ms、60 ms、40 ms和0 ms。

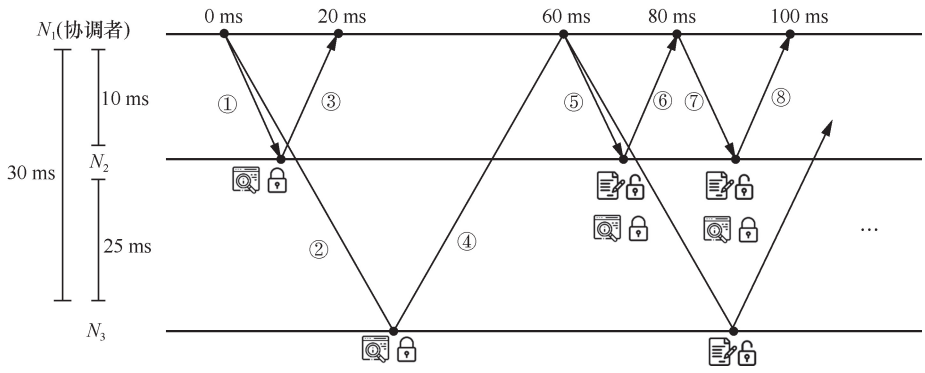


图 6 2PL 协议 +2PC 协议的执行逻辑

2.2 优化空间

如果不通过副本迁移[12-13]方法改变数据分布，分布式事务的锁争用时长至少为RTT（参与者在事务执行完毕后通知协调者，协调者在收到消息后可以随即确定事务的状态，并通知参与者释放锁）。在图6中，子事务T_{11}和T_{12}在执行结束后，需要等到T_{13}的执行结果后才能确定事务的最终状态。在这段时间里，与T_{11}和T_{12}在数据项上存在冲突的事务都将被阻塞或者回滚，这限制了系统的并发度。本文将锁争用时长等于网络RTT的情况称为“必要锁争用”；如果锁争用时长大于网络RTT，则认为其存在“无效锁争用”现象。

在理想情况下，所有子事务的锁争用时长都不存在“无效锁争用”，即事务在单个数据项上的锁争用时长等于对应网络的RTT。那么，事务T_1~T_4各数据项上锁争用时长之和分别为80 ms、20 ms、20 ms和0 ms。不难看出，这与实际场景中的执行情况差异较大。随着网络时延和差异性的增大，这种现象将会更加凸显。

在跨空间域分布式事务中，各参与者节点间的时延有较大差异，会出现大量的“无效锁争用”现象。为了缓解跨空间域分布式事务对系统性能的影响，本文在不影响正确性的前提下，提出了一种新的事务处理算法Harp，通过延迟子事务的执行，减少锁争用时长，提升数据库系统的吞吐量。

3 Harp：跨空间域分布式事务优化算法

3.1 算法式描述

在图6的实例中，子事务T_{11}和T_{12}分别在0 ms和20 ms时刻执行完毕，但是需要等到T_{13}返回结果才可以进入提交阶段，过早的执行T_{11}和T_{12}不会减少事务T_1的执行时延，反而会增加数据项上的争用时长。假设本地事务的执行时间远小于跨空间域节点间网络时延，通过延迟一部分子事务的执行，可以有效地减少事务的锁争用时长。但子事务的延迟时间并不是没有约束的，以T_1为例，如果T_{12}延迟到50 ms处执行，那么它将在70 ms返回N_1。在这种情况下，T_1事务状态的确定时间会从原先的

60 ms延长到70 ms，这是Harp不希望发生的。

Harp希望最大限度地减少事务的锁争用时长。对于事务而言，锁争用时长可以表示为：

$$\sum_{k_i \in T.ws \cup T.rs} \left[\mathrm{UnLock}(k_i) - \mathrm{Lock}(k_i) \right] \quad (1)$$

其中，k_i是T需要操作的数据项，$\mathrm{Lock}(k_i)$和$\mathrm{UnLock}(k_i)$分别是T对该数据项加锁和解锁的时刻。

此外，Harp对加锁逻辑进行了修改。在远程事务T_I的加锁过程中与当前持锁的事务T_J存在冲突时，如果T_J是本地事务，则T_I可以等待，否则将T_I回滚。由于本地事务的执行通常较快，在修改了加锁逻辑后，远程事务可以很快执行完毕，并将执行结果返回协调者。

根据2PC协议，在协调者收到所有子事务的执行结果后才可以确定事务的最终状态，即事务的执行时延取决于最慢的子事务。如果用$\tau_{c,j}$表示协调者N_c和参与者N_j之间的网络时延，根据Harp的加锁逻辑，T的执行时间可以近似表示为$2\cdot\max\{\tau_{c,j}\}$。用$\mathrm{DelayTime}(k_i)$表示k_i所在子事务t_i的延迟发送时间，那么$\mathrm{Lock}(k_i)$可以用$\mathrm{DelayTime}(k_i)+\tau_{c,\mathrm{node}(k_i)}$表示，而$\mathrm{UnLock}(k_i)$可以用$2\cdot\max\{\tau_{c,j}\}+\tau_{c,\mathrm{node}(k_i)}$表示。可以将式(1)改写为：

$$\sum_{k_i \in T.ws \cup T.rs} \left[2\cdot\max\{\tau_{c,j}\} - \mathrm{DelayTime}(k_i) \right] \quad (2)$$

此外，Harp不希望子事务延迟执行导致事务的执行时延大于$2\cdot\max\{\tau_{c,j}\}$，因此各子事务应在$2\cdot\max\{\tau_{c,j}\}$之前返回结果。结合式(2)，可以写出Harp的优化目标和约束方程：

$$\begin{cases} \min \sum_{k_i \in T.ws \cup T.rs} \left[2\cdot\max\{\tau_{c,j}\} - \mathrm{DelayTime}(k_i) \right] \\ \mathrm{DelayTime}(k_i) + 2\cdot\tau_{c,\mathrm{node}(k_i)} \leqslant 2\cdot\max\{\tau_{c,j}\} \end{cases} \quad (3)$$

由于$2\cdot\max\{\tau_{c,j}\}$是一个常量，可以将式(3)进一步改写为式(4)：

$$\begin{cases} \max \sum_{k_i \in T.ws \cup T.rs} \mathrm{DelayTime}(k_i) \\ \mathrm{DelayTime}(k_i) \leqslant 2\cdot\max\{\tau_{c,j}\} - 2\cdot\tau_{c,\mathrm{node}(k_i)} \end{cases} \quad (4)$$

将约束方程代入目标函数得到式(5)：

$$\sum_{k_i \in T.ws \cup T.rs} \mathrm{DelayTime}(k_i) \leqslant \sum_{k_i \in T.ws \cup T.rs} 2\cdot\max\{\tau_{c,j}\} - 2\cdot\tau_{c,\mathrm{node}(k_i)} \quad (5)$$

可以看出，在满足式(5)的取等条件时，目标函数取得最大值。设数据项k_i属于子事务t_i，那么的延迟时间应该满足：

$$\mathrm{DelayTime}(t_i) = 2\cdot\max\{\tau_{c,j}\} - 2\cdot\tau_{c,\mathrm{node}(t_i)} \quad (6)$$

在事务进入执行阶段之前，Harp根据式(6)对子事务的发送时间进行调整，调整算法如算法1所示，该算法的时间复杂度为$O(n)$。

算法1：调整子事务的发送时间

输入：G：网络时延图；T：事务

输出：子事务的发送时间

1. Function Calculate Send Time (G, T)

2.max_latentcy←0, start_time←get_sys_clock()

3. for t_i∈T.subtxn do

4. j←get_node (t_i)

5. t_i.lat←$G[c][j]$

6. max_latentcy←max{max_latentcy, t_i.lat}

7. for t_i∈T.subtxn do

8. t_i.send_time←start_time+2·(max_latentcy−t_i.lat)

在Harp调度下，**图6**中实例的执行过程将发生改变。如**图7**所示，在事务执行前，通过算法1计算得出T_{12}和T_{13}的发送时间分别为40 ms和0 ms，T_{11}将在60 ms时执行，T_{21}和T_{31}延迟20 ms执行，T_4为单机事

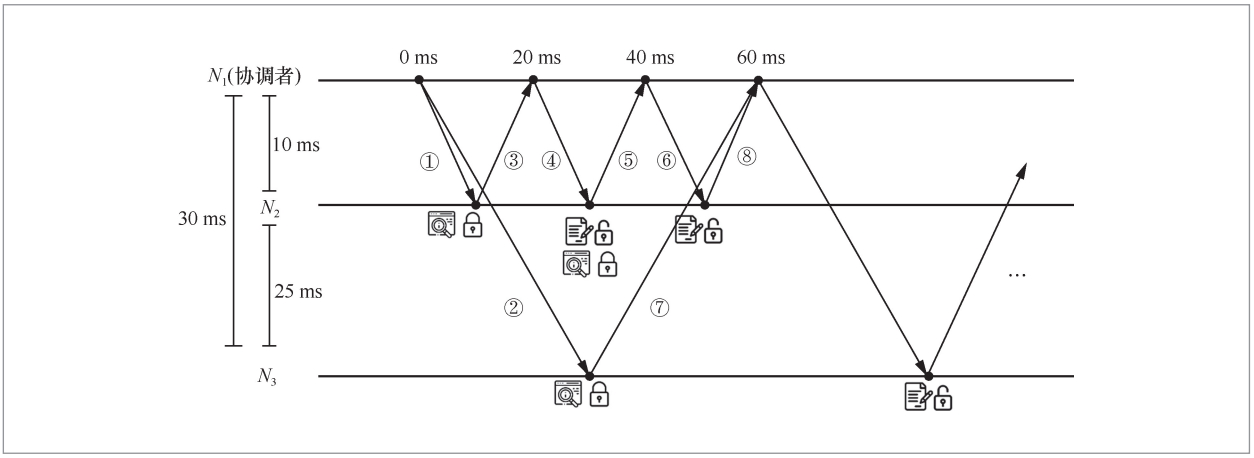


图 7 优化后的执行逻辑

务不做调整。经过调整后，T_1~T_4可以在60 ms内执行完毕。

在0 ms处，T_1~T_3子事务（包括本地子事务）的延迟执行使得T_4在执行$W_4(x)$时，数据项x上并没有锁，故T_4的执行不会被阻塞，并且可以立刻提交。同时，N_1会将子事务T_{21}发送给N_2（消息1）、T_{13}发送给N_3（消息2）。在20 ms处，N_1接收到子事务T_{22}的返回结果（消息3），同时执行T_2的本地子事务T_{21}，并根据T_{21}和T_{22}的执行结果确定事务状态，并将事务状态发给N_2（消息4）。经过10 ms后，N_2收到该消息。在修改了T_2的事务状态后，释放了数据项u的锁。此时，T_{32}获取了数据项u的锁，并且返回执行结果（消息5）。至此，T_2~T_4在40 ms内执行完毕。此时，协调者N_1向N_2发送子事务T_{12}（消息6）。在60 ms时，N_1执行本地子事务T_{11}，并收到了T_{12}和T_{13}的执行结果（消息7和消息8）。根据各子事务的执行结果确定T_1的最终状态。见**表1**，在2PL协议和2PC协议的调度下，T_2~T_4需要被阻塞直到T_1执行完毕。Harp将T_1的子事务T_{11}和T_{12}延迟执行，在不影响T_1的情况下，减少了T_1在数据项x、y和z上的锁争用时长，使T_1~T_4都可以在较短的时间内执行完成。

表1 实例中各事务执行时间

事务	开始时间/ms	结束时间/ms		时延/ms	
		优化前	优化后	优化前	优化后
T_1	0	60	60	60	60
T_2	0	80	20	80	20
T_3	0	100	40	100	40
T_4	0	100	0	100	0

同时，Harp还调整了数据库系统发送消息的相关逻辑，如**图8**所示。在队列中有消息需要处理时，从消息队列的队首取出消息，比较当前系统时间和消息的发送时间。如果当前系统时间大于消息的发送时间，说明子事务需要被执行，立刻发送该消息；否则该消息包含的子事务仍可以推迟执行，将其压入队尾，并继续处理队列中的其他消息。为了进一步优化锁争用时长，在接收消息时，对涉及锁操作的消息赋予一个较高的优先级，优先将这些消息分配给工作线程进行处理。

3.2 正确性分析

可串行化是保证事务正确调度的黄金

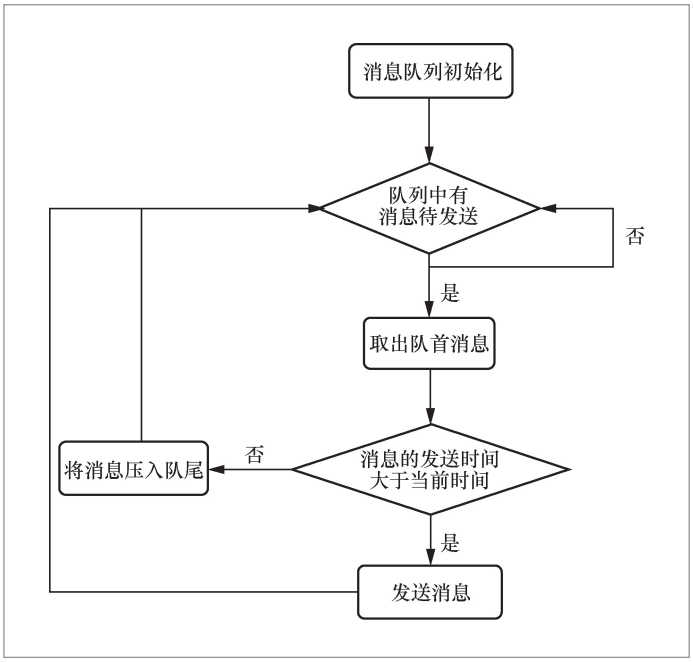


图 8 发送消息流程

法则[6]。Harp会根据网络差异调整子事务执行的顺序，优化事务的锁争用时长，但加锁方式仍然遵循两阶段封锁协议。同时，Harp利用No-Wait[4]和Wait-Die[14]策略进行死锁预防，不会出现依赖成环的情况。众所周知，遵循两阶段封锁协议的调度满足冲突可串行化[15]，因此利用Harp进行事务调度，仍然可以达到可串行化的隔离级别。

3.3 可行性分析

在跨空间域场景下，节点间的网络时延和差异均比较大，Harp根据操作涉及数据项的分布情况和网络特点，在不影响事务时延的基础上延迟执行部分子事务，以此缩短热点数据的锁争用时长，提高系统的并发度。Harp与确定性数据库最大的不同是Harp不需要对事务进行批处理。此外，对于不存在逻辑谓词的存储过程、单语句的交互式事务以及多语句的交互式事务的第一条语句，仍然可以利用Harp对它们进行优化。

此外，DTP模型也是当前数据库应用中处理跨域事务的方法。事务的调度可以在多个层面上实现，包括在数据库内核层面、中间件层面以及客户端层面。在DTP模型中，客户端应用通过中间件连接各数据库，并通过事务管理器（transaction manager，TM）完成2PC/3PC以保证事务的原子性和独立性。在分布式数据库中，由协调者节点负责事务的调度，而在DTP模型中，由中间件负责协调各个数据库中的事务，中间件在感知各目标节点的网络时延差异的基础上，使用本文的算法思想判断子事务的延迟时间，并延迟发送子事务，同样可以达到减少锁争用时长、提高系统并发度的效果。因此，本文算法思想在DTP模型中仍然是有效的。

3.4 针对性讨论

Harp的目标是在跨空间域场景下，利用网络时延差异，延迟部分子事务的执行时间，减少热点数据的锁争用时长。然而，Harp延迟发送的子事务可能因为数据项被其他并发的分布式事务占有而无法立刻返回，从而达不到算法预期的效果。目前的工作在遇到这种情况时会直接回滚，避免等待分布式事务。这是一种较为激进的方法，在后续的工作中可以增加元信息记录历史数据，预测各子事务从被工作线程处理到获取锁之间的时延，并对式（3）进行修正。

除了悲观并发控制算法2PL，乐观并发控制算法（OCC）、多版本并发控制（multi-version concurrency control，MVCC）[16]等都是数据库系统中常用的并发控制算法，它们在跨空间域场景下也会遇到性能较差、回滚率较高等问题。MVCC只能支持快照隔离级别，其与2PL、OCC等结合才能够达到可串行化隔离级

别，如MV2PL[17]、MVOCC[18]等。OCC为了保证事务的原子性和隔离性，在事务提交前会验证该事务的读写集，如果其他并发事务在此期间修改了相关数据项，会导致事务验证失败从而被回滚。在跨空间域场景下，节点之间的网络时延较长，事务从访问数据项到验证该数据项需要经历至少一轮网络通信，在这期间该数据项被修改的可能性变大，从而系统的回滚率会上升。利用Harp的技术思想，可以根据网络时延差异延迟部分子事务的执行，可以缩短对应子事务从执行到验证的时间间隔，从而减小数据项被其他并发事务修改的可能性，降低系统的回滚率。

此外，在多副本系统中副本一致性的开销在2PC协议中，而Harp的优化针对事务的执行阶段。因此，Harp的优化与副本一致性是正交的，同样可以应用在多副本系统中。在传统的2PC+Paxos事务执行逻辑中，Harp仍然可以延迟执行阶段的加锁时刻，从而缩短事务的锁争用时长。如果采用TAPIR[19]和G-PAC[20]这类对执行阶段进行了调整的事务执行逻辑，Harp是同样有效的。在执行阶段，TAPIR会将读写操作发给每一个副本，并在每一个副本中进行并发控制。在这种场景下，Harp在向各个副本发送读写操作前，会考虑协调者节点到各个副本的时延，将其代入算法1获得各子事务的延迟时间，通过延迟子事务的执行，仍然可以减少事务的锁争用时长，提高系统的并发度。

4 实验评估

4.1 环境测试

本文使用C++语言在分布式事务测试平台Deneva[21]中对Harp进行了实现，在相同的实验环境和实验负载下比较算法间的性能差异。共有8台服务器参与实验，每台服务器的参数见**表2**。其中，使用4台服务器作为客户端，另外4台作为服务端，客户端负责生成事务并发送给服务端执行。通过在网卡上增加网络时延模拟跨空间域场景，将4台服务器分为2组模拟“两地三中心”场景，城市间（跨国）的网络时延设置为300 ms，同城市的不同数据中心间的网络时延设置为25 ms，数据中心内的网络时延设置为10 ms。

本文基于YCSB负载进行性能测试。YCSB是一个模拟大规模互联网应用的综合基准测试，其数据集是1个包含10个属性的关系，其中1个属性为主键。在这个关系中，每条记录大小约为1 KB，数据集通过水平分区分布在不同节点上。2PL是悲观并发控制的经典算法，可以采用Wait-Die策略避免死锁问题，在数据库产品中有着广泛应用，例如，MySQL[22]、OceanBase[23]等。因此，本文的实验选择在YCSB[24]上对2PL和Harp进行性能测试。

4.2 性能测试

首先，笔者测试了倾斜因子对吞吐量和回滚率的影响，并将结果绘制在**图9**中，倾斜因子越大代表热点数据的争用越严重。可以看出，Harp在测试的所有场景中性能都优于Wait-Die。通过计算，发现

表2 实验服务器配置

配置	具体信息
操作系统	CentOS Linux release 7.4.1708 (Core)
CPU	Intel Core Processor (Skylake)
内存	32 GB

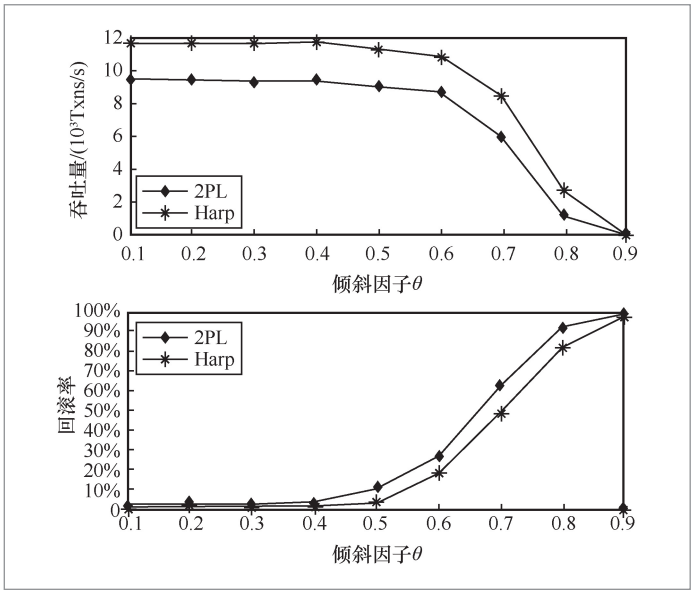


图 9 不同倾斜因子的性能测试

Harp在倾斜因子为0.8时的吞吐量相比2PL提升了1.36倍，同时回滚率下降了11%。

接着，笔者在分布式事务比例固定为50%的情况下，测试了Harp和2PL在不同读写比下的性能，结果如**图10**所示。随着写操作比例的增加，Harp和2PL都会遇到吞吐量下降和回滚率上升的问题。可以看出，Harp的性能下降较为缓和，在写读比为0时，系统处理的事务均为只读事务，2PL算法对读操作需要加共享锁，因此在这种场景下不会出现事务冲突，两者性能接近。随着读写事务比例的增加，Harp的性能优势逐渐明显，在写读比为100%时，即全是写事务时，Harp的性能达到了2PL的2.39倍，回滚率也远低于Wait-Die。

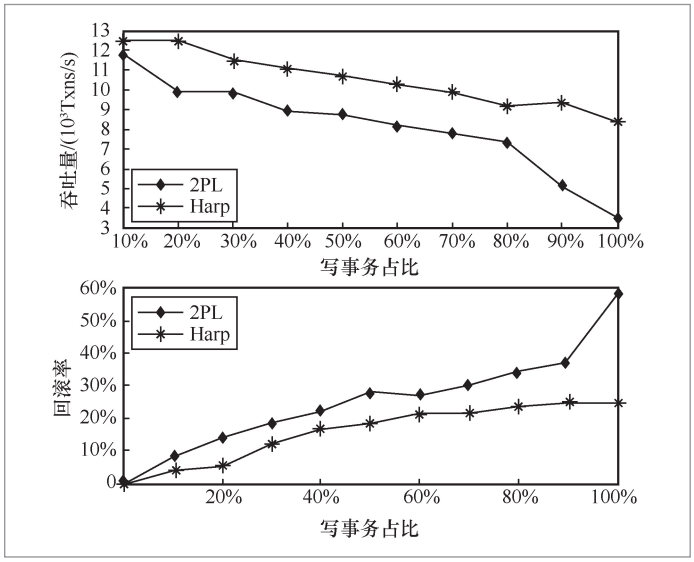


图 10 不同写事务占比的性能测试

最后，在中等争用场景下（θ=0.6），测试了Harp和2PL在多种分布式事务比例中的吞吐量和回滚率，结果如**图11**所示。因为Harp并没有对单机事务进行优化，在处理单机事务负载时（分布式事务占比为0）2PL的性能略好于Harp，这说明Harp的额外开销较小（2.7%）。随着分布式事务占比上升，两种算法的性能都在下降，但是Harp的吞吐量一直高于2PL。在处理跨空间域分布式事务，Harp根据网络差异推迟子事务的执行，缩短了锁争用时间。在实验中笔者统计了数据项的平均锁争用时长。实验结果表明，Harp的锁争用时长远低于2PL，在分布式事务占比为100%时，Harp的平均锁争用时长为2PL平均锁争用时长的26.8%。

Harp的性能提升得益于3个方面：①Harp根据网络差异性延迟执行部分子事务，这样可以缩短事务的锁争用时长，提高了系统的并发度，同时锁资源也得到了充分的利用；②Harp修改了加锁逻辑，不会出现远程事务相互等待的现象，虽然会发生不必要的回滚，但避免了事务过长的等待时间，并且实验表明Harp的回滚率在多种场景下都低于2PL；③Harp对消息队列进行了改进，使涉及锁操作的消息能更快地被响应，进一步缩短了事务的锁争用时长。

5 相关工作

分布式并发控制算法通常可以分为静

态定序算法和动态定序算法[6]。在静态定序算法中静态地确定事务的顺序。OCC的主体思想就是在事务提交时检查数据项是否被修改或者是否能修改，通常将进入验证阶段的时间戳作为依据确定顺序。Tu等人[25]提出的Slio就是OCC的变体，Silo维护了事务的读集和写集，在验证阶段通过检测自己的读集是否被其他事务修改来判断是否存在冲突。悲观并发控制算法，如2PL，通过对冲突数据项授予的锁的顺序对事务排序。此外，Thomson等人提出的Calvin[26]、Aria[27]和 Caracal[28]等是确定性并发控制算法。在Calvin中，以批（batch）为单位处理事务，将一段时间内的事务静态定序之后按照事务顺序为事务加锁。动态定序算法会为每个事务分配一个时间戳区间，并按一定的规则在区间中选择提交时间戳。在提交事务时，动态更新调整与它有关的事务时间戳区间[29]。Boksenbaum等人[30]首次在分布式并发控制中使用动态时间戳调整。近年来，围绕动态时间戳调整算法的工作也有很多，如MaaT[31]、Sundial[32]和TicToc[33]等。其中，MaaT利用额外的元数据和事务队列来维护数据项的访问痕迹，并记录了并发事务间的依赖关系，通过调整事务的逻辑时间戳区间的上界和下界，选择一个合理的提交区间，从而动态地确定事务之间的顺序。近年来，为了充分利用各种并发控制的优势，Liu等人[34]基于Actor模型提出了结合确定性并发控制和非确定性并发控制的混合并发控制算法。

在高争用场景下，并发事务之间存在大量的冲突和依赖，为了保证事务的ACID原则，避免数据异常，将会导致大量的事务被回滚。产业界也提出了一些针对高争用场景下锁争用时长优化的方案。Faleiro等人[35-36]提出了惰性执行方案（lazy execution scheme）和早期写可见性（early write visibility），旨在减少这些系统中的数据争用。Guo等人[37]在Bamboo中打破了两阶段锁协议的约束，在事务执行过程中有条件的释放锁，减少了争用现象，提高了事务执行的效率。Li等人[38]提出了SwitchTx，将部分事务逻辑下放到可编程交换机，减少网络调度的开销和数据争用。此外，Zamanian等人[5]提出了Chiller，在事务执行之前通过事务依赖图分析得出热点数据，将热点数据放置在本地，以此减少热点数据的锁争用。

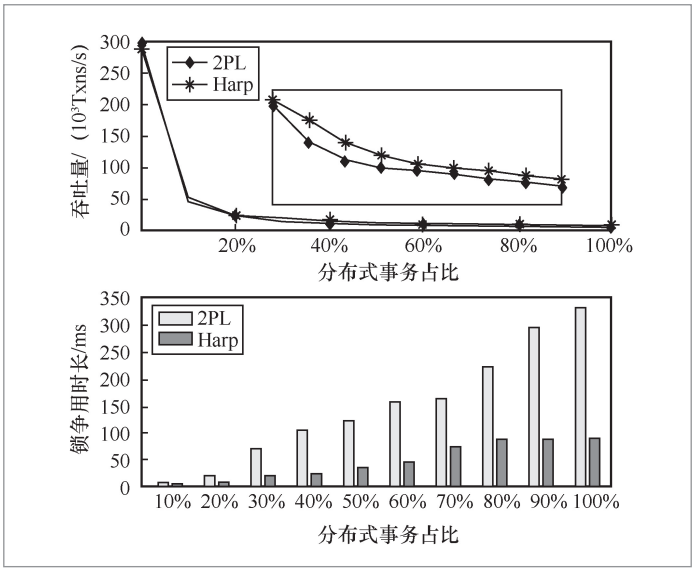


图11 不同分布式事务占比的性能测试

在跨空间域场景下，数据库将数据项进行分区以实现可扩展性，并利用多副本技术以实现高可用性。在处理访问多个分区的分布式事务时，Megastore[39]、Spanner和CockroachDB需要引入多轮网络通信以完成事务的执行、2PC、日志同步以及事务状态的复制。广域网中的多轮通信导致了事务的高延迟。为了缩短延迟，MDCC[40]、RedT[41]和TAPIR等并行执行2PC和日志同步。Yang等人[42]提出的Natto考虑了跨空间域网络差异，在预估事务到达最远参与者的时刻后，根据该时刻分配

时间戳，以此降低高优先级事务的延迟。此外，有大量的工作聚焦于优化跨空间域的数据分布。CLOCC[43]在客户端使用缓存。然而，客户端的缓存有限，但某些工作负载需要大量的缓存，并且保持缓存一致性会引入额外的开销。Cloud SQL Server[44]通过限制事务只能访问单节点上的数据来避免2PC。此外，在工作负载变化时Akkio[45]在数据中心之间移动数据，以增加数据局部性，但其并没有提供事务保证。

6 结束语

在跨空间域场景下，节点之间的网络时延更长且存在差异性，传统的事务处理算法会遇到新的挑战。本文在分析了分布式事务存在的问题和优化空间后，提出了面向跨空间域的分布式事务优化算法Harp。Harp根据网络的差异性，延迟部分子事务的执行，减少了事务的锁争用时长，同时提高了锁资源的利用率，使分布式数据库的性能得到明显提升。作为跨空间域分布式事务优化的初步研究，Harp可以为后续的跨域数据管理研究提供一定的参考价值。

参考文献:

[1] 柴云鹏，李彤，范举，等. 跨域数据管理的内涵与挑战[J]. 中国计算机学会通讯，2022，18(11)：29-33.
CHAI Y P, LI T, FAN J, et al. Connotation and challenges of cross-domain data management [J]. Communications of China Computer Society, 2022, 18(11): 29-33.

[2] TAFT R, SHARIF I, MATEI A, et al. CockroachDB: the resilient geo-distributed SQL database[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2020: 1493-1509.

[3] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems, 2013, 31(3): 1-22.

[4] BERNSTEIN P A, GOODMAN N. Concurrency control in distributed database systems[J]. ACM Computing Surveys, 13(2): 185-221.

[5] ZAMANIAN E, SHUN J L, BINNIG C, et al. Chiller: contention-centric transaction execution and data partitioning for modern networks[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2020: 511-526.

[6] 赵泓尧，赵展浩，杨皖晴，等. 内存数据库并发控制算法的实验研究[J]. 软件学报，2022，33(3)：867-890.
ZHAO H Y, ZHAO Z H, YANG W Q, et al. Experimental study on concurrency control algorithms in in-memory databases[J]. Journal of Software, 2022, 33(3): 867-890.

[7] KUNG H T, ROBINSON J T. On optimistic methods for concurrency control[J]. ACM Transactions on Database Systems, 1981, 6(2): 213-226.

[8] MOHAN C, LINDSAY B. Efficient commit protocols for the tree of processes model of distributed transactions[C]//Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing. New York: ACM Press, 1983: 76-88.

[9] BINNIG C, CROTTY A, GALAKATOS A, et al. The end of slow networks[J]. Proceedings of the VLDB Endowment, 2016, 9(7): 528-539.

[10] BALLA D, MALIOSZ M, SIMON C, et al. Bounded latency with RoCE[C]//Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos. New York: ACM Press, 2019: 134-135.

[11] STAMOS J W, CRISTIAN F. A low-cost atomic commit protocol[C]//Proceedings

Ninth Symposium on Reliable Distributed Systems. Piscataway: IEEE Press, 2002: 66–75.

[12] LU Y, YU X, MADDEN S. STAR: scaling transactions through asymmetric replication[J]. arXiv preprint, 2018, arXiv: 1811.02059.

[13] LIN Q, CHANG P F, CHEN G, et al. Towards a non–2PC transaction management in distributed database systems[C]// Proceedings of the 2016 International Conference on Management of Data. New York: ACM Press, 2016: 1659–1674.

[14] 王珊, 萨师煊. 数据库系统概论(第5版)[M]. 北京: 高等教育出版社, 2014.
WANG S, SA S X. Introduction to database system(5th ed.)[M]. Beijing: Higher Education Press, 2014.

[15] ROSENKRANTZ D J, STEARNS R E, LEWIS P M. System level concurrency control for distributed database systems[J]. ACM Transactions on Database Systems, 1978, 3(2): 178–198.

[16] WU Y J, ARULRAJ J, LIN J X, et al. An empirical evaluation of in–memory multi–version concurrency control[J]. Proceedings of the VLDB Endowment, 2017, 10(7): 781–792.

[17] DuBourdieuD. Implementation of distributed transactions[C]//Proceedings of 6th Berkeley Workshop on Distributed Data Management and Computer Networks. [S.l.:s.n.], 1982.

[18] LARSON P Å, BLANAS S, DIACONU C, et al. High–performance concurrency control mechanisms for main–memory databases[J]. Proceedings of the VLDB Endowment, 2011, 5(4): 298–309.

[19] ZHANG I, SHARMA N K, SZEKERES A, et al. Building consistent transactions with inconsistent replication[J]. ACM Transactions on Computer Systems, 2018, 35(4): 1–37.

[20] MAIYYA S, NAWAB F, AGRAWAL D, et al. Unifying consensus and atomic commitment for effective cloud data management[J]. Proceedings of the VLDB Endowment, 2019, 12(5): 611–623.

[21] HARDING R, AKEN D V, PAVLO A, et al. An evaluation of distributed concurrency control[J]. Proceedings of the VLDB Endowment, 2017, 10(5): 553–564.

[22] WIDENIUS U M, AXMARK D. MySQL reference manual – documentation from the source[M]. [S.l.]: O'Reilly, 2002.

[23] YANG Z K, YANG C H, HAN F S, et al. OceanBase: a 707 million tpmC distributed relational database system[J]. Proceedings of the VLDB Endowment, 2022, 15: 3385–3397.

[24] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB[C]//Proceedings of the 1st ACM Symposium on Cloud Computing. New York: ACM Press, 2010: 143–154.

[25] TU S, ZHENG W T, KOHLER E, et al. Speedy transactions in multicore in–memory databases[C]//Proceedings of the 24th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2013: 18–32.

[26] THOMSON A, DIAMOND T, WENG S C, et al. Calvin: fast distributed transactions for partitioned database systems[C]// Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2012: 1–12.

[27] LU Y, YU X Y, CAO L, et al. Aria[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2047–2060.

[28] QIN D, BROWN A D, GOEL A. Caracal: contention management with deterministic concurrency control[C]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. New York: ACM Press, 2021: 180–194.

[29] BAYER R, ELHARDT K, HEIGERT J, et al. Dynamic timestamp allocation for transactions in database systems[C]// International Symposium on Distributed Data Bases. [S.l.:s.n.], 1982: 9–20.

[30] BOKSENBAUM C, CART M, FERRIÉ

J, et al. Certification by intervals of timestamps in distributed database systems[C]//Proceedings of the 10th International Conference on Very Large Data Bases. New York: ACM Press, 1984: 377–387.

[31] MAHMOUD H A, ARORA V, NAWAB F, et al. MaaT: effective and scalable coordination of distributed transactions in the cloud[J]. Proceedings of the VLDB Endowment, 2014, 7(5): 329–340.

[32] YU X Y, XIA Y C, PAVLO A, et al. Sundial: harmonizing concurrency control and caching in a distributed OLTP database management system[J]. Proceedings of the VLDB Endowment, 2018, 11(10): 1289–1302.

[33] YU X Y, PAVLO A, SANCHEZ D, et al. TicToc: time traveling optimistic concurrency control[C]//Proceedings of the 2016 International Conference on Management of Data. New York: ACM Press, 2016: 1629–1642.

[34] LIU Y J, SU L, SHAH V, et al. Hybrid deterministic and nondeterministic execution of transactions in actor systems[C]//Proceedings of the 2022 International Conference on Management of Data. New York: ACM Press, 2022: 65–78.

[35] FALEIRO J M, ABADI D, HELLERSTEIN J. High performance transactions via early write visibility[J]. Proceedings of the VLDB Endowment, 2017, 10(5): 613–624.

[36] FALEIRO J M, THOMSON A, ABADI D J. Lazy evaluation of transactions in database systems[C]//Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2014: 15–26.

[37] GUO Z H, WU K, YAN C, et al. Releasing locks as early as you can: reducing contention of hotspots by violating two-phase locking[C]//Proceedings of the 2021 International Conference on Management of Data. New York: ACM Press, 2021: 658–670.

[38] LI J R, LU Y Y, ZHANG Y M, et al. SwitchTx: scalable in-network coordination for distributed transaction processing[J]. Proceedings of the VLDB Endowment, 2022, 15(11): 2881–2894.

[39] BAKER J, BOND C, CORBETT J C, et al. Megastore: providing scalable, highly available storage for interactive services[C]//Proceedings of the 5th Biennial Conference on Innovative Data Systems Research. [S.l.:s.n.], 2011: 223–234.

[40] KRASKA T, PANG G, FRANKLIN M J, et al. MDCC: multi-data center consistency[J]. arXiv preprint, 2012, arXiv: 1203.6049.

[41] ZHANG Q, LI J Y, ZHAO H Y, et al. Efficient distributed transaction processing in heterogeneous networks[J]. Proceedings of the VLDB Endowment, 2023, 16(6): 1372–1385.

[42] YANG L G, YAN X N, WONG B. Natto: providing distributed transaction prioritization for high-contention workloads[C]//Proceedings of the 2022 International Conference on Management of Data. New York: ACM Press, 2022: 715–729.

[43] LISKOV B, CASTRO M, SHRIRA L, et al. Providing persistent objects in distributed systems[C]//Proceedings of the ECOOP'99-Object-Oriented Programming. Heidelberg: Springer, 1999: 230–257.

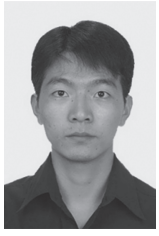
[44] BERNSTEIN P A, CSERI I, DANI N, et al. Adapting microsoft SQL server for cloud computing[C]//Proceedings of 2011 IEEE 27th International Conference on Data Engineering. Piscataway: IEEE Press, 2011: 1255–1263.

[45] ANNAMALAI M, RAVICHANDRAN K, SRINIVAS H, et al. Sharding the Shards: managing datastore locality at scale with Akkio[C]//Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation. New York: ACM Press, 2018: 445–460.

作者简介

庄琪钰（2000– ），男，中国人民大学信息学院博士生，主要研究方向为分布式数据库系统、事务处理。

李彤（1989– ），男，博士，中国人民大学信息学院副教授，主要研究方向为新一代互联网体系结构、跨域数据管理和大数据。



卢卫（1981– ），男，博士，中国人民大学信息学院教授、博士生导师，中国计算机学会数据库专业委员会委员，主要研究方向为数据库基础理论、大数据系统研制、时空背景下的查询处理和云数据库系统和应用。

杜小勇（1963– ），男，博士，中国人民大学信息学院二级教授、博士生导师，主要研究方向为数据库系统、大数据管理与分析、智能信息检索。

收稿日期： 2023-02-28

通信作者： 杜小勇，duyong@ruc.edu.cn

基金项目： 国家自然科学基金资助项目（No.61972403，No.61732014，No.62202473）；国家重点研发计划资助项目（No.2020YFB2104100）

Foundation Items: The National Natural Science Foundation of China (No.61972403, No.61732014, No.62202473), The National Key Research and Development Program of China(No.2020YFB2104100)