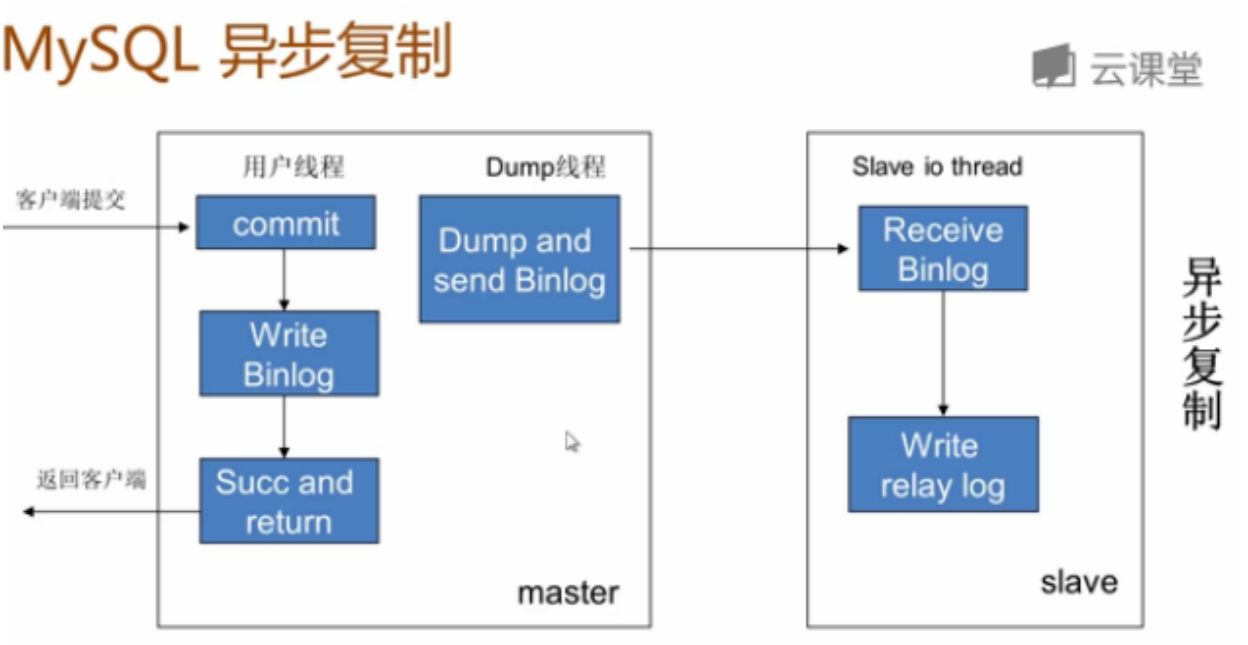
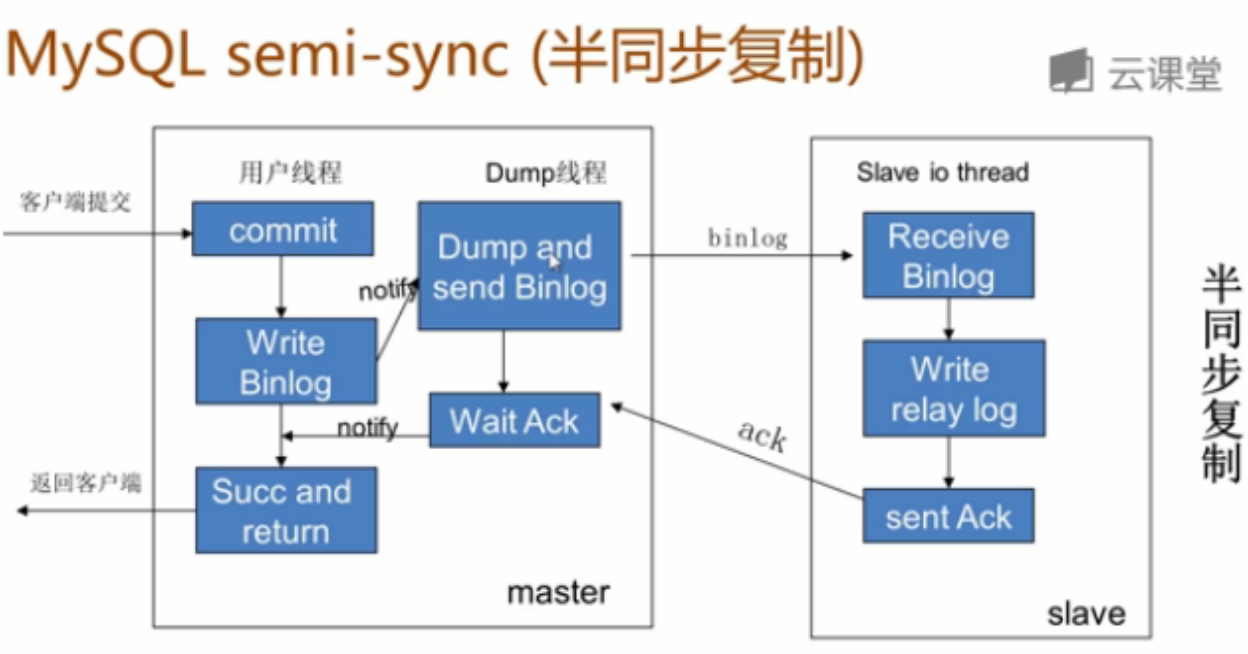
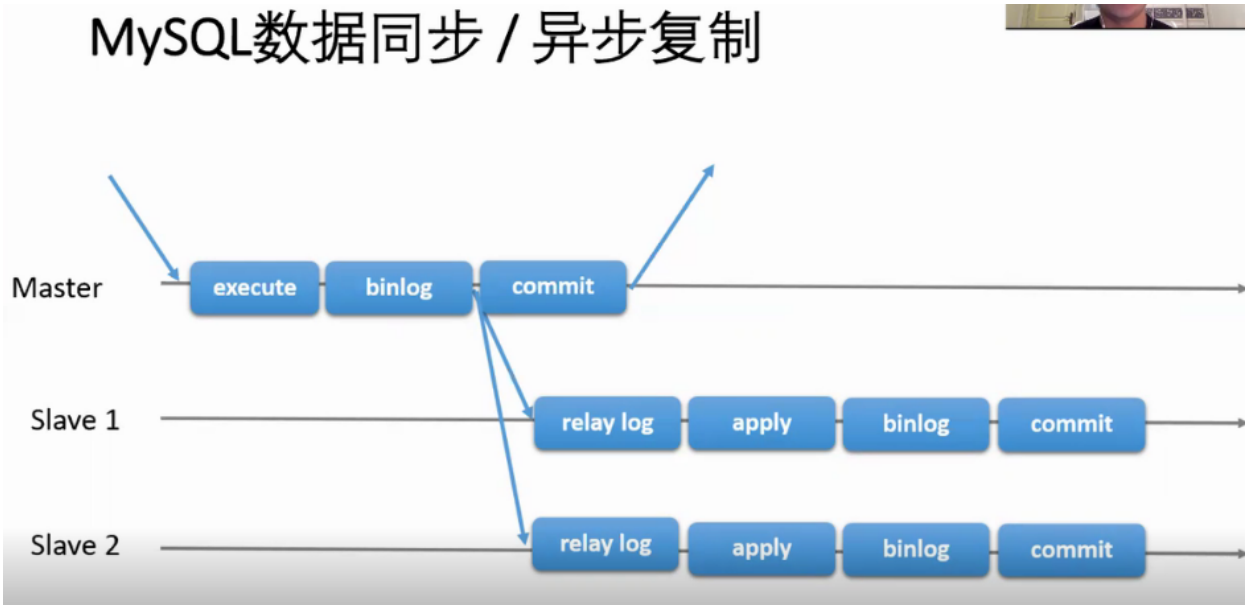
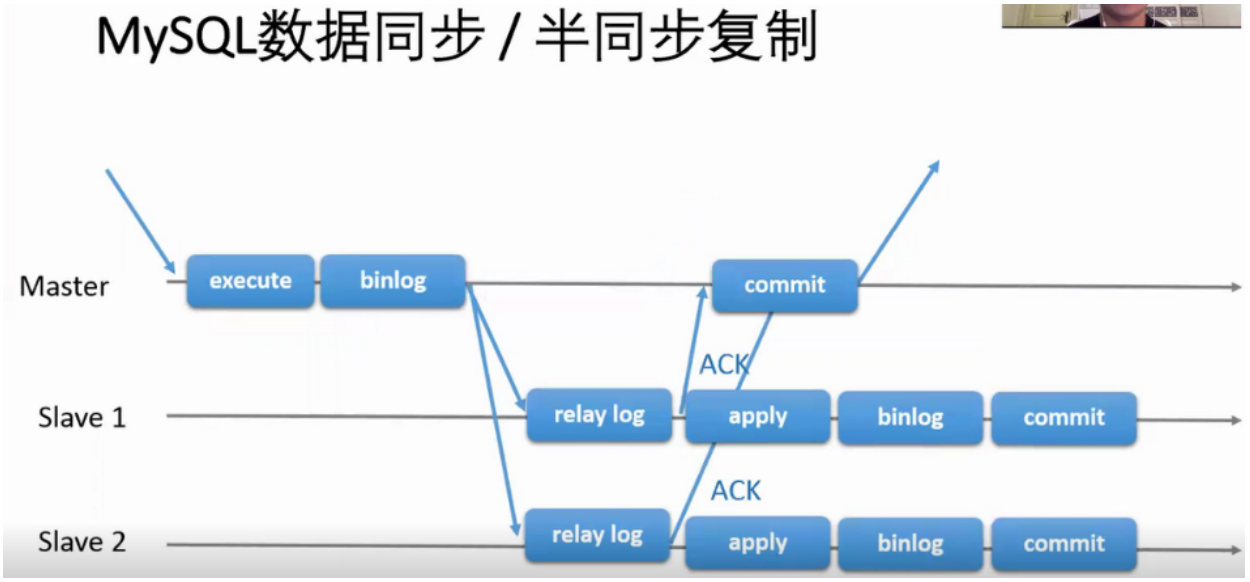
一、创建主从同步

1.1 基本原理









1.2 重点

1、利用mysqldump新建一个从库（GTID + row）

2、复制相关的线程

IO_thread 负责把主库数据，写到本地relay log

SQL_thread 负责把relay log中的记录 应用到从库

3、复制的几种方式： 异步复制， 半同步复制， 增强版同步复制。 和io_thread 相关

4、sql_thread 并发：

5.6 database级别

5.7 binlog group comit 级别

5.7.22 writeset

8.0 writeset

1.3 关键参数

```
########replication settings########
```

```
#server-id=1
master_info_repository = TABLE
log_bin = mysql-bin
binlog_row_image=full
binlog_format = row
expire_logs_days=7
slave_net_timeout=20
log_slave_updates
 sync_binlog = 1
 slave_checkpoint_group=512
 slave_checkpoint_period=300
 slave_preserve_commit_order=on
 slave_rows_search_algorithms=TABLE_SCAN,INDEX_SCAN
  ####启用GTID####
 gtid_mode = on #开启gtid模式
 enforce_gtid_consistency = on #强制gtid一致性，开启后对于特定create table不被支持
  ####从库 crash-safe####
 relay_log_info_repository = TABLE
 relay_log_recovery = 1
  ####半同步####
 plugin-load="rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
 loose-rpl_semi_sync_master_enabled=on
 loose-rpl_semi_sync_slave_enabled=on
 loose-rpl_semi_sync_master_timeout=100000
 loose-rpl_semi_sync_master_wait_for_slave_count=1
 loose-rpl_semi_sync_master_wait_point=AFTER_SYNC
  ####组提交###
 binlog_group_commit_sync_delay=100
 binlog_group_commit_sync_no_delay_count=10
 binlog_order_commits=off
  ####并行复制###
 transaction_write_set_extraction=XXHASH64
 binlog_transaction_dependency_tracking=writeset_session
 binlog_transaction_dependency_history_size=25000
 slave_parallel_type=LOGICAL_CLOCK
 slave_parallel_workers=4 #太多的线程会增加线程间同步的开销，建议4-8个slave线程
```

1.4 基于 GTID 的复制 环境搭建

1.4.1 要求

```
开启binlog
每个实例要配置server-id
启用GTID
创建复制用的账号
原始主库上数据备份
```

1.4.2 主库关键配置

```
server_id=2003306
log_bin=mysql_bin
binlog_format=row
gtid_mode=on
enforce_gtid_consistency=on
```

1.4.3 创建复制账号

```
create user 'repl'@'%' identified by '123456';
grant replication slave on *.* to 'repl'@'%';
```

1.4.4 从库关键配置

```
server_id=2013306
log_bin=mysql_bin
binlog_format=row
gtid_mode=on
enforce_gtid_consistency=on
```

1.5 开启过程(主、从库都是新初始化环境)

重置主库状态（已经运行的线上环境不要运行此命令）

```
reset master;
```

重置从库状态

```
reset master;
```

查看主库状态

```
show master status;
```

在从库配置复制

```
change master to master_host='192.168.123.201',\
master_port=3306,\
```

```
master_user='repl',\
master_password='123456',\
master_auto_position=1;
```

> 配置master_auto_position 等于1 ，表示，当从库启动复制时，将已经有的“Executed_Gtid_Set ”传递给主库（show master status 命令查看到的Executed_Gtid_Set）。主库会将从库传递过来的“Executed_Gtid_Set ”和自己的“Executed_Gtid_Set ”做差集。 从而计算出，从库中不存在的事务（需要被复制的事务），然后将这些事务传递给从库进行复制。

启动从库复制线程

```
start slave io_thread;
start slave sql_thread;
```

1.6 开启过程（主库已经存在数据）

备份主库数据，可以使用xtrabackup 或者其它备份工具，本例使用mysqldump工具

```
mysqldump --master-data=2 --single-transaction -A > db3306-`data +%Y%m%d`.sql
```

重置从库状态（从库中不能存在GTID）,在从库中执行

```
reset master;
```

在从库上恢复主库备份的数据

```
mysql -uroot -p < db3306.sql
```

查看主库状态

```
show master status;
```

在从库配置复制

```
change master to master_host='172.18.0.12',\
master_port=3306,\
master_user='repl',\
master_password='repl4slave',\
master_auto_position=1;
```

启动从库复制线程

```
start slave io_thread;
start slave sql_thread;
```

1.7 从库复制延迟时间

```
mysql >show slave status;
 Seconds_Behing_Master: 193 # 从库复制相较于主库的落后的时间
```

1.8 开启半同步

（1）安装插件，三台MySQL服务器都要执行：

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
mysql> INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

我们可以通过以下语句查看是否安装成功：

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS FROM INFORMATION_SCHEMA.PLUGINS
WHERE PLUGIN_NAME LIKE '%semi%';
+----------------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+----------------------+---------------+
| rpl_semi_sync_master | ACTIVE |
| rpl_semi_sync_slave | ACTIVE |
+----------------------+---------------+
2 rows in set (0.00 sec)
```

在三台MySQL的配置文件当中添加如下的参数：

```
rpl_semi_sync_master_enabled=1 #开启半同步复制 主库设置
rpl_semi_sync_slave_enabled=on; #打开半同步复制 从库设置
```

然后重启数据库，不安装semisync_master.so的话是不能识别这个参数的，所以说这个参数要等安装完以后在重启。

(2)重启完成后，默认就是半同步复制了，开始看一下半同步相关的参数：

```
mysql> show variables like 'rpl_se%';
+-------------------------------------------+------------+
| Variable_name | Value |
+-------------------------------------------+------------+
| rpl_semi_sync_master_enabled | ON |
| rpl_semi_sync_master_timeout | 10000 |
| rpl_semi_sync_master_trace_level | 32 |
| rpl_semi_sync_master_wait_for_slave_count | 1 |
| rpl_semi_sync_master_wait_no_slave | ON |
| rpl_semi_sync_master_wait_point | AFTER_SYNC |
| rpl_semi_sync_slave_enabled | ON | ##从库设置
| rpl_semi_sync_slave_trace_level | 32 |
+-------------------------------------------+------------+
8 rows in set (0.00 sec)
```

下面我们看一下这些参数具体是有什么含义：

rpl_semi_sync_master_enabled ： 主库是否打开半同步复制

rpl_semi_sync_master_timeout： 毫秒为单位，当主库等待从库ACK的实践超过这个值，就会自动退化为异步复制。rpl_semi_sync_master_no_times参数记录了服务器关闭半同步的次数。可以通过这个参数的增长查看是否发生过退化

rpl_semi_sync_master_trace_level： master的trace 级别，分为四个（1,16,32,64），分别记录不同的信息，32能够输出更详细的网络延迟等信息，也是默认值

rpl_semi_sync_master_wait_for_slave_count： 至少有N个slave接收到日志，一主多从的情况下只要有一个slave的ACK返回给了主库，就会进行commit

rpl_semi_sync_master_wait_no_slave：默认为ON，当半同步复制转换为异步复制后，如果从库的日志追赶上了主库，会自动转换为半同步复制，设置为OFF的话就不会再进行转换。

rpl_semi_sync_slave_enabled： 从库是否打开半同步复制功能

rpl_semi_sync_slave_trace_level： trace 级别

rpl_semi_sync_master_wait_point： 这是MySQL5.7新增的功能，可以设置两个值AFTER_SYNC 和AFTER_COMMIT，AFTER_COMMIT的模式下master将每个事务写入binlog ,传递到slave 刷新到磁盘(relay log)，同时主库提交事务。master等待slave反馈收到relay log，只有收到ACK后master才将commit OK结果反馈给客户端。AFTER_SYNC 情况下master 将每个事务写入binlog，传递到slave 刷新到磁盘(relay log)。master等待slave 反馈接收到relay log的ack之后，再提交事务并且返回commit OK结果给客户端。 即使主库crash，所有在主库上已经提交的事务都能保证已经同步到slave的relay log中。我们推荐使用默认AFTER_SYNC 的情况，这样可以提高性能，减少等待时间。

此外在MySQL5.7的半同步复制当中还移除了dump thread对binlog的互斥锁， 解决了在高并发环境下串行读取binlog的问题。

1.9 开启并行复制

1.9.1 主库配置

```
binlog_group_commit_sync_delay=100
binlog_group_commit_sync_no_delay_count=10
binlog_order_commits=off
transaction_write_set_extraction=XXHASH64
binlog_transaction_dependency_tracking=writeset_session
binlog_transaction_dependency_history_size=25000
```

以上配置开启master库组提交

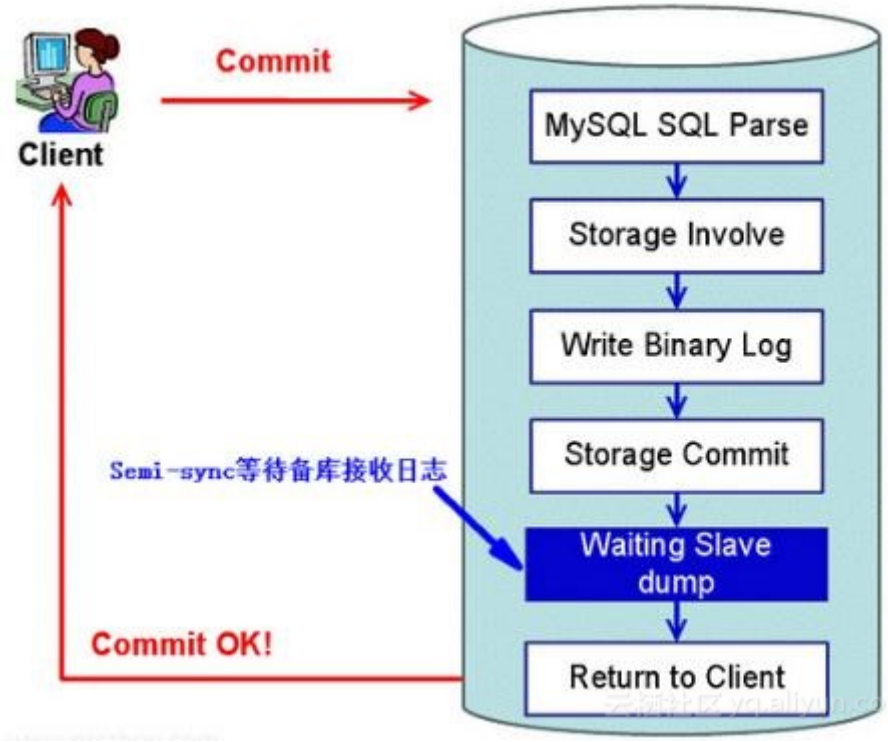
1.9.2 从库配置

```
transaction_write_set_extraction=XXHASH64
binlog_transaction_dependency_tracking=writeset_session
binlog_transaction_dependency_history_size=25000
######以上三个参数 需要5.7.22以上版本支持 属于mysql 8.0 的新特性######
slave_parallel_type=LOGICAL_CLOCK
slave_parallel_workers=8 #太多的线程会增加线程间同步的开销，建议4-8个slave线程
master_info_repository=TABLE
relay_log_info_repository=TABLE
relay_log_recovery=ON
```

slave_parallel_type有两个之，DATABASE和LOGICAL_CLOCK，DATABASE： 默认值，兼容5.6以schema维度的并行复制， LOGICAL_CLOCK： MySQL 5.7基于组提交的并行复制机制。

1.10 半同步和增强半同步的区别

半同步：



整个流程：

master write binlog -> slave sync binlog -> master commit -> salve ack -> master return result。

存在的问题

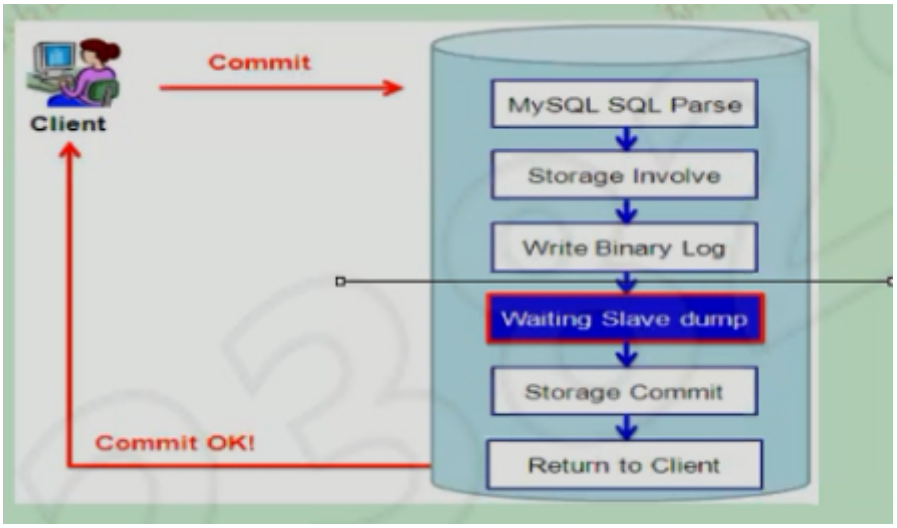
1. 会出现脏读的情况，也就是在事务提交之后，slave 确认之前，客户端A还没有获得结果返回，但是客户端B能够读取A提交的结果；

2. 可能会出现我们上文提到的数据丢失的问题（如果storage commit 提交之后，master 宕机）；

增强半同步

为了解决半同步的问题， 5.7推出了增强半同步。

rpl_semi_sync_master_wait_point=AFTER_SYNC， 5.7之前是 AFTER_COMMIT



整个流程：

master write binlog -> slave sync binlog -> salve ack -> master commit -> master return result。

这样就保证 master commit 的事务已经同步到 slave 了，防止数据丢失。

Storage Commit 之后，如果master宕机，因为 主从库都已经提交了数据，如果write binary log 和 Waiting slave dump 宕机，主库还没有提交数据，从库也没有同步数据，所有两边数据也是一致的。

但是，还存在另外一个问题。主库如果在write binary log 和 Waiting slave dump 之间宕机，当主库重启起来之后，做recovery时，会在binlog中找到宕机前没有提交的数据，进行提交。这样的数据不会同步到从库，导致从库丢失一条数据。

解决办法：

在腾讯云和阿里云中，当主库宕机之后，会直接抛弃主库，让其中一个从库升主。然后重新创建原来的主库

在mysql的 MGR 中， 当主库宕机并重启之后，会检查虽然已经记录在binlog中，但还没有提交的记录，是否已经同步到从库中，如果没有的话，主库会直接舍弃这个事务。

mysql中，事务提交的大概过程mysql中事务提交的大概过程.note

1.11 其它操作

1.11.1 查看主库的从库信息

```
root@repl1 09:19: [(none)]> show slave hosts;
+-----------+------+------+-----------+--------------------------------------+
| Server_id | Host | Port | Master_id | Slave_UUID |
+-----------+------+------+-----------+--------------------------------------+
| 1023306 | | 3306 | 1013306 | 00d9f849-131d-11e9-896b-0242ac120066 |
+-----------+------+------+-----------+--------------------------------------+
1 row in set (0.00 sec)
```

1.11.2 开启延时复制

```
mysql> stop slave;
mysql> change master to master_delay = N;
mysql> start slave;
```

1.11.3 查看从库应用线程的状态

```
root@repl2 13:03:24 [db1] select * from performance_schema.replication_applier_status_by_worker\G
*************************** 1. row ***************************
 CHANNEL_NAME:
 WORKER_ID: 1
 THREAD_ID: NULL
 SERVICE_STATE: OFF
LAST_SEEN_TRANSACTION: 74ba5f08-8de7-11e9-9593-0242ac12006f:24
 LAST_ERROR_NUMBER: 1062
```

```
 LAST_ERROR_MESSAGE: Worker 1 failed executing transaction '74ba5f08-8de7-11e9-9593-0242ac12006f:24' at master log mysql-bin.000004, end_log_pos 1763; Could not execute Write_rows event on table db1.repl_1; Duplicate entry '3' for key 'PRIMARY', Error_code: 1062; handler error HA_ERR_FOUND_DUPP_KEY; the event's master log mysql-bin.000004, end_log_pos 1763
 LAST_ERROR_TIMESTAMP: 2019-06-14 13:02:24
*************************** 2. row ***************************
 CHANNEL_NAME:
 WORKER_ID: 2
 THREAD_ID: NULL
 SERVICE_STATE: OFF
LAST_SEEN_TRANSACTION:
 LAST_ERROR_NUMBER: 0
 LAST_ERROR_MESSAGE:
 LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
*************************** 3. row ***************************
 CHANNEL_NAME:
 WORKER_ID: 3
 THREAD_ID: NULL
 SERVICE_STATE: OFF
LAST_SEEN_TRANSACTION:
 LAST_ERROR_NUMBER: 0
 LAST_ERROR_MESSAGE:
 LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
*************************** 4. row ***************************
 CHANNEL_NAME:
 WORKER_ID: 4
 THREAD_ID: NULL
 SERVICE_STATE: OFF
LAST_SEEN_TRANSACTION:
 LAST_ERROR_NUMBER: 0
 LAST_ERROR_MESSAGE:
 LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
4 rows in set (0.00 sec)
```

1.11.4 只复制部分数据库和表

```
CHANGE REPLICATION FILTER REPLICATE_DO_TABLE=(dbName.tbName);
```

1.11.5 启动sql_thread，指定只应用到指定的GTID

```
start slave sql_thread until SQL_BEFORE_GTIDS='xxxxx:108579';
```

这样，binlog应用到108578之前一个GTID时，就会停止

1.11.6 从库响应超时

当主库告知从库有新数据后，从库会返回 ACK 响应给主库。如果主库无法收到 ACK 响应会怎样

主库中有rpl_semi_sync_master_timeout 参数，默认为10000（单位毫秒），如果超过这个时间，半同步会退化为异步。但是考虑到数据的一致性，一般不允许退化。所以会把此值设置很大。另外，可以挂多个从库，避免超时的出现。

二、复制中的重要参数及优化

2.1 master 配置优化

建议主库配置：

```
biglog_format=row
binlog_row_image=full
gtid_mode = on #开启 GTID
enforce_gtid_consistency = on #开启GTID

binlog_group_commit_sync_delay=100
biglog_group_commit_sync_no_delay_count=10
binlog_order_commits=off

transaction_write_set_extraction=XXHASH64
binlog_transaction_dependency_tracking=writeset_session
binlog_transaction_dependency_history_size=25000
```

binlog_transaction_dependency_tracking为COMMIT_ORDER 时，binlog_transaction_dependency_history_size 和 transaction_write_set_extraction 不生效。

要想使用writeset 特性，需要设置transaction_write_set_extraction 以及binlog_transaction_dependency_tracking。建议将binlog_transaction_dependency_tracking

设置为writeset_session。

writeset 特性虽然是 用来提高从库应用relay log并行能力的。但是主库也需要设置这几个参数。

解释：

1：binlog_group_commit_sync_delay

全局动态变量，单位微妙，默认0，范围：0～1000000（1秒）。

表示binlog提交后等待延迟多少时间再同步到磁盘，默认0，不延迟。设置延迟可以让多个事务在用一时刻提交，提高binlog组提交的并发数和效率，提高slave的吞吐量。

当设置了 sync_binlog=0 或者 sync_binlog=1时，每个binlog 组提交同步前都会进行这个延迟。如果sync_binlog 的值为n（n大于1）时，每n个binlog组提交之后，会进行延迟。

设置binlog_group_commit_sync_delay可以增加主服务器的并发提交事务的数量。

如果从服务器开起了并行复制，也可以增加从库的并发执行量。要想从中受益，从服务器必须指定**slave_parallel_type=LOGICAL_CLOCK**，并且当设置了binlog_transaction_dependency_tracking=COMMIT_ORDER时效果更佳显著。

在调整binlog_group_commit_sync_delay的设置时，必须考虑主服务器的吞吐量和从服务器的吞吐量。

请注意，设置binlog_group_commit_sync_delay会增加服务器上事务的延迟，这可能会影响客户端应用程序。 此外，在高度并发的工作负载上，延迟可能会增加争用，从而降低吞吐量。 通常，设置延迟的好处超过了缺点，但应始终进行调整以确定最佳设置。

MySQL 5.7 学习：新增配置参数.note

2：binlog_group_commit_sync_no_delay_count

全局动态变量，单位个数，默认0，范围：0～1000000。

表示等待延迟提交的最大事务数，如果上面参数的时间没到，但事务数到了，则直接同步到磁盘。若binlog_group_commit_sync_delay没有开启，则该参数也不会开启。

关于binlog 组 提交的相关理论请看MySQL并发复制系列一：binlog组提交.note

MySQL 5.7新特性：并行复制原理（MTS）.note

从库配置

```
######以下几个参数 需要5.7.22以上版本支持 ######
transaction_write_set_extraction=XXHASH64
```

```
binlog_transaction_dependency_tracking=writeset_session
binlog_transaction_dependency_history_size=25000
slave-parallel-type=LOGICAL_CLOCK
slave-parallel-workers=8 #太多的线程会增加线程间同步的开销，建议4-8个slave线程
#########当从库总是落后主库时，可以考虑配置以下两个参数
innodb_flush_log_trx_commit=2
sync_binlog=0
```

2.2 json binglog 优化

```
binlog_row_image=image
binlog_row_value_options=PARTIAL_JSON
```

2.3 io_thread

```
slave_net_timeout=20|30 #和主库断开连接的超时时间，单位秒
```

io_trhead更多的优化： change master to相关参数

```
MASTER_CONNECT_RETRY=60
MASTER_CONNECT_COUNT=24*3600
MASTER_AUTO_POSITION=1
MASTER_DELAY=0
MASTER_Bind='' #指定复制使用的网卡名， 建议指定为内网网卡
```

2.4 sql_thread

```
log_slave_updates #从库应用主库binlog数据时，同时在本地记录binlog
slave_parallel_type=LOGICAL_CLOCK #基于组提交的并行复制方式
slave_parallel_workers=4|8 #并行复制线程数（sql_thread,应用relay log的线程），建议4 或 8 个

slave_checkpoint_group=512
slave_checkpoint_period=300
slave_preserve_commit_order=on
slave_rows_search_algorithms=TABLE_SCAN,INDEX_SCAN
```

2.5 复制过滤

在从库执行

```
CHANGE REPLICATION FILTER
 REPLICATE_DO_DB=(db_list)
 REPLICATE_IGNORE_DB=(db_list)
 REPLICATE_IGNORE_TABLE=(tbl_list)
 REPLICATE_WILD_DO_TABLE=(wild_tbl_list)
```

2.6 延迟复制

开启延迟复制

延迟复制只是sql_thread延迟，io_thread 并没有延迟。也就是说，io_thread实时将主库binlog拉去到了本地，放在了relay log中。只是sql_thread 延迟应用relay log。

保持和主库一个小时的延迟

```
stop slave sql_thread;
change master to master_delay=3600;
start slave sql_thread;
```

关闭延迟复制

```
stop slave sql_thread;
change master to master_delay=0;
start slave sql_thread;
```

应用场景

延迟复制的从库可以作为主库的灾备环境。如主库数据被误删除，可以使用延迟复制的从库上的数据做快速恢复。

具体方法：可以将延迟时间设置长一些。当发生误删除等操作时，

先手动 停止从库的复制 stop slave

然后通过start slave [io_thread | sql_thread] until 让从库执行到某个误操作前的点。然后用从库数据 恢复 到主库

三、常见复制故障

3.1 从库宕机

如果在从库复制数据过程中，宕机了。很可能造成relay log 损坏。无法继续进行复制。启动复制时报错：

```
relay log read failure
```

遇到这种情况，有如下几个解决版本：

开启了 crash-safe replication

Mysql 5.6以后，可以在从库配置以下两个参数,可以自动恢复这个问题

```
relay_log_info_repository=table
relay_log_recovery=1
```

开启了GTID

如果开启了GTID，直接在从库

```
stop slave;
reset slave;
start slave;
```

没有开启GTID

首先使用show slave status;查看已经复制到的位置

```
show slave status;
 Relay_Master_Log_File: mysql-bin.000008
 Exec_Master_Log_Pos: 75
```

然后再CHANGE MASTER TO

```
stop slave;
change master to
 master_log_file=relay_master_log_file,
 master_log_pos=exec_master_log_pos,
 ....;
```

3.2 复制中断

1062错误

从库上出现写入数据，把自增ID占用 （主键冲突）出现1062错误

错误出现的方法

- **master**

```
use zst;
create table zst_1(
 id int not null,
 uname varchar(32),
 primary key (id)
);
insert into zst_1(id, uname) values(1, 'wubx'),(2, 'mysql');

```

- **slave**

```
set sql_log_bin=0;
```

```
insert into zst_1(id, uname) values(3, 'python');
```

- **master**

```
---从库操作完之后，再执行下面语句---
insert into zst_1(id, uname) values(3, 'java');
```

- **show slave status\G 错误出现**

```
Could not execute Write_rows event on table zst.zst_1;
Duplicate entry '3' for key 'PRIMARY', Error_code: 1062;
handler error HA_ERR_FOUND_DUPP_KEY;
the event's master log mysql-bin.000001, end_log_pos 1390
```

解决办法：

- 解决思路： 删除从库中的重复数据，再重新开始执行复制

```
set sql_log_bin=0;
delete from zst.zst_1 where id=3;
set sql_log_bin=1;
start slave sql_thread;
```

1032错误

主库上更新或删除的记录，在从库上不存在

错误复现

- master

```
use zst;
create table zst_1(id int not null,uname vachar(32), primary key(id));
insert into zst_1(id, uname) values (1, 'wubx'),(2, 'mysql');
insert into zst_1(id, uname) values (3, 'python');
```

- slave

```
set sql_log_bin=0;
delete from zst_1 where id=3;
set sql_lob_bin=1;
```

- master

```
--- 等待slave 操作完之后，再执行下面的语句---
update zst_1 set uname='java' where id=3;
```

- show slave status 错误出现

```
Could not execute Update_rows event on table wubx.zst_1;
Cannot find record in 'zst_1',Error_code:1032;
handler error HA_ERR_KEY_NOT_FOUND;
the event's master log mysql-bin.000012, end_log_pos 2072
```

解决方法

- 思路

出现这个错误的原因是从库上不存在这条记录。但是单纯从日志中没办法看出是哪一条记录。

但是:

日志中记录了这条信息在binlog中的结束为止:

在日志中记录了出现问题这条记录所在的mysql binlog 文件（event's master log mysql-bin.000012），以及记录的结束为止（end_log_pos 2072）。

slave status 信息中可以找到这条记录的开始位置

再结合 show slave status 信息中的记录的已经执行到的位置（Exec_Master_Log_Pos: 1975）。

使用mysqlbinlog 工具 从 binlog 中抓取这条记录

然后就可以使用这些信息，在主库的binlog中 抓取到这条记录

```
mysqlbinlog -v --base64-output=decode-rows --start-positon=1765 --stop-position=2072 mysql-bin.000012
```

在从库上生成这条记录（只需要插入非空字段就可以）

插入记录时，只插入非空字段就可以。因为row格式的binlog 中， 会记录这条记录的全部字段值。从库在根据主库binlog复制这条记录时，会根据binlog中的记录，恢复全部字段的数据

```
set sql_log_bin=0;
insert into zst_1(id) values (3);
set sql_log_bin=1;
```

从库启动sql_thread 线程

```
start slave sql_thread;
```

删除记录出现1302错误

以上是以update更新记录作为例子的，如果主库删除了一条从库不存在的记录时。解决方法有一下几种:

1、和update 采用一样的方式，人工补齐数据，再重新开始复制

2、在从库跳过这个事务，又分为两种情况:

如果没有开启GTID:

```
stop slave sql_thread;
set global skip_sql_slave_conter=1;
start slave sql_thread;
```

如果开启了GTID，需要在从库模拟空事务

```
stop slave sql_thread;
set gtid_next='master_server_uuid:gtid'; #出问题的这个事务的GTID
begin;commit; #人为的空事务，因为set gtid_next 中明确指定了下一个gtid，所以虽然这个事务是在从库上产生的，
```

```
 #但是更新的却是 gtid_next参数中指定的那个GITD（也就是主库的GTID）
select @@gtid_next # 查看下一个GTID
set gtid_next='automatic'; #将gtid改回 为 自动
start slave sql_thread;
```

1236错误

从库想要获取的GTID，已经不在主库的binlog中了

复现

- slave

```
stop slave;
```

- master

```

use zst;
create table zst_1(
 id int not null,
 uname varchar(32),
 primary key (id)
);
insert into zst_1(id, uname) values(1, 'wubx);
flush logs;
 insert into zst_1(id, uname) values (2, 'mysql');
 flush logs;
 purge master logs to 'mysql-bin.000002';
```

- slave

```
start slave
```

出现错误

这时候，slave会向master获取mysql-bin.000001 和 mysql-bin.000002中的GTID 数据。但是因为主库上已经没有了这两个binlog。从库就会报错 1236

解决方法

1、重建从库，从新从主库复制数据

2、忽略掉主库已经不存在的GTID， 从库只应用主库中还存在的binlog中的事务（不建议这么做，如果从库可以下线的话，建议采用从建从库的方式）

两种方式

1、如果相差的事务不是很多

```
#需要暂停从库的服务
set gtid_next='xxxx:n';
```

```
begin;commit;
set gtid_next='xxxxxx:n+1';
begin;commit;
#然后在使用 pt-table-checksum 和 pt-table-sync工具对数据进行校验.
#花费时间较多
```

2、如果相差的事务稍微多一点

```
#需要暂停从库的服务
stop slave;
reset master;
set global gtid_purge='xxxxxxxx:1-28'; #此值可以查询主库的 gtid_purge 参数获取
start slave
#然后在使用 pt-table-checksum 和 pt-table-sync工具对数据进行校验。
#花费时间较多
```

3.3 复制延迟排查思路

```
mysql >show slave status;
 Seconds_Behing_Master: 193 # 从库复制相较于主库的落后的时间
```

排查思路

1、查看sql_thread 在执行什么；

在show slave stauts返回结果中，

Relay_Masster_Log_File 和 Exec_Master_Log_Pos 的结果表示从库最后应用的事务的位置。比如Relay_Masster_Log_File 为 mysql-bin.000013，Exec_Master_Log_Pos 为 3834

在主库上查看这个位置之后的事务，就是从库当前在执行的事务和sql。

使用如下命令解析主库binlog：

```
mysqlbinlog -v \
 --base64-output=decode-rows \
 --start-position=3834 mysql-bin.000013 > 13.sql
```

或者在主库执行

```
show binlog events in 'mysql-bin.000013' from 3834 limit 10;
```

可能的情况

- 主库上执行了大事务，并且 这张表上没有索引。

解决办法：

```
停掉从库
在从库上为这张表建索引
```

```
重启打开复制
找时间为主库上的表建索引
```

另外，还可以检查主库是否开启了binlog group commit 特性, 从库是否开启了writeset 特性

如果还是很慢，可以临时修改从库的以下参数，让从库跑的更快一点

```
#########当从库总是落后主库时，可以考虑配置以下两个参数
innodb_flush_log_trx_commit=2
sync_binlog=0
```

其它解决办法：

```
如果对业务非常了解，也知道了造成当前延迟的具体语句。
也可以先停掉复制，然后在从库上跳过慢的GTID，然后手动执行被跳过的事务。最后再从新开始复制

另外一个办法。因为binlog format 格式为 row时，批量删除和更新操作会按照表中数据，
一条一条的记录在binlog中（
 例如，主库中执行的 delete from tb1,记录在binlog中就会变成
 delete from tb1 where @1=2 @2='abc2';
 delete from tb1 where @1=3 @2='bcd';
）。
也就是主库上的批量操作，在从库应用时会变成一条一条执行。针对这种情况，
在主库批量操作前，可以临时修改format的格式为statement格式。
```

如何避免复制延迟

主库是开启binlog group commit 特性, 从库是开启了writeset 特性

如果还是很慢，可以临时修改从库的以下参数，让从库跑的更快一点

```
#########当从库总是落后主库时，可以考虑配置以下两个参数
innodb_flush_log_trx_commit=2
sync_binlog=0
```