MySQL 外部 XA 及其在分布式事务中的应用分析

何登成

1 原理

关于 XA，分布式事务处理的原理，可见[3]；关于 MySQL XA 的说明，可见[1][2]。

MySQL XA 分为两类，内部 XA 与外部 XA；内部 XA 用于同一实例下跨多个引擎的事务，由大家熟悉的 Binlog 作为协调者；外部 XA 用于跨多 MySQL 实例的分布式事务，需要应用层介入作为协调者(崩溃时的悬挂事务，全局提交还是回滚，需要由应用层决定，对应用层的实现要求较高)；

本文，假设读者已经知道 MySQL 外部 XA 的使用，而将重点放在 MySQL 如何处理外部 XA 的 crash recover，以及面对不同的 crash recover 的情形，应用程序如何处理，才能够保证分布式事务的一致性。最后，本文简单分析一下目前 MySQL 外部 XA 支持存在的问题，以及可选的解决方案。

源代码分析基于 MySQL 5.1.49，MySQL 5.5.16。

2 MySQL 处理流程

2.1 MySQL 外部 XA –正常处理流程

MySQL 外部 XA 的正常处理流程，这里不准备介绍，可以参考[1][2][3]。接下来我重点描述一下 MySQL 外部 XA 的崩溃恢复流程，毕竟此流程跟应用程序如何正确使用外部 XA 息息相关。

2.2 MySQL 外部 XA –崩溃恢复流程

若一个运行外部 XA 事务的 MySQL 节点发生崩溃，那么其重启之后的崩溃恢复，涉及到外部 XA 处理的流程如下：
Crash recover：

```
// 1. 读取 binlog 文件，将文件中的 xid 存入 commit_list hash 表
//    顾名思义，所谓的 commit_list，就是说此 list 中对应 prepare 状态的 xid
//    在崩溃恢复过程中均可以被提交，而不在 commit_list 中的 xid，均须回滚
//    binlog 中的 xid，都是属于内部 xid，由 MySQL 产生，用于内部 XA
Log.cc::TC_LOG_BINLOG::recover
```

```
// 2. 遍历底层所有的事务引擎，收集处于 XA_PREPARED 状态的所有 xid
//    这些 xid 列表，既包括内部 xid，也包括外部 xid，存储引擎内部不做区分
Handler.cc::ha_recover(commit_list)

    // 执行各引擎层面提供的 recover 方法，收集所有的处于 prepared 状态的 xid
    // 根据 xid 分类：
    // 3. 若 xid 属于内部 xid，那么在 commit_list 中查找此 xid，
    //    若存在，则提交此 xid 对应的事务；否则，回滚此事务
    // 4. 若 xid 属于外部 xid，那么则将 xid 插入 xid_cache hash 表
    //    xid_cache 中的所有 xid，将会通过 xa recover 命令返回，等待外部程序决策
    Handler.cc::xarecover_handlerton

        // 5. 收集 InnoDB 引擎中，处于 prepare 状态的所有 xid，并返回
        got = hton->recover(innobase_xa_recover)

        my_xid x = info->list[i].get_my_xid();
        if (!x)
            // 若当前为外部 xid，那么将 xid 插入 xid_cache hash 表
            xid_cache_insert(&xid_cache, x);
        else
            if (x in commit_list)
                // 若当前为内部 xid，同时此 xid 在 binlog 中存在，则提交
                hton->commit_by_xid();
            else
                // 若当前为内部 xid，同时此 xid 在 binlog 中不存在，则回滚
                hton->rollback_by_xid();
```

通过以上的分析，可以总结出：

- MySQL 内部，会对 xid 做区分。内部 xid 有 MySQL 自己产生(MySQL 内部 xid 格式，将在本文下面给出)，用于多引擎间事务的一致性；外部 xid 由应用程序给出，用于跨多 MySQL 实例的分布式事务。但是存储引擎层不做区分(区分在 MySQL 上层)。

- crash recover 时，存储引擎负责将引擎内部，处于 prepare 状态的事务收集，并返回 MySQL 上层。

- Binlog 作为内部 XA 的协调者[5]，在 binlog 中出现的内部 xid，在 crash recover 时，由 binlog 负责提交；在 binlog 中未出现的 xid，由 binlog 负责回滚。(这是因为，binlog 不进行 prepare，只进行 commit，因此在 binlog 中出现的内部 xid，一定能够保证其在底层各存储引擎中已经完成 prepare)。

- 外部 XA 事务的 xid，在 crash recover 过程中仅仅是插入 xid_cache 中，而不做其他处理。等到用户发起 xa recover 命令时，将 xid_cache 中处于 prepare 状态的 xid 返回。

- xa recover 命令的流程处理如下。

xa recover 命令处理流程：

```
sql_parse.cc::mysql_execute_command
    case SQLCOM_XA_RECOVER:
        mysql_xa_recover();
            // 遍历 xid_cache，找出其中的状态处于 XA_PREPARED 的事务，发送客户端
            while (xs = hash_element(&xid_cache,))
                if (xs->xa_state == XA_PREPARED)
                    protocol->write();
```

根据 xa recover 命令收集到的各 MySQL 实例返回的 xid 列表，然后再对比应用程序端日志，决定这些 xid，哪些全局 commit，哪些 rollback。

由于测试中只有一个 MySQL 实例，因此此时可以直接选择 commit 处于 prepare 状态的 xid。

2.3 MySQL 内部 xid 格式

上面提到，MySQL 有外部 XA 与内部 XA，内部 XA 对应的 xid 由 MySQL 内部产生，有特定的格式：

- MySQL 内部 xid 格式：　MYSQL_XID_PREFIX + server_id + my_xid

  MYSQL_XID_PREFIX：　MySQLXid(源码写死)　8 bytes

  server_id：　MySQL 实例的 id，ulong，　4 bytes

  my_xid：　内部自增序列，ulonglong，　8 bytes

  MySQL 内部 xid 由以上 3 部分组成，总长度为 20。

  判断是否为内部 xid 的代码如下：

```
gtrid_length == MYSQL_XID_GTRID_LEN
&&bqual_length == 0
&&!memcmp(data, MYSQL_XID_PREFIX, MYSQL_XID_PREFIX_LEN)

其中：MYSQL_XID_GTRID_LEN = 20；MYSQL_XID_PREFIX_LEN = 8；
```

  例如：”MySQLXid 0004”

  server_id = ‘’；my_xid = 4

  因此，使用时应该注意，不要在外部构造这种形式的 xid，否则 MySQL 就会将内部 xid 与外部 xid 混淆。

  在测试中，我构造了一个外部 xid = ‘MySQLXidxxxx00100000’，长度为 20 bytes，前八个字符为’MySQLXid’。在事务完成 xa prepare 之后，关闭 MySQL 数据库。MySQL 在 crash recover 时，直接将此 xid 认为是内部 xid，并在内部由 Binlog 直接 rollback 此事务，导

致使用 xa recover 命令无法看到任何 prepare 状态的 xa 事务。

但是，反过来考虑，若是应用程序本身不想处理悬挂事务，那么将外部 xid 构造成内部的形式不失为一种较好的策略，由 binlog 来负责处理悬挂事务的提交与回滚。付出的代价则是：崩溃时，未提交事务在各个 MySQL 实例上的状态可能不一致(部分节点提交；部分节点回滚)。

2.4 MySQL 崩溃恢复& Binlog

前面提到了 MySQL 外部 XA 的崩溃恢复流程。在本小节我们简单分析一下崩溃恢复过程中的 Binlog 文件的读取问题。

通过跟踪 TC_LOG_BINLOG::open 函数，发现在 crash recover 过程中，MySQL 全量读取最后一个 Binlog 文件，这与 MariaDB WorkLog#164：Extend crash recovery to recover non-prepared transactions from binlog[6]中的说法一致：*...The existing scan always scans the full last binlog file, and we should keep this...*

但是这样就带来一个疑问：
为什么仅仅全量读取最后一个 Binlog 文件就可以呢？如果最后一个 binlog 文件很短，如何保证底层引擎处于 prepare 状态的事务不会出现在前一个 Binlog 文件之中？

回答这个疑问，需要从目前 MySQL 写 Binlog 与底层存储引擎(InnoDB)写 redo log 的方式分析：

1. 同一事务只能写到同一个 Binlog 文件中，不能跨文件。
2. 为了保证底层引擎 Commit 顺序与 Binlog 顺序一致，目前 MySQL+InnoDB 不支持 group commit(新版的 Precona，MariaDB 除外)，同一时间只有一个事务可以进行提交(内部的 XA 事务，二阶段提交)：InnoDB prepare + Binlog flush + InnoDB commit 这一系列操作。因此下一个事务开始进行 InnoDB prepare 时，前一个事务的系列动作一定结束，事务已经提交。意味着 crash recovery 时，最多只有一个 InnoDB 事务处于 prepare 状态。
3. 结合 1，2 可得，最后一个 prepare 事务一定位于最后的 Binlog 文件中。

上面说到，由于 MySQL+InnoDB 不支持 group commit，因此只读最后一个 Binlog 是可行的，那么如果是最新版的 Percona/MariaDB，已经支持 group commit (关于 group commit 的具体实现，可以参考我的另外一篇短文：MariaDB&PerconaXtraDB Group Commit 实现简要分析[7])，那么仍旧读取最后一个 Binlog 文件是否一样可行呢？

答案是肯定的，因为目前 Percona/MariaDB 的最新版本实现中，仍旧采用的是全量读取最后一个 Binlog 文件的策略，那么此时又是如何保证前一个 Binlog 文件中所有的日志对应的事务，其在底层 InnoDB 引擎中已经完成提交动作了呢？
经过阅读 MariaDB 5.3.4 的代码，我找到了答案：

1. 同一事务只能写在同一 Binlog 文件中，不能跨文件，这个要求仍旧保留。
2. Binlog 在进行 group commit 时，需要统计参与本次 group commit 的所有内部 XA 事务的数量(prepared_xids，何用？)。

3. 若当前 Binlog 文件已经超出指定的大小，需要切换，那么在切换之前，必须等待当前 Binlog 文件对应的 prepared_xids 归零(换句话说，也就是要保证当前 Binlog 文件中的所有内部 XA 事务，在存储引擎中全部提交，完成 commit & fsync)。如此一来，就能够保证切换到新的 Binlog 文件之后，老的 Binlog 文件对应的所以事务，都已经确定提交。
4. prepared_xids 归零前提？要让 prepared_xids 归零，首先必须将新的 Binlog group commit 暂停，通过对 LOCK_log mutex 加锁即可实现(LOCK_log mutex 功能可见[7]，新的 binlog group commit 开始前，必须获得此 mutex)。
5. prepared_xids 归零操作？Binlog 模块(TC_LOG_BINLOG)提供一个 unlog 方法，该方法每调用一次，对 prepared_xids --，直到 prepared_xids 归零，即可进行 binlog 文件的切换操作。每个事务，在完成所有的 commit 步骤(包括底层的存储引擎 commit)，返回用户之前，调用此方法；若 binlog group commit 中有事务失败，同样调用此方法。因此，只要 binlog 中的事务对应的底层引擎全部完成 commit，prepared_xids 一定为 0，也意味着可以切换 Binlog 文件。
6. 总结：group commit 下的 crash recovery，同样只需要遍历最后一个 Binlog 文件即可。MariaDB 在实现 group commit 的过程中，已经改动 binlog 的实现，用于支持此方法。

同样还是在 MariaDB WL#164[6]中，提到了遍历 binlog 的一种优化，目前，InnoDB redo log 在 commit 日志中已经记录了对应的 Binlog 日志的(文件名，位置)信息。只要将此信息返回，就可以从指定位置开始遍历 Binlog，如此一来，使用更大的 Binlog 文件，也不会影响 crash recovery 时，读取 Binlog 文件的性能。

3 MySQL 外部 XA 分析

3.1 作用分析

MySQL 外部 XA 可以用在分布式数据库代理层，例如开源的代理工具：ameoba[4]，网易的 DDB，淘宝的 TDDL，B2B 的 Cobar 等等。

通过 MySQL 外部 XA，这些工具可以提供跨库的分布式事务。当然，这些工具也就成了外部 XA 事务的协调者角色。在 crash recover 时控制悬挂事务是全局 commit，或者 rollback。

在 crash recover 之后，外部应用程序可能会遇到以下几种情况：

- 情况一：分布式事务对应的 MySQL 实例，部分完成 prepare，部分未完成 prepare。此时直接回滚完成 prepare 的实例即可。n_prepared <Total Nodes (处于 prepare 状态的节点数量要小于参与分布式事务的所有节点总数)。

- 情况二：分布式事务对应的 MySQL 实例，全部完成 prepare，未开始进行 commit。此时即可提交此事务，也可回滚此事务(根据分布式事务原理，所有节点都完成 prepare，应该提交)。n_prepared = Total Nodes。

- 情况三：分布式事务对应的 MySQL 实例，全部完成 prepare，并且部分节点已经完成

commit。此时应该提交该事务处于 prepare 状态的节点。n_prepared < Total Nodes。对比情况三与情况一，仅仅通过 prepare 节点的数量无法区分，因此应用程序需要在 prepare 完成之后记录日志(此时，应用程序起着事务协调者(Transcaction Coordinator)的角色，而根据 MariaDB WorkLog#132[5]的说法，TC 角色是可以进行"middle engine"优化的，不需要 prepare 过程，所有 MySQL 节点 xa prepare 返回之后，应用程序直接写 commit 标识即可，然后再对每个 MySQL 节点进行 xa commit 操作。)，从而用于区分情况一与情况三。

- 情况四：分布式事务对应的 MySQL 实例，全部完成 commit。此时事务已经提交成功，xid 不会出现在执行 xa recover 的任一个节点。不需要特殊处理。

- 情况五：未记录任何 prepare 日志。那么所有的事务，在各个存储引擎的 crash recover 时，都会被回滚，不需要外部特殊处理。

3.2 MySQL 外部 XA 不足

通过前面的分析，可知应用程序配合 MySQL 的 XA 事务功能，能够较好的支持分布式环境下的事务。但是，这个支持并不完美，根据我的分析，有可能会出现以下几个问题：

- 问题一：主备库数据不一致。
  MySQL 的主备库的同步，通过 Binlog 的复制完成。而 Binlog 是 MySQL 内部 XA 事务的协调者，并且 MySQL 为 binlog 做了优化——binlog 不写 prepare 日志，只写 commit 日志。

  考虑前面提到的情况二，所有的参与节点 prepare 完成，在进行 xa commit 前 crash。crash recover 如果选择 commit 此事务。由于 binlog 在 prepare 阶段未写，因此主库中看来，此分布式事务最终提交了，但是此事务的操作并未写到 binlog 中，因此也就未能成功复制到备库，从而导致主备库数据不一致的情况出现。

  在 MySQL 5.5.16 版本中做过测试，这个问题实际存在。crash recover 之后，对 xa recover 返回的事务运行 xa commit，对应事务提交，但是操作并未写入 binlog，因此无法复制到备库。

  那么是否回滚所有 prepare 的事务，就可以避免此问题呢？结论是仍旧不行，不仅不能解决问题一，甚至可能引起问题二。

- 问题二：同一事务，在各参与节点，最终状态不一致(部分提交，部分回滚)。
  若回滚所有 prepare 状态的分布式事务，会产生问题二。考虑情况三(所有节点完成 prepare，部分节点完成 commit)，该分布式事务对应的节点，部分已经提交，无法回滚，而部分节点回滚。最终导致同一分布式事务，在各参与节点，最终状态不一致。

- 问题三：源码级别问题。MySQL 5.1.49 源码对于外部 XA 事务处理存在 bug，在 MySQL 5.5.16 版本中，此 bug 已经被 fix。经过验证发现，在我已下载的 MySQL 5.1.61 与之后的所有版本，此 bug 均已经被 fix。

在 MySQL 5.1.49 中，所有 xa recover 返回的外部 xid，都不能被提交。原因如下：

当运行 xa commit ‘xid_name’命令时，MySQL 会判断当前 xid_name 的错误信息，若存在错误信息，那么就在内部将 xa commit 命令强制转换为 xa rollback。xid_name 的状态存于 xid_cache 中，在 crash recover 阶段，由函数 Handler.cc::xarecover_handlerton 调用 xid_cache_insert(&xid_cache, x)函数完成插入。MySQL 5.1.49 在实现 xid_cache_insert 函数有 bug。

```
…
xs->xa_state=xa_state;
xs->xid.set(xid);
xs->in_thd=0;
xs->rm_error=0;
res=my_hash_insert(&xid_cache, (uchar*)xs);
…
```

MySQL 5.1.49 中，缺少了 xs->rm_error =0 这一行，未初始化 rm_error，导致 xa commit 时判断出错，无法 commit。MySQL 5.5.16 已经 fix 此 bug，加上了黑色这一行的初始化，应用程序可以 xa commit。

3.3 不足的解决方案

从 MySQL 外部 XA 不足的分析可以看出，除了实现 bug 之外，产生其余两个问题的最大原因，还是在于 MySQL 针对 binlog 做的”middle engine”优化，binlog 的 prepare 不写日志。在 MySQL 内部 XA 事务中，这个优化是可行的，因为 Binlog 本身的角色就是事务协调者(Transaction Coordinator)，事务协调者可以不进行 prepare [5]。

但是对于 MySQL 外部 XA 事务，Binlog 已经不是事务协调者的角色，其也是一个参与者，或者说是 Resource Manager。因此 Binlog 的 prepare 日志是不可省略的。

为了解决 MySQL 外部 XA 事务 crash recover 过程中出现的问题，我觉得只能修改 binlog 模块。使 binlog 模块在正常运行过程中也区分内部 XA 事务与外部 XA 事务。内部 XA 事务可以仍旧沿用现在的方案；而外部 XA 事务，需要增加写 prepare 日志的功能，已经 crash recover 时处理 prepare 日志的功能。

4 参考资料

[1] Sergei Golubchik.Distributed Transaction Processing with MySQL XA

[2] http://dev.mysql.com/doc/refman/5.1/en/xa.html

[3] X/Open.Distributed TP: The XA Specification

[4] 陈思儒. Amoeba

[5] MariaDB WorkLog#132: Transaction coordinator plugin

[6] MariaDB WorkLog#164: Extend crash recovery to recover non-prepared transactions from binlog
[7] 何登成. MariaDB&PerconaXtraDB Group Commit 实现简要分析
[8]
[9]