MySQL学习笔记（Day042：replication_3）

MySQL学习

一. 作业解析

1.1. Seconds_Behind_Master

`Seconds_Behind_Master` 字面意思为Slave落后Master的 `秒数` ，但是实际上并不是十分准确。

计算的方式为： `Slave`的`SQL`线程执行时刻的时间 减去 `event`产生时刻的时间 （ *event_header 中的前 4 个字节就是timestamp* ）

> `Seconds_Behind_Master` 测算主从延时的时间的方式并 `不准确` ，因为回放的SQL线程可能落后很多

1.2. pt-heartbeat

> Percona ToolKit Download
>
> pt-heardbeat 官方文档

```
+-----------------+      (2)      +-----------------+
|      Master     |  Replication  |      Slave      |
|    DB:pt_test   +-------------->    DB:pt_test   |
| TABLE:heartbeat |               | TABLE:heartbeat |
+--------^--------+               +--------+--------+
         |                                 |
(1)update|                                 |(3)read
    row  |                                 |   row
timestamp|                                 |timestamp
         |   +---------------------------+ |
         |   |   pt-heartbeat server     | |
         |   +------------+--------------+ |
         +---+  --update  |  --monitor <---+
             +------------+--------------+

(4) now_sys_time - timestamp_from_slave = seconds behind master
```

> 1. pt-heartbeat 可以运行在独立的服务器上，或者直接运行在Master / Slave上，只要运行在 `同一台服务器` 上（*即 update 和 monitor 的时间源一致，都是本服务器的系统时间*），就能确保得到的主从落后的差值是准确的。
> 2. 如果 `--update` 运行在Master上， `--monitor` 运行在Slave上，则要确保Master和Slave的时间一致（即指向同一个NTP时钟源服务器）

`pt-heartbeat` 的原理：**在 Master 节点上创建一个** `带有时间戳字段的表` **， `pt-heartbeat --update` 每隔一段时间写入时间戳，通过主从复制，将表中的内容（*带有时间戳的记录*）复制到Slave节点，通过运行 `pt-heartbeat --monitor` 的服务器的本地时间戳 减去 复制到Slave节点的记录的时间戳，就可知道Slave落后Master的时间**

由于我们使用的是5.7.11版本，已经没有test库了，所以我们先创建一个库，供 `pt-heartbeat` 使用

```
--
-- Master端
--
mysql> create database pt_test;
Query OK, 1 row affected (0.02 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

> 测试过程中，发现 `CentOS 7.X` 在运行 `pt-heartbeat` 后，仅可以创建 `pt_test.heartbeat` 表，但是无法往表里面写入记录，且没有任何错误信息(*使用2.2.17二进制包和rpm包均不可以*)，目前原因未知；
>
> 现在使用一台 `CentOS 6.5` 进行 `pt-heartbeat`的运行
> *`pt_heartbeat` 可以运行在任何一台服务器上，仅需要保证可以 `连接到主从数据库` 即可*

```
##
## 安装 pt-heartbeat 的 CentOS 6.5
##
# 安装依赖包
[root@Master bin]> yum install perl-DBD-MySQL

# 启动pt-heartbeat
[root@Master bin]> ./pt-heartbeat -D pt_test --update -u root -p 123 -h 172.18.14.70  -P 3306 --create-table  --daemonize
# -D 指定一个数据库，这里是我们创建的测试数据库
# --update
# -h -P  Master端的IP和端口
# --create-table 第一次创建的时候需要创建表，后续使用不要使用该参数了
# --daemonize 以后端进程的方式运行 （可选）
```

运行 `pt-heartbeat` 后，实际上是在创建的数据库 `pt_test` 中创建了表后，往表里面填充时间戳

```
--
-- Master
--
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| burn_test          |
| mysql              |
| performance_schema |
| pt_test            |
| sys                |
+--------------------+
6 rows in set (0.00 sec)

mysql> use pt_test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------+
| Tables_in_pt_test |
+------------------+
| heartbeat         |
+------------------+
1 row in set (0.00 sec)

mysql> desc heartbeat;
+-----------------------+---------------------+------+-----+---------+-------+
| Field                 | Type                | Null | Key | Default | Extra |
+-----------------------+---------------------+------+-----+---------+-------+
| ts                    | varchar(26)         | NO   |     | NULL    |       |  -- 时间戳
| server_id             | int(10) unsigned    | NO   | PRI | NULL    |       |  -- 服务器的Server-id
| file                  | varchar(255)        | YES  |     | NULL    |       |
| position              | bigint(20) unsigned | YES  |     | NULL    |       |
| relay_master_log_file | varchar(255)        | YES  |     | NULL    |       |
| exec_master_log_pos   | bigint(20) unsigned | YES  |     | NULL    |       |
+-----------------------+---------------------+------+-----+---------+-------+
6 rows in set (0.00 sec)

mysql> select * from pt_test.heartbeat\G
*************************** 1. row ***************************
                  ts: 2016-04-01T14:47:00.002390
           server_id: 100
                file: bin.000013
            position: 15753
relay_master_log_file: NULL
  exec_master_log_pos: NULL
1 row in set (0.00 sec)


--
-- Slave 端
--
mysql> select @@hostname;
+------------+
| @@hostname |
+------------+
| Slave1     |
+------------+
1 row in set (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| burn_test          |
| mysql              |
| performance_schema |
| pt_test            |  -- 同步过来的pt_test
| sys                |
+--------------------+
6 rows in set (0.01 sec)
```

```
##
## 安装 pt-heartbeat 的 CentOS 6.5
##
# Slave的IP地址是 172.18.14.71
[root@MyServer bin]> ./pt-heartbeat  -D pt_test --monitor -u root -p 123 -h 172.18.14.71 -P 3306
0.00s [  0.00s,  0.00s,  0.00s ]
0.00s [  0.00s,  0.00s,  0.00s ]
0.00s [  0.00s,  0.00s,  0.00s ]
0.00s [  0.00s,  0.00s,  0.00s ]
0.00s [  0.00s,  0.00s,  0.00s ]

# --monitor 从Slave的pt_test.heartbeat中找到Master同步过来的记录，
#           然后和本地系统时间 做差值计算，得到落后时间
```

二. loss less semi-sync replication

之前测试的复制，都是 异步复制 ， Master并不关心数据是否被Slave节点所获得 ，所以复制效率很高，但是数据有可能会丢失。

从 MySQL5.5 开始，MySQL推出了 **semi-sync replication**（半同步复制）

- **至少有一个**Slave节点收到binlog后再返回（ **IO**线程接收到即可 ）
- 减少数据丢失风险
- **不能完全避免数据丢失**
- **超时后，切换回异步复制**

从 MySQL5.7 开始，MySQL推出了 **lossless semi-sync replication**（无损复制）

- 二进制日志（binlog）先写远程（ **IO**线程接收到即可 ）
- 可保证数据完全不丢失

2.1. loss less / semi-sync replication插件安装

1. 手工安装

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

2. 方式二 – 写入配置文件

```
[mysqld]
plugin_dir=/usr/local/mysql/lib/plugin
plugin_load = "rpl_semi_sync_master=semisync_master.so;rpl_semi_sync_slave=semisync_slave.so"
```

上述操作 仅仅是加载了插件 ，还 未启动 对应的功能，需要配置额外的参数：

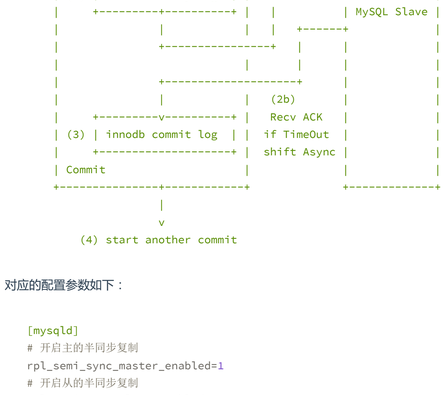
```
[mysqld]
# 等同于 rpl_semi_sync_master_enabled = 1
loose_rpl_semi_sync_master_enabled = 1
# 等同于 rpl_semi_sync_slave_enabled = 1
loose_rpl_semi_sync_slave_enabled = 1
# 超时5秒后，则切换回异步方式
loose_rpl_semi_sync_master_timeout = 5000
```

> 使用 loose_ 前缀表示如果没有加载 semi_sync 的插件，则 忽略该参数
> 当Slave在Timeout后，又追上了Master了（ IO线程 ），则会 自动切换回半同步复制

注意：半同步复制 / 无损复制 在 主从 上都要 安装插件和开启功能

2.2. semi-sync replication

semi-sync replication 称为 半同步复制 ，在一个事物 提交（commit） 的过程时，**在 InnoDB 层的 commit log 步骤后**，Master节点需要收到 至少一个 Slave节点回复的 ACK（表示 收到了binlog ）后，才能继续下一个事物；
如果在一定时间内（Timeout）内 没有收到ACK ，则 切换为异步模式 ，具体流程如下：



对应的配置参数如下：

```
[mysqld]
# 开启主的半同步复制
rpl_semi_sync_master_enabled=1
# 开启从的半同步复制
rpl_semi_sync_slave_enabled=1
# 超时1秒，切回异步
rpl_semi_sync_master_timeout=1000
# 至少收到 1 个 slave发回的ack
rpl_semi_sync_master_wait_for_slave_count=1
```

2.3. loss less semi-sync replication

loss less semi-sync replication 称为 无损复制 ，在一个事物 提交（commit） 的过程时，**在 MySQL 层的 write binlog 步骤后**，Master节点需要收到 至少一个 Slave节点回复的 ACK（表示 收到了binlog ）后，才能继续下一个事物；
如果在一定时间内（Timeout）内 没有收到ACK ，则 切换为异步模式 ，具体流程如下：

对应的配置参数如下：

```
[mysqld]
# 开启主的半同步复制
rpl_semi_sync_master_enabled=1
# 开启从的半同步复制
rpl_semi_sync_slave_enabled=1
# 超时1秒，切回异步
rpl_semi_sync_master_timeout=1000

[mysqld57]
# 控制 半同步复制 还是 无损复制  的参数
# - AFTER_SYNC 表示的是无损复制：（5.7 默认）
# - AFTER_COMMIT 表示的是半同步复制；
rpl_semi_sync_master_wait_point=AFTER_SYNC
# 至少收到 1 个 slave发回的ack
rpl_semi_sync_master_wait_for_slave_count=1
```

2.4. 两种复制方式的对比

1. **等待ACK的 时间点 不同**
   - semi-sync replication（半同步复制）在 InnoDB层的Commit Log后（第三步），等待ACK
   - loss less semi-sync replication（无损复制）在 MySQL层的Write binlog后（第二步），等待ACK
2. **主机宕机后主从数据一致性的不同**
   假设主从复制时产生异常（*比如 Master 宕机了*），Master的binlog还没有传递到Slave上，此时两种复制方式，在主从数据一致性上的表现是不一样的（ 其实都是主有从没有 ）。
   - semi-sync replication（半同步复制）在commit完成后，才传输binlog，意味着在Master节点上，这个 刚刚提交的事物对数据库的修改， 对其他事物是可见的（即在**Master**上该事物已经提交了） ，假如此时Master宕机了，且发生 主从切换 ，此时的 Slave提升为New Master ，但是此时的 New Master 上是 没有 之前提交的事物的内容的，这样就产生了主从数据的不一致。
     - 对App而言，之前读取到的内容，现在读取不到了；
   - loss less semi-sync replication（无损复制）在write binlog完成后，就传输binlog， 但还没有去写commit log，意味着当前这个事物对数据库的修改， 其他事物也是不可见的（即在**Master**上还没有提交） ，假如此时Master宕机了，且发生了 主从切换 ，此时的 Slave提升为New Master ，由于Master上对该事物还没有提交，且此时的 New Master 上同样也没有该事物的内容，此时主从的数据是一致的。
     - 如果此时Slave提升为New Master，且原来的Master又恢复了，需要让原来的Master不要提交宕机前的那个事物。
     - 使用Flashback，人工介入，进行人工回滚，因为宕机的那一刻提交的事物，用户是不知道是否成功的，我们可以让他成功，也可以让他回滚。

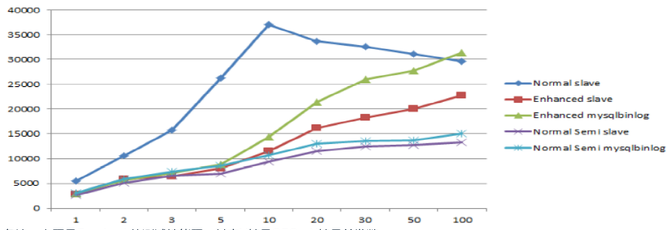
   > **举例来说，记录row=1修改为row=2**
   >
   > 1. semi-sync replication（半同步复制）模式下，Master上的row=1改成row=2，且commit成功， 此时其他事物是可以读到row=2的 ；若此时binlog还没来得及传给Slave，Master就宕机了，那在Slave上记录还是row=1； 发生主从切换后，App读取到的内容是前后是 不一致的 ；
   > 2. loss less semi-sync replication（无损复制）模式下，Master上的row=1改成row=2，但是还没有commit完成（仅到第二步）， 此时其他事物读取到的记录仍为row=1 ；若此时binlog还没来得及传给Slave，Master就宕机了，但是在Slave上记录也还是row=1； 发生主从切换后，App读取到的内容前后是 一致的 ；
3. 当 主机恢复 后，且主从还未切换，则在两种复制模式下，主从的数据都是 最终一致 的（配置是crash_safe的）

> 在数据一致性要求较高的场合，比如金融行业（ *可靠性要求大于可用性需求* ），则可以把参数 rpl_semi_sync_master_timeout 设置的很大，不让他切成异步；此时Master节点会hang住，然后需要人工介入。

2.5. 查看无损 / 半同步复制状态

```
mysql> show global status like "%rpl%";
+--------------------------------------------+-------+
| Variable_name                              | Value |
+--------------------------------------------+-------+
| Rpl_semi_sync_master_clients               | 0     |
| Rpl_semi_sync_master_net_avg_wait_time     | 0     |
| Rpl_semi_sync_master_net_wait_time         | 0     |
| Rpl_semi_sync_master_net_waits             | 0     |
| Rpl_semi_sync_master_no_times              | 0     |
| Rpl_semi_sync_master_no_tx                 | 0     |
| Rpl_semi_sync_master_status                | ON    | -- status 正常
| Rpl_semi_sync_master_timefunc_failures     | 0     |
| Rpl_semi_sync_master_tx_avg_wait_time      | 0     |
| Rpl_semi_sync_master_tx_wait_time          | 0     |
| Rpl_semi_sync_master_tx_waits              | 0     |
| Rpl_semi_sync_master_wait_pos_backtraverse | 0     |
| Rpl_semi_sync_master_wait_sessions         | 0     |
| Rpl_semi_sync_master_yes_tx                | 0     |
| Rpl_semi_sync_slave_status                 | OFF   |
+--------------------------------------------+-------+
15 rows in set (0.00 sec)
```

2.6. 两种复制方式的性能



备注：上图是Facebook的测试性能图；其中Y轴是QPS，X轴是并发数

- 蓝色的 `Normal Slave` 是 异步复制
  - 性能很好，但是随着并发数的增长，性能有所下降
- 绿色的 `Enhanced mysqlbinlog` 是 无损复制
  - 随着并发数的增长，性能几乎是线性增长的，在高并发下，性能会优于异步复制
- 紫色的 `Normal Semi Slave` 是 半同步复制
  - 性能较低

无损复制性能优于半同步复制的原因

1. 就等待ACK回包问题上，其实两种复制的开销是一样的，没有区别，都是网络的等待开销。
2. 无损复制由于在 `write binlog`（*commit 的第二步*）后，需要等待ACK，后续的事物无法提交，这样就 堆积 了很多需要落盘的事物（*半同步复制由于已经提交了事物，没有堆积事物的效果*），通过 组提交 机制，*一次* `fsync` 的事物变多了（*半同步复制也有组提交，只是一次 fsync 的事物数没那么多*），相当于提高了`I/O`性能。

所以**线程（*事物*）越多，效果越明显**，以至于有上图中超过异步复制的效果。（无损复制的组提交比例比原版的高3~4倍）

> *产生上述测试效果的前提*：测试用例是 `IO Bound` 的（*比如数据量有 100G，而 buffer pool 只有 10G*），且并发数足够多。

下面这两个参数不要去设置，设置了反而性能差

```
mysql> show variables like "%binlog_group%";
+-----------------------------------------+-------+
| Variable_name                           | Value |
+-----------------------------------------+-------+
| binlog_group_commit_sync_delay          | 0     |
| binlog_group_commit_sync_no_delay_count | 0     | -- 等待一组里面有多少事物我才提交
+-----------------------------------------+-------+
2 rows in set (0.00 sec)

mysql> show variables like "%binlog_max%";
+-------------------------------+-------+
| Variable_name                 | Value |
+-------------------------------+-------+
| binlog_max_flush_queue_time | 0     | -- 等待多少时间后才进行组提交
+-------------------------------+-------+
1 row in set (0.00 sec)
```

2.7. rpl_semi_sync_master_wait_for_slave_count

该参数控制Master在收到 多少个 Slave的 `ACK` 后，才可以继续commit。配置多个ACK和配置一个ACK的效果是类似的，因为他们是 并行执行 的（*理论上来说不会有两倍的等待时间*），取决于最慢的那个。

三. 并行复制（Multi-Threaded Slave）

姜老师的博客 – MYSQL 5.7 并行复制实现原理与调优

3.1. 介绍

在官方文档中，并行复制的叫法为 `Multi-Threaded Slave`（MTS）
MySQL的并行复制基于组提交： 一个组提交中的事务都是可以并行执行的 ，因为既然处于组提交中，这意味着事务之间没有冲突（不会去更新同一行数据），否则不可能在同一个组里面。

Slave上开启并行复制，需要在配置文件中增加以下参数：

```
[mysqld]
# DATABASE -- 基于库级别的并行复制，如果只有一个库，就还是串行（为了兼容5.6）。
# LOGICAL_CLOCK -- 逻辑时钟，主上怎么并行执行的，
#                  从上也是怎么并行回放的。
slave-parallel-type=LOGICAL_CLOCK
# 并行复制的线程数，一般设置为一个组内提交的事物数，线上设置为32足够了
slave-parallel-workers=4
# Slave上commit的顺序保持一致，必须为1，否则可能会有GAP锁产生
slave_preserve_commit_order=1
```

slave_preserve_commit_order的说明

3.2. 动态调整复制线程数

配置并行复制后，Slave节点上的效果如下，可以看到4个 `Coordinator` 线程

```
mysql> show processlist;
+----+-------------+-----------+------+---------+-------+-----------------------------------------------------------------+------------------+
| Id | User        | Host      | db   | Command | Time  | State                                                           | Info             |
+----+-------------+-----------+------+---------+-------+-----------------------------------------------------------------+------------------+
|  1 | system user |           | NULL | Connect | 26277 | Waiting for master to send event                                | NULL             |
|  2 | system user |           | NULL | Connect | 26217 | Slave has read all relay log; waiting for more updates          | NULL             |
|  4 | system user |           | NULL | Connect | 26277 | Waiting for an event from Coordinator                           | NULL             |
|  5 | system user |           | NULL | Connect | 26277 | Waiting for an event from Coordinator                           | NULL             |
|  6 | system user |           | NULL | Connect | 26277 | Waiting for an event from Coordinator                           | NULL             |
|  7 | system user |           | NULL | Connect | 26277 | Waiting for an event from Coordinator                           | NULL             |
| 10 | root        | localhost | NULL | Query   |     0 | starting                                                        | show processlist |
+----+-------------+-----------+------+---------+-------+-----------------------------------------------------------------+------------------+
7 rows in set (0.00 sec)
```

动态调整方式如下：

```
mysql> set global slave_parallel_workers=8;
Query OK, 0 rows affected (0.00 sec)

mysql> stop slave;  -- 一定要重启一下slave才能有效
Query OK, 0 rows affected (0.01 sec)

mysql> start slave;
Query OK, 0 rows affected (0.12 sec)

mysql> show processlist;
+----+-------------+-----------+------+---------+------+-----------------------------------------------------------------+------------------+
| Id | User        | Host      | db   | Command | Time | State                                                           | Info             |
+----+-------------+-----------+------+---------+------+-----------------------------------------------------------------+------------------+
| 10 | root        | localhost | NULL | Query   |    0 | starting                                                        | show processlist |
| 11 | system user |           | NULL | Connect |    6 | Waiting for master to send event                                | NULL             |
| 12 | system user |           | NULL | Connect |    5 | Slave has read all relay log; waiting for more updates          | NULL             |
| 13 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
| 14 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
| 15 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
| 16 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
| 17 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
| 18 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
| 19 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
| 20 | system user |           | NULL | Connect |    6 | Waiting for an event from Coordinator                           | NULL             |
+----+-------------+-----------+------+---------+------+-----------------------------------------------------------------+------------------+
11 rows in set (0.00 sec)
```

> **特别注意：这里的 并行复制 指的是 `SQL Thread`（回放线程），而 非`IO Thread`（IO线程）**
> `Waiting for master to send event` 这个 `State` 在 `show processlist` 中只有一个，即只有一个 `IO Thread`

线上环境可以配置成两台Slave做无损复制（保证数据不丢），其他的Slave做异步复制（配置为只读，用于负载均衡），都指向同一台Master。