文章编号: 1000-5641(2016)05-0018-09

内存数据库事务提交的关键技术与挑战

胡 爽, 周 欢, 钱卫宁

(华东师范大学 数据科学与工程研究院, 上海 200062)

摘要: ARIES 作为传统事务提交机制从 20 世纪 90 年代问世以来, 一直是主流商业或开源数据库系统普遍采用的方法. 随着应用的高通量化, 基于传统硬件设备实现的事务提交机制成为了系统性能提升的首要瓶颈. 然而大内存、多核等高性能硬件技术的发展又为事务提交机制的优化提供了新的契机. 本文详细分析并总结了传统事务提交机制中存在的问题, 然后归纳并讨论了现有基于新型硬件的事务提交技术的应用现状与优缺点, 最后探讨了事务提交优化的未来发展与挑战.

关键词: 内存数据库; 事务提交; 日志处理技术

中图分类号: TP391 **文献标识码**: A **DOI:** 10.3969/j.issn.1000-5641.2016.05.003

Key techniques and challenges of transaction commit in main-memory database systems

HU Shuang, ZHOU Huan, QIAN Wei-ning

(*Institute for Data Science and Engineering, East China Normal University, Shanghai* 200062, *China*)

Abstract: Since its debut in the 1990s, ARIES, as the traditional transaction commit, has been widely adopted by the mainstream commercial or open source database systems. With the performance of applications increasingly improved, transaction commit based on the traditional hardwares has become the first bottleneck for performance of systems. However, the developments of high-performance hardwares, such as memory and multiple CPU cores, offer a new opportunity for the optimization of transaction commit. In this paper, we analyze and summarize the existing bottlenecks of traditional transaction commit in detail. Furthermore, key techniques of transaction commit based on new hardwares are discussed and concluded, including their application status, advantages and disadvantages. Finally, challenges and future developments for optimization of transaction commit are discussed.

Key words: main-memory database systems; transaction commit; techniques of log processing

收稿日期: 2016-05

基金项目: 国家自然科学基金重点项目(61332006); 国家 863 计划项目(2015AA015307)

第一作者: 胡 爽, 女, 硕士研究生, 研究方向为分布式数据库系统. E-mail: hu621000@sina.com.

通信作者: 钱卫宁, 男, 教授, 博士生导师, 研究方向为可扩展事务处理和海量数据管理与分析. E-mail: wnqian@sei.ecnu.edu.cn.

0　引　　言

内存数据库是以内存为主要存储介质的数据库系统, 由于内存读写速度非常快和避开了磁盘I/O操作, 相对于传统磁盘数据库来说, 其具有高吞吐量和低延迟的显著优势[1]. 虽然内存发展迅速, 但容量远不及磁盘而不足以支持整个数据库. 所以, 目前多数内存数据库系统是混合存储介质的, 除了内存它还需要磁盘存储, 让部分数据存在磁盘. 另外, 联机事务处理(On-Line Transaction Processing, OLTP)是数据库系统的重要内容之一, 在金融、互联网、电信等领域应用广泛, 也是当前学术研究的热门方向. 针对这种新型的事务型内存数据库系统来说, 在传统单机环境下, 内存的读写访问速度与磁盘相差太多, 导致涉及磁盘I/O的日志结构成为瓶颈. 而在分布式的多机环境下多台机器并发执行事务, 日志仍然是集中式的结构, 高并发的负载使其必然成为系统瓶颈.

日志是支持系统容灾恢复的基础技术[2], 也是内存数据库系统的一个重要结构, 它和数据是紧密联系的两个部分. 日志记录于非易失性存储器(磁盘等)上, 以一种稳定的方式记录数据库变更的历史, 不会因故障(如系统断电, 关机等)而丢失, 以保证系统在重新启动后进行及时恢复. 事实上, 日志不仅仅涉及I/O, 它涉及事务提交的整个流程, 与缓冲区管理, 锁管理等模块紧密联系. 日志缓冲区冲突以及日志引发的锁竞争等问题越来越严重, 导致事务提交性能跟不上事务的执行速度, 从而使事务提交成为系统的性能瓶颈.

作为数据库系统的一个核心服务, 事务提交是数据库系统的研究热点. 随着计算机硬件的飞速发展, 数据库系统开始面向大容量内存、多核处理器和高速网络等, 并试图通过这些硬件基础来提高事务提交性能. 近几年来, 陆续有研究利用多核特性, 充分协调每个核的调度请求以提高系统的并行度. 在传统成组提交[3]的基础上, 已有一系列优化技术解决了不同瓶颈来提升事务提交的性能. 新型存储介质的出现与快速发展, 非易失内存(Non-Volatile Memory, NVM)[4]以其数据的非易失性和高效的访问性能, 成为日志存储介质的不二选择. 现基于NVM日志存储的事务提交机制的研究处于探索和原型阶段[5-6]. 另外, 利用快速读写、质量轻以及能耗低特点的闪存(Flash)等固态存储和高速网络(High-Speed Data Center Networks), 一些高性能日志服务系统相继问世[7-9].

传统数据库系统事务执行流程需要记录两方面的内容, 包括修改数据和日志. 事务的数据操作在内存中进行, 而操作被记录在全局的日志结构. 为了缓冲从内存到磁盘的读写性能差异, 日志先互斥地写到集中式缓冲区, 并产生全局唯一的日志序列号LSN(Log Sequence Number), 以保证日志有序性. 当有事务提交, 缓冲区日志根据LSN被写到磁盘, 并在刷盘成功后响应客户端. 其中, 事务在数据操作时获得的锁会保留到日志刷盘成功后才释放. 图1表示的是事务提交流程: ①阶段事务写缓冲区; ②阶段缓冲区日志刷磁盘; ③阶段释放锁并响应客户端. 其中, 前两个阶段频繁涉及缓冲区管理、锁管理以及磁盘I/O等, 是主要产生瓶颈的阶段. 目前事务提交技术主要是对这两个阶段涉及的结构和硬件存储进行优化.

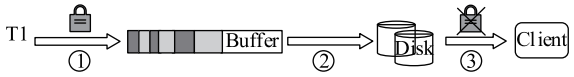

图1　事务提交的流程

Fig. 1　Process of transaction commit

本文主要从三个部分展开论述. 第一部分针对目前新型内存数据库系统高并发的负载特点, 分析传统事务提交机制存在的瓶颈; 第二部分对利用不同硬件特性优化的事务提交关键技术进行整理, 讨论现有事务提交关键技术的应用现状和优缺点, 以及归纳它们各自解决的传统事务提交的瓶颈; 第三部分对事务提交机制存在的挑战与未来的发展趋势进行探讨.

1 传统事务提交流程与分析

内存数据库在处理数据库恢复问题(Database Recovery)[10]时, 目前大多数系统都基于数据库日志[11]. 基于日志的ARIES传统事务提交机制是大多数系统的选择. 然而, 随着可扩展事务处理技术的发展, 传统事务提交机制已无法满足新型数据库系统的需求. 下面对ARIES的事务提交机制进行分析, 归纳出其存在的瓶颈.

1.1 Write Ahead Logging 方法

由 IBM 提出的 ARIES 算法是经典的数据库恢复原型算法, 其中预写日志(Write Ahead Logging, WAL)[12], 自 20 世纪 90 年代问世一直是事务提交的标准. 尽管目前大量数据库系统已使用多核硬件及大容量内存, 但这些系统仍使用这种集中式的 ARIES 预写日志提交机制[2,13].

传统事务提交机制的核心是数据落入磁盘必须发生在这些修改记录日志, 且日志被写到磁盘之后. 事务执行过程中所有操作产生的日志先填写到日志缓冲区, 直到发出事务提交请求时才会把缓冲区的日志写到非易失性存储器(磁盘), 数据后于日志落盘.

传统事务提交机制遵循“先写日志”的原则, 以确保数据安全. 数据落盘和日志落盘是两个不同的阶段, 这两个阶段之间有可能发生故障. 如果数据先落盘, 而日志没有存下来, 那么故障恢复时, 日志与数据是不一致的, 系统无法恢复. 如果先写了日志, 还没来得及将数据更新到磁盘, 那么在系统重启后, 根据日志文件中的记录, 重新将更新数据补录, 确保数据安全. 因此, 传统事务提交机制严格地保持数据和日志的一致性, 系统出现故障后将已提交的事务根据日志文件进行重做, 而将未提交的事务恢复到事务开始前的状态.

1.2 事务提交的瓶颈分析

随着多机、大内存以及多核系统的发展, 数据库负载变得复杂和高并发化, 系统无法达到足够高的并行性以满足硬件需要. 日志作为全局的数据结构, 被每个运行的事务共享, 在事务执行中成为了一个全局互斥点. 事务执行因日志的严格互斥性, 在提交阶段发生阻塞, 大量事务为争用日志结构而不断等待, 降低事务并行速度从而严重影响系统的吞吐量.

Stonebraker 等人对传统数据库系统做了一次分析[14]. 实验分析, 系统只有约 10% 时间在实际处理事务, 而剩下巨大的 90% 都花费在三大部分: 缓冲区管理, 锁管理和日志管理. 为强有力的说明事务提交已成为内存数据系统的一大瓶颈, Johnson R等人对 Shore-MT 系统[15]进行分析: Shore-MT 随事务线程数逐渐增加, 事务执行的时间分布情况, 使用TATP Update Location transaction. 其中 Shore-MT 是 Shore[16]优化的 48 核数据库系统. 实验显示: 随线程数增加, 日志管理约占事务执行总时间的 20%, 锁管理约占 10%, 其他竞争约占 5%, 其他 latch 约占 25%, 而剩下 40% 都花费在日志竞争上. 由此看出, 集中式的事务提交明显成为了主要开销. 表 1 主要从 4 种延迟来分析与归纳其对事务提交的影响.

表 1　事务提交瓶颈总结

Tab. 1　Analysis and conclusions for bottlenecks of transaction commit

延迟	原因	结果
I/O	磁盘 I/O 慢, 日志刷盘成为事务执行中耗时最多的部分.	短事务并发执行, 日志不断刷盘, 增加多个事务的响应时间.
上下文切换	日志刷盘除磁盘 I/O 还有上下文切换的延迟. 上下文切换占用 CPU, 必然会消耗额外开销.	核数增长导致调度器负载变大, 度的上下文切换使系统处于高负过载, 低 CPU 利用率的状态.
日志引起的锁竞争	事务执行获得的锁直到日志刷盘成功才释放. 其他事务必须等锁释放后, 才能获得锁继续执行.	高并发的负载下, 日志刷盘保留锁, 使得多数事务被阻塞而不断地竞争锁.
日志缓冲区竞争	集中式缓冲区和日志互斥填写缓冲区的方式, 使得大量事务阻塞.	事务被阻塞, 受日志大小的影响, 影响系统的并行性和可扩展性.

2　事务提交的关键技术

根据传统事务提交的整个流程以及其存在的各种瓶颈, 本文主要研究与讨论基于多核、NVM 和高速网络等硬件设备的事务提交关键技术.

2.1　基于多核的事务提交关键技术

面对多核在新型内存数据库系统中的广泛应用和高并发负载需求, 已有一系列成熟的技术充分利用每个核的调度请求来提高系统的并行性, 减少系统的性能瓶颈, 主要分为缓冲区填写日志阶段和日志写入磁盘阶段的优化.

2.1.1　成组提交(Group Commit)

事务每进行一次提交, 缓冲区中日志刷磁盘阶段都会产生磁盘 I/O. 多数事务因处于等待状态而增加延迟, 从而造成了系统瓶颈.

为减少每次事务提交产生的磁盘 I/O, 成组提交提出了一种新的解决方案, 即每隔时间T、或者每提交X个事务、或者提交的N个事务日志长度达到L字节时, 才将缓冲区中的积累的所有日志统一刷磁盘.

成组提交的特点是打包多个事务日志进行成组处理, 将原本多次磁盘 I/O 减少为一次, 降低磁盘 I/O 的开销. 虽然该技术优化了传统日志的 I/O 延迟, 提高了系统性能, 但系统仍存在上下文切换、锁竞争和缓冲区竞争等问题.

目前有不少商业或开源数据库系统使用成组提交, 其中包括 MySQL[17] 和 Oracle[18]等. 其中, Hekaton[19]对成组提交进行了改进以适用于自身的架构. Hekaton 是微软为大内存和多核 CPU 研发的新型内存事务引擎, 现已集成到了 SQL Server. Hekaton 日志流是存储在磁盘的数据, 与常规 SQL Server 日志相同. 不同的是, Hekaton 每个事务只生成单一日志, 日志记录有关该事务插入和删除操作的所有版本, 足以在故障恢复时重做. 事务日志流作为事务提交的重要结构, 减少日志生成数量和空间可提高效率. 因此 Hekaton 在每个事务提交时才生成日志填入缓冲区, 并成组写入磁盘以减少多次磁盘 I/O 切换.

2.1.2　异步提交(Asynchronous Commit)

针对磁盘 I/O 的延迟问题, 异步提交[20-21]对成组提交进行了扩展. 相比与传统的事务提交, 它增加了日志提交线程(Commit Threads), 专门进行日志刷盘活动. 每个事务提交时, 处理线程(Handle Threads)填完缓冲区日志后就响应客户端. 缓冲区日志成组(每隔时间 T、每 X 个事务或提交的 N 个事务日志长度达到 L 字节)后, 提交线程将缓冲区成组的日志刷盘. 图 2 是异步

提交的基本流程: ① 处理线程填写缓冲区日志; ② 处理线程响应客户端, ③ 提交线程将成组的缓冲区日志刷盘.

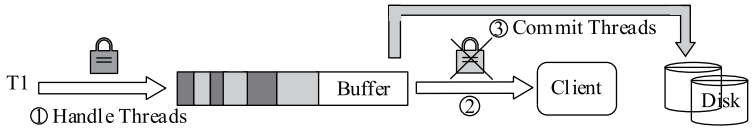

图 2 异步提交流程

Fig. 2 Process of asynchronous commit

异步提交的特点是事务不等待日志刷盘成功就响应客户端, 商业和开源数据库系统 Oracle[20]和 PostgreSQL[21]就采用了这种提交方式. 异步提交将耗时较长的日志刷盘阶段完全隐藏不仅缩短了响应客户端的时间, 还减少了锁竞争和过度的上下文切换. 但该提交下的数据是不安全, 如果系统故障而崩溃, 已提交的日志可能因还没有被刷到磁盘而丢失.

2.1.3 提前释放锁(Early Lock Release, ELR)

日志写入磁盘阶段除了磁盘 I/O 的延迟, 还存在的问题是事务直到日志持久化成功后才能释放锁. 事务长期持锁使得其他并行事务阻塞而产生锁冲突, 因此增加事务提交的延迟, 并导致系统的并行度无法提高.

针对这一问题, DeWitt 等人[1]早在 20 世纪 80 年代就提出了在一定条件下事务可在日志刷盘前安全释放锁的观点. 事务被分为两种类型: 预提交事务和依赖事务. 提前释放锁的事务称为预提交事务; 访问预提交事务的更新数据的事务称为依赖事务, 依赖于预提交事务. 在缓冲区填写阶段之前, 事务就已确定全局的提交顺序. 提交过程中, 未依赖于其他事务的预提交事务可提前释放锁, 无依赖关系的事务得到锁后就可继续执行, 提高了事务提交效率. 只有少数依赖事务被阻塞, 需等到预提交事务日志刷盘成功并响应客户端. 过了十多年, 观点的正确性才被文献[22]证明. 另外, 提前释放锁必须满足两个条件才能维持系统的可恢复性.

(1) 每个依赖事务必须在它依赖的预提交事务日志刷盘完成后进行刷盘.

(2) 当一个预提交事务被终止了, 那么所有该预提交事务的依赖事务也将被终止.

提前释放锁的特点是消除了日志刷盘带给其他事务的延迟, 只有依赖事务必须等待其刷盘后才能完成提交操作, 其他事务可立即获得锁继续执行. 早在正确性被验证之前, 文献 [23] 和 IBM 的 IVS[24]事务处理工具就已采用了这种技术. 对于锁竞争激烈的负载, 提前释放锁使系统性能显著提升. 与异步提交比较, 它能够保证数据安全.

2.1.4 流水线提交(Flush Pipelining)

近几年已有一些研究[25-26]通过闪存优化来减少 I/O 延迟, 但闪存不能完全消除日志刷盘的高延迟. 在多核数据库系统中, 日志写入磁盘阶段仍存在着频繁的上下文切换延迟.

Johnson. R 等人提出了流水线提交[27]: 流水线提交是将传统的事务处理分成两个阶段: 事务处理阶段和事务提交阶段, 分别对应多个事务代理线程(Agent Threads)和一个事务守护线程(Daemon Threads). 事务代理线程负责负责写缓冲区、响应客户端; 守护线程负责刷盘. 事务提交时, 代理线程写完缓冲区日志就将当前待提交任务压入响应客户端队列, 而自己执行其他事务. 每隔时间T、每X个事务或提交的N个事务日志长度达到L字节后, 守护线程将那部分缓冲区日志刷磁盘. 日志刷盘后, 守护线程通知代理线程最新一批已完成的事务. 代理线程唤醒队列中的待提交任务, 完成提交并响应客户端. 图 3 是流水线提交的基本流程.

流水线提交的特点是将事务处理拆分成两个阶段进行, 长时间的日志刷盘阶段交给专门线程执行. Johnson R 等人在 Shore-MT 的 Aether 日志组件中对它进行了实现, 对比传统机制, 系统性能提高了22%. 流水线提交不仅减少了频繁的上下文切换的瓶颈, 还能保证数据安全, 从而

大大提高了系统的吞吐量.

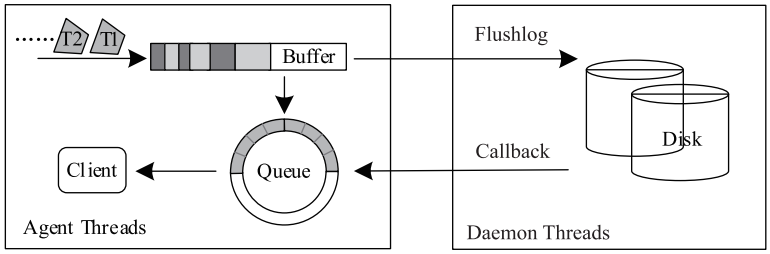

图 3　流水线提交流程

Fig. 3　Process of flush pipelining

2.1.5　可扩展的缓冲区(Scalable Log Buffer)

传统事务提交机制的每个日志都分配全局唯一的日志序列号LSN, 作为数据存入磁盘的时间戳, 缓冲区写日志阶段必须是互斥的. 并且传统事务处理采用的是集中式日志缓冲区.

为永久地消除日志缓冲区竞争, 并且使其不受日志大小和线程数量的影响, Johnson R 等人提出两种可扩展的日志缓冲区设计[27]: 合并缓冲区分配(Consolidating Buffer Allocation)和解耦缓冲区填写(Decoupling Buffer Fill).

合并缓冲区分配的优化目的是减少竞争, 并使事务提交不受线程数的影响. 多个线程同时竞争锁, 获得锁的线程生成 LSN 并填写缓冲区日志. 未获得锁的多线程抢占一个固定大小的“合并数组”(Consolidation Array)[28]. 抢占成功的多个线程成组等待锁释放. 一旦释放锁, 第一个成员将获得锁并生成 LSN, 并将 LSN、缓冲区位置和偏移通知下一成员. 所有成员依次推算出 LSN 和缓冲区位置后, 并行填写缓冲区日志. 所有成员填写完后清空数组, 并由该数组最后一个成员的线程负责缓冲区日志刷盘和缓冲区空间的释放.

解耦缓冲区填写进一步提高了日志缓冲区填写的并行度. 在传统事务提交机制基础上, 生成 LSN 的线程在获得缓冲区位置的同时释放锁, 另一个线程就可获得锁并确定缓冲区位置. 确定缓冲区位置的线程可填写缓冲区日志, 使得日志进行流水线式的填写. 防止创建空白日志而丢失可恢复性, 缓冲区空间必须按 LSN 顺序释放, 使日志按 LSN 依次写入磁盘.

虽然在合并缓冲区分配的优化下, 填写缓冲区日志的时间开销与组内成员日志大小和释放缓冲区空间成正比, 但对于组内的多个事务提交性能得到了一定的提升. 而解耦缓冲区填写的优化技术, 虽然使得大量线程可同时对缓冲区进行日志填写, 提高了事务提交的并发性, 但是顺序释放缓冲区的操作成为了关键延迟.

2.2　基于 NVM 的事务提交关键技术

新兴的 NVM 技术从根本上改变了事务提交的设计原则. NVM 因其数据非易失和高效访问的特性, 使得事务日志不再需要暂存缓冲区后刷到磁盘的缓慢流程, 以保证事务的持久性. 但由于 NVM 的成本较高, 且访问模式与传统的内存和二级存储都不相同, 加上目前 NVM 技术本身仍然不够成熟, 基于 NVM 日志存储的事务提交机制研究处于起步阶段.

基于 NVM 的分布式日志提交(NVM-based Distributed Logging)[6]是一种基于 NVM 日志存储的事务提交原型. 多核数据库系统的每个核都分配一个 NVM 缓冲区. 事务填完 NVM 缓冲区日志就完成事务提交, 这是因为 NVM 的非易失性保证了事务的持久性. 当 NVM 缓冲区的日志填满时, 系统再将这些日志转存入磁盘. 如何对日志分区和保证日志次序成为了难点. 该提交技术采用两种策略: 事务分区和页分区, 分别依据事务和数据页的相关性来分配. 相对于传统事务提交机制的LSN, 该提交技术采用基于逻辑时钟[29]的全局序列号(Global Sequence Number)以保证生成日志的顺序. 另外, 日志从易失性 CPU 写到 NVM 缓冲区的阶段采用被动

的成组提交(Passive Group Commit)以保证系统的可恢复性.

基于 NVM 的分布式日志提交解决了磁盘 I/O 延迟和日志缓冲区竞争的瓶颈, 实验证明其具有低冲突和高吞吐的性能. 随着 NVM 技术的日益成熟, 基于 NVM 的事务提交技术将越来越热门.

2.3 基于高速网络与高性能固态存储的事务提交关键技术

高速网络的发展使网络延迟变得越来越小, 目前网络传输已能够达到 100 Gb/s. 闪存等固态存储设备也发展迅速, 其读写性能与磁盘相比高出约 10^4 IOPS. 越来越多的学者开始研究基于高速网络与高性能固态存储的存储系统. 基于这两种硬件的事务提交技术应运而生.

Hyder[30]事务提交是基于高速网络与闪存硬件, 面向具有共享的网络寻址存储(Network-Addressable Storage)的集群. Hyder事务提交的流程如下: 分布式系统的一个站点的事务执行产生的更新操作暂存在本地缓存, 并广播给所有站点; 同时该站点将生成的日志通过高速网络存入闪存优化的共享可扩展日志结构(Shared Striped Log); 其他站点接收消息后, 更新到本地缓存以保证全局日志的连续性.

Hyder 事务提交目前应用于 Hyder 系统, 微软研究的一个多版本的日志文件/数据库系统. 该提交技术不仅解决了锁竞争, 还降低了日志缓冲区的冲突, 使系统性能大大提升. 除了 Hyder, 还有 Flash-log[7]、CORFU[8]、Tango[9], 也利用专门的大容量固态存储服务器, 配以定制的事务提交机制, 提供共享的高性能日志服务. 但是这些系统要求高性能的网络和存储服务器, 无法直接在现有的新型内存数据库系统中使用.

2.4 事务提交关键技术的小结

本文分析的事务提交关键技术利用新型数据库系统不同硬件特点, 针对性地解决了传统事务提交机制存在的一些瓶颈. 这些事务提交关键技术各自存在优缺点. 表 2 是对事务提交技术解决不同的瓶颈进行的归纳.

表 2 事务提交关键技术各自解决的瓶颈

Tab. 2 The bottlenecks that key techniques of transaction commit respectively have resolved

	I/O延迟	锁竞争	上下文切换	缓冲区竞争
成组提交	√			
异步提交	√	√	√	
提前释放锁		√		
流水线提交			√	
可扩展的缓冲区		√		√
基于NVM的分布式提交	√		√	√
Hyder提交		√		√

3 总 结

面对越来越复杂的高并发事务处理, 目前新型内存数据库系统已无法达到足够的高吞吐量以满足应用需求. 传统事务提交机制因存在局限性而逐渐成为了系统的性能瓶颈. 本文重点整理和分析了利用多核特性的成组提交、异步提交、提前释放锁和流水线提交等事务提交技术, 基于 NVM 的分布式事务提交技术以及基于高速网络和高性能固态存储的事务提交技术, 这些关键技术在不同程度上地解决传统事务提交机制的瓶颈. 此外, 本文对它们各自的应用现状和优缺点进行分析与归纳.

随着海量数据事务处理的复杂化, 以及计算机硬件设备的极速更新, 传统事务提交机制无法满足系统需求. 越来越多的学者利用新型内存数据库系统的多核和大内存等特性来研究事务提交的优化方法. 而事务提交涉及数据库系统的多种模块管理结构, 包括缓冲区管理, 锁管理等,

这些模块需要相互权衡. 日志不是一个全序关系, 而是一个偏序关系, 如何对事务提交进行解耦合是目前事务提交优化的困难之一. 对提交事务进行预处理是优化的趋势, 但由于事务间复杂的依赖关系, 事务的合理分区成为难点. 而在解耦合过程中, 提高事务并发度使得各阶段的校验变得更加复杂. 另外, 传统事务提交机制并没有考虑数据库系统所处的网络环境, 分布式数据库系统的日志同步代价和成本是巨大的, 需进一步改进才能在实际环境中应用. 所以设计出一种能够同时解决所有传统事务提交机制的瓶颈, 又能适应于多种环境的事务提交新技术面临着巨大的挑战.

同时 NVM 等一些非易失性内存的发展和 SSD、Flash 等一些非易失性固态存储器成本的不断降低, 将有力推动事务提交优化技术的发展. 采用这些新型硬件设备, 可以减少事务提交中日志持久化过程, 大大提高事务提交效率. 作为未来研究热点的可扩展事务提交将是避免产生瓶颈的有效手段, 其中采用 lock-free 和 latch-free 的数据结构可减少事务提交过程中的锁竞争; 采用 CAS 原子操作可实现日志串行化以提高吞吐量. 此外, 解耦传统数据库架构, 将日志单独作为服务系统也是事务提交技术未来研究的方向之一.

[参　考　文　献]

[1] DEWITT D J, KATZ R H, OLKEN F, et al. Implementation techniques for main memory database systems[J]. Acm Sigmod Record, 1984, 14(2): 1-8.

[2] RAGHU R, JOHANNES G. Database Management Systems. [M]. 3th ed. New York: McGraw-Hill, 2003.

[3] HELLAND P, SAMMER H, LYON J, et al. Group commit timers and high volume transaction systems[C]//High Performance Transaction Systems. USA: Springer Berlin Heidelberg, 1987: 301-329.

[4] PELLEY S, CHEN P M, WENISCH T F. Memory persistency[J]. Acm Sigarch Computer Architecture News, 2014, 42(3): 265-276.

[5] FANG R, HSIAO H I, HE B, et al. High performance database logging using storage class memory[C]//IEEE International Conference on Data Engineering. [S.l.]: IEEE, 2011: 1221-1231.

[6] WANG T, JOHNSON R. Scalable logging through emerging non-volatile memory[J]. Proceedings of the VLDB Endowment, 2014, 7(10): 865-876.

[7] BALAKRISHNAN M, BERNSTEIN P A, MALKHI D, et al. Brief Announcement Flash-Log-A High Throughput Log[J]. Lecture Notes in Computer Science, 2010, 1(6343): 401-403.

[8] BALAKRISHNAN M, MALKHI D, DAVIS J D, et al. CORFU: A distributed shared log[J]. Acm Transactions on Computer Systems (TOCS), 2013, 31(4): 879-889.

[9] BALAKRISHNAN M, MALKHI D, WOBBER T, et al. Tango: Distributed data structures over a shared log[C]//Twenty-Fourth ACM Symposium on Operating Systems Principles. New York: ACM, 2013: 325-340.

[10] MALVIYA N, WEISBERG A, MADDEN S, et al. Rethinking main memory oltp recovery[C]//International Conference on Data Engineering. [S.l.]: IEEE, 2014: 604-615.

[11] GRAY J, REUTER A. Transaction Processing: Concepts and Techniques[M]. San Francisco: Margan Kaufmann, 2015.

[12] MOHAN C, HADERLE D, LINDSAY B, et al. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging[J]. Acm Transactions on Database Systems (TODS), 1992, 17(1): 94-162.

[13] MOHAN C. Repeating history beyond ARIES[J]. VLDB, 1999, 99: 1-17.

[14] STONEBRAKER M, WEISBERG A. The VoltDB main memory DBMS[J]. IEEE Data Eng Bull, 2013, 36(2): 21-27.

[15] JOHNSON R, PANDIS I, HARDAVELLAS N, et al. Shore-MT: a scalable storage manager for the multicore era[C]//Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology. New York: ACM, 2009: 24-35.

[16] CAREY M J, DEWITT D J, FRANKLIN M J, et al. Shoring Up Persistent Applications[M]. New York: ACM, 1994.

[17] MySQL A B. MySQL: The world's most popular open source database [EB/OL]. (2005-12-01) [2016-05-23]. http://www.mysql.com.

[18] LONEY K, MCCLAIN L. Oracle Database 10g: The Complete Reference [M]//Oracle 8: The Complete Reference. New York: Osbome/McGraw-Hill, 1997, 10(6): 179.

[19] DIACONU C, FREEDMAN C, ISMERT E, et al. Hekaton: SQL server's memory-optimized OLTP engine[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2013: 1243-1254.

[20] THOME B, GAWLICK D, PRATT M. Event processing with on oracle database[C]//Proceedings of the 2005 ACM SIGMOD Interational Conference on Management of Data. New York: ACM, 2005: 863-867.

[21] 彭智勇, 彭煜炜. PostgreSQL 数据库内核分析[M]. 北京: 机械工业出版, 2011: 343-404.

[22] SOISALON-SOININEN E, YLÖNEN T. Partial strictness in two-phase locking[C]//Intemational Conference on Database Theory. USA: Springer Berlin Heidelberg, 1995: 139-147.

[23] LIU C C, MINOURA T. Effect of update merging on reliable storage performance[C]//Second International Conference on Data Engineering. [S.l.]: IEEE, 1986: 208-213.

[24] GAWLICK D, KINKADE D. Varieties of concurrency control in IMS/VS fast path[J]. IEEE Database Eng Bull, 1985, 8(2): 3-10.

[25] CHEN S. FlashLogging: Exploiting flash devices for synchronous logging performance[C]//Proceedings of the International Conference on Management of Data. New York: ACM, 2009: 73-86.

[26] LEE S W, MOON B, PARK C, et al. A case for flash memory ssd in enterprise database applications[C]//Proceedings of the ACM SIGMOD International Conference on Management of Data. New York: ACM, 2008: 1075-1086.

[27] JOHNSON R, PANDIS I, STOICA R, et al. Aether: A scalable approach to logging[J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 681-692.

[28] MOIR M, NUSSBAUM D, SHALEV O, et al. Using elimination to implement scalable and lock-free fifo queues[C]//Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures. New York: ACM, 2005: 253-262.

[29] LAMPORT L. Time, clocks, and the ordering of events in a distributed system[J]. Communications of the ACM, 1978, 21(7): 558-565.

[30] BERNSTEIN P A, REID C W, DAS S. Hyder-A transactional record manager for shared flash[J]. CIDR, 2011(11): 9-20.

(责任编辑: 张 晶)