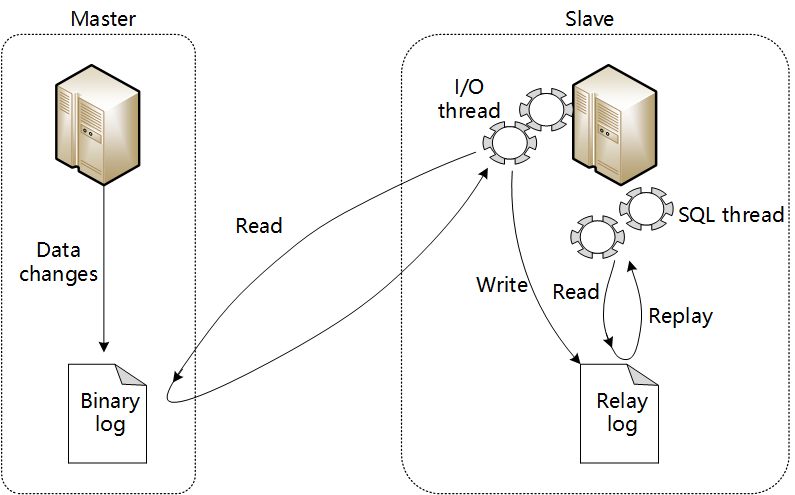
**复制的作⽤**

1. 横向扩展读能⼒。 读写分离

2. 很多关键的⽇常操作可放到从库进⾏，如备份、数据分析等。

3. 数据灾备。 高可用，一主两从，一主一从

4. 有丰富的⽣态圈。

**复制原理图**

**复制的流程**

1. 从库开启主从复制（start slave），会创建⼀个 IO thread 来连接主库。

2. 主库接受到连接请求后，会创建⼀个 Binlog Dump thread 来响应它。这个线程会读取 binlog ⾥⾯的⼆进制 ⽇志事件，发送给 Slave IO thread。Slave IO thread 接收到事件后，会将其写⼊到本地的 Relay log 中。

3. 从库的 SQL thread 会读取Relay log中的⼆进制⽇志事件，然后重放。

搭建

1. 相关配置
   1. Server-id 主从库 server-id不能一样
   2. log\_bin 二进制文件路径
   3. binlog\_format 二进制文件格式，一般为ROW
2. 复制用户与权限
   1. 用户名：username@’’
   2. 权限 replicantion slave
3. 从库设置 change master to
   1. Master\_host=’ip’
   2. Master\_user=’username’
   3. Master\_password=’password’
   4. Get\_master\_public\_key=1 如果使用的是caching\_sha2\_password就需要设置此参数
   5. MASTER\_CONNECT\_RETRY=10;
4. 基于位置复制
   1. Master\_log\_file=’主库二进制日志文件名’
   2. Master\_log\_pos=’主库二进制日志文件起始位置’，主库上使用show master status和show binlog events in ’binlog文件名’
5. 从库操作命令
   1. Show slave status\G
      1. 关键信息
         1. Slave\_IO\_Running
         2. Slave\_SQL\_Running
   2. select \* from mysql.slave\_master\_info\G
   3. select \* from mysql.slave\_relay\_log\_info\G
   4. Start slave
   5. Stop slave
   6. Reset slave [all]

**GTID**

GTID（Global transaction identifier，全局事务ID，会为每个事务分配⼀个唯⼀的事务ID，事务ID的格式如下:

source\_id:transaction\_id

source\_id：事务是在哪个实例上产⽣的，通常⽤实例的server\_uuid来表示。 transaction\_id：事务的序列号，按照事务提交的顺序从1开始分配。

这样，可保证GTID在整个复制拓扑中都是全局唯⼀的。

GTID的优点

GTID（Global Transaction ID）的推出旨在简化数据库复制和故障恢复过程。其主要⽬的是提供全局唯⼀的标识 符，⽤于标识数据库中的每个事务。

GTID的优点包括：

1. 全局唯⼀标识符： GTID为每个事务提供⼀个全局唯⼀的标识符，避免了在复制拓扑中使⽤⼆进制⽇志⽂件和 位置的复杂性。

2. 简化配置： 使⽤GTID可以更轻松地配置主从复制，⽆需⼿动跟踪⼆进制⽇志⽂件和位置。

3. 故障恢复： GTID简化了故障恢复过程，使得在主从之间重新建⽴复制关系更为容易。

4. 避免重复应⽤： GTID可以帮助防⽌在主从之间重复应⽤相同的事务，提⾼数据⼀致性。

总体⽽⾔，MySQL GTID的推出旨在提⾼复制的可靠性、可维护性和容错性，使得数据库管理更加简便。

配置：

1. gtid\_mode=ON
2. enforce\_gtid\_consistency=ON
3. log\_slave\_update=ON
4. log\_replication=ON //8.0使用

从库设置：

master\_auto\_position=1

source\_auto\_position=1 //8.0使用

CHANGE REPLICATION SOURCE TO

SOURCE\_HOST = '主库ip',

SOURCE\_USER = 'username',

SOURCE\_PASSWORD = 'password',

SOURCE\_PORT = 3306,

SOURCE\_AUTO\_POSITION = 1,

GET\_SOURCE\_PUBLIC\_KEY = 1;

注意：使用change master to 或 change replication source to 都是可以的，但8.0后使用change master to会提示警告并且change master to也会在今后的版本弃用

GITD的启用，如果不是在新环境中，最好不要直接启用，需按 off->off\_permissive->on\_permissive->on的顺序逐步开启，如需关闭则按相反顺序操作；

enforce\_gtid\_consistency则是warn->on 关闭则按相反顺序操作;

先设置enforce\_gtid\_consistency 再设置gtid\_mode

当gtid\_mode要设置为ON时，需要先检查正在进⾏的匿名事务的数量

SHOW STATUS LIKE 'ONGOING\_ANONYMOUS\_TRANSACTION\_COUNT';

若为0，则进行下一步在从库上执行此函数

select MASTER\_POS\_WAIT('主库二进制日志文件名',起始位置);

该函数会一直阻塞直到位置点的事务在从库上应用

GTID相关参数 show variables like ‘%gtid%’  
 gtid\_mode：是否开启GTID复制。在MySQL 5.7.6之前，如果要开启或关闭GTID 复制，必须重启实例。从 MySQL 5.7.6开始，可在线开启 GTID 复制。

gtid\_mode 可设置以下选项：

OFF：所有新的事务和复制中的事务都必须是匿名事务（⾮ GTID 事务，每个事务前⾯有⼀个 ANONYMOUS\_GTID\_LOG\_EVENT）。

OFF\_PERMISSIVE：新的事务是匿名事务，复制中的事务既可以是匿名事务，也可以是GTID事务.

ON\_PERMISSIVE：新的事务是GTID事务，复制中的事务既可以是匿名事务，也可以是GTID事务。

ON：所有新的事务和复制中的事务都必须是GTID事务。

对于该参数的修改，只能逐步修改，不能直接修改。如将OFF调整为ON，只能按照 OFF -> OFF\_PERMISSIVE -> ON\_PERMISSIVE -> ON的顺序进⾏修改。

enforce\_gtid\_consistency：

⽤来检测否有违反GTID限制的 SQL，其有如下取值：

OFF：关闭检测。

ON：开启检测。如果操作违反了上述限制，会直接报错。

WARN：开启检测。如果操作违反了上述限制，不会报错，只是会在错误⽇志中打 印Warning。

注意，如果要将gtid\_mode设置为ON，则enforce\_gtid\_consistency必须为ON。

gtid\_executed：MySQL已经执⾏过的GTID集。

gtid\_purged：MySQL已经清除的GTID集，通常是被删除的binlog中的GTID集。

gtid\_next：⽤来设置下⼀个GTID值，只能在会话级别设置，有如下取值：

AUTOMATIC：⾃动获取下⼀个GTID值，默认值。

ANONYMOUS：匿名事务。

UUID:NUMBER：显式设置GTID值。常⽤于主从复制环境，从库跳过指定事务。

gtid\_executed\_compression\_period：MySQL会定期对 mysql.gtid\_executed 表进⾏压缩，压缩的频率由 gtid\_executed\_compression\_period 参数控制。在MySQL 8.0.23 之前，该参数的默认值为1000，即每插⼊ 1000个事务，会对表进⾏⼀次压缩。从MySQL 8.0.23开始，该参数的默认值为0，这个时候会根据需要进⾏ 压缩。压缩后，连续的GTID会以区间的形式表示。

**GTID复制的限制**

GTID复制存在以下限制：

1. 不⽀持 CREATE TABLE ... SELECT ... 语句。从MySQL 8.0.21开始，该限制被取消，CREATE TABLE ... SELECT ... 会作为⼀个原⼦DDL被处理。

2. 不允许在事务、存储过程、函数、触发器中执⾏ CREATE TEMPORARY TABLE 或 DROP TEMPORARY TABLE 操作。从 MySQL 8.0.13 开始，如果 binlog 格式为 ROW 或 MIXED，该限制被取消。

3. 不能将InnoDB表和⾮InnoDB表放到⼀个事务内操作。

设置 gtid\_purged 的注意事项

在 MySQL 8.0 之前，设置gtid\_purged的前提是gtid\_executed必须为空，⽽要gtid\_executed为空，只能执⾏ reset master操作。

在 MySQL 8.0 中，⽆需 gtid\_executed 为空，但需满⾜以下条件：

1. 设置的 gtid\_purged 不能包含 gtid\_executed 中任何尚未清除的 GTID 集。

2. 设置的 gtid\_purged 必须是当前 gtid\_purged 的超集。

**半同步复制**

**异步复制（Asynchronous replication）**

MySQL 默认的复制即是异步的，主库在执⾏完客户端提交的事务后会⽴即将结果返给给客户端，并不关⼼从库是 否已经接收并处理，这样就会有⼀个问题，主如果crash掉了，此时主上已经提交的事务可能并没有传到从上，如 果此时，强⾏将从提升为主，可能导致新主上的数据不完整。

**全同步复制（Fully synchronous replication）**

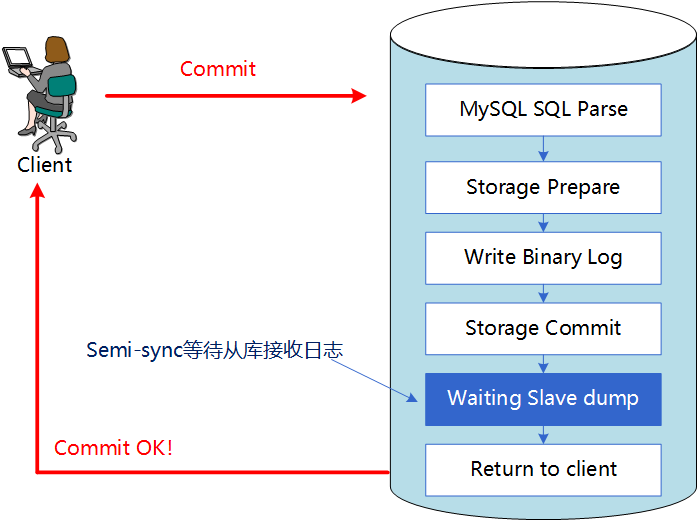
指当主库执⾏完⼀个事务，所有的从库都执⾏了该事务才返回给客户端。因为需要等待所有从库执⾏完该事务才能 返回，所以全同步复制的性能必然会收到严重的影响。

**半同步复制（Semisynchronous replication）**

介于异步复制和全同步复制之间，主库在执⾏完客户端提交的事务后不是⽴刻返回给客户端，⽽是等待⾄少⼀个从 库接收到并写到relay log中才返回给客户端。相对于异步复制，半同步复制提⾼了数据的安全性，同时它也造成了 ⼀定程度的延迟，这个延迟最少是⼀个TCP/IP往返的时间。所以，半同步复制最好在低延时的⽹络中使⽤。

**半同步复制的原理**

半同步复制的原理图：



图中，等待从库的反馈是发⽣在引擎层提交之后，这会带来⼀个潜在的问题，在引擎层提交之后，从库反馈之前， 所有其它的会话都可以看到这次提交的事务。如果此时主库crash了，切换到从上，客户端就会发现切换之前和切 换之后的数据不⼀致，也就是切换之前看到的数据丢失了。

针对上述问题，MySQL 5.7引⼊了⼀种新的半同步⽅案：⽆损复制（Lossless Replication）。

相对于之前的半同步⽅案，它将上图中的等待从库反馈的阶段调整到了引擎层提交之前。

具体使⽤哪种⽅案，由rpl\_semi\_sync\_master\_wait\_point参数控制：

AFTER\_SYNC：⽆损复制，默认值。

AFTER\_COMMIT：⽼的半同步复制⽅案。

相关插件

1. 主库 rpl\_semi\_sync\_master semisync\_master.so
2. 从库 rpl\_semi\_sync\_slave semisync\_slave.so
3. 安装 mysql提示符下
   1. 主库 install plugin rpl\_semi\_sync\_master soname ’semisync\_master.so’;
   2. 从库 install rpl\_semi\_sync slave soname semisync\_slave.so

相关配置

1. 主库 rpl\_semi\_sync\_master\_enabled=ON
2. 从库 rpl\_semi\_sync\_slave\_enabled=ON
3. plugin-load="rpl\_semi\_sync\_master=semisync\_master.so;rpl\_semi\_sync\_slave=semisync\_slave.so"
4. 需要安装插件启用半同步后，再修改配置文件，如果一来就修改配置文件，会导致无法启动，报错rpl\_semi\_sync\_master\_enabled为不识别参数

相关参数 show global variables like '%semi%';

1. rpl\_semi\_sync\_master\_wait\_point
   1. AFTER\_SYNC 无损复制，默认值
   2. AFTER\_COMMIT 老的半同步复制方式
2. rpl\_semi\_sync\_master\_enabled ON
3. rpl\_semi\_sync\_master\_timeout 10000 默认为10S
4. rpl\_semi\_sync\_master\_trace\_level 32
5. rpl\_semi\_sync\_master\_wait\_for\_slave\_count 1 半同步从库数
6. rpl\_semi\_sync\_master\_wait\_no\_slave ON 是否为半同步
7. rpl\_semi\_sync\_master\_wait\_point AFTER\_SYNC
8. rpl\_semi\_sync\_slave\_enabled OFF
9. rpl\_semi\_sync\_slave\_trace\_level 32

相关命令

1. 主库 show status like 'Rpl\_semi\_sync\_master\_status';
2. 从库 show status like 'Rpl\_semi\_sync\_slave\_status';
3. show variables like '%semi\_sync%';
4. show status like ’%semi\_sync%’;
   1. Rpl\_semi\_sync\_master\_clients 1 半同步复制从库个数
   2. Rpl\_semi\_sync\_master\_net\_avg\_wait\_time 0
   3. Rpl\_semi\_sync\_master\_net\_wait\_time 0
   4. Rpl\_semi\_sync\_master\_net\_waits 0
   5. Rpl\_semi\_sync\_master\_no\_times 0
   6. Rpl\_semi\_sync\_master\_no\_tx 0 异步复制状态下执行事务的个数
   7. Rpl\_semi\_sync\_master\_status ON
   8. Rpl\_semi\_sync\_master\_timefunc\_failures 0
   9. Rpl\_semi\_sync\_master\_tx\_avg\_wait\_time 0
   10. Rpl\_semi\_sync\_master\_tx\_wait\_time 0
   11. Rpl\_semi\_sync\_master\_tx\_waits 0
   12. Rpl\_semi\_sync\_master\_wait\_pos\_backtraverse 0
   13. Rpl\_semi\_sync\_master\_wait\_sessions 0
   14. Rpl\_semi\_sync\_master\_yes\_tx 0 半同步复制状态下执行事务的个数
   15. Rpl\_semi\_sync\_slave\_status OFF

⽐较重要的有以下⼏个：

Rpl\_semi\_sync\_master\_clients

当前半同步复制从的个数，如果是⼀主多从的架构，并不包含异步复制从的个数。

Rpl\_semi\_sync\_master\_no\_tx

主库切换为异步复制状态下执⾏的事务的个数。

Rpl\_semi\_sync\_master\_yes\_tx

主库在半同步复制状态下执⾏的事务的个数

1. mysql终端使用命令设置半同步后，需要重启从库io\_thread

**半同步复制的注意事项**

1. 半同步复制在主库和从库上必须同时开启。

2. 从库响应的条件是事务对应的⼆进制⽇志事件写⼊到Relay log中。

3. 在规定的时间内（由rpl\_semi\_sync\_master\_timeout参数决定，默认是10s），主库如果没有收到从库的响 应，半同步复制会切换为异步复制，同时，错误⽇志中会提示以下信息，  
[Warning] [MY-011153] [Repl] Timeout waiting for reply of binlog (file: mysql- bin.000004, pos: 9720138), semi-sync up to file mysql-bin.000004, position 9718392.   
后续如果收到了从库的响应，则异步复制⼜会恢复成半同步复制。如果不想半同步复制退化为异步复制，可将rpl\_semi\_sync\_master\_timeout设置为⼀个较⼤值。

4. 在⼀主多从的架构中，如果要开启半同步复制，并不要求所有的从库都是半同步复制。

5. 半同步复制会给事务的提交带来⼀定程度的延迟，这个延迟⾄少是⼀个 TCP/IP 往返的时间。所以，建议半同步复制在低延时的⽹络中使⽤。

**多源复制的概念**

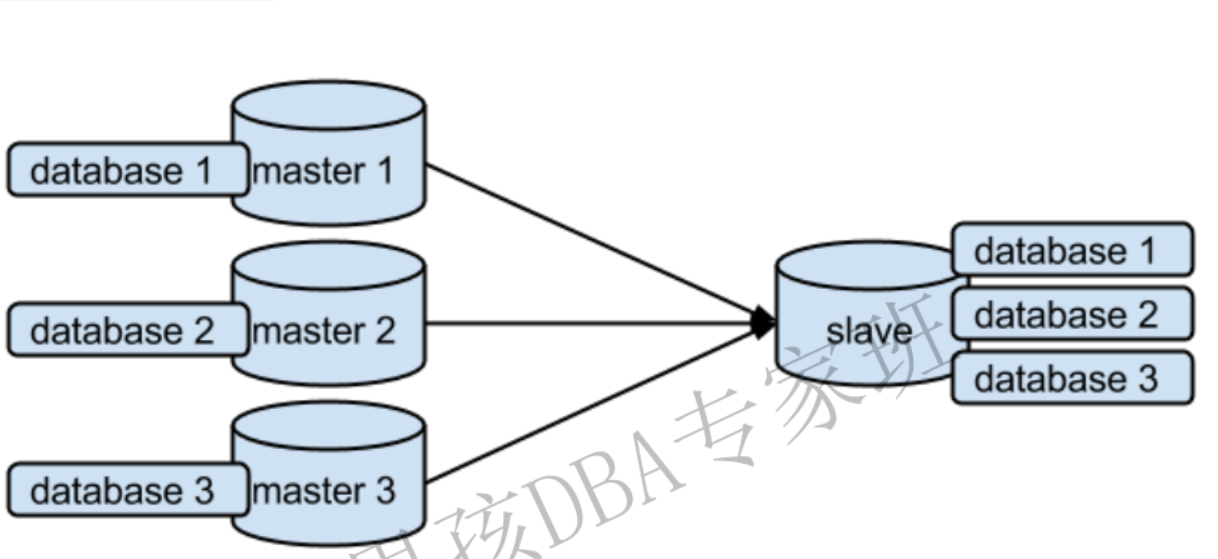
**多源复制指的是将多个主库的数据复制到同⼀个从库上。常⻅的应⽤场景有：**

1. 数据灾备。

2. 分库分表场景，将多个分⽚的数据汇总到⼀个实例上进⾏数据分析。

3. 数据聚合。将多个实例的数据聚集到⼀个实例上。

多源复制的原理图



**多源复制的搭建**

相对于普通的复制，多源复制的搭建相对⽐较简单。

1. 参数master-info-repository和relay-log-info-repository必须设置为TABLE。

2. 执⾏CHANGE MASTER TO命令时，需显式指定CHANNEL值。

CHANGE MASTER TO

MASTER\_HOST='192.168.244.10',

MASTER\_USER='repl',

MASTER\_PASSWORD='repl123',

MASTER\_AUTO\_POSITION = 1

FOR CHANNEL 'source\_1';

**多源复制搭建过程中的注意事项**

1. 在多源复制中，对主的复制模式没有限制，可以其中⼀个主库是GTID复制，另⼀个主库是基于binlog位置点 的复制。

如果主库的复制模式不⼀样，则从库的GTID\_MODE必须设置为ON\_PERMISSIVE或OFF\_PERMISSIVE，不能 设置为ON或OFF。

2. 如果是基于现有的主库搭建多源复制，有如下⼏点需要注意：

1. 备份还原⼯具的选择

要么全部使⽤逻辑备份⼯具，要么第⼀个主库使⽤Xtrabackup。为什么不能全部使⽤ Xtrabackup呢？ 因为Xtrabackup是物理备份，会拷⻉所有⽂件，在恢复的时候，不同 实例之间的系统表（ibdata）会存 在冲突。

2. GTID\_PURGED的设置

在MySQL 8.0之前，如果主库都开启了GTID，则在导⼊第⼆个主库时会报如下错误：

$ mysql < full\_backup\_master1.sql

$ mysql < full\_backup\_master2.sql

ERROR 1840 (HY000) at line 24: @@GLOBAL.GTID\_PURGED can only be set when @@GLOBAL.GTID\_EXECUTED is empty.

提示很明显，设置GTID\_PURGED的前提是GTID\_EXECUTED必须为空。

此时，可在从库上执⾏RESET MASTER操作，待第⼆个主库导⼊后，同样执⾏RESET MASTER操作，最后在从库上执⾏SET @@GLOBAL.GTID\_PURGED操作。GTID\_PURGED的值取master1和master2 GTID\_PURGED的交集，如：

$ grep "GTID\_PURGED" full\_backup\_master1.sql

...

SET @@GLOBAL.GTID\_PURGED='0ed33867-d1b0-11e8-8f58-000c2914fb06:1-9';

...

$ grep "GTID\_PURGED" full\_backup\_master2.sql

...

SET @@GLOBAL.GTID\_PURGED='ff693ae0-d1b2-11e8-b732-000c2927cfea:1-4';

SET @@GLOBAL.GTID\_PURGED=/\*!80000 '+'\*/ '453a5124-020e-11ec-8719- 000c29f66609:1-12';

命令中的/\*!80000 '+'\*/是注释，仅当MySQL服务端版本⼤于8.0.00时才会执⾏。

3. 如果⼆进制⽇志是ROW格式，不建议复制mysql库，否则很容易导致主从复制中断。

**多源复制的管理**

在执⾏相关管理操作时需指定CHANNEL的值。譬如，

SHOW SLAVE STATUS FOR CHANNEL 'master-1'\G START SLAVE FOR CHANNEL 'master-1';

STOP SLAVE FOR CHANNEL 'master-1';

RESET SLAVE FOR CHANNEL 'master-1';

RESET SLAVE ALL FOR CHANNEL 'master-1';

如果不指定CHANNEL的话，则默认对所有CHANNEL⽣效。

**多线程复制 （并行复制）**

从库上设置以下参数。

slave\_parallel\_type = LOGICAL\_CLOCK

slave\_parallel\_workers = 16

slave\_preserve\_commit\_order = ON

slave\_parallel\_type

设置从库并⾏复制的类型。该参数有以下取值：

DATABASE：基于库级别的并⾏复制。MySQL 8.0.27 之前的默认值。

LOGICAL\_CLOCK：基于组提交的并⾏复制。

slave\_parallel\_workers

设置 Worker 线程的数量。开启了多线程复制，原来的 SQL 线程将演变为 1 个 Coordinator 线程和多个 Worker 线程。

slave\_preserve\_commit\_order

事务在从库上的提交顺序是否与主库保持⼀致，建议开启。

注意，调整这三个参数，需重启复制⽣效。

从 MySQL 5.7.22、MySQL 8.0 开始，可使⽤ WRITESET ⽅案进⼀步提升并⾏复制的效率，此时，需在主库上设置 以下参数。

binlog\_transaction\_dependency\_tracking = WRITESET\_SESSION

transaction\_write\_set\_extraction = XXHASH64

binlog\_transaction\_dependency\_history\_size = 25000 binlog\_format = ROW

注意，基于 WRITESET 的并⾏复制⽅案，只在 binlog 格式为 ROW 的情况下才⽣效。

**过滤复制**

CHANGE REPLICATION FILTER

CHANGE REPLICATION FILTER filter[, filter] [, ...] [FOR CHANNEL channel]

filter: {

REPLICATE\_DO\_DB = (db\_list)

| REPLICATE\_IGNORE\_DB = (db\_list)

| REPLICATE\_DO\_TABLE = (tbl\_list)

| REPLICATE\_IGNORE\_TABLE = (tbl\_list)

| REPLICATE\_WILD\_DO\_TABLE = (wild\_tbl\_list)

| REPLICATE\_WILD\_IGNORE\_TABLE = (wild\_tbl\_list) | REPLICATE\_REWRITE\_DB = (db\_pair\_list)

db\_list:

db\_name[, db\_name][, ...]

tbl\_list:

db\_name.table\_name[, db\_name.table\_name][, ...]

wild\_tbl\_list:

'db\_pattern.table\_pattern'[, 'db\_pattern.table\_pattern'][, ...] db\_pair\_list:

(db\_pair)[, (db\_pair)][, ...]

db\_pair:

from\_db, to\_db

其中，

REPLICATE\_DO\_DB：复制指定库的操作，相当于--replicate-do-db。

REPLICATE\_IGNORE\_DB：忽略指定库的操作，相当于--replicate-ignore-db。

REPLICATE\_DO\_TABLE：复制指定表的操作， 相当于--replicate-do-table。

REPLICATE\_IGNORE\_TABLE：忽略指定表的操作。 相当于--replicate-ignore-table。

REPLICATE\_WILD\_DO\_TABLE：复制指定表的操作，⽀持正则匹配，相当于--replicate-wild-do-table。 REPLICATE\_WILD\_IGNORE\_TABLE：忽略指定表的操作，⽀持正则匹配，相当于--replicate-wild-ignore- table。

REPLICATE\_REWRITE\_DB：将主库A库的操作重写到从库B库中，相当于--replicate-rewrite-db。

在执⾏完 CHANGE REPLICATION FILTER 后，对应的规则写到参数⽂件中。

-- 不指定 CHANNEL，默认创建的是全局规则，对所有 channel 都⽣效。 CHANGE REPLICATION FILTER

REPLICATE\_DO\_DB = (d1), REPLICATE\_IGNORE\_DB = (d2);

CHANGE REPLICATION FILTER REPLICATE\_DO\_DB = (d1) FOR CHANNEL channel\_1;

CHANGE REPLICATION FILTER

REPLICATE\_WILD\_DO\_TABLE = ('db1.old%');

CHANGE REPLICATION FILTER

REPLICATE\_WILD\_IGNORE\_TABLE = ('db1.new%', 'db2.new%');

CHANGE REPLICATION FILTER

REPLICATE\_REWRITE\_DB = ((dbA, dbB), (dbC, dbD))   
-- 清除过滤规则

CHANGE REPLICATION FILTER

REPLICATE\_DO\_DB = (), REPLICATE\_IGNORE\_DB = ();   
生产中，如果有库级别的过滤，建议使用   
REPLICATE\_WILD\_DO\_TABLE，REPLICATE\_WILD\_IGNORE\_TABLE  
而不是直接使用REPLICATE\_DO\_DB , REPLICATE\_IGNORE\_DB ;

**延迟复制**

开启较为简单，只需设置CHANGE MASTER TO语句中的MASTER\_DELAY选项。 搭建时设置：

CHANGE MASTER TO

MASTER\_HOST='192.168.79.10',

MASTER\_USER='repl',

MASTER\_PASSWORD='123456',

MASTER\_PORT=3306,

MASTER\_LOG\_FILE='mysql-bin.000001', MASTER\_LOG\_POS=155,

MASTER\_DELAY=28800;

运⾏时设置：

STOP SLAVE;

CHANGE MASTER TO MASTER\_DELAY = 28800;

START SLAVE;

延迟复制的相关信息可通过 show slave status\G 查看。

SQL\_Delay: 28800

SQL\_Remaining\_Delay: 28774

Slave\_SQL\_Running\_State: Waiting until MASTER\_DELAY seconds after master executed event

输出中，SQL\_Delay 是期望延迟时间，即 CHANGE MASTER TO 语句中设置的 MASTER\_DELAY。 SQL\_Remaining\_Delay指的是当前被暂停的事务需要等待多久，才能到达期望延迟时间。

**查看主库的状态**

SHOW MASTER STATUS

mysql> show master status;

mysql> show master status;

+------------------+----------+--------------+------------------+-------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set |

+------------------+----------+--------------+------------------+-------------------+

| mysql-bin.000012 | 157 | | | |

+------------------+----------+--------------+------------------+-------------------+

1 row in set (0.00 sec)

**查看从库复制的状态**

SHOW SLAVE STATUS [FOR CHANNEL channel]

常⻅字段的含义。

Master\_Host，Master\_User，Master\_Port：分别代表主库的地址（主机名或 IP ），复制⽤户及端⼝。

Master\_Log\_File，Read\_Master\_Log\_Pos：IO 线程当前正在接受的⼆进制⽇志事件在主库 binlog 的位置。

Relay\_Master\_Log\_File，Exec\_Master\_Log\_Pos：SQL 线程当前正在重放的⼆进制⽇志事件在主库 binlog 的位置。

Slave\_IO\_Running：IO 线程是否处于正常运⾏的状态。有 Yes，No，Connecting 三种状态。

Slave\_SQL\_Running：SQL 线程是否处于正常运⾏的状态。

Replicate\_Do\_XXX，Replicate\_Ignore\_XXX：基于库表的复制。

Until\_Condition，Until\_Log\_File，Until\_Log\_Pos：SQL 线程应⽤到指定位置点时停⽌。

Seconds\_Behind\_Master：从库相对于主库的延迟。

SQL\_Delay，SQL\_Remaining\_Delay：常⽤在延迟复制中。SQL\_Delay 指的是期望延迟时间，对应 CHANGE MASTER TO 中的 MASTER\_DELAY ，SQL\_Remaining\_Delay 指的是当前被暂停的事务需要等待多久，才能 到达期望延迟时间。两者的单位都是秒。

Retrieved\_Gtid\_Set：接受过的事务所对应的 GTID 集。

Executed\_Gtid\_Set：从库执⾏过的事务所对应的 GTID 集。

Auto\_Position：对应 CHANGE MASTER TO 中的 MASTER\_AUTO\_POSITION。

Channel\_Name：Channel 名，常⽤在多源复制中。

**建⽴复制**

CHANGE MASTER TO option [, option] ... [ channel\_option ]

8.0开始最好将MASTER 替换为 SOURCE

CHANG MASTER TO 替换为 CHANGE REPLICATION SOURCE TO

常用选项

SOURCE\_HOST

SOURCE\_USER

SOURCE\_PASSWORD

SOURCE\_PORT

SOURCE\_AUTO\_POSITION = 1

SOURCE\_HEARTBEAT\_PERIOD = 3

SOURCE\_CONNECT\_RETRY = 10

SOURCE\_LOG\_FILE = '',

SOURCE\_LOG\_POS = ,

GET\_SOURCE\_PUBLIC\_KEY = 1;

FOR CHANNEL 'channel name'

在开启复制后，如果要修改某个选项的值，需在停⽌复制后，重新执⾏ CHANGE MASTER TO 语句，但这次只需 指定对应的选项，不⽤带上其它选项。

CHANGE MASTER TO MASTER\_PASSWORD = '123456';

重放 relay log 中的内容。

CHANGE MASTER TO

RELAY\_LOG\_FILE = 'node2-relay-bin.000004',

RELAY\_LOG\_POS = 744;

**开启复制**

开启复制的命令是 START SLAVE。但其实，该命令⽀持众多选项，其完整语法如下。

START {SLAVE | REPLICA} [thread\_types] [until\_option] [connection\_options]

[channel\_option]

thread\_types:

[thread\_type [, thread\_type] ... ]

thread\_type:

IO\_THREAD | SQL\_THREAD

until\_option:

UNTIL { {SQL\_BEFORE\_GTIDS | SQL\_AFTER\_GTIDS} = gtid\_set

| MASTER\_LOG\_FILE = 'log\_name', MASTER\_LOG\_POS = log\_pos

| SOURCE\_LOG\_FILE = 'log\_name', SOURCE\_LOG\_POS = log\_pos

| RELAY\_LOG\_FILE = 'log\_name', RELAY\_LOG\_POS = log\_pos

| SQL\_AFTER\_MTS\_GAPS }

connection\_options:

FOR CHANNEL channel

如果不指定任何选项，则默认是开启 IO 线程和 SQL 线程，也可通过以下命令单独启动某个线程。

start slave io\_thread;

start slave sql\_thread;

这⾥，重点说说 until\_option，这个选项会让 SQL 线程应⽤在指定位置点时停⽌。

常⻅的使⽤场景有：

1. 在⼀主多从且基于位置点的复制场景中，变更从库之间的拓扑。譬如，将其中⼀个从库切换为另⼀个从库的从库。

2. 在延迟复制中，恢复到指定位置点。

下⾯看看 until\_option 的具体选项：

SQL\_BEFORE\_GTIDS，SQL\_AFTER\_GTIDS：针对 GTID 复制。这⾥的 GTIDS 是个集合，以下⾯这个 GTIDS 为例。

bd6b3216-04d6-11ec-b76f-000c292c1f7b:31-39

对于 SQL\_BEFORE\_GTIDS，SQL 线程在应⽤到bd6b3216-04d6-11ec-b76f-000c292c1f7b:31 这个事务时（不包括这个事务）停⽌。

对于 SQL\_AFTER\_GTIDS，SQL 线程在应⽤完 bd6b3216-04d6-11ec-b76f-000c292c1f7b:39 这个事务后停⽌。

MASTER\_LOG\_FILE，MASTER\_LOG\_POS：主库 binlog 的位置点，对应 SHOW SLAVE STATUS 中的 Relay\_Master\_Log\_File，Exec\_Master\_Log\_Pos。

RELAY\_LOG\_FILE，RELAY\_LOG\_POS：从库 relay log 的位置点，对应 SHOW SLAVE STATUS 中的 Relay\_Log\_File，Relay\_Log\_Pos。

在使⽤ until\_option 时，需要注意以下⼏点。

1. until\_option 只对 SQL 线程有效果，对 IO 线程没有效果 。

2. 在 GTID 复制中，也可指定位置点。

3. 执⾏ STOP SLAVE 会清空 until\_option。

**停⽌复制**

STOP SLAVE [ IO\_THREAD | SQL\_THREAD ][FOR CHANNEL channel]

不指定任何选项，则默认是停⽌ IO 线程和 SQL 线程，也可通过以下命令单独停⽌某个线程。

stop slave io\_thread;

stop slave sql\_thread;

命令中的 channel ⽤来指定 CHANNEL 名，常⽤于多源复制。如果不指定，默认会对所有 CHANNEL ⽣效。

**在主库上查看从库IP，端⼝信息**

在主库上查看从库IP，端⼝信息，可通过以下两种⽅式。

1. SHOW PROCESSLIST

2. SHOW SLAVE HOSTS | SHOW REPLICAS

**查看当前实例拥有的 Binlog**

SHOW {BINARY | MASTER} LOGS

**删除 Binlog**

PURGE { BINARY | MASTER } LOGS { TO 'log\_name' | BEFORE datetime\_expr }

既可基于⽂件名，也可基于时间点删除。如，

purge binary logs to 'mysql-bin.000002'

purge binary logs before '2024-01-14 09:10:20'

在执⾏ PURGE 操作时，注意：

1. 第⼀个 PURGE 操作只会删除 mysql-bin.000004 之前的 binlog。

2. 在 GTID 复制中，执⾏ PURGE BINARY LOGS 操作还会同步修改 gtid\_purged 的值。

**切换⽇志**

FLUSH BINARY LOGS

该命令会关闭当前的 binlog，并重新打开⼀个新的 binlog。

**查看 Binlog 的内容**

SHOW BINLOG EVENTS [IN 'log\_name'] [FROM pos] [LIMIT [offset,] row\_count]

如果不指定 log\_name，则默认是 SHOW BINARY LOGS 中的第⼀个 binlog 。

**跳过指定事务**

基于位置点的复制

mysql> stop slave;

mysql> set global sql\_slave\_skip\_counter = 1;

mysql> start slave

在GTID 复制中， 跳过事务的本质是注⼊空事务。

mysql> show slave status\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

Last\_SQL\_Error: Error 'Table 't1' already exists' on query. Default

database: ''. Query: 'create table slowtech.t1(id int primary key)'

...

Retrieved\_Gtid\_Set: bd6b3216-04d6-11ec-b76f-000c292c1f7b:11664-11711

Executed\_Gtid\_Set: bd6b3216-04d6-11ec-b76f-000c292c1f7b:1-11690

...

mysql> stop slave;

mysql> set session gtid\_next='bd6b3216-04d6-11ec-b76f-000c292c1f7b:11691';

mysql> begin;

mysql> commit;

mysql> set session gtid\_next='automatic';

mysql> start slave;

这⾥⾯，第⼀个 gtid\_next 的设置是关键，需取 Executed\_Gtid\_Set 的最⼤值 + 1。

**操作不写⼊ Binlog**

mysql> set session sql\_log\_bin=0;

**判断主库的某个操作是否已经在从库上执⾏**

位置点复制，推荐使⽤ MASTER\_POS\_WAIT

SELECT MASTER\_POS\_WAIT('master\_log\_file', master\_log\_pos [, timeout][, channel])

给出具体的 binlog 及位置点，如果从库 SQL 线程还没重放到这个位置，该命令会⼀直阻塞。也可指定 timeout，设置命令的超时时间，如果在超时时间内还没重放到指定位置，该命令会返回 -1 。

GTID 复制，推荐使⽤ WAIT\_FOR\_EXECUTED\_GTID\_SET，其使⽤⽅式如下，

SELECT WAIT\_FOR\_EXECUTED\_GTID\_SET(gtid\_set[, timeout])

**RESET MASTER，RESET SLAVE 和 RESET SLAVE ALL 三者的区别**

RESET MASTER

该命令会执⾏以下操作。

1. 删除所有的 binlog，并从头开始⽣成⼀个新的 binlog。

2. 清空 binlog 索引⽂件（ mysql-bin.index ）。

3. 在 GTID 复制中，会同时清空变量 gtid\_executed、gtid\_purged及表

mysql.gtid\_executed 的内容。 如：

mysql> show binary logs;

+------------------+-----------+-----------+

| Log\_name | File\_size | Encrypted |

+------------------+-----------+-----------+

| mysql-bin.000001 | 139239 | No |

| mysql-bin.000002 | 1055792 | No |

| mysql-bin.000003 | 1053576 | No |

| mysql-bin.000004 | 196 | No |

+------------------+-----------+-----------+

4 rows in set (0.01 sec)

mysql> reset master;

Query OK, 0 rows affected (0.09 sec)

mysql> show binary logs;

+------------------+-----------+-----------+

| Log\_name | File\_size | Encrypted |

+------------------+-----------+-----------+

| mysql-bin.000001 | 156 | No |

+------------------+-----------+-----------+

1 row in set (0.00 sec)

该命令既可在主库上执⾏，也可在从库上执⾏。在执⾏的时候注意。

1. 在正常的主从复制环境中，切记不要在主库上执⾏ RESET MASTER 命令，否则，很容易导致主从复制中断， 甚⾄主从数据不⼀致。

2. 执⾏完 RESET MASTER ，默认情况下，⽣成的第⼀个 binlog 的序号是 000001 ，也可通过 RESET MASTER TO xxx 命令显式指定第⼀个 binlog 的序号。

RESET SLAVE

该命令会执⾏以下操作。

1. 删除所有的 relay log，并从头开始⽣成⼀个新的 relay log。

2. 清空 relay log 索引⽂件。

3. 清空 mysql.slave\_relay\_log\_info 表的内容。

4. 清除 mysql.slave\_master\_info 表中 binlog 的位置点信息，具体来说， Master\_log\_name 和 Master\_log\_pos ，但是会保留连接信息。

既然连接信息依然存在，所以可直接通过 START SLAVE 重启复制。在 GTID 复制中，因为 gtid\_purged ，这样操作是没有问题的，复制可继续进⾏。但如果是基于位置点的复制，切记不要这样操作，因为 IO 线程会从主库现有的第⼀个 binlog 开始重新拉取数据，这样很容易导致主从数据不⼀致。

RESET SLAVE ALL

相对于 RESET SLAVE ，RESET SLAVE ALL 会直接清空 mysql.slave\_master\_info 表的所有内容，包括连接信息。

这个时候，如果要重启复制，只能执⾏ CHANGE MASTER TO 命令。

⽆论是 RESET SLAVE 还是 RESET SLAVE ALL，在执⾏的时候注意。

1. 在执⾏之前，必须停⽌复制。

2. 不会清除 GTID 相关的任何信息。

**MySQL 主从延迟的常见原因及解决方法**

**⼀、Seconds\_Behind\_Master 的实现逻辑**

sql/rpl\_slave.cc

/\*

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;

}

\*/

以上是计算 Seconds\_Behind\_Master 的伪代码。

1. Seconds\_Behind\_Master 为 0 的条件是 SQL 线程已经重放完所有的 relay log 且 Slave\_IO\_Running 为 Yes

。

2. 在以下两种情况中，Seconds\_Behind\_Master 会为 NULL。

Slave\_SQL\_Running 为 No 。

Slave\_SQL\_Running 为 Yes ，在重放完所有的 relay log 时，Slave\_IO\_Running 不为 Yes。

sql/rpl\_slave.cc

if (mi->rli->slave\_running) {

if ((mi->get\_master\_log\_pos() == mi->rli->get\_group\_master\_log\_pos()) &&

(!strcmp(mi->get\_master\_log\_name(),

mi->rli->get\_group\_master\_log\_name()))) {

if (mi->slave\_running == MYSQL\_SLAVE\_RUN\_CONNECT)

protocol->store(0LL);

else

protocol->store\_null();

} else {

long time\_diff = ((long)(time(nullptr) - mi->rli->last\_master\_timestamp) - mi->clock\_diff\_with\_master);

protocol->store(

(longlong)(mi->rli->last\_master\_timestamp ? max(0L, time\_diff) : 0)); }

} else {

protocol->store\_null();

}

Seconds\_Behind\_Master 的计算逻辑是由如下代码实现的。

long time\_diff = ((long)(time(nullptr) - mi->rli->last\_master\_timestamp) - mi->clock\_diff\_with\_master);

protocol->store(

(longlong)(mi->rli->last\_master\_timestamp ? max(0L, time\_diff) : 0));

这⾥⾯，

time(0)：从库当前的系统时间。

mi->rli->last\_master\_timestamp：SQL 线程当前重放的事务在主库开始执⾏的时间戳，具体到 binlog 中，

即事件注释部分的时间戳，如下⾯这个事件中的 210912 9:53:25 。

#210912 9:53:25 server id 1 end\_log\_pos 26045 CRC32 0xcca7e8ab GTID

last\_committed=66 sequence\_number=67 rbr\_only=no

mi->clock\_diff\_with\_master：IO 线程启动时，主从之间的系统时间差。

注意，如果 IO 线程启动后调整过系统时间，需重启复制，否则会影响Seconds\_Behind\_Master 的计算结果。

所以，Seconds\_Behind\_Master = 从库当前的系统时间 - SQL 线程当前重放的事务在主库的开始时间戳 - 主从之间的系统时间差。

**⼆、如何分析主从延迟**

分析主从延迟⼀般会采集以下三类信息。

**2.1 从库服务器的负载情况**

为什么要⾸先查看服务器的负载情况呢？因为软件层⾯的所有操作都需要系统资源来⽀撑。

常⻅的系统资源有四类：CPU、内存、IO、⽹络。对于主从延迟，⼀般会重点关注 CPU 和 IO 。

分析 CPU 是否达到瓶颈，常⽤的命令是 top，通过 top 我们可以直观地看到主机的 CPU 使⽤情况。以下是 top 中

CPU 相关的输出。

Cpu(s): 0.2%us, 0.2%sy, 0.0%ni, 99.5%id, 0.0%wa, 0.0%hi, 0.2%si, 0.0%st

下⾯我们看看各个指标的具体含义。

us：处理⽤户态（ user ）任务的 CPU 时间占⽐。

sy：处理内核态（ system ）任务的 CPU 时间占⽐。

ni：处理低优先级进程⽤户态任务的 CPU 时间占⽐。

进程的优先级由 nice 值决定，nine 的范围是 -20 ～ 19 ，值越⼤，优先级越低。其中，1 ～ 19 称之为低优先级。

id：处于空闲状态（ idle ）的 CPU 时间占⽐。

wa：等待 IO 的 CPU 时间占⽐。

hi：处理硬中断（ irq ）的 CPU 时间占⽐。

si：处理软中断（ softirq ）的 CPU 使⽤率。

st：当系统运⾏在虚拟机中的时候，被其它虚拟机占⽤（ steal ）的 CPU 时间占⽐。

**⼀般来说，当 CPU 使⽤率 （ 1 - 处于空闲状态的 CPU 时间占⽐ ）超过 90% 时，需引起⾜够关注。毕竟，对于数据库应⽤来说，CPU 很少是瓶颈，除⾮有⼤量的慢 SQL 。**

接下来看看 IO。

查看磁盘 IO 负载情况，常⽤的命令是 iostat

# iostat -xm 1

avg-cpu: %user %nice %system %iowait %steal %idle

4.21 0.00 1.77 0.35 0.00 93.67

Device: rrqm/s wrqm/s r/s w/s rMB/s wMB/s avgrq-sz avgqu-sz

await r\_await w\_await svctm %util

sda 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

0.00 0.00 0.00 0.00 0.00

sdb 0.00 0.00 841.00 3234.00 13.14 38.96 26.19 0.60

0.15 0.30 0.11 0.08 32.60

命令中指定了 3 个选项，其中

-x：打印扩展信息。

-m：指定吞吐量的单位是 MB/s ，默认是 KB/s 。

1：每隔 1s 打印⼀次。

下⾯看看输出中各指标的具体含义。

rrqm/s：每秒被合并的读请求的数量。

wrqm/s：每秒被合并的写请求的数量。

r/s：每秒发送给磁盘的读请求的数量。

w/s：每秒写⼊磁盘的写请求的数量。注意，这⾥的请求是合并后的请求。r/s + w/s 等于 IOPS 。

rMB/s：每秒从磁盘读取的数据量。

wMB/s：每秒写⼊磁盘的数据量。rMB/s + wMB/s 等于吞吐量。

avgrq-sz：I/O 请求的平均⼤⼩，单位是扇区，扇区的⼤⼩是 512 字节。⼀般⽽⾔，I/O 请求越⼤，耗时越⻓。

avgqu-sz：队列⾥的平均 I/O 请求数量。

await：I/O 请求的平均耗时，包括磁盘的实际处理时间及队列中的等待时间，单位 ms 。

其中，r\_await 是读请求的平均耗时，w\_await 是写请求的平均耗时。

svctm：I/O 请求的平均服务时间，单位 ms 。注意，这个指标已弃⽤，在后续版本会移除。

%util：磁盘饱和度。反映了⼀个采样周期内，有多少时间在做 I/O 操作。

⼀般来说，我们会重点关注 await 和 %util。

对于只能串⾏处理 I/O 请求的设备来说，%util 接近 100% ，就意味着设备饱和。但对于 RAID、SSD 等设备，因为它能并⾏处理，故该值参考意义不⼤，即使达到了 100% ，也不意味着设备出现了饱和。⾄于是否达到了性能上限，需参考性能压测下的 IOPS 和吞吐量。

**2.2 主从复制状态**

对于主库，执⾏ SHOW MASTER STATUS 。

mysql> show master status;

+------------------+----------+--------------+------------------+----------------------

-----------------------+

| File | Position | Binlog\_Do\_DB | Binlog\_Ignore\_DB | Executed\_Gtid\_Set

|

+------------------+----------+--------------+------------------+----------------------

-----------------------+

| mysql-bin.000004 | 1631495 | | | bd6b3216-04d6-11ec

b76f-000c292c1f7b:1-5588 |

+------------------+----------+--------------+------------------+----------------------

-----------------------+

1 row in set (0.00 sec)

SHOW MASTER STATUS 的输出中重点关注 File 和 Position 这两个指标的值。

对于从库，执⾏ SHOW SLAVE STATUS 。

mysql> show slave status\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

...

Master\_Log\_File: mysql-bin.000004

Read\_Master\_Log\_Pos: 1631495

...

Relay\_Master\_Log\_File: mysql-bin.000004

...

Exec\_Master\_Log\_Pos: 1631495

SHOW SLAVE STATUS 的输出中重点关注 Master\_Log\_File，Read\_Master\_Log\_Pos，Relay\_Master\_Log\_File， Exec\_Master\_Log\_Pos 这四个指标的值。

接下来，重点⽐较以下两对值。

第⼀对：( File , Position ) & ( Master\_Log\_File , Read\_Master\_Log\_Pos )

这⾥⾯， ( File , Position ) 记录了主库 binlog 的位置。

( Master\_Log\_File , Read\_Master\_Log\_Pos ) 记录了 IO 线程当前正在接收的⼆进制⽇志事件在主库 binlog 中的位置。

如果 ( File , Position ) ⼤于 ( Master\_Log\_File , Read\_Master\_Log\_Pos ) ，则意味着 IO 线程存在延迟。

第⼆对：( Master\_Log\_File , Read\_Master\_Log\_Pos ) & ( Relay\_Master\_Log\_File , Exec\_Master\_Log\_Pos )

这⾥⾯，( Relay\_Master\_Log\_File, Exec\_Master\_Log\_Pos ) 记录了 SQL 线程当前正在重放的⼆进制⽇志事件在主库 binlog 的位置。

如果 ( Relay\_Master\_Log\_File, Exec\_Master\_Log\_Pos ) < ( Master\_Log\_File, Read\_Master\_Log\_Pos ) ，则意味着 SQL 线程存在延迟。

**1.3 主库 binlog 的写⼊量**

主要是看主库 binlog 的⽣成速度，⽐如多少分钟⽣成⼀个。

**三、主从延迟的常⻅原因及解决⽅法**

下⾯分别从 IO 线程和 SQL 线程这两个⽅⾯展开介绍。

**3.1 IO 线程存在延迟**

下⾯看看 IO 线程出现延迟的常⻅原因及解决⽅法。

1. ⽹络延迟。

判断是否为⽹络带宽限制。如果是，可开启 slave\_compressed\_protocol 参数，启⽤ binlog 的压缩传输。

2. 磁盘 IO 存在瓶颈 。

可调整从库的双⼀设置或关闭 binlog。

注意，在 MySQL 5.6 中，如果开启了 GTID ，则会强制要求开启 binlog ，MySQL 5.7 ⽆此限制。

3. ⽹卡存在问题。

这种情况不多⻅，但确实碰到过。当时是⼀主两从的架构，发现⼀台主机上的所有从库都延迟了，但这些从库对应集群的其它从库却没有延迟，后来通过 scp 远程拷⻉⽂件进⼀步确认了该台主机的⽹络存在问题，最后经系统组确认，⽹卡存在问题。

⼀般情况下，IO 线程很少存在延迟。

**3.2 SQL 线程存在延迟**

下⾯看看 SQL 线程出现延迟的常⻅原因及解决⽅法。

**3.2.1 主库写⼊量过⼤，SQL 线程单线程重放**

具体体现如下：

1. 从库磁盘 IO ⽆明显瓶颈。

2. Relay\_Master\_Log\_File , Exec\_Master\_Log\_Pos 也在不断变化。

3. 主库写⼊量过⼤。如果磁盘使⽤的是 SATA SSD，当 binlog 的⽣成速度快于 5 分钟⼀个时，从库重放就会有瓶颈。

这个是 MySQL 软件层⾯的硬伤。要解决该问题，可开启并⾏复制。

**3.2.2 STATEMENT 格式下的慢 SQL**

具体体现，在⼀段时间内 Relay\_Master\_Log\_File , Exec\_Master\_Log\_Pos 没有变化。

看下⾯这个示例，对 1 张千万数据的表进⾏ DELETE 操作，表上没有任何索引，在主库上执⾏⽤了 7.52s，观察从库的 Seconds\_Behind\_Master，发现它最⼤达到了 7s 。

mysql> show variables like 'binlog\_format';

+---------------+-----------+

| Variable\_name | Value |

+---------------+-----------+

| binlog\_format | STATEMENT |

+---------------+-----------+

1 row in set (0.00 sec)

mysql> select count(\*) from sbtest.sbtest1;

+----------+

| count(\*) |

+----------+

| 10000000 |

+----------+

1 row in set (1.41 sec)

mysql> show create table sbtest.sbtest1\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: sbtest1

Create Table: CREATE TABLE `sbtest1` (

`id` int NOT NULL,

`k` int NOT NULL DEFAULT '0',

`c` char(120) NOT NULL DEFAULT '',

`pad` char(60) NOT NULL DEFAULT ''

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci

1 row in set (0.00 sec)

mysql> delete from sbtest.sbtest1 where id <= 100;

Query OK, 100 rows affected (7.52 sec)

对于这种执⾏较慢的 SQL ，并⾏复制实际上也是⽆能为⼒的， 此时只能优化 SQL。

在 MySQL 5.6.11 中，引⼊了参数 log\_slow\_slave\_statements ，可将 SQL 重放过程中执⾏时⻓超过long\_query\_time 的操作记录在慢⽇志中。

**3.2.3 表上没有任何索引，且⼆进制⽇志格式为 ROW**

同样，在⼀段时间内，Relay\_Master\_Log\_File , Exec\_Master\_Log\_Pos 不会变化。

如果表上没有任何索引，对它进⾏操作，在主库上只是⼀次全表扫描。但在从库重放时，因为是 ROW 格式，对于每条记录的操作都会进⾏⼀次全表扫描。

还是上⾯的表，同样的操作，只不过⼆进制⽇志格式为 ROW ，在主库上执⾏⽤了 7.53s ，但Seconds\_Behind\_Master 最⼤却达到了 723s ，是 STATEMENT 格式下的 100 倍。

mysql> show variables like 'binlog\_format';

+---------------+-------+

| Variable\_name | Value |

+---------------+-------+

| binlog\_format | ROW |

+---------------+-------+

1 row in set (0.00 sec)

mysql> delete from sbtest.sbtest1 where id <= 100;

Query OK, 100 rows affected (7.53 sec)

如果因为表上没有任何索引，导致主从延迟过⼤，常⻅的优化⽅案如下：

1. 在从库上临时创建个索引，加快记录的重放。注意，尽量选择⼀个区分度⾼的列添加索引，列的区分度越⾼，重放的速度就越快。

2. 将参数 slave\_rows\_search\_algorithms 设置为 INDEX\_SCAN,HASH\_SCAN 。

设置后，对于同样的操作，Seconds\_Behind\_Master 最⼤只有 53s 。

**3.2.4 ⼤事务**

这⾥的⼤事务，指的是⼆进制⽇志格式为 ROW 的情况下，操作涉及的记录数较多。

还是上⾯的测试表，只不过这次 id 列是⾃增主键，执⾏批量更新操作。更新操作如下，其中，N 是记录数，M 是 ⼀个随机字符，每次操作的字符均不⼀样

update sbtest.sbtest1 set c=repeat(M,120) where id<=N

接下来我们看看不同记录数下对应 Seconds\_Behind\_Master 的最⼤值。

|  |  |  |
| --- | --- | --- |
| **记录数** | **主库执⾏时⻓(s)** | **Seconds\_Behind\_Master最⼤值(s)** |
| 50000 | 0.76 | 1 |
| 200000 | 3.10 | 8 |
| 500000 | 17.32 | 39 |
| 1000000 | 63.47 | 122 |

可⻅，随着记录数的增加，Seconds\_Behind\_Master 也是不断增加的。

所以对于⼤事务操作，建议分⽽治之，每次⼩批量执⾏。

**3.2.5 从库上有查询操作**

从库上有查询操作，通常会有两⽅⾯的影响：

1.消耗系统资源。

2.锁等待。

常⻅的是从库的查询操作堵塞了主库的 DDL 操作。看下⾯这个示例。

mysql> show processlist;

+----+-----------------+-----------------+------+---------+------+---------------------

-------------+----------------------------------------+

| Id | User | Host | db | Command | Time | State

| Info |

+----+-----------------+-----------------+------+---------+------+---------------------

-------------+----------------------------------------+

| 5 | event\_scheduler | localhost | NULL | Daemon | 2239 | Waiting on empty

queue | NULL |

| 17 | root | localhost | NULL | Query | 0 | init

| show processlist |

| 18 | root | localhost | NULL | Query | 19 | User sleep

| select id,sleep(1) from sbtest.sbtest1 |

| 19 | system user | connecting host | NULL | Connect | 243 | Waiting for source

to send event | NULL |

| 20 | system user | | | Query | 13 | Waiting for table

metadata lock | alter table sbtest.sbtest1 add c2 int |

+----+-----------------+-----------------+------+---------+------+---------------------

-------------+----------------------------------------+

5 rows in set (0.00 sec)

**3.2.6 从库上存在备份**

常⻅的是备份的全局读锁阻塞了 SQL 线程的重放。看下⾯这个示例。

mysql> show processlist;

+----+-----------------+-----------------+------+---------+------+---------------------

-------------+----------------------------------------+

| Id | User | Host | db | Command | Time | State

| Info |

+----+-----------------+-----------------+------+---------+------+---------------------

-------------+----------------------------------------+

| 5 | event\_scheduler | localhost | NULL | Daemon | 4177 | Waiting on empty

queue | NULL |

| 17 | root | localhost | NULL | Query | 0 | init

| show processlist |

| 18 | root | localhost | NULL | Query | 36 | User sleep

| select id,sleep(1) from sbtest.sbtest2 |

| 19 | system user | connecting host | NULL | Connect | 2181 | Waiting for source

to send event | NULL |

| 20 | system user | | | Query | 2 | Waiting for global

read lock | alter table sbtest.sbtest1 add c1 int |

| 28 | root | localhost | NULL | Query | 17 | Waiting for table

flush | flush tables with read lock |

+----+-----------------+-----------------+------+---------+------+---------------------

-------------+----------------------------------------+

6 rows in set (0.00 sec)

**3.2.7 磁盘 IO 存在瓶颈**

这个时候可调整从库的双⼀设置或关闭 binlog。

**四、总结**

主从延迟的常见原因及解决方法

IO 线程存在延迟

网络延迟

可开启 slave\_compressed\_protocol 参数，启用压缩传输二进制日志

磁盘 I0 存在瓶颈

可调整从库的双一设置 关闭 bir1og

网卡存在问题

可通过scp远程拷贝文件测试网络的带宽

SQL 线程存在延迟

SOL 线程单线程重放 开启并行复制

STATEMENT 格式下的慢 SQL 开启 log\_slow\_slave\_statements，优化慢 SQL

ROW 格式

表上没有任何索引

添加索引

将 slave rows search algorithms 设置为INDEX SCAN,HASH SCAN

大事务

分批执行

从库上有查询操作

查询占用了大量系统资源

锁等待，从库的查询操作堵塞了主库的 DDL 操作

备份

全局读锁堵塞了 Sql线程的重放操作