计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.20210231
59(4):721-736,2022

内存事务中并发控制协议研究综述

姜天洋　张广艳　李之悦
(清华大学计算机科学与技术系　北京　100084)
(北京信息科学与技术国家研究中心(清华大学)　北京　100084)
(ty-jiang18@mails.tsinghua.edu.cn)

Survey on Concurrency Control Protocols of In-Memory Transactions

Jiang Tianyang, Zhang Guangyan, and Li Zhiyue
(*Department of Computer Science and Technology, Tsinghua University, Beijing* 100084)
(*Beijing National Research Center for Information Science and Technology (Tsinghua University), Beijing* 100084)

Abstract Transactions provide strong guarantees for the upper-level applications based on databases and other systems. NoSQL databases obtain higher scalability by weakening the support for transactions, but it is difficult to meet the transactional requirements of applications such as OLTP. To retrieve the support for highly consistent transactions, the NewSQL database architecture proposed later takes into account the efficient management of massive data. NewSQL databases therefore migrate data accessed by transactions from disk to memory to accelerate transaction processing. However, existing concurrency control protocols of in-memory transactional systems are challenged by the incompatibility with emerging storage and network technology. We categorized the concurrency control protocols of in-memory transactions in the past ten years from the three dimensions of processing strategy, version control, and conflict resolution. Subsequently, we offer detailed comparisons of typical concurrency control protocols from the three aspects of performance, scalability, and durability. Then four technical solutions are summarized to improve concurrency control protocols of in-memory transactions including eliminating the bottleneck of scalability, accelerating transaction processing with emerging hardware, reducing the probability of transaction aborts and ensuring the durability of in-memory transactions efficiently. Finally, we point out the future research direction of concurrency control protocols of in-memory transactions.

Key words in-memory transaction; concurrency control; distributed system; remote direct memory access; persistent memory

摘　要　事务为数据库等系统的上层应用提供了强大的保证.NoSQL 数据库通过弱化对事务的支持来获得更高的扩展性,却难以满足 OLTP 等应用的事务性需求.之后提出的 NewSQL 数据库架构回归了高一致性的事务支持,并兼顾了海量数据的高效管理.因此,NewSQL 数据库逐步将事务执行所需的数据从硬盘迁移到内存中以提升事务执行的效率.但是,已有内存事务的并发控制协议与新兴的存储、

收稿日期:2021-03-18;**修回日期**:2021-08-18
基金项目:国家自然科学基金项目(62025203,61672315);国家重点研发计划项目(2018YFB0203902)
This work was supported by the National Natural Science Foundation of China (62025203, 61672315) and the National Key Research and Development Program of China (2018YFB0203902).
通信作者:张广艳(gyzh@tsinghua.edu.cn)

网络设备并不适配.从处理策略、版本控制、冲突解决 3 个维度对近 10 年来的内存事务中并发控制协议进行了分类阐述,进而从性能、扩展性、持久性 3 个方面比较了有代表性的并发控制协议.之后总结了 4 个改进内存事务并发控制协议的技术思路:消除事务扩展瓶颈,利用新硬件加速事务处理,降低事务中止概率,高效保证事务持久性.最后指出了内存事务并发控制协议的未来研究方向.

关键词 内存事务;并发控制;分布式系统;远程直接内存访问;持久性内存

中图法分类号 TP333

事务指一个有限的操作序列[1],是数据库或其他系统的一个执行逻辑单元.事务具有 ACID (atomicity, consistency, isolation, durability)[2]的特性:事务中的所有操作必须全部执行或全部不执行(原子性);确保系统从一个一致的状态转变为另一个一致的状态(一致性);多个事务并发执行时不互相影响执行结果(隔离性);已提交事务对系统的修改将被永久保存(持久性).事务支持应用通过数据库等系统来处理各种复杂的业务逻辑,对于在线交易和实时分析等领域有着重要作用.

数据库系统在不同发展阶段提供不同程度的事务支持.以 Oracle[3],SQL Server[4]为代表的关系型数据库[5]为用户提供严格的事务支持,用户通过结构化查询语言 SQL 对数据进行一系列事务操作.以 Redis[6],MongoDB[7]等为代表的 NoSQL 数据库[8]放松了对事务的支持,实现了数据读写的高性能以及数据增长时的可扩展.然而,因为只提供弱一致性保证,NoSQL 数据库往往难以满足联机事务处理 (online transaction processing, OLTP)[9]等应用的需要.以 Spanner[10],CockroachDB[11]为代表的新兴 NewSQL 数据库[12]设计了适应更高计算、网络和存储能力的并发控制协议,在提供海量数据存储管理能力的同时,实现了满足 OLTP 等应用的严格事务支持.

内存事务处理是 NewSQL 数据库系统的核心.早期的硬盘事务指内存缓存事务执行的部分数据以及保留必要的元数据,硬盘存储所有事务执行的数据,其特点是事务执行延迟高、并发程度低.内存事务指内存存储所有事务执行需要的数据,硬盘等持久性介质通过存储日志等方式来保证事务的持久性.内存事务并发控制协议主要为单机多核及分布式场景设计,显著缩短了事务的执行时间.计算、网络、存储等基础设施的能力提升,促使 NewSQL 数据库提供高性能、可扩展、持久化的内存事务支持.例如 2020 年“双 11”,阿里巴巴公司的云数据库 PolarDB 每秒事务处理峰值达到 1.4 亿次[13].

为了聚焦于内存事务,研究人员将数据库中事务处理部分分离,形成内存事务系统进行单独研究.内存事务系统主要包括并发控制层、索引层和存储层 3 个层次[14].其中,并发控制层是内存事务的核心,也是本文主要讨论内容.并发控制协议通过合理安排并发执行的多个内存事务中的数据读写操作的顺序,避免事务的执行结果相互影响,在保证多个内存事务之间隔离性的同时,实现事务处理的高效率.并发控制协议设计,需要结合事务系统的数据布局、网络架构和上层应用的业务模式,广义上的并发控制协议还包括内存事务的持久化方案.

使用场景的变迁和设备工艺的发展使得传统并发控制协议面临诸多挑战:1)数据访问热点的存在使得多核架构中的并发控制协议必须通过跨核通信才能发现并解决事务冲突,而处理器核数和内存规模的增长使得跨核访问的延迟明显上升,如何在解决事务冲突的前提下减少跨核通信对并发控制协议设计是一个挑战.2)并发控制协议往往通过集中式锁管理器、集中式时间戳分配器等集中控制点来协调解决事务和数据的冲突,这些集中的协调环节容易成为分布式系统的性能瓶颈,使得系统在规模不断增长的数据中心集群里面临扩展性难题.3)原有事务管理基于磁盘或固态盘进行异步持久化,面对速度快 1 到 2 个数量级的非易失内存(non-volatile memory, NVM)[15]时,异步持久化方案已不再适用,如何将持久化操作加入并发控制协议中,使得并发控制协议适应 NVM 的性能特点,是保证事务系统性能和可用性的一个挑战.

研究人员从增强扩展性、利用新硬件加速事务处理、减少事务中止次数、保证持久性等多条技术路线出发,对内存事务的并发控制协议进行了探索.1)通过重构时间戳分配机制、缓存热点数据等技术,逐步解决了系统扩展中的瓶颈问题.2)结合计算能力的提升和新兴网络技术来改进并发控制协议,缩短了事务的执行时间.3)通过采用更加灵活的提交策略以及数据的多版本技术在不同场景中降低事务

中止概率,提升了事务系统的吞吐率.4)内存事务利用非易失内存和多副本技术,提高了持久性保证的实现效率,并增强了事务系统的容错能力.

本文从内存事务中并发控制协议所面临的机遇与挑战出发,梳理了最近10年内存事务相关的研究工作.首先,从处理策略、版本控制、冲突解决3个维度,分类阐述了现有内存事务的并发控制协议.接着从性能、扩展性、持久性等方面论述并比较了代表性的并发控制协议,并通过实验比较了部分并发控制协议的性能,总结了内存事务并发控制协议优化的技术思路.最后,指出了内存事务中并发控制协议的未来研究方向,并对相关的综述工作进行了总结.

1 内存事务中并发控制协议的机遇与挑战

内存事务系统架构一般包括3层,从上至下依次为并发控制层、索引层、存储层,如图1所示.并发控制层包含事务的并发控制协议,是内存事务处理的核心.对于单机事务系统,并发控制层维护了事务的读写集合、数据的锁、版本号、读写时间戳等信息.对于分布式事务系统,并发控制层还需要引入网络通信,通过远程过程调用(remote procedure call, RPC)等技术去读写远端的数据及元数据信息.事务进入系统后,并发控制协议会根据实时并发、有读写依赖关系的其他事务,确定该事务读写数据的时机,以确保事务并发执行时的隔离性,并追求事务处理的高性能.

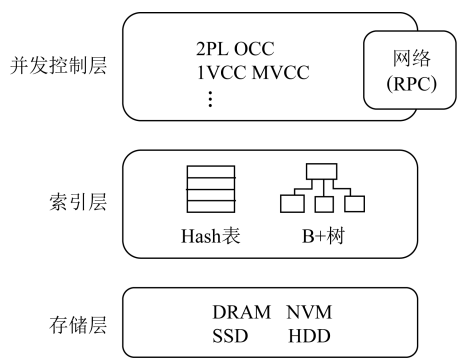


Fig. 1 Architecture of in-memory transaction system

图1 内存事务系统架构

在并发控制层之下,索引层为事务提供数据所在的位置信息.在分布式场景中,索引层可以通过静态Hash、一致性Hash等分布式索引方式,也可以通过元数据服务器动态地维护数据分布表等中心化索引方式,帮助事务找到数据所在的服务器.在单台数据服务器上,索引层则利用Hash表、B^+树这2种在内存事务系统中使用最广泛的数据结构,查找数据在对应服务器上的具体位置.

找到了数据的具体位置,内存事务系统在存储层对数据进行读写操作.存储层综合包括了内存、硬盘等多种介质构成的存储空间,可以被组织成块存储或对象存储等不同形式.

1.1 发展机遇

关于事务的研究一直是计算机系统领域一个热点问题.针对事务的并发控制协议与实现在20世纪80年代就有了大量的学术研究[16-17],Oracle,MySQL[18],SQL Server等关系型数据库的兴起带动了事务从学术研究走向工业应用.近10年新兴网络、存储技术的出现带给了并发控制协议新的机遇,具体可归纳为2点:

1) 网络能力的提升降低了分布式事务的集群通信开销

由于数据分布式地存放在集群的不同节点内,分布式事务必须通过网络通信读写远端节点的数据.传统网络通信存在内核调用栈复杂、缓冲区拷贝频繁等问题,网络通信时间更多耗费在了服务器端而非链路上.新兴的数据平面开发套件(data plane development kit, DPDK)[19]和远程直接内存访问(remote direct memory access, RDMA)[20-21]技术都聚焦于绕过复杂的系统调用来实现简单高效的网络通信,它们具有的低延迟和高吞吐率特性可以帮助分布式事务减少网络通信开销.

同时,RDMA技术提供的新型网络原语,也给进一步加速分布式事务带来了机遇.RDMA提供的远程读写、远程原子操作可以在远端CPU不参与的前提下,绕过内核直接读写远端内存.这种做法不仅降低了本地事务操作获取远端数据的延迟,还降低了远端CPU的负载,因此提升了事务系统的整体吞吐率.远程原子操作甚至能帮助并发控制协议实现部分复杂的逻辑,在完全没有远端CPU的参与下解决事务间的冲突.

2) 新型存储硬件的出现加速了数据持久化并减少了事务冲突

传统内存事务的持久化需要借助固态盘或磁盘,它们读写粒度大、延迟高,事务的批量持久化策略会带来崩溃后大量事务重试的风险,进而影响恢复速度,事务的持久化完成时间通常明显落后于事务

在内存中的完成时间.非易失内存能提供字节粒度的数据访问,访存时延接近 DRAM 内存,且掉电后数据不丢失.在目前的计算机存储层级结构中,它可以作为内存或块设备工作,从而有效加速内存事务的持久化.

此外,硬件事务内存技术(hardware transactional memory, HTM)的发展有助于从硬件角度解决事务访存竞争并减少事务冲突.作为一个典型实例,Intel 公司的受限事务内存(restricted transactional memory, RTM)[22]使用一级缓存跟踪读写集,并依赖缓存一致性检测数据竞争.RTM 的效率比软件实现的并发控制协议更高,却存在读写集大小受限、不支持网络 I/O 等不足.研究人员可以思考如何利用 RTM 实现高效的分布式事务并发控制协议.

1.2 面临的技术挑战

网络和存储技术的进步以及使用场景的变迁也让内存事务中并发控制协议的发展面临新的挑战,主要包括 3 个方面:

1) 多核场景中大量跨 NUMA 节点的通信增加了事务的延迟

随着处理器技术的发展和内存规模的增加,单个高端服务器可以拥有上百个核心及 TB 级内存.单机数据库系统在存储数据时往往采用共享内存[23]的模式,任何线程都可以自由地访问所有数据.事务在执行时会从共享内存中读写相应数据,并将读集、写集写入共享内存中供其他事务验证冲突.然而,随着核数增多,这种模式造成大量跨 NUMA 节点的通信开销,增加了事务的延迟.此外,不同线程对热点数据的同步操作也会造成 CPU 资源的等待浪费,即使是 CAS(compare and swap)等原子指令,在很多线程同时发起指令的情况下也会导致阻塞.采用硬分区技术的事务系统将数据按核划分,单核内执行事务避免了核间通信和数据竞争,但一旦遇到难以简单分区的负载,系统就会因为频繁的跨分区事务而出现性能降级.因此,如何在多核场景中减少事务执行时的跨核通信,尽早检测并避免数据竞争,是设计并发控制协议面临的重要挑战.

2) 分布式系统的集中控制点限制了事务系统的扩展性

分布式数据库的规模从同机房单一机架规模,逐步扩展到异地多数据中心,其扩展性问题变得更加突出.数据库处理分布式事务存在很多集中控制点,包括集中式锁管理节点、集中式时间戳分配节点等.并发控制协议利用这些集中控制点协调事务处理顺序,解决事务读写数据时产生的冲突等,保证事务的隔离性等级.所有事务的执行路径都会包括这些集中控制点,因此成为分布式数据库的瓶颈,影响系统的扩展性.设计者需要改进事务处理流程,将事务的协调开销、数据冲突分摊到各个节点,避免单个节点访问量过大成为瓶颈.同时,将事务分散到各个节点可能造成事务之间的冲突无法协调,元数据通信不及时会导致系统状态的不一致,最终导致事务中止增多、系统吞吐率降低.如何平衡两者关系,合理地减少分布式事务的集中控制点,是设计者面临的挑战.

3) 原有协议难以发挥新设备优势来提高持久性保证的实现效率

原有事务的持久化方案为固态盘或磁盘设计,由于磁盘访存延迟高、访问粒度大,原有方案采用异步持久化方案,将操作日志批量聚合成块后发往磁盘.因此简单地将磁盘替换为 NVM 无法发挥新设备低延迟、低访问粒度的特性.在并发控制协议中加入持久化、网络备份操作来适配 NVM 的特性,尽可能避免影响事务原有性能,是一个重要挑战.一方面,由于 NVM 的访存性能和 DRAM 仍有数倍差距,主存中 NVM 和 DRAM 还将并存.并发控制协议需要设计 NVM 区域中事务日志信息的存储格式,在保证正确处理事务冲突的前提下加入持久化操作.另一方面,事务可以将数据的修改通过网络快速持久化到后备节点的 NVM 上,避免主节点崩溃后数据不可用.然而,直接在存储层应用副本技术会因为割裂并发控制层和存储层而增加了通信开销.

2 内存事务并发控制协议的分类

本文从 3 个维度对内存事务中并发控制协议进行分类.首先是从事务处理策略的角度,分为悲观并发控制和乐观并发控制.悲观并发控制要求事务在执行时锁定数据,以阻止其他事务修改数据,如果锁定失败,则该事务选择继续等待或中止.2 阶段锁(2-phase lock, 2PL)是一种常见的悲观并发控制,在业界有着广泛应用.乐观并发控制(optimistic concurrency control, OCC)强调事务在读写时不是锁定或独占数据,而是在提交阶段对读写的数据进行验证,如果验证通过则成功提交,否则说明事务读写的数据在执行阶段被其他事务干扰,该事务必须中止并回滚.

在事务冲突较少时,OCC 的事务被干扰中止的概率更低,吞吐率表现更加优异,而 2PL 更适合冲突较多的场景.

如图 2 所示,2PL 在执行过程中分为 2 阶段:1)持锁阶段,事务在执行过程中获取读写数据所需的锁,并阻塞其他事务执行;2)放锁阶段,事务释放所有已经获得的锁,将数据的修改展示给其他事务.OCC 则分为 3 个阶段:执行阶段、验证阶段、提交阶段.事务在执行阶段读取数据及相应版本号,对于写请求缓存在本地或当前线程的缓冲区;在验证阶段验证事务的读集没有被其他事务的写请求修改,以及事务的写集不与其他事务的写集冲突;在提交阶段修改数据,并将修改展示给其他事务.

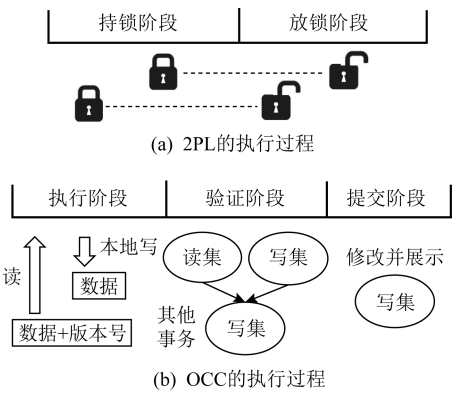


Fig. 2 Executive processes of two concurrency control schemes

图 2 2 种并发控制方案的执行过程

从数据版本控制的角度,并发控制协议可以分为单版本并发控制(1-version concurrency control, 1VCC)和多版本并发控制(multi-version concurrency control, MVCC)[24].单版本并发控制的数据只维护数据的最新版本,这种方式简单直接并且易于进行内存管理.然而,当事务的并发程度增加,维护数据的多个历史版本能让不同事务分别读取对应时间点的数据,减少了事务之间的冲突,增加了系统的吞吐率.同时,为每个数据维护多个历史版本会给系统增加额外的内存开销,定期回收、整合数据的不同版本给系统增添了 CPU 开销.

根据数据竞争的解决方式,并发控制协议又可以分为基于锁、基于时间戳、混合使用 3 种类型.锁是处理数据竞争最直接有效的方式,对于"写-写"数据冲突的事务可以通过互斥锁来解决,"读-写"数据冲突的事务可以借助读写锁来解决.但是,锁的使用会降低事务的并发率,例如 2PL 为了满足可串行化的隔离等级,要求事务在提交前获得所有相关数据的锁,这给事务并发处理增添了很多"虚假"的冲突,也增加了系统出现"死锁"的风险.基于时间戳的协议通过给事务分配开始时间戳、提交时间戳以及数据版本的时间戳,确定事务之间正确执行必须存在的依赖关系,缩小了事务处理中由锁保护的临界区的长度,提高了并发程度.通常基于时间戳的协议需要 2 次验证来确定事务能否提交,这给处理"写-写"冲突带来较大开销,因为一旦事务中止,回滚事务需要很多额外操作.因此,研究人员提出了利用时间戳解决"读-写"冲突、利用锁解决"写-写"冲突的混合并发控制协议.

以上 3 种分类是依据并发控制协议的实现方式.不同协议实现的隔离等级亦不相同,本文涉及的协议包括 3 种隔离等级[25]:快照隔离性(snapshot isolation)、可串行化(serializability)、严格可串行化(strict serializability).更低的隔离等级对于应用编程不友好,因此本文不作讨论.快照隔离性要求事务的读操作看到数据库中一个一致的版本快照,事务的写操作只要在读到快照时间点后不与并发的写冲突即可成功提交.因此快照隔离性会出现"写偏斜"的异常.可串行化保证了并发事务的执行结果一定和某个串行化顺序的执行结果一致,也可以理解为事务的读写从结果上看在同一时间点发生;严格可串行化是最强的隔离级别,在可串行化基础上要求,从物理时间上看,如果事务 A 在事务 B 开始之前已经提交,那么在可串行化序中事务 A 必须先于事务 B.从实现方式看,基于锁的并发控制协议利用锁保护临界区解决冲突,天然地具备了满足严格可串行化的特性:后发生的事务要么看到已经提交的修改,要么看到其他事务持锁而阻塞;基于时间戳的并发控制则受制于分布式时钟的精度误差,必须通过不确定性等待等额外操作实现严格可串行化.然而,兼顾严格可串行化和事务系统性能并不容易,根据 SNOW 理论[26],当并发控制中同时处理只读事务和读写事务时,为实现严格可串行化,只读事务的读请求要么需要一次以上的信息交互,要么需要阻塞、等待等操作,这影响了事务的执行延迟.因此,部分并发控制协议选择提供可串行化隔离等级,以换取性能的提升.

表 1 列举了近 10 年提出的 19 个内存事务系统,并对其并发控制协议进行分类.

Table 1 Classification of Cocurrency Control Protocols of In-Memory Transaction Systems

表 1 内存事务系统并发控制协议分类

系统名称	年份	系统场景	协议类型	版本控制	冲突解决方式	隔离等级
FaRM[27-28]	2014	分布式	OCC	单版本	锁	严格可串行化
TAPIR[29]	2015	分布式	OCC	多版本	时间戳	可串行化
DrTM[30]	2015	分布式	2PL	单版本	锁	严格可串行化
FaSST[31]	2016	分布式	OCC	单版本	锁	严格可串行化
DrTM+R[32]	2016	分布式	OCC	单版本	锁	严格可串行化
NAM-DB[33]	2017	分布式	OCC	多版本	混合	快照隔离性
DrTM+H[34]	2018	分布式	OCC	单版本	锁	严格可串行化
Sundial[35]	2018	分布式	OCC	单版本	混合	可串行化
FaRM v2[36]	2019	分布式	OCC	多版本	混合	严格可串行化
DST[37]	2021	分布式	OCC/2PL	多版本	混合	可串行化
DAST[38]	2021	分布式	OCC	单版本	时间戳	可串行化
HEKATON[39]	2011	单机多核	OCC	多版本	混合	可串行化
Silo[40]	2013	单机多核	OCC	单版本	混合	可串行化
Doppel[41]	2014	单机多核	OCC	多版本	锁	可串行化
TicToc[42]	2016	单机多核	OCC	单版本	混合	可串行化
Cicada[43]	2017	单机多核	OCC	多版本	时间戳	可串行化
DUDETM[44]	2017	单机多核	OCC/2PL	单版本	混合	可串行化
Pisces[45]	2019	单机多核	OCC	多版本	混合	快照隔离性
Bamboo[46]	2021	单机多核	2PL	单版本	混合	可串行化

3 并发控制协议的代表性工作

内存事务系统从不同技术路线出发,对并发控制协议进行了很多探索.本节整理了最近 10 年具有代表性的内存事务系统及其并发控制协议,按照技术路线可以分为 4 类:消除事务扩展瓶颈、利用新硬件加速事务处理、降低事务中止概率、高效保证事务持久性.

3.1 消除事务扩展瓶颈

扩展性是内存事务系统兴起面临的主要难题.基于传统中心化架构的事务系统难以满足高并发、瞬时暴涨的业务需求,同时计算、网络能力的提升又推动了事务系统架构的变革.减少传统架构中的集中控制点、分摊负载热点成为消除事务扩展瓶颈的主要手段.

1) 改进全局时间戳分配方法

为了实现严格可串行化的隔离等级,并发的事务在决定执行顺序时依赖物理时间对照,事务在开始和提交时需分配对应时间戳,多版本事务系统中每个数据版本也需要时间戳加以区分.但是,分布式场景中节点之间物理时钟的误差只经过松散的同步则远大于数据传输的延迟,传统事务系统采用时间发号器统一全局物理时间.系统在时钟服务器上维护时间发号器,所有节点在事务开始、提交以及新数据版本写入时,都向时钟服务器请求获取时间戳.时钟服务器收到请求后,原子地将发号器加 1 并返回该请求,因此每个请求得到了全局唯一、和物理时间序吻合的全局时间戳.

上述做法显然会使时间发号器成为整个系统的瓶颈. Google 提出的地理上分布数据库 Spanner[10] 虽然是一个基于硬盘事务的分布式数据库,但其技术被广泛应用于之后的内存事务系统中.Spanner 采用了新的物理时钟同步方案 TrueTime,每个节点维护全局统一的物理时钟,避免获取时间戳时在关键路径上向集中式时钟服务器发送请求.TrueTime 通过周期性地向时钟服务器发送网络数据包,获得本地时钟和时钟服务器的时钟偏移.每次本地请求获取时间戳时,返回一个时间区间,代表当前时间点可能的时间范围$[L,U]$,区间长度的一半即为时钟误差.系统将时间戳分配给事务或数据版本时,将 U 作为当前时间戳,在休眠 $U-L$ 时间后分配给对应

对象并对系统中的其他节点可见,以保证该时间戳可见后不存在任何节点能获得比 U 更小的时间戳,实现全局时间戳分配的单调增长.Spanner 在地理上分布的数据中心场景中借用 GPS 实现了这一方案,时钟误差为 7 ms.但是,对于分布式事务,保证严格可串行化需要增加高达 7 ms 的执行延迟,这会降低系统吞吐率,影响用户体验. FaRM v2[36] 是 Microsoft 提出的一个基于 RDMA 的分布式计算平台,它延用 Spanner 的方法,在单个数据中心中应用 RDMA 技术加速时钟同步的网络传输,将时钟误差降低到 10 μs.

除了同步物理时钟,研究者们还关注如何增强时间发号器的可扩展性.布朗大学提出的分布式数据库 NAM-DB[33] 没有取消集中式的时间发号器,它通过 RDMA 提供的远程读写原语优化了时间发号器的可扩展性.NAM-DB 将原本时间发号器的单一变量扩展成 N 个变量,每个变量代表一个节点,整个变量组构成一个时钟向量.某节点请求获得一个数据版本时间戳时,只需远程修改时间发号器对应的变量即可,而为事务获得开始时间戳时,需要远程读取整个向量时钟.节点通过比较单个变量和整个向量时钟,就可以获得正确的逻辑顺序.读取、修改远端时间发号器变量可以利用 RDMA 提供的远程读写原语实现,该原语绕过了远端 CPU,直接读写远端内存,这将瓶颈从 CPU 端转移到网卡端,提升了时间发号器的扩展性. 实验结果显示该发号器可以支持到 500 台服务器的集群规模.但是,NAM-DB 的缺点是只能实现快照隔离性,不能实现可串行化和严格可串行化的隔离等级.

瑞士联邦理工学院提出的 Clock-SI[47] 同样是一个实现了快照隔离性的时钟同步方案,它没有一个中心化的时间发号器,每个节点都可以作为事务的协调者记录时间戳,所有的节点之间采用松散同步时钟(loosely synchronized clock)同步物理时间,即不借助额外的时钟同步技术同步节点之间的物理时间.Clock-SI 通过精心设计事务协议来避免松散同步时钟的不一致性影响事务快照读的正确性.Clock-SI 的协议规定,当一个事务读取一个远端节点的数据时,会提前获取本地时间戳 T 作为事务的快照时间戳,并将 T 和远端节点正在提交或已经提交的事务的时间戳比较,只有当 T 大于这些事务的时间戳时,才会返回读请求.通过推迟读请求的返回,Clock-SI 借助一个完全分布式的时钟实现了支持快照隔离性的事务系统.虽然 Clock-SI 实现在硬盘事务系统上,但其引入的分布式时间发号器的做法也可以迁移到内存事务中.Clock-SI 的缺点是读阻塞的时间取决于节点之间的时钟偏移,如果时钟偏差过大会降低只读事务的性能.

DST[37] 是上海交通大学提出的一个分布式时钟方案,它发现目前的并发控制协议如 OCC 或 2PL 已经通过数据间的冲突给出了一个事务之间可串行化的顺序,它通过在并发控制协议中捎带传输这个顺序,维护了一个可扩展的分布式时间戳.具体做法是,DST 为每个事务在开始时分配一个本地时间戳,当该事务在访问远端数据时,同时获取远端数据的时间戳,最后取远端时间戳和本地时间戳的最大值为该事务的提交时间戳.这种方案保证了一个事务如果读写其他事务的修改,就会被串行化到这些事务之后.文献[37]的作者将 DST 分别部署在 OCC 和 2PL 这 2 种协议上,通过 DST 使原有协议支持了数据的多版本机制,进而支持了一致的快照读,并控制快照读获取的数据的滞后时间不超过 2 倍的集群中最大物理时钟偏移.

Silo[40] 是麻省理工学院设计的基于多核机器的内存数据库,较早地提出了通过减少时间戳分配次数、分批处理事务的方式解决该问题.Silo 以 40 ms 为 1 个时间片分配 1 个时间戳,不同时间片之间的事务保证严格可串行化,同一时间片内的事务保证可串行化,但无法显式地表达执行顺序.如果用户要求严格可串行化,Silo 支持推迟事务的完成时间以满足该性质.通过这样的权衡,Silo 降低了时间发号器的竞争,提升了系统的吞吐率. 表 2 对以上 5 种时间戳分配技术进行了总结对比.

2) 减少热点数据竞争

热点数据在分布式系统中广泛存在,如何减少热点数据竞争、实现负载均衡是事务内存设计普遍面临的问题.缓存是解决热点数据竞争的常见手段,通过缓存读写数据既降低访存延迟,也减少热点数据的访问量.在内存事务中利用缓存需要注重保证缓存一致性.麻省理工学院提出的分布式内存并发控制协议 Sundial[35] 将并发控制和缓存结合,利用缓存来降低远程访问的开销.Sundial 对于每个数据对象引入了租约机制,在数据从远端缓存到本地的同时获得一个租约的期限,协议保证租约期限内数据的有效性.缓存协议要求节点对数据的修改以写通过的方式更新,即节点更新本地缓存的同时需要对远端数据进行修改.租约的引入避免了传统缓存协议中由于数据不断更新而产生频繁的缓存无效,

提升缓存命中率的同时减少了节点间的通信次数.DrTM 是一个结合 RDMA 和 HTM 的分布式事务处理系统[30],它在应用缓存技术时充分考虑了 RDMA 技术延迟低、绕过远端 CPU 的特点.DrTM 在读写远端数据时通过 RDMA 的远程读写原语完成查找索引、读写数据的操作.在分布式系统中维护缓存一致性需要较大开销,相比热点数据,热点的索引结构更容易被多节点频繁访问成为瓶颈.同时 RDMA 远程访问比普通网络带宽更高,DrTM 选择缓存数据存放位置而不是数据本身,降低了热点数据的查找开销并保证了数据对于所有节点的透明性,充分利用了 RDMA 技术的带宽优势.

Table 2 Comparison of Timestamp Allocation Techniques

表 2 时间戳分配技术对比

时钟名	是否需要等待	关键路径的信息交互次数	分布式时钟误差对性能影响	扩展性
TrueTime[10]	事务协调者在事务开始时进行不确定等待	0	大	中
NAM-DB[33]	否	1(只读事务) 2(读写事务)	无	低
Clock-SI[47]	事务参与者在执行事务时进行等待	0	大	高
DST[37]	否	0	小	高
Silo[40]	否	0	无	中

除了使用缓存,还可以将数据竞争分摊到不同的节点或线程上.多核内存数据库 Doppel[41] 通过本地维护热点数据的副本实现多核之间的负载分摊.Doppel 对于数据的更新记录在本地线程绑定的内存中,避免跨 NUMA 节点访问引起的开销.当有事务想要读取这些数据时,Doppel 会推迟这些读请求,在收集读请求数目达到阈值后,对分散在各线程上的数据进行归并整合,并返回这些读请求.需要注意的是,Doppel 需要数据的更新操作满足交换律时,才能支持事务进行数据热点的负载分摊.

Bamboo[46] 是一个基于传统 2 阶段锁优化的并发控制协议.对于热点数据的持锁更新会阻塞其他事务的执行,而 2 阶段锁要求事务在释放任何数据的锁之前必须获得事务执行需要的所有数据的锁,这进一步加剧了热点数据上的冲突和阻塞.Bamboo 允许事务在 2 阶段锁过程中对于数据提前释放锁,即允许某些事务读取尚未提交的“脏数据”.同时为了保证可串行化的隔离等级,Bamboo 会利用锁表记录这种数据间的依赖,当事务中止时对相关事务进行级联中止.但是,级联中止会使事务之间产生长依赖链,一旦出现会造成大量事务回滚,对于吞吐影响巨大.Bamboo 通过理论分析指出了级联中止的劣势和提前放锁的增益之间的临界点,并说明当数据库系统内数据规模远大于系统正在运行事务数量和事务访存规模时,Bamboo 的方案会取得性能优势.

3.2 利用新硬件加速事务处理

网络、存储能力的提升帮助分布式内存事务的延迟达到了亚微秒级别.面对新型硬件及技术,设计更适合的并发控制协议及事务系统架构、最大程度发挥它们的性能优势是系统设计者的当务之急.

1) 改进并发控制协议及事务系统架构

传统的并发控制协议为事务调度者设计了复杂的调度协议.FaRM 是 Microsoft 提出的一个基于 RDMA 的分布式计算平台[27-28],提供基于分布式事务的共享内存读写接口,它简化了协议并将 RDMA 技术运用到并发控制中.FaRM 事务的执行分为执行和提交阶段,图 3 包括 1 个事务调度者 C,3 个事务参与者 P_1,P_2,P_3 以及备份节点 B_1,B_2,B_3.在执行阶段,事务调度者 C 远程读取参与者 P_1,P_2,P_3 的数据,并执行事务的相应逻辑,将数据的修改保存在本地.在提交阶段,FaRM 对事务的写集数据通过远程过程调用向数据所在节点 P_1,P_2 申请锁并同时验证相应读集的数据,在获得所有写集的锁后,向包含数据仅属于读集的节点 P_3 发送请求,验证读集数据是否被修改.如果数据版本一致,则事务

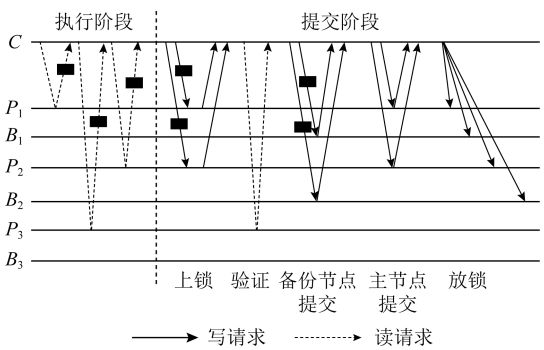


Fig. 3 Transaction processing in FaRM

图 3 FaRM 的事务处理流程

可以继续,调度者将数据的修改依次发送给各备份节点(B_1,B_2)及主节点(P_1,P_2),并提交给上层应用.FaRM 读取数据使用了 RDMA 的 read 原语,避免在关键路径上引入远端 CPU,节省了远端 CPU 开销,发挥了 RDMA 的优势.

布朗大学的研究者分析了已有内存事务架构的不足:shared-nothing 架构为了避免网络通信,节点之间不共享数据,这难以满足目前真实应用涉及数据众多、数据访问特征变化频繁的特点;shared-memory 架构需要成熟的缓存一致性协议,这是一般事务系统不具备的,另外事务管理模块往往希望能对内存数据进行全方位的管理,而 shared-memory 只能提供有限的接口.因此他们提出了适用于 RDMA 网络的 NAM 架构[48].NAM 架构分离了计算节点和存储节点,计算节点负责事务的执行逻辑,存储节点负责数据的存储、响应读写数据的请求.相比于之前的架构,存储层可以独立于计算节点支持更大规模的扩展,且存储层可以嵌入负载平衡策略动态支持数据迁移.NAM 架构在存储节点之间交互数据时使用 RDMA 单边操作,减少了存储层的 CPU 使用率.

2) 充分发挥 RDMA 性能特性

RDMA 技术的迅速发展促进了更多关于内存事务通信优化的研究.研究发现在分布式场景中使用 RDMA 进行通信,网卡管理的通信链路越多,网卡的缓存命中率就越低,通信延迟越高,吞吐率越低.因此,卡内基梅隆大学的 Kalia 等人[31]提出在节点较多的数据中心中,使用 RDMA 的 UD 模式代替常用的 RC 模式.UD 模式可以利用较少的通信链路实现任意节点之间的通信,网卡吞吐率不受节点扩展的影响.UD 模式的局限在于底层无拥塞控制逻辑,但是他们声称在实际数据中心的测试中丢包的现象根本不会出现.所以他们设计了分布式事务系统 FaSST,采用 RDMA 的 UD send 原语代替 RC read/write,以 FaRM 的并发控制协议为基础设计了分布式内存事务系统,在 OLTP 负载下吞吐率达到 FaRM 的 1.87 倍.然而,仅从性能的角度考虑 RDMA 技术的应用不够全面,内存占用、CPU 使用率等指标同样应该纳入系统的评价中,而后两者更能得益于 RDMA 提供的独特的单边原语操作.

DrTM+H 是上海交通大学提出的一个混合分布式事务系统[34],尝试在事务运行的每个阶段都选择最适合的 RDMA 操作来缩短事务的执行时间.DrTM+H 以 OCC 为并发控制协议,将事务的运行分为执行、验证、日志记录和提交 4 个阶段.RDMA 的操作包括单边操作 RDMA read/write、单边原子操作 RDMA CAS 和双边操作 RDMA send(即 RPC 调用,远端 CPU 配合执行相应读写并返回结果).在执行阶段,采用单边操作 RDMA read 和双边操作 RDMA send 结合的方式更加高效,当本地缓存了远端数据位置时,通过单边操作可以在一个往返时延中获取数据,反之,则应该尽早让远端 CPU 介入寻找数据;在验证阶段,使用 RDMA read 和 RDMA CAS 更合适,因为需要验证的数据位置之前往往已经被缓存了;在记录日志阶段,简单的记录日志操作可以直接由 RDMA write 完成,无需引入远端 CPU;在提交阶段,双边操作可以通过捎带信息的方式合并请求和回复的信息,减少通信次数,因此比单边操作更适合本阶段.

3.3 降低事务中止概率

事务在并发执行中由于数据竞争会产生资源等待.为了避免资源的循环等待导致的死锁,大部分并发控制协议会在资源竞争时中止某个事务以保证其他事务的顺利执行,被中止的事务则会在等待一段时间后重新发起该事务.事务频繁中止不仅会影响系统的吞吐率,还可能出现事务"饥饿"的问题.减少事务的中止概率对于内存事务系统十分重要.

1) 采取更加灵活的提交策略

传统基于时间戳的内存事务系统会在事务的提交阶段获取提交时间戳并验证事务是否能提交,一旦发现数据不一致,则事务中止.这种确定性的事务执行流程造成了很多"虚假"冲突,实际上系统可以为并发的事务确定可串行化顺序.图 4 有 2 个事务 A,B 和 2 个数据 x,y,纵轴代表了逻辑时间.事务 B 在事务 A 读 x 后写 x 并成功提交.在事务 B 提交后,事务 A 写 y 并准备提交,系统发现此时 x 已经被 B 修改过,验证失败,事务 A 被中止.实际上,系统可以将事务 A 的提交时间点提前到时刻 1,即在串行化顺序中先完成事务 A 后完成事务 B,避免了事务 A 的中止.麻省理工学院提出的新型乐观并发控制协议 TicToc[42] 利用了这个特点,标记了每个数据的有效时间区间.事务在执行阶段会获得相应数据的有效时间区间.当事务进入提交阶段,事务会对所有获得的有效时间区间求并集,若并集不为空,则可以选择并集内任一时间作为提交时间点顺利提交,否则需要进一步验证.TicToc 通过在事务提交阶段灵活确定事务提交时间点降低了事务的中止率.此外,TicToc 还使用逻辑时间代替物理时间标记

有效时间区间,避免了使用时间发号器带来的扩展性问题.实验显示 TicToc 相比传统基于乐观并发控制协议的事务系统降低了 3.3 倍的事务中止率.TicToc 灵活地选取了事务的提交时间,本质上是通过弱化事务提供的隔离等级来降低事务中止率.

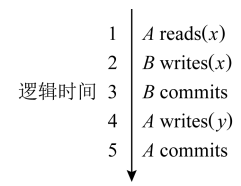


Fig. 4 An example of transaction executing using TicToc

图 4 TicToc 中事务执行例子

香港大学提出的地理上分布的事务系统 DAST[38] 侧重通过推后提交时间解决事务的延迟问题.跨地域事务由于涉及多个数据中心,一旦中止重试延迟增加明显;本地域事务的执行,由于跨地域事务执行时间长,会被阻塞导致延迟上升.因此,DAST 提出了一种灵活的提交策略,通过推后跨地域事务的提交时间避免其重试.跨地域事务先通过一个准备阶段,让事务的参与者独立地预估该事务在本节点的提交时间,事务选取预估提交时间的最大值作为真实提交时间.预估时间保证了跨地域事务不会因为生成过小的时间戳导致中止.对于本地域事务,生成对应的逻辑时间戳,支持其在跨地域事务执行之前穿插执行,以避免其被跨地域事务阻塞.通过这 2 个技术,DAST 最高降低了跨地域事务 70.4%的尾延迟以及本地域事务 93.2%的尾延迟.

2) 提供多版本避免事务中止

单版本的乐观并发控制协议被广为诟病的问题是频繁的事务中止.对于只读事务,单版本协议往往需要 2 次读取读集的数据,才能正确提交事务.多版本协议对于一个数据维护了多个历史版本,处理只读事务时,读取小于只读事务提交时间戳的最新版本的数据就可以满足数据的一致性.因此,多版本协议通过支持读取历史版本降低事务中止率,对读密集的负载更友好.

Hekaton[39] 是早期多版本技术在内存数据库中的重要尝试,它通过 3 阶段提交实现了多版本并发控制协议.Hekaton 避免了数据上的锁结构(如自旋锁),采用原子指令 CAS 保证数据一致.在数据布局上,数据按线程物理地划分,每个线程作为事务执行主体,维护事务的状态及日志信息.对于每个数据版本,Hekaton 维护了它的开始时间和结束时间.然而,Hekaton 维护多版本时采用的链表结构降低了缓存命中率,并产生了较大的存储和计算开销.同时,Hekaton 在读取未提交事务修改的数据时采取的预测读策略引入了事务级联中止的问题.

Cicada 是卡内基梅隆大学设计的多核内存数据库[43],针对传统多版本事务系统提出了一些解决方案.①尽力而为地将事务最频繁访问的数据版本缓存在固定位置,增加版本遍历的一次命中率.②采取了"早中止"策略,通过维护更多数据版本信息,在事务执行和验证阶段尽早判断事务是否会由于冲突中止.③通过自学习的方式确定事务中止后的重试时间避免事务频繁重试导致的中止率的提升.Cicada 在多核场景中相比其他系统取得了 3 倍的吞吐率提升,但是其仍存在部分问题.Cicada 的时间戳分配方案使得事务的执行无法满足严格的物理时间序,甚至会出现一个线程读不到本线程已提交事务的情况.

然而,在分布式场景中多版本并发控制协议并不是主流实现.这是因为频繁地比较、修改数据的版本号等信息会增加节点之间的交互次数,相比于多核间的跨核访问,网络通信开销更大,如何在分布式场景中简化多版本并发控制协议,是系统设计者亟待解决的难题.

3) 在硬件事务内存中减少事务中止次数

DrTM 尝试在基于硬件事务内存的分布式事务系统中降低事务的中止概率.RTM 作为英特尔的硬件事务内存产品,通过 CPU 的私有缓存追踪事务读写集,将事务的支持下沉到硬件层面.然而,RTM 对于分布式事务的支持非常局限,因为网络 I/O 会中止本地 RTM 正在执行的事务.为了减少分布式事务执行的中止率,DrTM 采用 2PL 协议,通过 RDMA 将远程的读写集数据加锁并搜集到本地,再通过 RTM 执行本地事务,最后将远程的修改通过 RDMA 写回.这种方式可以避免 RTM 执行事务时需要访问远端数据而中止事务.更进一步,DrTM+R[32] 针对 DrTM 中存在的读写集必须在事务执行前提前确定的问题,基于 OCC 修改了分布式事务协议,在执行阶段确定读写集,在提交阶段进行验证提交.此外,还有一些工作针对 RTM 读写集大小受限的问题进行了探索,提出了拆分读写集、只用 RTM 保护元数据等方法.这些工作都从降低硬件事务内存中止率的角度出发优化内存事务.

4) 中止后保证透明性避免重试事务再次中止

FaRM v2 分布式事务系统聚焦于为事务使用

者提供透明性的保证.成功提交的事务往往能保证读取数据的一致性,但是中止的事务却无法满足这一点.在乐观并发控制协议中,大部分事务的中止源于版本验证失败,这表示事务在中止后向上层应用提供的是不一致的读数据.用户难以通过不一致的数据判断事务中止的具体原因,很可能造成之后重试的事务再次中止,加重系统不必要的负担.FaRM v2 将 FaRM 的单版本协议改进为多版本协议,在事务的开始阶段分配时间戳,事务读取数据时依据该时间戳,在所读数据的历史版本中寻找旧于该时间戳的最新版本数据,保证事务读到系统在该时间戳的快照.即使事务在执行阶段提前中止或在提交阶段因为验证失败而中止,事务所读到的数据都能保持一致性.FaRM v2 这种分布式多版本事务系统和 Cicada 这种单机多版本数据库的最大区别在于:前者是在分布式 OCC 协议基础上进行延伸,通过多版本支持一致的快照读和透明性;后者是以时间戳为并发控制协议的核心,利用时间戳标记多个数据版本,以解决数据间的冲突.

3.4 高效保证事务持久性

实现内存事务的持久性需要事务将数据的修改记录在持久性介质中.磁盘或固态盘作为持久性介质,已经无法匹配内存事务的执行速度,难以提供高速的持久化解决方案;在分布式场景中,并发控制协议也需要协调各个节点所持久化的数据,保证机器损坏、掉电后数据的持久性和系统的可用性.

1) 利用 NVM 为单机事务保证持久性

传统基于硬盘的内存事务持久化方案无法满足如今上层应用的性能需求,NVM 的出现和相关产品的问世给单机事务的持久化带来了机遇.NVM 具有延迟低、支持字节粒度寻址、掉电不易失的特性,但直接用 NVM 替换内存是不可取的.持久化数据到 NVM 上需要调用持久化指令(如 CLFLUSHOPT,CLWB 指令)将数据从 CPU 缓存刷写到 NVM,再通过内存栅栏指令(如 SFENCE)保证指令顺序执行,2 种指令的序列化操作增加了开销.数据的持久化可以采用日志写的方法,包括撤销日志(undo log)和重做日志(redo log),保证日志持久化到 NVM.清华大学提出的持久化事务架构 DUDETM[44] 结合 2 种日志的特点,将事务的持久化拆解为执行、持久、再现 3 个独立步骤,异步批量处理完成持久化,减少了持久化指令和内存栅栏指令这 2 种指令次数,提升持久化事务吞吐率.但是,DUDETM 中存在再现线程不可扩展的问题.针对这一问题,上海交通大学提出的持久化事务内存 Pisces[45] 采用双版本并发控制,将数据的日志作为一个临时副本,减少维护多版本存在的空间和查找开销.此外,Pisces 采用 3 阶段提交协议,将事务的提交划分成持久化、并发提交、写回 3 阶段,让数据更新的持久化在提交第 1 阶段完成,并发提交阶段事务原子地获取提交时间戳进行提交,在写回阶段事务将持久化的数据更新日志写回数据原地.

加州大学默塞德分校的研究者们针对基于 NVM 的内存事务性能进行了优化,提出了 ArchTM[49].利用 NVM 实现事务持久性的方法有 2 种,记录日志和异地更新,前者会写 2 倍的数据增加对 NVM 的负载,后者由于元数据更新产生的随机小写请求不符合 NVM 擅长的访存特性. 因此,ArchTM 提出了将元数据更新放置于内存中,将事务的更新以异地写的方式保存在 NVM 中的策略. 为了保证持久性,ArchTM 在每个数据上都标记了其版本对应的事务序号,用于事务系统的崩溃恢复;为了将事务系统对 NVM 的写聚合成 NVM 擅长的顺序写,ArchTM 设计了全局空闲空间队列和后台定期整理碎片的机制,将一起到来的空间分配请求尽量分配到连续 NVM 空间,提高事务性能.

2) 利用副本为分布式事务保证持久性

在分布式场景中,副本是保证数据持久性最常见的手段.相比文件系统、键值存储的副本策略,事务系统的持久化对一致性的要求更高,很多要求在事务执行过程中完成持久化操作.Spanner 借助 Paxos 共识协议实现了多个副本数据的强一致性,并将持久化时间隐藏在分配时间戳的休眠时间中.FaRM 采用主从备份的方案,每个节点组利用 $f+1$ 个节点容纳 f 个错误.如图 3 中,在提交阶段先将事务的修改提交给各备份节点,当收到所有的确认消息后,再提交给数据的主节点,完成事务的持久化.在事务提交到所有备份节点后,FaRM 要求事务必须在一个主节点成功提交后才能向上层提交,这是为了保证在每个节点组都出现 f 个错误后已提交的事务仍能顺利恢复.如果提前向上层提交事务,恢复后可能丢失事务已经通过验证阶段的信息,进而中止已提交的事务.

与前 2 例不同,华盛顿大学通过独特的思路解决事务持久化问题.他们认为在事务系统保证强一致性的前提下,再使用共识协议保证副本的一致性是一种过度的保护,会影响系统的吞吐率.他们提出的事务并发控制协议 TAPIR[29] 基于弱一致性的副本

协议,每个节点组没有中心化的协调者.客户端作为事务的协调者,将事务的读发送到节点组中距离最近的节点上,写缓存在本地,在提交阶段再分发给节点组中的所有节点.在事务的提交阶段,TAPIR 利用时间戳作为依据,要求副本节点对事务的操作顺序达成共识.通过这样的协议,在常规路径下,TAPIR 将 Spanner 中事务执行的至少 2 个往返时延(Spanner 中操作由事务调度者发送给事务参与者,再由事务参与者发送给备份节点)减少到 1 个往返时延(直接由事务调度者发送给所有事务参与者及备份节点).

3.5 实验验证

为了展示不同并发控制协议对于内存事务性能的影响,本节以开源数据库 Deneva[14] 为例进行了性能实验.Deneva 数据库采用以太网传输,为适配新兴网络技术,我们重构了 Deneva 的网络传输层,使用基于 Infiniband 网络的 RDMA 技术进行消息通信.我们使用 4 台服务器测试了数据库常用的 YCSB[50] 和 TPC-C[51] 两种负载.

图 5 展示了 5 种并发控制协议在 YCSB 负载中随偏移因子变化的系统吞吐,偏移因子越大代表数据的冲突越密集.在数据访问较均衡时,5 种协议的性能相近.当数据冲突初步增加后,基于时间戳的 5 种并发控制协议 TIMESTAMP 和 MVCC 性能优异,这得益于它们在执行过程中避免了持锁的临界区,允许事务在不违反隔离等级的前提下以对应的时间戳插入串行化序列中.当数据冲突进一步升级后,事务中止率上升,系统吞吐下降,各种并发控制协议的性能表现趋同.

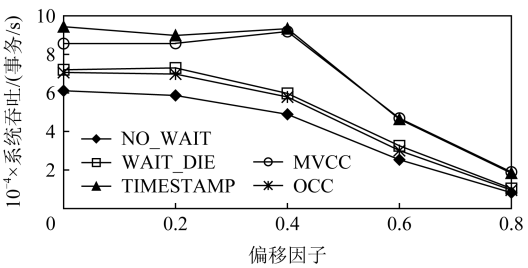


Fig. 5 Throughput when varying the skew factor in YCSB

图 5 YCSB 中偏移因子变化下的吞吐性能

图 6 展示了各种协议执行 TPC-C 负载中 2 种事务 new_order 和 payment 的性能.这 2 种事务都包含对数据条目的读—修改—写,且执行周期长.OCC 的验证由于执行周期长而导致频繁失败,2PL 由于持锁时间长会阻塞其他事务的执行,因此这 2 种协议性能低下;TIMESTAMP 和 MVCC 在这种场景中表现优异.

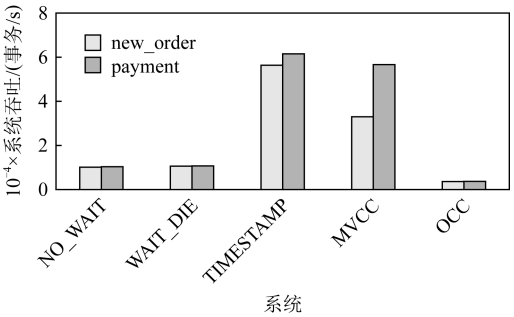


Fig. 6 Throughput of two kinds of transactions in TPC-C

图 6 TPC-C 中 2 种事务的吞吐性能

3.6 小 结

最近 10 年内存事务的并发控制协议,都聚焦于性能、扩展性、持久性这 3 个维度,从不同技术路线入手对并发控制协议进行了探索.这些工作主要遵循 2 条设计思路:1)利用新型技术、设备去减少内存事务系统中遇到的瓶颈,包括 NAM-DB 利用 RDMA 优化中心时钟、DrTM 利用 RTM 实现高效的分布式事务.它们充分分析并发控制协议中所遇到的瓶颈环节,借助新技术提供的优势解决相关问题.2)使用已有技术去解决新场景中的相应问题,包括 Sundial 利用缓存缓解热点数据冲突、Cicada 提供多版本减少事务中止次数等.随着网络、存储能力的提升,原有的事务处理模型和事务系统变得低效,在并发控制协议里使用一些更加适应当前场景的技术可以取得较好的效果.此外,还有少量工作依据上层应用或负载,对并发控制协议进行针对性的优化.

根据第 1 节提到的内存事务并发控制协议目前面临的 3 个挑战,现有技术的应对方案可以进行如下总结.单机多核数据库中跨分区事务不可避免地产生大量跨 NUMA 节点的访问,降低事务中止概率(3.3 节)减少了事务中止后的重试次数,降低了事务跨 NUMA 节点访问次数.这不仅减少了本事务的延迟,还通过减少跨核访问次数减轻了该事务对其他资源竞争事务的冲突.分布式事务受集中控制点和数据热点的制约难以扩展,现有技术通过改进时间戳分配(3.1.1 节)、并发控制协议及事务框架(3.2.1 节)将集中控制点移出事务的关键路径,通过缓存、聚合请求、提前放锁(3.1.2 节)和使用新硬件(3.2.2 节)等方式增加热点数据的并发.针对现有协议难以发挥新设备优势来保证高效的持久性,研究

者从单机事务和分布式事务2个维度出发,分别结合非易失内存和副本重新设计并发控制协议并进行相应优化(3.4节).

4 未来研究方向

随着业务需求和使用场景的进一步变化,内存事务中一些问题会变得更加突出.结合本文提出的内存事务面临的挑战,其未来的研究热点方向包括3点:

1) 设计感知负载特征的并发控制协议.短期内服务器单机多核的架构不会发生革命性的变动,跨NUMA节点的通信开销仍是一个不可忽视的问题.目前已有不少工作从并发控制层面尝试减少热点数据的同步操作和跨NUMA的访问,但是结合OLTP等上层负载特征优化数据布局的思路还缺乏实践.目前内存事务系统多数采用物理分区的方式平均划分数据,避免共享内存的引入.这种方式减少了核间竞争,但带来了多核、多节点负载不均的问题.面对不同负载,内存事务系统应当支持更灵活的数据分布策略,针对负载热点进行相应数据的动态迁移.如何在该场景中权衡迁移带来的性能增益和一致性开销,是研究者面临的挑战.更进一步,研究者可以结合负载特征设计混合、自适应的并发控制协议,根据负载变化调整协议解决冲突的方式,如自适应地使用锁或时间戳解决冲突.2PL适合冲突频繁的负载,OCC在短事务、较少冲突场景中性能更佳,MVCC适合读事务主导的负载,系统可以结合各自协议特点适应不同负载.

2) 解决地理上分布系统中事务的扩展性问题.数据库的扩展性问题一直是研究者面临的重要挑战,在初步解决了单核到多核、单机到分布式的扩展性问题后,地理上分布数据库的扩展性受到了更多关注.容灾需求的发展迫使数据库支持多地备份,而在单机或单个数据中心场景中,核间通信和网络通信的开销已经减小,但是跨地域的网络通信仍存在较大开销.因此,地理上分布的数据库不能接受集中控制点的设计,也难以在多次信息交互的协议中提供低延迟.Spanner为地理上分布的场景提出了一个解决方案,但存在跨地域交互过多、依赖物理硬件等不足,所以减少事务执行时跨地域的网络交互更加重要.未来的研究路线可以从3方面入手:①进一步耦合并发控制协议和副本协议,从协议角度减少跨地域的交互,将关键路径上的同步交互改为异步交互;②通过合理的数据布局减少跨地域事务的数量,利用多副本的读写带宽,在保证一定程度一致性的前提下,灵活切换数据分布,多节点同时提供服务;③通过NVM等高速存储器件代替传统持久化介质,在跨地域事务执行时提供持久化保证,加速事务的处理及灾后恢复进程.

3) 探索内存事务的服务质量保证方法.在用户追求事务系统高吞吐率、低延迟的同时,长久以来用户也关注着事务的服务质量.用户难以接受小部分事务阻塞时间过长,内存事务系统的长尾延迟问题亟待解决.长尾延迟的原因包括事务由于请求不到资源的频繁中止、事务重试频率过高造成冲突加剧、事务在热点数据上竞争过多等.并发控制协议可以合理地调整节点执行本地事务和服务其他节点的数据请求的顺序,提前预测可能出现的耗时操作并优先执行,来降低内存事务的长尾延迟.

5 相关工作

数据库及系统研究者针对近年来兴起的内存事务进行了广泛的研究,形成了一些关于并发控制协议的综述文章.针对多核场景,中国人民大学的研究[52]以PostgreSQL的优化技术为依托,着重分析多核架构对内存事务的影响,从锁管理、日志管理、缓冲区管理等多方面描述适配多核场景的新兴技术;新加坡国立大学的研究[53]从协议设计、数据存储、垃圾回收和索引管理等角度比较了现有的单机多版本并发控制协议,并在同一框架内实现验证;华东师范大学的研究[54]对比了现有内存事务并发控制协议和传统基于磁盘协议的差异,以OceanBase为例从并发控制和日志恢复2个角度分析现有技术.针对分布式场景,Deneva[14]针对分布式事务,选取多个经典的并发控制协议进行总结对比,通过实验指出不同负载下性能最优的协议;分布式事务中分区策略会影响并发控制协议的性能,印度浦那工程学院的研究[55]总结了各种分区策略,并对比了不同协议采用不同分区策略的效果.

相比以上综述,本文结合硬件发展趋势,分析了多核架构、机器规模、新型器件对内存事务带来的挑战,着重对比分析了单机事务和分布式事务的不同优化策略.优化内存事务的技术思路在单机和分布式场景是相通的,但实践方式却大不相同,本文将这些技术汇总比较,指出研究者面对不同应用场景

使用技术时的考虑,为之后系统设计者进一步优化提供参考.

6 总 结

为了应对内存事务中并发控制协议所面临的机遇与挑战,研究者主要采用如下技术思路来开展研究工作:1)通过改进全局时间戳分配方法、减少热点数据竞争等方法来消除事务系统的扩展瓶颈.2)通过改进并发控制协议、选取恰当的 RDMA 原语等方法来利用新硬件加速事务处理.3)通过采取更加灵活的提交策略、提供多版本避免事务中止、在硬件事务内存中减少事务中止次数、实现事务透明性避免重试事务再次中止等方法来降低事务中止概率.4)利用非易失内存和副本来高效保证单机和分布式事务的持久性.

结合业务应用场景和硬件工艺的发展趋势,内存事务中并发控制协议的未来研究预期将聚焦如下方向:设计感知负载特征的并发控制协议、解决地理上分布的场景中事务系统的扩展性问题,以及保证内存事务的服务质量等.

作者贡献声明:姜天洋负责调研文献、撰写全文并完成相关实验;张广艳负责确定论文思路、设计文章框架并讨论修改全文;李之悦负责实现代码并协助分析实验结果.

参 考 文 献

[1] Wikipedia. Database transaction [OL]. [2021-07-03]. https://en.wikipedia.org/wiki/Database_transaction

[2] Bernstein P A, Hadzilacos V, Goodman N. Concurrency Control and Recovery in Database Systems [M]. New York: Addison-Wesley, 1987

[3] Oracle. Oracle cloud [OL]. [2021-07-03]. https://www.oracle.com/index.html

[4] Microsoft. SQL server [OL]. [2021-07-03]. https://www.microsoft.com/zh-cn/sql-server

[5] Codd E F. A relational model of data for large shared data banks [M] //Software Pioneers. Berlin: Springer, 2002: 263-294

[6] Redis. Redis database [OL]. [2021-07-03]. https://redis.io/

[7] Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage [M]. Sebastopol, CA: O'Reilly Media, Inc, 2013

[8] Cattell R. Scalable SQL and NoSQL data stores [J]. ACM SIGMOD Record, 2011, 39(4): 12-27

[9] Baidu. OLTP [OL]. [2021-07-03]. https://baike.baidu.com/item/OLTP/5019563

[10] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally distributed database [J/OL]. ACM Transactions on Computer Systems, 2013, 31(3). [2021-07-03]. https://dl.acm.org/doi/pdf/10.1145/2491463

[11] Taft R, Sharif I, Matei A, et al. CockroachDB: The resilient geo-distributed SQL database [C] //Proc of the 46th Int Conf on Management of Data. New York: ACM, 2020: 1493-1509

[12] Pavlo A, Aslett M. What's really new with NewSQL? [J]. ACM SIGMOD Record, 2016, 45(2): 45-55

[13] Aliyun. PolarDB [OL]. [2021-07-03]. https://developer.aliyun.com/article/778595

[14] Harding R, Van Aken D, Pavlo A, et al. An evaluation of distributed concurrency control [J]. Proceedings of the VLDB Endowment, 2017, 10(5): 553-564

[15] Yang Jian, Kim J, Hoseinzadeh M, et al. An empirical guide to the behavior and use of scalable persistent memory [C] // Proc of the 18th USENIX Conf on File and Storage Technologies (FAST 20). Berkeley, CA: USENIX Association, 2020: 169-182

[16] Gray J, Reuter A. Transaction Processing: Concepts and Techniques [M]. Amsterdam: Elsevier, 1992

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

[18] Wikipedia. MySQL [OL]. [2021-07-03]. https://zh.wikipedia.org/wiki/MySQL

[19] DPDKProject. DPDK [OL]. [2021-07-03]. https://www.dpdk.org/

[20] Wikipedia. RDMA [OL]. [2021-07-03]. https://en.wikipedia.org/wiki/Remote_direct_memory_access

[21] Kalia A, Kaminsky M, Andersen D G. Design guidelines for high performance RDMA systems [C] //Proc of the 22nd USENIX Annual Technical Conf (USENIX ATC 16). Berkeley, CA: USENIX Association, 2016: 437-450

[22] Wikipedia. Transactional synchronization extensions [OL]. [2021-07-03]. https://en.wikipedia.org/wiki/Transactional_Synchronization-_Extensions#RTM

[23] Adve S V, Gharachorloo K. Shared memory consistency models: A tutorial [J]. Computer, 1996, 29(12): 66-76

[24] Bernstein P A, Goodman N. Multiversion concurrency control—Theory and algorithms [J]. ACM Transactions on Database Systems, 1983, 8(4): 465-483

[25] Berenson H, Bernstein P, Gray J, et al. A critique of ANSI SQL isolation levels [J/OL]. ACM SIGMOD Record, 1995, 24(2): 1-10

[26] Lu Haonan, Hodsdon C, Ngo K, et al. The SNOW theorem and latency-optimal read-only transactions [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation (OSDI 16). Berkeley, CA: USENIX Association, 2016: 135-150

[27] Dragojević A, Narayanan D, Castro M, et al. FaRM: Fast remote memory [C] //Proc of the 11th USENIX Symp on Networked Systems Design and Implementation (NSDI 14). Berkeley, CA: USENIX Association, 2014: 401-414

[28] Dragojević A, Narayanan D, Nightingale E B, et al. No compromises: Distributed transactions with consistency, availability, and performance [C] //Proc of the 25th Symp on Operating Systems Principles. New York: ACM, 2015: 54-70

[29] Zhang I, Sharma N K, Szekeres A, et al. Building consistent transactions with inconsistent replication [J/OL]. ACM Transactions on Computer Systems, 2018, 35(4) [2021-07-03]. https://dl.acm.org/doi/pdf/10.1145/3269981

[30] Wei Xingda, Shi Jiaxin, Chen Yanzhe, et al. Fast in-memory transaction processing using RDMA and HTM [C] //Proc of the 25th Symp on Operating Systems Principles. New York: ACM, 2015: 87-104

[31] Kalia A, Kaminsky M, Andersen D G. FaSST: Fast, scalable and simple distributed transactions with two-sided (RDMA) datagram RPCs [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation (OSDI 16). Berkeley, CA: USENIX Association, 2016: 185-201

[32] Chen Yanzhe, Wei Xingda, Shi Jiaxin, et al. Fast and general distributed transactions using RDMA and HTM [C/OL] // Proc of the 11th European Conf on Computer Systems. New York: ACM, 2016 [2021-07-03]. https://dl.acm.org/doi/pdf/10.1145/2901318.2901349

[33] Zamanian E, Binnig C, Harris T, et al. The end of a myth: Distributed transactions can scale [J]. Proceedings of the VLDB Endowment, 2017, 10(6): 685-696

[34] Wei Xingda, Dong Zhiyuan, Chen Rong, et al. Deconstructing RDMA-enabled distributed transactions: Hybrid is better! [C] //Proc of the 13th USENIX Symp on Operating Systems Design and Implementation (OSDI 18). Berkeley, CA: USENIX Association, 2018: 233-251

[35] Yu Xiangyao, Xia Yu, 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

[36] Shamis A, Renzelmann M, Novakovic S, et al. Fast general distributed transactions with opacity [C] //Proc of the 45th Int Conf on Management of Data. New York: ACM, 2019: 433-448

[37] Wei Xingda, Chen Rong, Chen Haibo, et al. Unifying timestamp with transaction ordering for MVCC with decentralized scalar timestamp [C] //Proc of the 18th USENIX Symp on Networked Systems Design and Implementation (NSDI 21). Berkeley, CA: USENIX Association, 2021: 357-372

[38] Chen Xusheng, Song Haoze, Jiang Jianyu, et al. Achieving low tail-latency and high scalability for serializable transactions in edge computing [C] //Proc of the 16th European Conf on Computer Systems. New York: ACM, 2021: 210-227

[39] 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

[40] Tu S, Zheng Wenting, Kohler E, et al. Speedy transactions in multicore in-memory databases [C] //Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 18-32

[41] Narula N, Cutler C, Kohler E, et al. Phase reconciliation for contended in-memory transactions [C] //Proc of the 11th USENIX Symp on Operating Systems Design and Implementation (OSDI 14). Berkeley, CA: USENIX Association, 2014: 511-524

[42] Yu Xiangyao, Pavlo A, Sanchez D, et al. Tictoc: Time traveling optimistic concurrency control [C] //Proc of the 42nd Int Conf on Management of Data. New York: ACM, 2016: 1629-1642

[43] Lim H, Kaminsky M, Andersen D G. Cicada: Dependably fast multi-core in-memory transactions [C] //Proc of the 43rd Int Conf on Management of Data. New York: ACM, 2017: 21-35

[44] Liu Mengxing, Zhang Mingxing, Chen Kang, et al. DudeTM: Building durable transactions with decoupling for persistent memory [J]. ACM SIGPLAN Notices, 2017, 52(4): 329-343

[45] Gu Jniyu, Yu Qianqian, Wang Xiayang, et al. Pisces: A scalable and efficient persistent transactional memory [C] // Proc of the 25th USENIX Annual Technical Conf (USENIX ATC 19). Berkeley, CA: USENIX Association, 2019: 913-928

[46] Guo Zhihan, Wu Kan, Yan Cong, et al. Releasing locks as early as you can: Reducing contention of hotspots by violating two-phase locking [C] //Proc of the 47th Int Conf on Management of Data. New York: ACM, 2021: 658-670

[47] Du Jiaqing, Elnikety S, Zwaenepoel W. Clock-SI: Snapshot isolation for partitioned data stores using loosely synchronized clocks [C] //Proc of the 32nd IEEE Int Symp on Reliable Distributed Systems. Piscataway, NJ: IEEE, 2013: 173-184

[48] Binnig C, Crotty A, Galakatos A, et al. The end of slow networks: It's time for a redesign [J]. Proceedings of the VLDB Endowment, 2016, 9(7): 528-539

[49] Wu Kai, Ren Jie, Peng I, et al. ArchTM: Architecture-aware, high performance transaction for persistent memory [C] //Proc of the 19th USENIX Conf on File and Storage Technologies (FAST 21). Berkeley, CA: USENIX Association, 2021: 141-153

[50] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB [C] //Proc of the 1st ACM Symp on Cloud computing. New York: ACM, 2010: 143-154

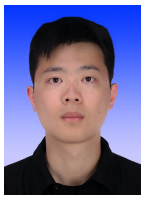
[51] TPC. TPC benchmark C [OL]. [2021-07-03]. http://www.tpc.org/tpcc/

[52] Zhu Yue'an, Zhou Xuan, Zhang Yansong, et al. A survey of optimization methods for transactional database in multi-core era [J]. Chinese Journal of Computers, 2015, 38(9): 1865-1879 (in Chinese)
(朱阅岸, 周烜, 张延松, 等. 多核处理器下事务型数据库性能优化技术综述[J]. 计算机学报, 2015, 38(9): 1865-1879)

[53] Wu Yingjun, Arulraj J, Lin Jiexi, et al. An empirical evaluation of in-memory multi-version concurrency control [J]. Proceedings of the VLDB Endowment, 2017, 10(7): 781-792

[54] He Xiaolong, Ma Haixin, He Yukun, et al. Technology and implementation of a new OLTP system [J]. Journal of East China Normal University: Natural Sciences, 2018 (5): 107-119 (in Chinese)
(贺小龙, 马海欣, 何毓锟, 等. 新型 OLTP 系统的技术与实践[J]. 华东师范大学学报: 自然科学版, 2018, (5): 107-119)

[55] Bharati R D, Attar V Z. A comprehensive survey on distributed transactions based data partitioning [C/OL] // Proc of the 4th Int Conf on Computing Communication Control and Automation (ICCUBEA). Piscataway, NJ: IEEE, 2018 [2021-07-03]. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8697589



Jiang Tianyang, born in 1996. PhD candidate. His main research interests include storage systems and distributed systems.
姜天洋, 1996 年生. 博士研究生. 主要研究方向为存储系统和分布式系统.

Zhang Guangyan, born in 1976. PhD, associate professor, PhD supervisor. Distinguished member of CCF. His main research interests include big data computing, network storage system and distributed computing.
张广艳, 1976 年生. 博士, 副教授, 博士生导师. CCF 杰出会员. 主要研究方向为大数据计算、网络存储系统和分布式计算.

Li Zhiyue, born in 1998. Master candidate. His main research interests include transaction systems and storage arrays.
李之悦, 1998 年生. 硕士研究生. 主要研究方向为事务系统和存储阵列.