Scalable Commit 功能设计文档

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-06-15	Scalable Commit 功能设计文档	周欢 胡爽	

1 需求分析

1.1 Cedar事务执行流程

Cedar数据库的内存事务引擎采用Flush Pipelining将事务执行过程分为3个阶段：事务处理，事务提交和事务发布，分别对应三种线程事务处理线程（worker）、事务提交线程（committer）和事务发布线程（publisher），每一个阶段完成之后将当前任务压入下一个阶段的任务队列之后再处理下一个任务请求。Cedar的事务执行流程下：

1. 事务处理阶段，多线程对并发的事务执行预处理，包括加行锁，进行逻辑判断（是否能执行insert等操作）；将待更新数据写入事务上下文的临时空间；之后在事务提交时确定每个待提交事务的事务版本号（trans_id），并按trans_id的先后顺序将任务压入提交队列；最后采用Early Lock Release技术提前释放行锁，维护多版本并发控制信息（更新memtable链表和版本号）；
2. 事务提交阶段，单线程处理待提交事务，包括更新系统已提交事务版本号（trans_committed_id）、生成日志校验和（checksum）、并按事务先后顺序确定日志信息（log_id,file_id和file_offset）;然后将事务日志顺序地拷贝到集中式日志缓冲区中；之后当缓冲区中的日志大小超过2M或者系统空闲时，单线程触发日志写盘和同步备机操作；同时提交线程循环地判断同步到备机的最大日志号，确定已经完成提交的事务，然后更新系统的发布版本号（trans_published_id）和memtable信息（row_counter_、checksum和last_trans_id）,最后将已完成提交的事务加入事务发布线程池中；
3. 事务发布阶段，由多线程对已完成提交的事务进行处理，包括释放事务上下文并响应客户端。

> 在事务提交方面，Cedar相较于传统数据库不仅避免了事务处理线程因为日志I/O而触发的上下文切换（尤其在如今计算机的物理核数不断增大的情况下）而且减少了系统内部对共享数据（日志缓冲区）的冲突访问。

1.2 Cedar事务提交瓶颈

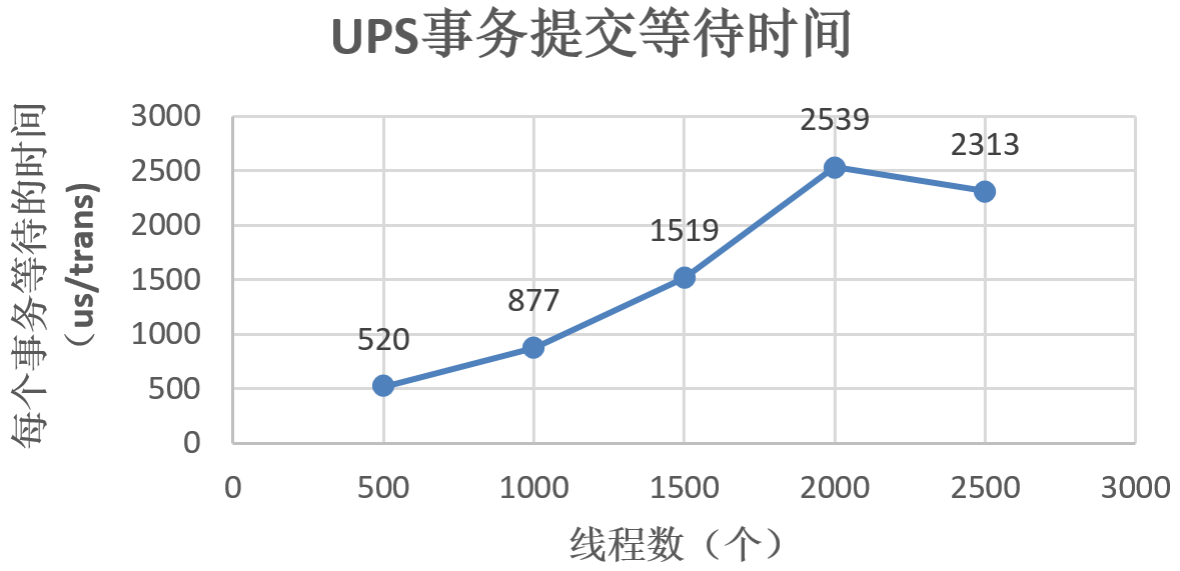

图1 事务提交等待时间

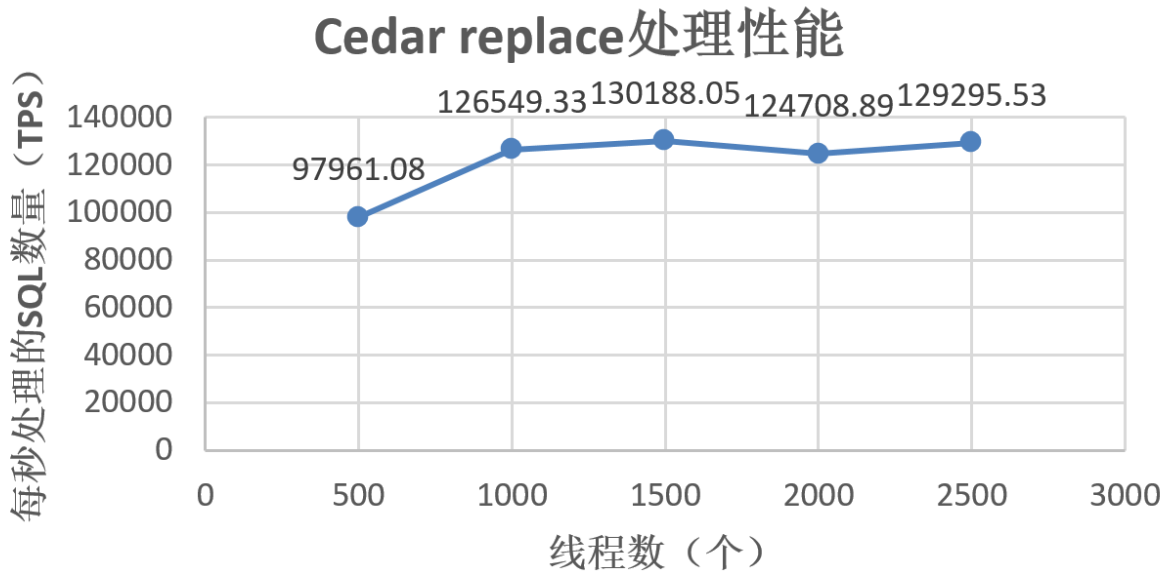

图2 Cedar replace处理性能

Cedar采用流水线式事务执行架构虽然避免了传统事务执行架构的弊端，但单线程处理事务提交的设计使得系统性能受限于提交线程的处理能力。随着客户端auto-commit事务请求的增多，大量事务将因提交线程性能达到极限而堆积在提交队列中，而此时事务处理线程和事务发布线程均未达到极限。因此单线程提交限制了系统整体的事务吞吐量。

图1，2所示是在8个MS、3个CS和1个RS、UPS的集群环境下进行replace操作的实验结果。实验结果显示，当Cedar的处理性能（TPS）达到13万之后，事务在提交队列中的等待时间由1519us突增到2539us,而此时事务提交线程的CPU利用率已达到97%。由此可以看出，当前Cedar提交线程的处理量在13万左右，当超过13万之后大量事务在提交队列中堆积从而导致吞吐量降低最后趋于稳定。

事务提交过程主要包括更新已提交事务号id（update_committed_id）、生成memtable校验和（checksum）、填充日志缓冲区（fill_log）、写磁盘并同步备机（commit_log）和发布事务（publish）操作。通过统计事务提交过程中各操作消耗的时间可以发现，随着客户端数量的增多（UPS压力的逐渐增大），update_commiited_id和生成checksum时间基本不变，commit_log时间越来越少，而fill_time和publish时间却逐渐增多，最后稳定在一定的范围内。如图3所示为2000客户端时各操作的时间比，其中fill_log和publish操作所占时间比例最大，分别为42%和37%。Fill_log操作中的大量内存拷贝和publish操作中的大量循环判断是限制提交线程处理性能的主要原因。

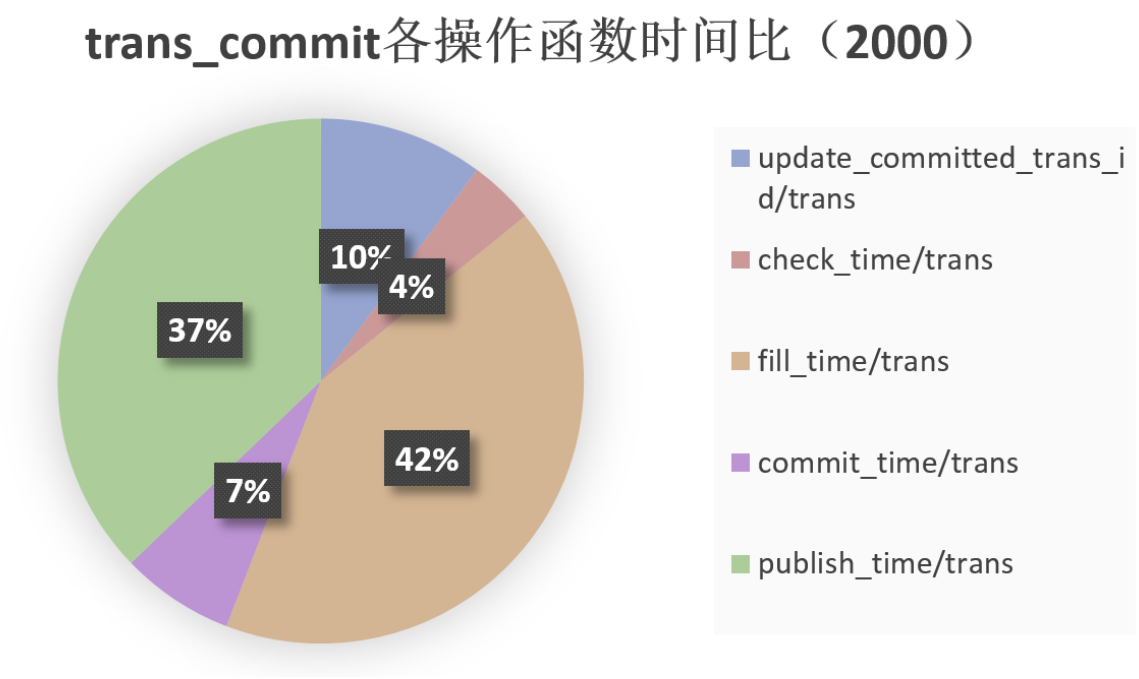

图3 事务提交各操作函数时间比

2 功能简述

为了提高提交线程的处理能力，需要优化提交过程中的大量内存拷贝和计算，在保证日志提交顺序的基础上，实现多线程填充日志缓冲区并减少对待发布事务的计算。另外，成组提交到单线程处理，依次将当前待提交的任务进行刷磁盘和同步备机。并由多线程循环地判断待提交事务是否能响应客户端，如果能响应则多线程并发的更新memtable信息，然后释放事务上下文并响应客户端。

3 设计思路

UpdateServer中存在两种线程分别为多个事务处理线程（workers）和单事务提交线程（committer），处理线程负责并发处理事务、占位、解行锁、生成日志、填充缓冲区、提交写盘任务，最后响应客户端并回收事务上下文，事务提交线程负责日志写盘、同步备机、判断待发布的事务并唤醒工作线程响应客户端。在Scalable Commit架构中，事务执行过程分为四个阶段分别是事务处理、事务预提交、事务提交和事务发布。具体的执行流程如下：

1. 事务处理阶段，由多线程对并发的事务执行预处理，包括加行锁，进行逻辑判断（是否能执行insert等操作）；然后将待更新数据写入事务上下文的临时空间；
2. 事务预提交阶段，当事务提交时多线程并发的为待提交事务抢占缓冲区的位置；如果抢占成功之后则生成日志信息，包括log_id、trans_id、file_id、file_offset和checksum，如果不成功则继续抢占；然后采用Early Lock Release技术提前释放行锁，维护多版本并发控制信息（更新memtable链表和版本号）；最后填充缓冲区并将预提交事务压入工作线程的私有队列中等待提交线程唤醒，再由缓冲区中最后一个完成日志填充的工作线程更新trans_committed_id并将缓冲区作为一个任务压入提交线程队列中；
3. 事务提交阶段，由单线程处理当前提交线程队列中的任务，如刷磁盘并同步备机；同时提交线程循环地判断同步到备机的最大日志号，确定已经完成提交的事务，然后用已完成提交事务的最大trans_id更新系统的发布版本号（trans_published_id）并清空当前缓冲区，最后唤醒工作线程私有队列中等待的事务；
4. 事务发布阶段，由多线程循环地判断私有队列中事务是否被唤醒，如果被唤醒则多线程并发的更新memtable信息（row_counter_、checksum和last_trans_id）,然后释放事务上下文并响应客户端。

4 参考文献

[1] 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.

[2] 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.