CommiLog强同步设计文档

修订历史

版本	修订日期	修订描述	作者	备注
0.1	2015-12-27	CommitLog强同步的设计文档	郭进伟 刘柏众	

1 系统设计

1.1 综述

OceanBase是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。OceanBase0.4支持单个机群中多UPS的配置，也支持一主多备多集群的配置。在所有UPS中，仅有主集群中的主UPS可以接收更新操作，该UPS将事务的CommitLog同步到本集群中其他备UPS，和其他备集群中的主UPS。OceanBase0.4通过CommitLog的同步机制来避免各种异常（如：程序异常，服务器宕机或者自然灾害）对数据库整体服务造成的影响。

在OceanBase0.4中，UpdateServer提供了配置项wait_slave_sync_type。该配置项为：备UpdateServer收到CommitLog后，应答主UpdateServer的时机。具体如下：

值	说明
0	在CommitLog回放前应答主UpdateServer
1	在CommitLog回放后且写盘前，应答主UpdateServer
2	在CommitLog写盘后，应答主UpdateServer

不幸的是，备UpdateServer设置了超时时间，无论wait_slave_sync_type的值设置为多少，备UpdateServer最终都会响应主UpdateServer（响应即代表成功）。

更糟的是，OceanBase0.4采用了主备同步中的最大可用模式。主UpdateServer为每个CommitLog的同步都设置有超时时间，如果在超时时间内未收到备UpdateServer的响应，主UpdateServer也会提交相应的事务，并将结果返回给客户端。

因此，需要提供一种机制，来保证CommitLog在备机写入磁盘后，主UpdateServer才能提交相应的事务。该机制在本文档中被称之为CommitLog强同步，类似于主备同步中的最大保护模式。但该机制可以配合多集群架构，只需要保证所有集群中半数以上的UpdateServer将Commit Log写入磁盘后，主UpdateServer就可以提交相应的事务。

CommitLog强同步的实现是基于三集群架构，即OceanBase配置为三集群，每个集群内有一个RootServer和一个UpdateServer。RootServer负责集群间选主，选主依据每个集群内UpdateServer的日志信息，通常拥有最新日志的集群可以作为主集群对外提供更新服务。

1.2 名词解释

RS：RootServer，管理集群内所有服务器，以及Tablet的分布和副本管理，在三集群架构中，负责集群间选主。

UPS：UpdateServer，存储更新数据，生成CommitLog，并将其同步到备机和写入到本地磁盘。

主RS：无特殊说明，主RS指的是主集群中的主RootServer。

备RS：无特殊说明，备RS指的是备集群中的主RootServer。

主UPS：无特殊说明，主UPS指的是主集群中的主UpdateServer。

备UPS：无特殊说明，备UPS指的是备集群中的主UpdateServer。

1.3 功能

主UPS每执行一个更新事务时，都会生成一条CommitLog，将其同步到备UPS并写入到本地磁盘。在三集群架构下，CommitLog的强同步策略要求至少成功同步一台备UPS（这里的成功同步意味着备UPS也将相应的CommitLog写盘成功），主UPS才可以提交相应的事务并应答客户端。

在主UPS中，CommitLog同步采用了异步同步备机和同步刷本地磁盘的方式。即主UPS仅需要将CommitLog消息包发送给备UpdateServer，不需要等待回复就可以将CommitLog刷入到本地磁盘。因此，主UPS磁盘中的日志有可能包含未决事务的日志。未决事务的日志有可能会造成系统中主备UPS上的数据不一致，为了避免该情况，需要引入已提交点，小于该提交点的CommitLog对应的事务是确定已经提交的事务。原主UPS宕机重启后，仅需要回放CommitLog到已提交点即可。

集群间选主通常是根据集群内UPS上的日志信息来进行。一种做法是采用日志序列号的方法，但是由于未决事务的存在，可能导致已提交事务对应数据的丢失。另一种做法是采用日志时间戳的方法，该做法可以避免上述的情况。因此，主UPS在生成每一条CommitLog的时候，在其中添加生成该日志的时间。

1.4 性能指标

同等三集群配置下，性能衰减不超过50%。

2 模块设计

根据功能需求，CommitLog强同步的实现，主要分为：

- 日志同步子模块
- 日志回放子模块
- 日志时间戳子模块

2.1 日志同步子模块设计

2.1.1 总体流程

基于三集群的CommitLog强同步要求任何一条日志需要在三个UPS中的两台确认写盘成功后，其对应的事务才能提交。日志同步的流程如下：

1. 主UPS将CommitLog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收到主UPS发送来的CommitLog后，先将CommitLog存储在缓冲区，然后默认等待一段时间让其进行回放并写入到本地磁盘；
3. 备UPS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（sync和nsync）；
4. 主UPS收到备UPS的响应后，更新本地保存的备UPS信息；
5. 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；

2.1.2 备UPS应答消息

OceanBase0.4中备UPS收到主UPS同步的CommitLog后，默认情况下，当日志存入缓冲区后，会立即返回应答消息。该应答消息仅表示备UPS已收到相应的日志，不包含其他信息。为了实现CommitLog的强同步，需要在应答消息中包含本地已刷入磁盘的最大日志号，为了后期的可选性，还需要增加备UPS的状态。

因此需要增加备UPS响应消息的结构体，并且实现序列化和反序列化，以满足网络传输需求，该结构体描述如下：

```
struct ObLogPostResponse
{
  /// 该日志号之前的日志都已刷到本地磁盘
  int64_t next_flush_log_id_;
  /// 本次日志同步消息在备UPS上的滞留时间
  int64_t message_residence_time_us_;
  /// 备UPS的状态，为2表示已与主UPS保持同步，为1表示还没有同步
  int64_t ups_status_;
};
```

2.1.3 备UpdateServer执行流程

OceanBase0.4中，备UPS收到主UPS同步的CommitLog后，给主UPS返回一个确认消息。为了实现CommitLog的强同步，返回的应答消息中需要包含必要的信息，如2.1.2所述。应答消息的构造需要在备UPS的日志接收处理函数中完成，该函数为ups_slave_write_log：

```
int ObUpdateServer::ups_slave_write_log(const int32_t version,
      common::ObDataBuffer& in_buff, easy_request_t* req,
      const uint32_t channel_id, common::ObDataBuffer& out_buff)
```

在上述函数中，需要对其原有流程进行改造，具体如下：

1. 调用ObUpsLogMgr的slave_receive_log函数。该函数将收到的CommitLog存入日志缓冲区中，并根据配置项wait_slave_sync_type的设置，采用不同的等待方式；
2. 调用ObLogReplayWorker的get_next_flush_log_id函数，并用返回值设置应答消息中的next_flush_log_id_；
3. 获取本机状态，并用其设置应答消息中的ups_status_；
4. 将应答消息返回给主UPS。

2.1.4 主UpdateServer执行流程

主UPS收到备UPS的响应消息后，需要将其保存至本地。主UPS保存有每个备UPS的信息，当收到备UPS的响应收，对其进行更新即可。保存某个备UPS信息的结构体描述如下：

```
struct SlaveNode
{
  /// 保存备机地址信息，包括IP和端口号
  ObServer server_;
  /// 以下信息与ObLogPostResponse的属性相对应
  int64_t next_flush_log_id_;
  ObSlaveStatus slave_status_;
};
```

所有备UPS的信息作为ObAckQueue的成员变量进行保存：

```
struct WaitNode
{
  ......
  /// 保存有所有备UPS的信息，备UPS的最大数量默认为6
  SlaveNode slave_array_[SLAVE_COUNT];
};
```

当主UPS收到备UPS的响应消息后，便会调用ObAckQueue中的callback函数。主UPS处理备UPS响应的具体流程如下：

1. 反序列化响应消息，并更新相应的备UPS的信息；
2. 根据多数派原则，从所有的备UPS信息中获取下一条可以提交的日志序列号，并将该值更新至本地；
3. 唤醒主UPS上的提交线程，对相应的事务进行提交。

2.2 日志回放子模块设计

2.2.1 关键件

为了避免未决事务有可能造成主备UPS数据不一致的问题，主UPS需要引入已提交点，并将其保存至固定的文件中。在三集群架构中，仅有主UPS会出现未决事务，因此每当UPS启动的时候，需要将自己的角色持久化到本地磁盘，以便下一次启动时使用。

- commit_point（已提交点）：记录主UPS可以进行提交的下一个日志序列号。该点仅由主UPS更新，并保存本地即可。
- was_master（是否当过主UPS）：标记UPS在最近一次工作时是否当过主UPS。0：未当过主UPS，1：当过主UPS。

commit_point的更新分为两种：

- 同步更新：每当主UPS要提交事务之前，将事务对应的日志序列号记录至commit_point中。
- 异步更新：由后台线程定时获取可以提交的事务对应的日志序列号，将其记录值commit_point中。

以上的commit_point的更新类型可配置，默认为同步更新，配置项说明如下：

配置项	默认值	说明
commit_point_sync_type	1	0：异步更新；1：同步更新。

was_master的更新在UPS刷写磁盘的时候，主备UPS刷写磁盘时调用的函数不同，因此可以在其中更新was_master的值。

2.2.2 UPS日志回放流程

UPS启动后的日志回放流程如下：

1. 判断自身是否为主UPS。如果是主UPS，则回放本地日志文件中“回放起点”后所有的CommitLog，否则执行下一步；
2. 读取was_master文件，判断上一次的运行是否为主UPS。如果不是主UPS，则回放本地日志文件中“回放起点”后所有的CommitLog，否则执行下一步；
3. 读取commit_log，获取已提交点。如果获取失败，则放弃“回放起点”后的所有CommitLog，从主UPS上获取所缺日志，否则执行下一步；
4. 回放本地日志至已提交点。

根据was_master和commit_point的不同取值，备UPS启动总共分为四种情况，每种情况下的日志回放方式如下表所示。其中第4种情况表明出现了“异常”，对应措施是不回放本地日志文件中“回放起点”后面的日志，所缺日志全部从主UPS拉取。

编号	was_master	commit_point	重启前为主	有效已提交点	类型	回放本地日志方式
0	不存在或0	有	否	是	正常	正常回放
1	1	有	是	是	正常	回放到已提交点
2	不存在或0	无	否	否	正常	正常回放
3	1	无	是	否	异常	不回放

注意：UPS启动后如果发现自身为主UPS，则正常回放本地日志即可。

2.3 日志时间戳子模块设计

在三集群架构下，RS之间的选举需要依据本集群中UPS的日志信息。为了避免已提交事务对应数据的丢失，这里的日志信息为日志文件中最大日志时间戳。

在OceanBase0.4中，没有在每条ComitLog中显式地记录该日志生成时间。因此，需要在每条日志的公共部分显式地保存生成该日志的时间戳。ObRecordHeader结构体中有一个int64_t类型的未使用的保留字段reserved_，可用于记录日志的时间戳：

```
struct ObRecordHeader
{
  ......
  /// reserved
  int64_t timestamp_;
  ......
}
```

UPS获取日志时间戳的步骤如下：

1. 获取日志文件中的最大日志时间戳：当UPS启动并回放完本地日志后，扫描日志文件，获取日志文件最后一条日志中的日志时间戳，用该时间戳作为本地日志文件中的最大日志时间戳。该过程仅执行一次；
2. 获取日志缓冲区中最大日志时间戳：UPS正常工作过程中，无论是Master还是Slave，都需要把日志刷入磁盘。因此，在把一批日志刷入磁盘前，记录下该批日志的

最后一条日志的时间戳为日志缓冲区中最大的日志时间戳；

3. 获取本地最大日志时间戳：比较日志文件中的最大日志时间戳和日志缓冲区中的最大日志时间戳，选其中较大的作为本地最大日志时间戳。

3 模块接口

3.1 对外接口

UPS启动后，需要设置多数派的个数，调用ObUpsSlaveMgr中的set_ack_queue_majority_count函数（三集群架构下该值设置为2）：

```
/**
 * @brief set the number of majority of UPS
 * @param majority_count [in]
 * @return OB_SUCCESS if success
 */
void set_ack_queue_majority_count(int32_t majority_count);
```

三集群架构下，RS在选举时需要通过RPC来获取UPS的日志时间戳，需调用ObRootRpcStub中的get_ups_max_log_timestamp函数：

```
/**
 * @brief get max log timestamp from updateserver
 * @param ups [in] the address of ups
 * @param max_log_timestamp [out] max log timestamp from updateserv
er
 * @return OB_SUCCESS if success
 */
virtual int get_ups_max_log_timestamp(const common::ObServer& ups,
              int64_t &max_log_timestamp, const int64_t timeout_u
s);
```

在UPS中，可以通过调用ObUpsLogMgr中的get_max_log_timestamp函数来获取最大日志时间戳：

```
/**
 * @brief get max log timestamp from local log file and log buffer
 * @param max_timestamp [out] max log timestamp
 * @return OB_SUCCESS if success
 */
int get_max_log_timestamp(int64_t& max_timestamp) const;
```

4 使用限制条件和注意事项

CommitLog强同步的实现是基于三集群架构，因此对更多集群的支持并不友好。

其他功能的使用方法及规则

新增配置项

UpdateServer配置项参考

参数	缺省值	说明
commit_point_sync_type	1	主UPS写commit point的方式。 0：异步方式；1：同步方式。
rs_election_lease_protection_time	150ms	RS租约保护时间，如果RS租约已陷入到保护时间内，主UPS则认为RS选举租约过期。
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