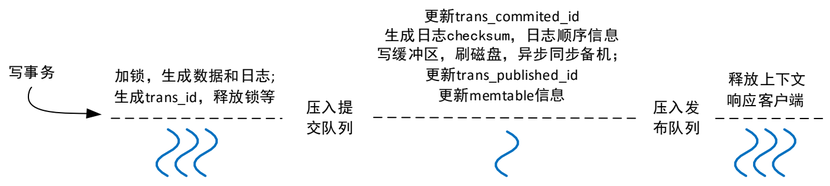
UPS事务提交优化

周欢 zhouhuan@stu.ecnu.edu.cn 2016.1

优化背景

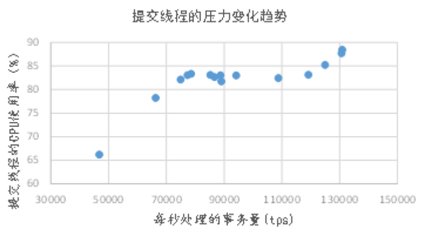
OceanBase0.4.2版本中UpdateServer的事务执行引擎分成三个阶段，分别是事务处理、事务提交和事务发布。事务处理阶段，由多个线程对并发的事务进行处理，包括加行锁和逻辑判断（是否能执行insert等操作），然后将待更新数据写入事务上下文的临时空间，最后在确定事务要提交的情况下将待提交事务压入提交队列，同时确定每个待提交事务的事务版本号并释放行锁、维护多版本并发控制（更新memtable链表和版本号）；事务提交阶段，由单个线程处理当前待提交的事务，包括更新trans_committed_id、生成日志checksum、生成日志顺序信息（log_id,file_id和file_offset），然后写日志缓冲区，成组写磁盘和异步同步备机，最后处理已同步备机成功的事务，包括更新trans_published_id，更新memtable信息（row_counter_、checksum和last_trans_id）和将事务压入发布队列；事务发布阶段，由多个线程对已完成提交的事务进行处理，包括释放事务上下文，响应客户端。



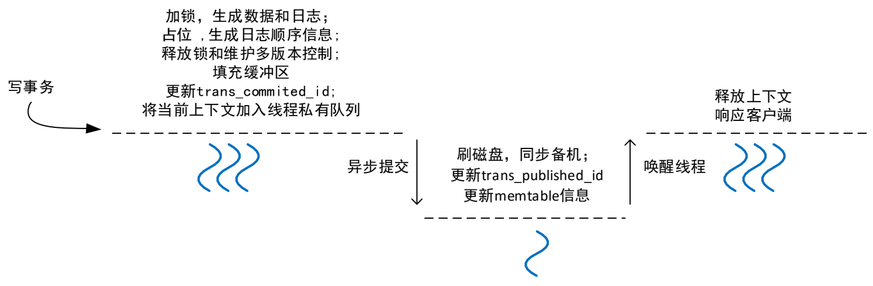
事务执行流程

由此存在一个性能瓶颈是：

单线程处理事务提交限制了事务引擎的整体性能。随着并发事务的增多，事务提交单线程的压力越来越大，当tps达到13万左右时，事务提交线程的CPU利用率已达到97%，从而导致大量并发事务阻塞在提交队列中。



总体设计



事务执行流程

新架构中，UpdateServer的事务执行引擎分成三个阶段，分别是事务处理、事务提交和事务发布。各阶段的操作流程如下：

事务处理阶段：

（1） 由多个线程对并发事务进行处理，包括加行锁和逻辑判断（是否能执行insert等操作），然后将待更新的数据写入事务上下文的临时空间；

（2） 确定事务要提交的情况下，为待提交事务抢占缓冲区中的位置，即占位操作；

（3） 如果占位成功，则生成日志顺序信息，包括log_id,trans_id,file_id，file_offset和checksum；如果占位不成功，则继续抢占；

（4） 释放行锁，维护多版本并发控制信息，如更新memtable链表和版本号；

（5） 填充缓冲区， 填充完成之后，将事务上下文压入线程私有队列中等待提交线程唤醒；

（6） 如果是缓冲区中最后一个完成事务日志填充操作的工作线程需要更新trans_committed_id，并发起异步提交。

事务提交阶段：

（1） 由单个线程处理当前缓冲区中的所有待提交的事务，如刷磁盘和同步备机；

（2） 更新trans_published_id和memtable信息，包括row_counter，checksum和last_trans_id；

（3） 最后唤醒已完成提交事务的工作线程。

事务发布阶段：

（1） 判断私有队列中上下文所涉及的事务是否已完成提交，如果完成则释放上下文并响应客户端；如果未完成则不做任何操作。

占位设计

此部分主要讲解**CAS占位操作**。

CAS占位操作流程

（1） 全局变量FLogPos next_pos_，初始化时group_id_，rel_id_，rel_offset_都为0；

（2） 工作线程根据自己的日志长度len，采用append方法乐观计算出自己的缓冲区位置cur_pos；

（3） 工作线程用128bit CAS操作抢占更新全局next_pos_，如果失败则返回(2)；

（4） 抢占成功，则根据cur_pos.group_id % GROUP_ARRAY_SIZE定位所属的group，然后确定日志信息，

如log_id = group.start_log_id_+cur_pos.rel_id_，trans_id = group.start_timestamp_+cur_pos.rel_id_，最后根据cur_pos.rel_offset_填充日志。

append(FLogPos& pos, int32_t len, int64_t limit)函数说明

（1） 如果工作线程向当前group追加长度为len的日志后还未超过limit，则追加成功，返回新追加日志的pos；

（2） 如果工作线程向当前group追加长度为len的日志后超过limit，则切换到下一个group。当前工作线程需要预留两条日志空间，一是当前group最后一条NOP或者SWITCH_LOG日志；二是下一个group的第一条日志（存储事务日志）。如果下一个group可用则返回当前group最后一条日志的pos，否则等待；

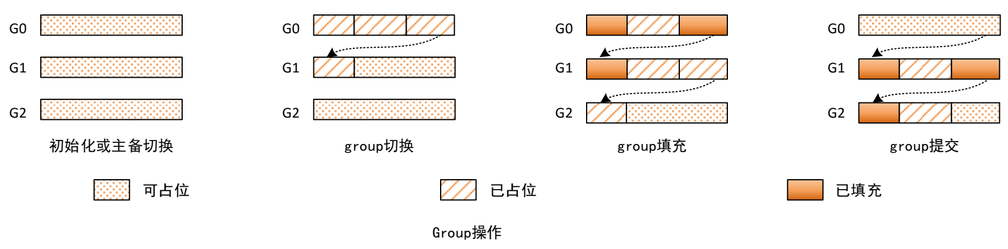
（3） 切换group时，append()返回一个特殊的错误码：OB_GROUP_SWITCHED，此时推算出下一个group第一条日志的pos，然后进行CAS抢占；

（4） 处理特殊写任务时，传递一个无效的len,表示冻结当前group，为当前group追加一条NOP或SWITCH_LOG日志，并切换group;？？？？？？？

（5） 实际调用时给append()传递的limit参数需要预留足够的空间写NOP或者SWITCH_LOG，此时SWITCH_LOG也有日志对齐功能。

缓冲区设计

此部分主要讲解**缓冲区操作**。



Group操作

缓冲区操作的主要步骤如下：

（1） 初始化或主备切换时，更新全局变量next_group_.group_id_所对应group的start_log_id_和start_timestamp_，然后设置ts_seq_=next_pos_.group_id_+READY；更新start_log_cursor_，然后设置log_cursor_seq_=next_pos_.group_id_+READY(标识当前group G0可占位，可填充，可提交)；

（2） group切换时，填充G0最后一条占位日志线程需等G1可占位之后，设置G1的start_log_id_和last_timestamp_,然后设置G1的last_ts_seq_=group_id + READY(标识G1可以修改group状态为可填充态)。填充G1第一条占位日志线程等G1的last_ts_seq_=group_id + READY之后，设置G1的start_timestamp_为last_timestamp_ + 1和当前时间戳中的较大者，然后设置ts_seq_=group_id + READY（标识G1可填充）；

（3） group填充时,G0的最后一个填充线程更新全局的committed_trans_id_ = start_timestamp_ + count_ - 1，等到G0为可提交状态之后负责异步提交，之后更新G1的start_log_cursor_，再设置G1的log_cursor_seq_ = group_id + READY（标识G1可提交）；

（4） group提交时，提交线程处理成功之后，设置当前group G0的ts_seq_ = group_id + GROUP_ARRAY_SIZE（标识G0可占位）。

缓冲区操作存在的等待情况：

（1） 等待group可占位，成组写磁盘和同步备机越快，group为可占位状态就越及时；group的数组够大，等待的时间也会相对减少；

（2） 等待group可填充，通常情况下，填充上一个group的最后一条占位日志和填充当前group的第一条占位日志是同一个线程，此线程设置当前group的可填充状态，那么其他工作线程即使占位成功也需要等待(等待填充NOP或者SWITCH_LOG的时间)；（可以考虑修改为group第一个填充线程设置group的可填充状态）；

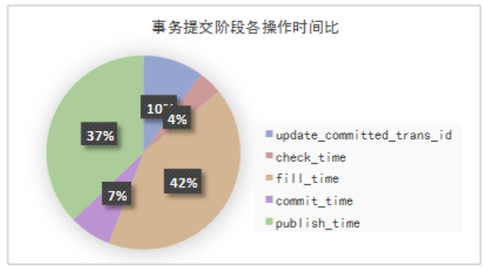
（3） 等待group可提交，只有当当前group的所有日志都已完成填充之后才会出现等待，所以需要等待的可能性不大。

提交设计

校验设计

优化目标

OceanBase0.4.2版本UpdateServer事务执行引擎的瓶颈在于单线程处理事务提交的能力有限。在事务提交阶段，消耗CPU最多的两个操作分别是填充缓冲区（大量的内存拷贝）和更新memtable信息，因此需要优化为多线程并发填充缓冲区和更新memtable信息，即多线程处理事务提交阶段。



基本原理

在OLTP系统中，Logging是唯一的全局同步点，因此在UpdateServer事务提交阶段需要严格保证写日志的先后次序，即日志内容在日志缓冲区中的先后位置。为了实现多线程并发填充缓冲区，工作线程需要在填充缓冲区之前确定事务日志顺序（在缓冲区中的位置）和与顺序紧密相关的日志信息，包括日志号(log_id)、事务版本号（trans_id）、文件编号(file_id)、文件偏移(file_offset)、日志校验和(checksum)。实现过程中需要满足以下条件：

（1） 并发填充缓冲区之前给日志分配一个缓冲区位置，这步操作称为占位操作；

（2） 并发释放行锁，维护多版本控制，生成日志之前必须决定log_id,trans_id；

（3） 并发填充缓冲区之前决定checksum；

（4） 必须提前给NOP和SWITCH_LOG日志预留log_id和缓冲区位置；

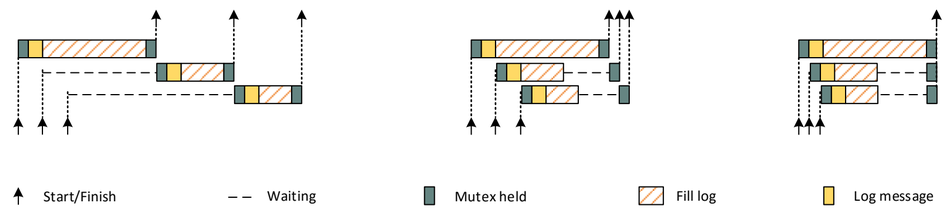
（5） 写磁盘之前，要决定file_id和file_offset；

（6） 每次写磁盘的日志不能超过2M。

为了充分利用多核扩展性避免使用latch和lock,工作线程并发处理待提交事务时应尽量使用原子操作决定log_id,trans_id,file_id，file_offset和checksum。

占位设计

占位操作，工作线程填充日志缓冲区之前先获得缓冲区的位置，并生成该条日志特有的日志信息，如log_id，trans_id，file_id，file_offset和log checksum。为了实现充分的并发性，减少id、offset和checksum的冲突，将待提交的事务分成多个group，实现时只需保证group之间的连续性，然后采用CAS原子操作为每个待提交的事务抢占缓冲区的位置即可。



日志填充演变过程

为了保证事务日志连续，group需要具备以下条件：

（1） 每个group内的log_id和trans_id都是连续的。只需确定group内第一个事务的log_id和trans_id，其余事务的log_id和trans_id可以根据自己在group内的相对编号(rel_id_)计算出来；

（2） 每个group内的日志是成组写磁盘。在成组提交时，只需确定group内第一个事务开始的file_id和file_offset以及group内日志的总长度即可；

（3） group之间的checksum是连续的。每个group采用不可交换的算法累积计算出一个checksum，而group内的每条日志采用可交换的算法计算出checksum，以group为粒度来保证主备日志一致性；？？？？？具体实现

（4） 每个group对应一个log buffer(多对一)。每个group有一个递增且连续的group_id，根据group_id % GROUP_ARRAY_SIZE来定位group对应的2M buffer的起始位置。group内的事务根据group_id和在group内的相对偏移（rel_offset_）来计算自己填充缓冲区的位置；

（5） group内最后一条日志预留给NOP或者SWITCH_LOG；

```
Struct FLogPos
{
  int64_t group_id_;
  int32_t rel_id_;          //相对id
  int32_t rel_offset_;      //相对偏移
  int append(FLogPos& pos, int32_t len, int64_t limit);
}
```

CAS占位结构

缓冲区设计

日志缓冲区，内存中临时存储数据库操作的结构，由多个2M大小的buffer组成，每个buffer称为一个group。每个group中记录了日志长度len_、日志个数count_、起始版本号start_timestamp_、起始日志号start_log_id_、写磁盘起始位置start_log_cursor_以及group状态标识。

```
struct Group
{
    enum { READY=1 };
    int64_t ref_cnt_;                       //group中已填充的日志数
    int64_t len_;                           //group的日志长度
    int64_t count_;                         //group中已占位的日志数
    int64_t ts_seq_ CACHE_ALIGNED;          //标识group的状态，group可占位，可填充；
    int64_t start_timestamp_;               //group起始版本号
    int64_t start_log_id_;                  //group起始日志号
    int64_t last_ts_seq_ CACHE_ALIGNED;     //标识start_log_id可用，可以修改group状态
    int64_t last_timestamp_;                //上一个group的最后一个版本号
    int64_t log_cursor_seq_ CACHE_ALIGNED;  //标识group状态，可提交
    ObLogCursor start_log_cursor_;          //group提交时写日志的位置
};
```

Group结构体

Group结构体：

（1） 每个group包含三种状态，分别为可占位、可填充、可提交。当ts_seq_ = group_id，表示group为可占位状态；当ts_seq_= group_id + READY,表示group为可填充状态；当log_cursor_seq_ = group_id + READY，表示group为可提交状态；

（2） 当group中的所有日志都刷盘成功并且同步成功之后，group才能重置为可占位状态，即由提交线程设置ts_seq_ = group_id + GROUP_ARRAY_SIZE；

（3） 设置start_log_id和start_timestamp_之后，group才能置为可填充状态，即填充group的第一条占位日志时设置ts_seq_ = group_id + READY(此处也可由group的第一个填充线程去设置)；

（4） 设置start_log_cursor_之后，group才能置为可提交状态，即由上一个group的最后一个日志填充线程设置当前group的log_cursor_seq_ = group_id + READY；

（5） group中最后一个占位线程设置len_和count_；

（6） 每个工作线程在填充时对ref_cnt_进行原子加1操作，最后一个完成填充的线程(ref_cnt_ = count_)将ref_cnt_置为0。

特殊写事务设计