日志同步优化功能开发文档

修订历史

版本	修订日期	修订描述	作者	备注
Cedar 0.2	2016-07-05	日志同步优化功能开发文档	储佳佳 郭进伟	

1 总体设计

1.1 综述

Cedar是由华东师范大学数据科学与工程研究院基于OceanBase 0.4.2 研发的可扩展的关系数据库，实现了海量数据集上的跨行跨表事务。支持主备集群的配置，每个集群内配置一台RS（集群管理节点）和一台UPS（事务管理节点），为了提供高可用性，主集群的UPS处理事务更新请求后，会将相关日志同步到备集群，备集群进行日志回放、写盘，从而达到和主集群一致的状态。主备集群UPS之间的日志传递、日志写盘、日志回放等一系列操作即是日志同步模块的主要功能。

Cedar 0.1版本中已经实现了三集群架构下的主备节点日志强同步&强一致的功能。但是在应用过程中发现，该功能存在性能优化空间，而且在一些极端场景中可能会出现日志丢失的情况。为了规避异常、提升整体性能，在Cedar 0.2中我们对日志同步流程进行了优化。优化目标主要有下述几点：

- 将最大可提交日志号写入日志记录中，避免额外线程频繁将提交点信息写入文件的操作，减少主节点非必需的资源开销；
- 调整备节点接收日志后的处理顺序为先将日志写入磁盘，再回复主UPS，最后回放日志，减小主节点答复客户端的延迟，提升更新操作的性能；
- 优化主备切换时对于未决日志的处理方式，规避日志丢失情况的出现。

1.2 名词解释

RS：RootServer，主控服务器，管理集群内所有服务器，以及Tablet的分布和副本管理，在多集群架构中，负责集群间选主。

UPS：UpdateServer，更新服务器，存储更新数据，生成事务日志，并将其同步到备机和写入到本地磁盘。

主RS：主集群中的主RootServer。

备RS：备集群中的主RootServer。

主UPS：主集群中的主UpdateServer。

备UPS：备集群中的主UpdateServer。

1.3 功能

在Cedar中，日志同步采取的是强同步&强一致的机制，每次主UpdateServer接收到事务的更新请求时，会生成操作日志，接着将更新操作的日志发送到所有的备UPS。备UPS接收到主UPS的日志后，确认将日志写入磁盘成功后才能响应主UPS，主UPS接收到多数派（超过半数）UpdateServer的响应之后才能提交对应的事务。Cedar 0.1中备机接收到日志后的处理流程为，先进行日志回放，再写入本地磁盘，最后回复主机。因为采取强一致的模型，主机需要收集到至少多数派备机的回复后才能进行事务的提交操作，备机的保守式操作（回复主机发生在确保回放、落盘操作都完成以后）会影响到主机的事务响应效率。所以在Cedar 0.2中，修改备机接收日志后的处理流程为，先将日志写入本地磁盘，再回复主机，最后备机进行日志回放。这样就能在保证强一致的前提下，尽可能减少主机响应客户端事务请求的延迟时间。

相较于Cedar 0.1版本中用单独的线程频繁地将日志提交点信息写入本地文件，0.2中修改了原有的日志记录的结构体，将日志提交点信息加入其中，每次主备节点同步日志时，日志提交点信息随着日志记录在主备节点之间进行同步。备机在接收到主机发送来的日志时，会解析日志获取到提交点信息，并将本地小于该提交点的日志进行提交。

当主节点因为租约过期、网络抖动等原因切换为备节点时，本地可能会有一些未决日志（由主机生成但还未同步到备机或是还未进行提交的日志），为了避免丢失日志的异常情况，当切换为备节点时，该节点需要对从主节点接收到的新日志和本地未决日志进行逐一进行比对的操作，若接收到的日志和本地未决日志相同，则保留本地未决日志，若不同，则删除不同点之后的所有未决日志，从主机重新拉取相应的日志。等到前任主UPS处理完本地未决日志，它将进入正常的日志同步流程。

1.4 性能指标

同等多集群环境配置下，性能提升为Cedar 0.1版本的1.3倍。

2 模块设计

根据功能需求，日志同步优化方案的实现，主要分为3个子模块：

- 主备UPS同步日志提交点信息
- 调换备UPS日志回放、日志写盘、回复主UPS的顺序
- 未决日志比对

2.1 主备UPS同步日志提交点信息

2.1.1 主要流程

优化方案中将日志提交点信息加入日志记录的结构体中，每次主备节点同步日志时，日志提交点信息随着日志记录在主备节点之间进行同步。主UPS每次生成日志时，需要将当前多数派备机已经回复的最大日志号信息写入日志记录中，再将日志异步地发送给所有备UPS。备UPS在接收到主UPS发送来的日志时，会解析日志获取到提交点信息，并将本地小于该提交点的日志进行提交。

2.1.2 关键数据结构

在日志记录的结构体中，添加提交点信息的属性，使得主备UPS之间可以借由日志同步来传递提交点信息：

```
//日志记录的结构体
struct ObRecordHeader
{
  int16_t magic_;            // magic number
  int16_t header_length_;    // header length
  int16_t version_;          // version
  int16_t header_checksum_; // header checksum
  int64_t timestamp_;        // reserved,must be 0
  ---------------------add begin-----------------------
  int64_t max_cmt_id_;       // commit point
  ----------------------add end-----------------------
  int32_t data_length_;      // length before compress
  int32_t data_zlength_;     // length after compress,
                             // if without compresssion
  int64_t data_checksum_;    // record checksum
  ObRecordHeader();
  …
}
```

备UPS接收到主UPS发送来的日志时，会调用slave_receive_log函数解析日志，并从日志中提取提交点信息：

```
int slave_receive_log()
{
  //解析日志，获取提交点信息
  parse_log_buffer(...replay_worker_->get_master_cmt_log_id());
}
```

2.2 调换备UPS日志回放、日志写盘、回复主UPS的顺序

2.2.1 主要流程

> 调换执行顺序后的主备UPS日志同步流程如下：
>
> 1. 主UPS将事务日志异步发送给所有的备UPS，然后将该批日志同步刷入到本地磁盘；
> 2. 备UPS将从主UPS接收到的CommitLog存入本地磁盘；
> 3. 备UPS将日志写盘成功后应答主UPS；并在应答消息中携带下一次刷磁盘的日志序列号和本机的状态；
> 4. 备UPS进行日志回放；
> 5. 主UPS收到备UPS的响应后，更新本地保存的备UPS信息；
> 6. 主UPS根据多数派规则，从所有的备UPS信息中获取到可以提交的日志号，从而相应的事务进入到提交阶段。

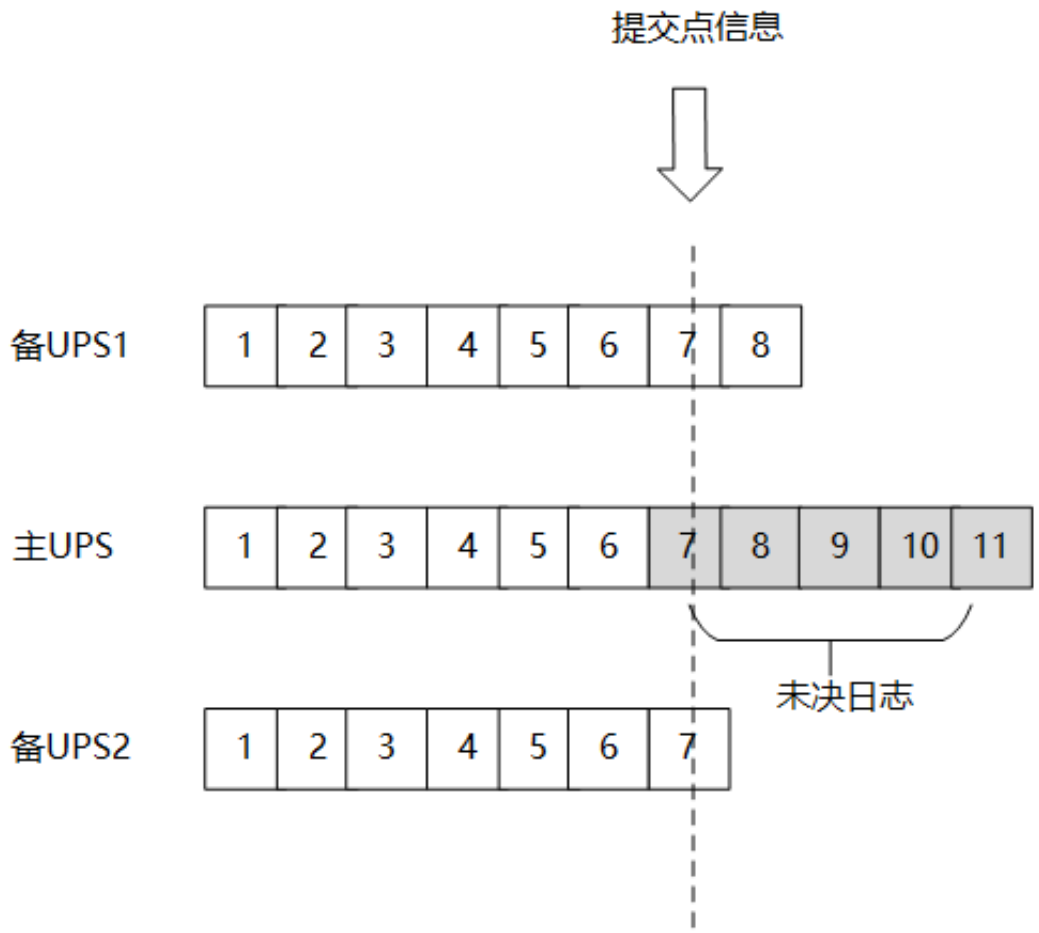
2.2.2 关键算法

修改replay_and_write_log函数，原来该函数中仅有提交日志回放任务的流程，优化方案中在提交回放任务之前，增加将日志写入磁盘的操作：

```
int replay_and_write_log()
{
  //日志写盘
  write_log_as_slave(start_id, buf, len);
  //更新刷写日志的cursor信息
  replay_worker_->set_next_flush_log_id(end_id);
  //提交回放任务
  replay_worker_->submit_batch(real_end_id, buf, len, RT_APPLY);
}
```

2.3 未决日志比对

当主节点因为租约过期、网络抖动等原因切换为备节点时，本地可能会有一些未决日志（如下图所示，由主机生成但还未同步到备机、未提交的日志），为了避免丢失日志的异常情况，当切换为备节点时，该节点需要对从主节点接收到的新日志和本地未决日志进行逐一进行比对的操作，若接收到的日志和本地未决日志相同，则保留本地未决日志，若不同，则删除不同点之后的所有未决日志，从主机重新拉取相应的日志。



进行未决日志比对时，需要调用函数compare_tmp_log，该函数会将日志提交点之后的每条未决日志，和接收到的日志进行比较。

```
int compare_tmp_log()
{
  ...
  while (OB_SUCCESS == err...)
  {
    log_entry.deserialize(log_data_received...);
    //从接收到的日志中读取datachecksum
    received_log_data_checksum = log_entry.header_.data_checksum_;
    //从日志文件中读取相应日志号的日志记录的datachecksum
    get_checksum_by_log_id(tmp_log_data_checksum...)
    //若未决日志与主UPS的日志不同，则用主UPS的日志覆盖本地未决日志
    if(received_log_data_checksum != tmp_log_data_checksum)
    {
      end_cursor = tmp_cursor;
      start_cursor = tmp_cursor;
    }
    ...
  }
  ...
}
```

如果接收到的日志和未决日志的data_checksum值相同(调用get_checksum_by_log_id函数获取指定未决日志中的data_checksum值)，则认为两条日志相同，则cursor前进不重复写盘。若两条日志的data_checksum的值不同，则认为两条日志不同（该未决日志在旧主UPS宕机前未同步到新主UPS，此时新主UPS已经生成了不同的新日志），说明该条日志开始至本地最后一条未决日志都是无效的，则该UPS将这条日志之后的未决日志用主UPS发送来的日志进行覆盖。之后将进入正常的日志同步流程。

```
int get_checksum_by_log_id(const char* log_dir,
                           const int64_t log_id,
                           ...
                           int64_t &checksum)
```

3 模块接口

UPS启动后，需要回放本地日志至提交点，需要调用get_max_cmt_id_in_file函数获取日志提交点信息：

```
/**
* @brief get max commit log id from log file
* @param[out] cmt_id  max commit log id from log file
* @return OB_SUCCESS if success
*/

int get_max_cmt_id_in_file(int64_t& cmt_id,
                           ObLogCursor &tmp_end_cursor) const;
```

可以调用get_cursor_by_log_id函数来获取日志文件中指定日志号对应的cursor：

```
/**
* @brief get log cursor by specified log id
* @param[in] log_dir  the directory containing log files
* @param[in] log_id   the log id specified by user
* @param[out] end_cursor  the cursor matching the specified log id
* @return OB_SUCCESS if success
*/

int get_cursor_by_log_id(const char* log_dir,
                         const int64_t log_id,
                         const ObLogCursor& start_cursor,
                         ObLogCursor& end_cursor);
```

主UPS切换为备UPS时，需要将本地未决日志和新主UPS发送来的日志做比较，判断两条日志是否相同主要需要比较两条日志的data_checksum信息，需要调用get_checksum_by_log_id函数获取本地未决日志的checksun信息：

```
/**
* @brief get checksum by specified log id
* @param[in] log_dir  the directory containing log files
* @param[in] log_id   the log id specified by user
* @param[out] checksum  the checksum of this get
* @return OB_SUCCESS if success
*/

int get_checksum_by_log_id(const char* log_dir,
                           const int64_t log_id,
                           const common::ObLogCursor& start_cursor,
                           common::ObLogCursor& end_cursor,
                           int64_t &checksum);
```

4 使用限制条件和注意事项

- 不支持手动指定任意集群成为主集群（或任意UPS成为主UPS）的功能，仅提供reelect的命令让各个集群重新选举，根据“多数派”“日志最新”等规则随机选出新的主集群。reelect命令如下：
  bin/rs_admin -r [主RS的IP] -p [主RS的端口号] reelect
- 不支持在集群切换过程中进行建表操作，有一定机率新建表的Schema信息未能成功同步至所有集群的所有CS，导致该表对于某些CS不可见。

其他功能的使用方法及规则

UpdateServer配置项参考

参数	缺省值	说明
message_residence_protection_time	150us	备UPS处理日志保护时间，即主UPS认为备UPS处理日志的最快时间，该配置项越小主UPS的发包频率越高。
message_residence_max_time	6ms	主UPS发包的最慢频率