

3.25 MySQL 从库出现 system lock 的原因

作者：高鹏

本文为笔者 2 年前写一篇说明性文章，发现很多同学都在问这个问题，因此做一次分享。

- 本文基于 5.7.17 源码
- 本文只考虑 row 格式 binlog
- 主要考虑 DML 语句，DDL 语句比较简单不做考虑
- 以单 sql 线程为例（非 MTS）

一、延迟的计算方式

其实每次 show slave status 命令的时候后台会调用函数 show_slave_status_send_data 进行及时计算，这个延迟并不是保存在哪里的。栈帧如下：

```
#0 show_slave_status_send_data (thd=0x7fffd8000cd0, mi=0x38ce2e0,
  io_gtid_set_buffer=0x7fffd800eda0 "e859a28b-b66d-11e7-8371-000c291f347d:42-100173",
  sql_gtid_set_buffer=0x7fffd8011ac0 "e859a28b-b66d-11e7-8371-000c291f347d:1-100173")
  at /MySQL/MySQL-5.7.17/sql/rpl_slave.cc:3602
#1 0x000000001867749 in show_slave_status (thd=0x7fffd8000cd0) at /MySQL/MySQL-5.7.17/sql/rpl_slave.cc:3982
#2 0x000000001867bfa in show_slave_status_cmd (thd=0x7fffd8000cd0) at /MySQL/MySQL-5.7.17/sql/rpl_slave.cc:4102
```

其计算方式基本就是这段代码，

```
time_diff= ((long)(time(0) - mi->rli->last_master_timestamp) - mi->
  clock_diff_with_master);
```

稍微解释一下：

- time(0)：取当前 slave 服务器系统时间。
- mi->rli->last_master_timestamp：是 event common header 中 timestamp 的时间 + exetime，其中 exetime 只有 query event 才有，其他全部是 0，这也导致了 binlog row 格式下的延迟最大基本是（2 乘以主库的执行的时间），但是 DDL 的语句包含在 query event 中索引延迟最大基本就是（1 乘以主库执行时间）
- mi->clock_diff_with_master：这是从库和主库时间的差值。
- 这里我们也看到 event 中 common header 中的 timestamp 和 slave 本地时间起了决定因素。因为每次发起命令 time(0) 都会增加，所以即便 event 中 common header 中的 timestamp 的时间不变延迟

也是不断加大的。

另外还有一段有名的伪代码如下：

```
/*
The pseudo code to compute Seconds_Behind_Master:
if (SQL thread is running)
{
    if (SQL thread processed all the available relay log)
    {
        if (IO thread is running)
            print 0;
        else
            print NULL;
    }
    else
        compute Seconds_Behind_Master;
}
else
    print NULL;
*/
```

其实他也来自函数 `show_slave_status_send_data`，有兴趣的自己在看看，我就不过多解释了。

二、Binlog 写入 Binlog 文件时间和 event 生成的时间

我发现有朋友这方面有疑问就做个简单的解释：

- binlog 真正从 binlog cache/tmp file 写入 binlog 文件是在 commit 的 flush 阶段然后 sync 阶段才落盘。
- event 生成是在语句执行期间，具体各个 event 生成时间如下：
 - 1) 如果没有显示开启事物，Gtid event/query event/map event/dml event/xid event 均是命令发起时间。
 - 2) 如果显示开始事物 Gtid event/xid event 是 commit 命令发起时间，其他 event 是 dml 语句发起时间。

所以 binlog Event 写入到 binlog 文件和 Event 的中的时间没有什么联系。下面是一个小事物典型的 event 生命周期，这是工具 `infobin` 生成的：

```
>GtidEvent:Pos:234(0Xea) N_pos:299(0X12b) Time:1513135186Event_size:65(bytes)
Gtid:31704d8a-da74-11e7-b6bf-52540a7d243:100009 last_committed=0 sequence_number=1
-->QueryEvent:Pos:299(0X12b) N_Pos:371(0X173) Time:1513135186Event_size:72(bytes)
Exe_time:0Use_db:test Statment(35b-trun):BEGIN/*!Trx begin!*/Gno:100009
---->MapEvent:Pos371(0X173) N_pos:415(0X19f) Time:1513135186Event_size:44(bytes)
TABLE_ID:108 DB_NAME:test TABLE_NAME:a Gno:100009
----->InsertEvent:Pos:415(0X19f) N_pos:455(0X1c7)
```

```

Time:1513135186Event_size:40(bytes)
Dml on table: test.a table_id:108Gno:100009
>XidEvent:Pos:455(0X1c7) N_Pos:486(0X1e6) Time:1513135186Event_size:31(bytes)
COMMIT; /*!Trx end*/Gno:100009

```

三、造成延迟的可能原因

这部分是我总结现有的我知道的原因：

- 大事物延迟，延迟略为 2 执行时间，状态为：reading event from the relay log
- 大表 DDL 延迟，延迟略为 1 执行时间，状态为：altering table
- 长期未提交的事物延迟，会造成延迟的瞬时增加
- 表上没有主键或者唯一键，状态为：system lock 或者 reading event from the relay log
- InnoDB 层锁造成延迟，状态为：system lock 或者 reading event from the relay log
- 从库参数设置如 sync_binlog, sync_relay_log, InnoDB_flush_log_at_trx_commit 等参数

这些原因都是我遇到过的。

接下来我想分析一下从库 system lock 形成的原因。

四、问题由来

问题主要是出现在我们的线上库的从库上，我们经常发现某些数据量大的数据库，sql thread 经常处于 system lock 状态下，大概表现如下：

```
MySQL> show processlist;
```

```

+-----+-----+-----+-----+-----+-----+-----+
| Id | User          | Host          | db   | Command | Time | State |
|-----+-----+-----+-----+-----+-----+-----+
| 3 | root          | localhost    | test | Sleep   | 426 |      |
| 4 | system user   |               | NULL | Connect | 5492 | Waiting for master to send event |
| 5 | system user   |               | NULL | Connect | 104 | System lock |
| 6 | root          | localhost    | test | Query   | 0 | starting |
| show processlist |
+-----+-----+-----+-----+-----+-----+-----+

```

对于这个状态官方有如下描述：

```
The thread has called MySQL_lock_tables() and the thread state has not been updated
since.
This is a very general state that can occur for many reasons.
For example, the thread is going to request or is waiting for an internal or external
system lock for the
table. This can occur when InnoDB waits for a table-level lock during execution of
LOCK TABLES.
If this state is being caused by requests for external locks and you are not using
multiple MySQLd
servers that are accessing the same MyISAM tables, you can disable external system
locks with the
--skip-external-locking option. However, external locking is disabled by default, so
it is likely
that this option will have no effect. For SHOW PROFILE, this state means the thread
is requesting the
lock (not waiting for it).
```

显然不能解决我的问题，一时间也是无语。而我今天在测试从库手动加行锁并且 sql thread 冲突的时候发现了这个状态，因此结合 gdb 调试做了如下分析，希望对大家有用，也作为后期我学习的一个笔记。

五、system lock 延迟的原因

这里先直接给出原因供大家直接参考，简单的说从库出现 system lock 应该视为正在干活，而不是名称看到的“lock”，这是由于 slave 端不存在语句（row 格式）的执行，都是 Event 的直接 apply，状态没有切换的机会，也可以认为是 slave 端状态划分不严谨，其实做一个 pstack 就能完全看出问题。下面是产生的必要条件：

- 由于大量的小事物，比如 UPDATE/DELETE table where 处理一行数据，这会出现只包含一行数据库的 DML event 的语句，如果 table 是一张大表，则会加剧这种可能。
- 这个表上没有主键或者唯一键，问题加剧。
- 由于类似 InnoDB lock 堵塞，也就是 slave 从库修改了数据同时和 sql_thread 也在修改同样的数据，问题加剧。

确实 I/O 扛不住了，可以尝试修改参数。

如果是大量的表没有主键或者唯一键可以考虑修改参数 slave_rows_search_algorithms 试试。关于 slave_rows_search_algorithms 在我的系列中有一章详细讨论，这里不做赘述。

六、system lock 延迟的问题分析

我们知道所有的 state 都是 MySQL 上层的一种状态，如果要发生状态的改变必须要调用

THD::enter_stage 来改变, 而 system lock 则是调用 mysql_lock_tables 进入的状态, 同时从库 SQL_THREAD 中还有另外一种状态重要的状态 reading event from the relay log。

这里是 rpl_slave.cc handle_slave_sql 函数中的很小的一部分主要用来证明我的分析。

```
/* Read queries from the IO/THREAD until this thread is killed */
while(!sql_slave_killed(thd,rli)) //大循环
{
    THD_STAGE_INFO(thd, stage_reading_event_from_the_relay_log); //这里进入 reading
    event from the relay log 状态
    if(exec_relay_log_event(thd,rli)) //这里会先调入 next_event 函数读取一条 event, 然后调用
        lock_tables 但是如果不是第一次调用 lock_tables 则不需要调用 MySQL_lock_tables
    //逻辑在 lock_tables 调用 mySQL_lock_tables 则会将状态置为 system lock, 然后进入 Innodb 层进
        行数据的查找和修改
}
```

这里还特地请教了阿里的印风兄验证了一下 mysql_lock_tables 是 myisam 实现表锁的函数 Innodb 会设置为共享锁。

这里我们先抛开 query event/map event 等。只考虑 DML event, 下面就是 system lock 出现的流程:

- 如果一个小事物只有一条 DML event 的场景下逻辑如下:

- >进入 reading event from the relay log 状态
- >读取一条 event(参考 next_event 函数)
- >进入 system lock 状态
- >Innodb 层进行查找和修改数据

- 如果是一个大事物则包含了多条 DML event 的场景逻辑如下:

- >进入 reading event from the relay log 状态
- >读取一条 event(参考 next_event 函数)
- >进入 system lock 状态
- >Innodb 层进行查找和修改数据
- >进入 reading event from the relay log 状态
- >读取一条 event(参考 next_event 函数)
- >Innodb 层进行查找和修改数据
- >进入 reading event from the relay log 状态
- >读取一条 event(参考 next_event 函数)
- >Innodb 层进行查找和修改数据
-直到本事物 event 执行完成

因此我们看到对于一个小事物我们的 sql_thread 会在加 system lock 的情况下进行对数据进行查找和修改。

因此得出了我的结论, 同时如果是 Innodb 层锁造成的 sqlthread 堵塞也会在持有 system lock 的状态下。

但是对于一个大事物则不一样，虽然出现同样的问题，但是其状态是 `reading event from the relay log`。

所以如果出现 `system lock` 一般就是考虑前文给出的结论。

七、分析中用到的断点

- `mysql_lock_tables` 本函数更改状态为 `system lock` gdb 打印 `:p tables[0]->s->table_name`
- `THD::enter_stage` 本函数改变状态 gdb 打印 `:p new_stage->m_name`
- `ha_innodb::index_read innodb` 查找数据接口 gdb 打印 `:p index->table_name`
- `ha_innodb::delete_row innodb` 删除数据接口
- `exec_relay_log_event` 获取 `event` 并且应用 gdb 打印 `:ev->get_type_code()`

3.26 什么是半一致性读？

作者：赵黎明

什么是半一致性读？

先看下官方的描述：

- 是一种用在 `Update` 语句中的读操作（一致性读）的优化，是在 `RC` 事务隔离级别下与一致性读的结合。
- 当 `Update` 语句的 `where` 条件中匹配到的记录已经上锁，会再次去 `InnoDB` 引擎层读取对应的行记录，判断是否真的需要上锁（第一次需要由 `InnoDB` 先返回一个最新的已提交版本）。
- 只在 `RC` 事务隔离级别下或者是设置了 `innodb_locks_unsafe_for_binlog=1` 的情况下才会发生。
- `innodb_locks_unsafe_for_binlog` 参数在 8.0 版本中已被去除（可见，这是一个可能会导致数据不一致的参数，官方也不建议使用了）。

测试案例

`InnoDB` 引擎的强大之处就在于它能完美地支持事务，而事务的一致性则是由事务隔离级别和并发事务锁来保证的。接下来，我们先通过 2 个测试案例来观察半一致性读会对事务产生哪些影响。

案例 1

`RC` 隔离级别，3 个 `Session` 执行事务语句