MySQL5.6 同步复制新特性详解

——— 贺春旸

继 5.5 半同步复制后，5.6 又对其进行了优化与改进，其中有两个地方较为重要：

1、对运维人员来说应该是一件大喜的事情，在主从切换后，在传统的方式里，你需要找到 binlog 和 POS 点，然后 change master to 指向，而不是很有经验的运维，往往会将其找错，造成主从同步复制报错，在 mysql5.6 里，你无须再知道 binlog 和 POS 点，你只需要知道 master 的 IP、端口，账号密码即可，因为同步复制是自动的，mysql 通过内部机制 GTID 自动找点同步。

2、多线程复制基于库。之前的版本，同步复制是单线程的，队列的，只能一个一个执行，在 5.6 里，可以做到多个库之间的多线程复制，例如 yourDB 库里，存放着用户表，商品表，价格表，订单表，那么将每个业务表单独放在一个库里，这时就可以做到多线程复制，但一个库里的表，多线程复制是无效的。

==========

新的名词解释：

server_uuid：服务器身份 ID。在第一次启动 Mysql 时，会自动生成一个 server_uuid 并写入到数据目录下 auto.cnf 文件里，官方不建议修改。

Important

The `auto.cnf` file is automatically generated; you should not attempt to write or modify this file.

Also beginning with MySQL 5.6, when using MySQL replication, masters and slaves know one another's UUIDs. The value of a slave's UUID can be seen in the output of `SHOW SLAVE HOSTS`. Once `START SLAVE` has been executed (but not before), the value of the master's UUID is available on the slave in the output of `SHOW SLAVE STATUS`.

并且 server_uuid 跟 GTID 有密切联系。

```
[root@mysql5_6 data]# pwd
/usr/local/mysql/data
[root@mysql5_6 data]# cat auto.cnf
[auto]
server-uuid=b0869d03-d4a9-11e1-a2ee-000c290a6b8f
```

GTID：全局事务标识符。当开始这个功能时，每次事务提交都会在 binlog 里生成一个唯一的标示符，它由 UUID 和事务 ID 组成。首次提交的事务 ID 为 1，第二次为 2，第三次为 3，依次类推。

查看 binlog，会看到如下：

```
# at 276
#120808  3:31:57 server id 25  end_log_pos 320  GTID [commit=yes]
SET @@SESSION.GTID_NEXT= 'B0869D03-D4A9-11E1-A2EE-000C290A6B8F:1'/*!*/;
```

```
# at 496
#120808  3:39:36 server id 25  end_log_pos 540  GTID [commit=yes]
SET @@SESSION.GTID_NEXT= 'B0869D03-D4A9-11E1-A2EE-000C290A6B8F:2'/*!*/;
```

```
# at 710
#120808  5:19:07 server id 25  end_log_pos 754  GTID [commit=yes]
SET @@SESSION.GTID_NEXT= 'B0869D03-D4A9-11E1-A2EE-000C290A6B8F:3'/*!*/;
```

```
# at 912
#120808  5:19:07 server id 25  end_log_pos 956  GTID [commit=yes]
SET @@SESSION.GTID_NEXT= 'B0869D03-D4A9-11E1-A2EE-000C290A6B8F:4'/*!*/;
```

开启 GTID 时，slave 在做同步复制时，无须找到 binlog 日志和 POS 点，直接 change master to master_auto_position=1 即可， 自动找点同步。

ine points for starting, stopping, or resuming the flow of data between master and slave. This means that, when you are using GTIDs for replication, you do not need (or want) to include `MASTER_LOG_FILE` or `MASTER_LOG_POS` options in the `CHANGE MASTER TO` statement used to direct a slave to replicate from a given master; in place of these options, it necessary only to use the `MASTER_AUTO_POSITION=1` option introduced in MySQL 5.6.5. For the exact steps needed to configure and start masters and slaves us-

==========

GTID 的工作流程是这样的：

1.在 master 上一个事务提交，并写入 binlog 里。

2.binlog 日志发送到 slave，slave 接收完并写入 relay log 中继日志里，slave 读取到这个 GTID，并设置 gtid_next 的值,例如：
SET@@SESSION.GTID_NEXT='B0869D03-D4A9-11E1-A2EE-000C290A6B8F:3';
然后告诉 slave 接下来的事务必须使用 GTID 并写入到它自己的 binlog 里。

3.slave 检查并确认这个 GTID 没有被使用并写入到它自己的 binlog 里。如果没有被使用，那么开始执行这个事务并写入到它自己的 binlog 里。

4.由于 gtid_next 的值不是空的，slave 不会尝试去生成一个新的 gtid，而是通过主从同步来获取 GTID。

==========

如何设置 GTID 方式的主从同步？

答：需要同时在 master 和 slave 上，在 my.cnf 文件上加入如下：

```
log-bin = mysql-bin
binlog_format = row
log_slave_updates
gtid-mode = ON
disable-gtid-unsafe-statements = ON
```

在 master 上导出：
mysqldump -uroot -p123456 -q --single-transaction -R -E --triggers
--default-character-set=utf8 -B yourDB > ./yourDB.sql

在 slave 上导入：
mysql -uroot -p123456 < ./yourDB.sql

然后做指向即可。
mysql> CHANGE MASTER TO
> MASTER_HOST = master-host,
> MASTER_PORT = master-port,
> MASTER_USER = repl-user,
> MASTER_PASSWORD = repl-password,
> MASTER_AUTO_POSITION = 1;

注：如果你使用了 GTID，那么就不能再使用传统 binlog 和 POS 方式
CHANGE MASTER TO
MASTER_HOST='master2.mycompany.com',
MASTER_USER='replication',
MASTER_PASSWORD='bigs3cret',
MASTER_PORT=3306,
MASTER_LOG_FILE='master2-bin.001',
MASTER_LOG_POS=4,
MASTER_CONNECT_RETRY=10;

否则会报错：

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000006',MASTER_LOG_POS=227;
ERROR 1775 (HY000): Parameters MASTER_LOG_FILE, MASTER_LOG_POS, RELAY_LOG_FILE and RELAY_LOG_POS cannot be set when MASTER_AUT
O_POSITION is active.
mysql>
```

==========

GTID 的局限性：

1.GTID 同步复制是基于事务。所以 Myisam 表不支持，这可能导致多个 GTID 分配给同一个事务。

2.CREATE TABLE ... SELECT 语句不支持。因为该语句会被拆分成 create table 和 insert 两个事务，并且这个两个事务被分配了同一个 GTID，这会导致 insert 被备库忽略掉。

```
mysql> create table t2 select * from t1;
ERROR 1785 (HY000): CREATE TABLE ... SELECT is forbidden when DISABLE_GTID_UNSAFE_STATEMENTS = 1.
mysql>
```

如果把 disable_gtid_unsafe_statements 参数关闭，启动 mysql 时会报错，也就是说开启 GTID,disable_gtid_unsafe_statements 参数必须开启。

```
120813  0:01:58 [ERROR] --gtid-mode=ON or UPGRADE_STEP_1 requires --disable-gtid-unsafe-statements
120813  0:01:58 [ERROR] Aborting
```

3.不支持 CREATE TEMPORARY TABLE、DROP TEMPORARY TABLE 临时表操作。

```
mysql> create temporary table t2(id int);
ERROR 1786 (HY000): When DISABLE_GTID_UNSAFE_STATEMENTS = 1, the statements CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE ca
n be executed in a non-transactional context only, and require that AUTOCOMMIT = 1.
mysql>
```

==========

设置 slave_parallel_workers 参数，开启基于库的多线程复制。默认是 0，不开启，最大并发数为 1024.

Multi-Threaded Slaves

Replication performance is improved by using multiple execution threads to apply replication events to the slave(s).

The multi-threaded slave splits processing between worker threads based on schema, allowing updates to be applied in parallel, rather than sequentially. This delivers benefits to those workloads that isolate application data using databases - e.g. multi-tenant systems

set global slave_parallel_workers = 4;
当设置为 4 个线程时，show processlist，你会发现：

```
mysql> show processlist;
+----+-------------+-----------+------+---------+------+----------------------------------------
-------+------------------+
| Id | User        | Host      | db   | Command | Time | State
      | Info             |
+----+-------------+-----------+------+---------+------+----------------------------------------
-------+------------------+
|  1 | system user |           | NULL | Connect |   25 | Waiting for master to send event
      | NULL             |
|  2 | system user |           | NULL | Connect |   23 | Slave has read all relay log; waiting fo
ate it | NULL             |
|  3 | system user |           | NULL | Connect |   25 | Waiting for an event from Coordinator
      | NULL             |
|  4 | system user |           | NULL | Connect |   25 | Waiting for an event from Coordinator
      | NULL             |
|  5 | system user |           | NULL | Connect |   25 | Waiting for an event from Coordinator
      | NULL             |
|  6 | system user |           | NULL | Connect |   25 | Waiting for an event from Coordinator
      | NULL             |
|  7 | root        | localhost | NULL | Query   |    0 | init
      | show processlist |
+----+-------------+-----------+------+---------+------+----------------------------------------
-------+------------------+
7 rows in set (0.01 sec)
```

有 4 个 Waiting for an event from Coordinator 线程

分别开两个终端，用 sysbench 分别对两个库进行压力测试，

sysbench --test=oltp --mysql-table-engine=innodb --oltp-table-size=100000 --max-requests=1000 --num-threads=16 --mysql-host=localhost --mysql-port=3306 --mysql-user=root --mysql-db=test --mysql-socket=/tmp/mysql.sock run

sysbench --test=oltp --mysql-table-engine=innodb --oltp-table-size=100000 --max-requests=1000 --num-threads=16 --mysql-host=localhost --mysql-port=3306 --mysql-user=root --mysql-db=test1 --mysql-socket=/tmp/mysql.sock run

然后 select * from mysql.slave_worker_info\G; 你会发现：

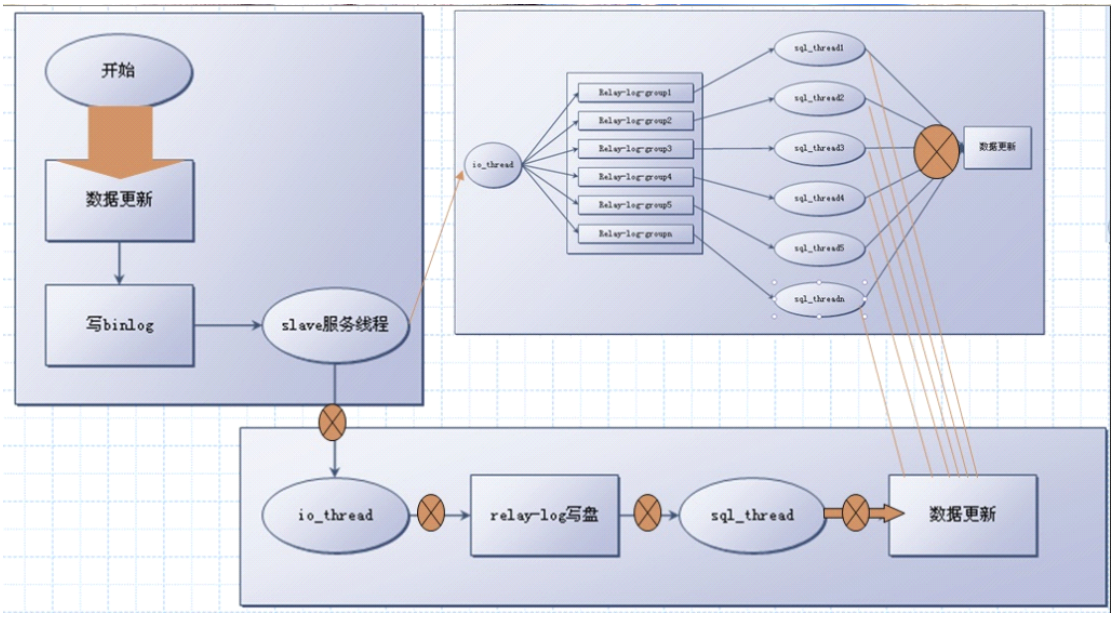
```
mysql> select * from mysql.slave_worker_info\G;
*************************** 1. row ***************************
                  Master_id: 165
                  Worker_id: 0
              Relay_log_name:
               Relay_log_pos: 0
             Master_log_name:
              Master_log_pos: 0
  Checkpoint_relay_log_name:
   Checkpoint_relay_log_pos: 0
Checkpoint_master_log_name:
  Checkpoint_master_log_pos: 0
           Checkpoint_seqno: 0
       Checkpoint_group_size: 64
     Checkpoint_group_bitmap:
```

*************************** 2. row ***************************
                Master_id: 165
                Worker_id: 1
              Relay_log_name:
               Relay_log_pos: 0
             Master_log_name:
              Master_log_pos: 0
  Checkpoint_relay_log_name:
   Checkpoint_relay_log_pos: 0
Checkpoint_master_log_name:
  Checkpoint_master_log_pos: 0
           Checkpoint_seqno: 0
       Checkpoint_group_size: 64
    Checkpoint_group_bitmap:
*************************** 3. row ***************************
                Master_id: 165
                Worker_id: 2
              Relay_log_name: ./mysql5_6-relay-bin.000009
               Relay_log_pos: 2091034
             Master_log_name: mysql-bin.000003
              Master_log_pos: 2090832
  Checkpoint_relay_log_name: ./mysql5_6-relay-bin.000009
   Checkpoint_relay_log_pos: 2082941
Checkpoint_master_log_name: mysql-bin.000003
  Checkpoint_master_log_pos: 2082739
           Checkpoint_seqno: 5
       Checkpoint_group_size: 64
    Checkpoint_group_bitmap: 0
*************************** 4. row ***************************
                Master_id: 165
                Worker_id: 3
              Relay_log_name: ./mysql5_6-relay-bin.000009
               Relay_log_pos: 2954634
             Master_log_name: mysql-bin.000003
              Master_log_pos: 2954432
  Checkpoint_relay_log_name: ./mysql5_6-relay-bin.000009
   Checkpoint_relay_log_pos: 2951772
Checkpoint_master_log_name: mysql-bin.000003
  Checkpoint_master_log_pos: 2951570
           Checkpoint_seqno: 1
       Checkpoint_group_size: 64
    Checkpoint_group_bitmap:
4 rows in set (0.01 sec)

有两个 Worker_id 的数值在不断的变化，那么多线程复制就开始起作用了。

注：如果一个库有 N 个请求，那么不会使用多线程复制，但是两个库有 N 个请求，那么会使用多线程复制。尽可能的把一个库中的表，按照业务逻辑拆分成多个库保存，这样在写操作时，slave 上就会开启多线程复制，减少了同步延时。2 个库 slave 就有 2 个 IO/SQL 线程，3 个库 slave 就有 2 个 IO/SQL 线程，依次类推。

如下图所示：



此外，
relay_log_info_repository=TABLE
master_info_repository=TABLE

会将 master.info 和 relay.info 保存在表中，默认是 Myisam 引擎，官方建议用
alter table　slave_master_info engine=innodb;
alter table　slave_relay_log_info engine=innodb;
alter table　slave_worker_info engine=innodb;
改为 Innodb 引擎，防止表损坏后自行修复。

- **Replication**: To provide a crash-safe slave, it was previously necessary to change the storage engine for the `slave_master_info`, `slave_relay_log_info`, and `slave_worker_info` tables from `MyISAM` to `InnoDB` manually, by issuing `ALTER TABLE`. To simplify the setup of replication using these slave log tables, they are now created using the `InnoDB` storage engine. (Bug #13538891)