阿里云RDS for MySQL的若干优化

Topic

- Double Sync Replication
- InnoDB Redo Replication
- Statement/Transaction Timeout
- InnoDB Asynchronous Optimization

Double Sync Replication

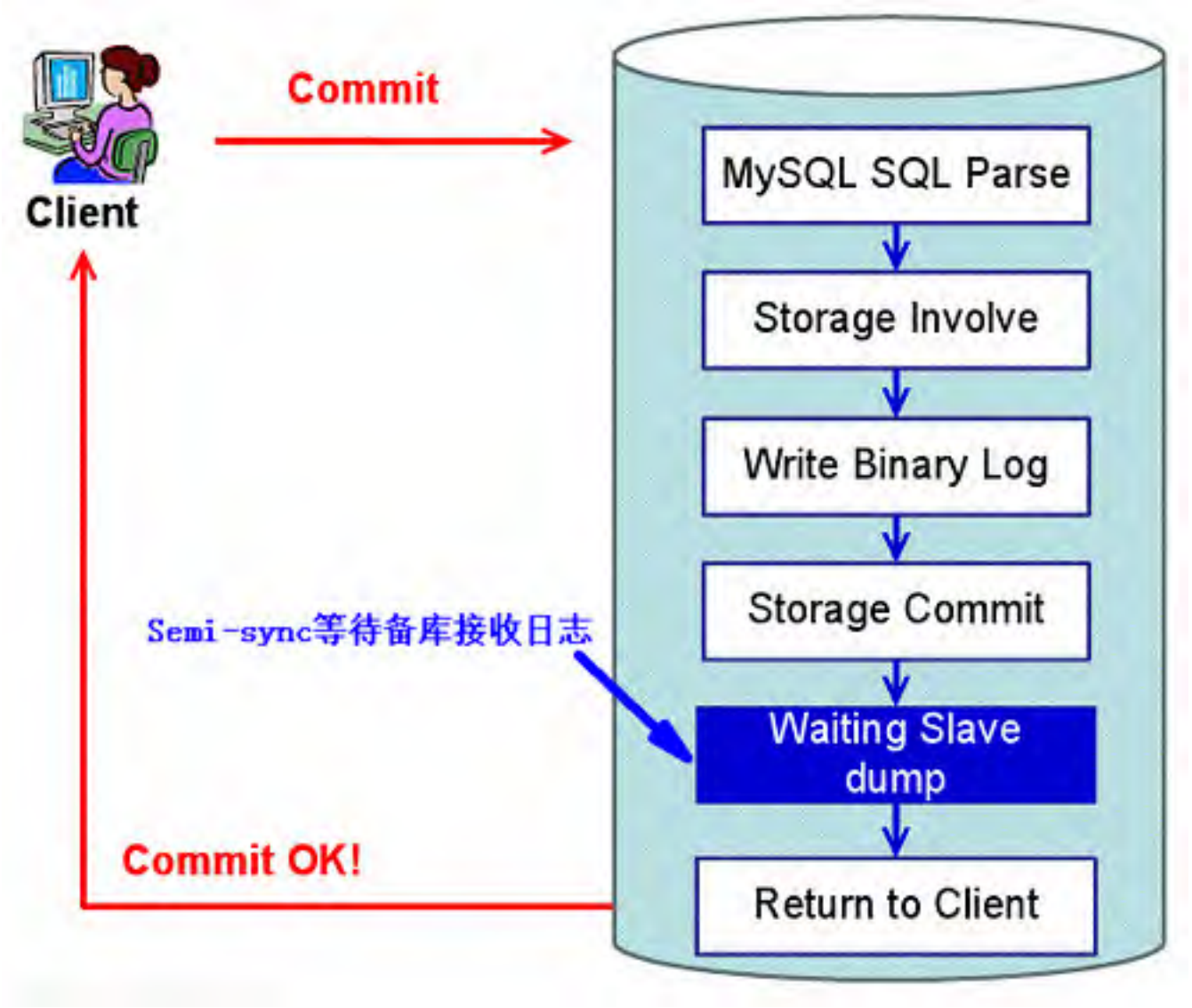
——对MySQL逻辑复制可靠性的改进

异步复制存在的缺陷

- 主库事务提交并不需要备库ACK
- 备库无法得知拖取的是否是最新的日志
- 宕机后无法利用备库本身的信息得知是否跟主库一致

- 所以，**备库无法及时得知主库的状态**

原生Semi-Sync Replication机制

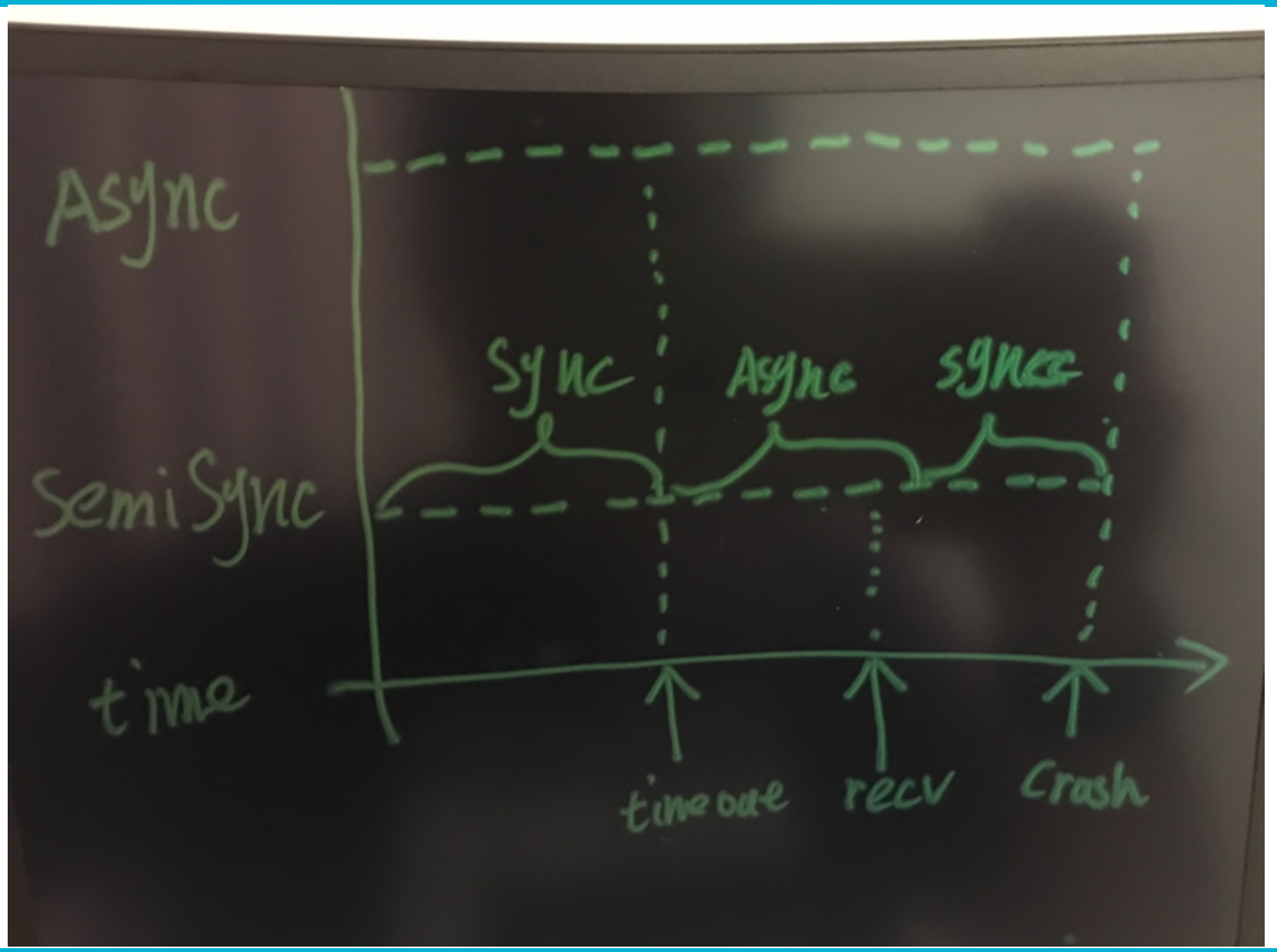


SemiSync存在的缺陷

- 主库事务提交需要备库ACK
- 网络超时后备库降级为异步复制
- 超时设太小，则经常发生超时
- 超时设太大，则经常导致主库hang
- 网络恢复后需要追赶日志，追赶期间备库状态依然不可知
- 因为无法得知宕机时备库是否跟主库是SemiSync状态
- 所以依然无法得知备库是否跟上主库

- 因此，**SemiSync并没有解决异步复制的根本缺陷**

异步复制/SemiSync存在的问题



我们要达成的目标

- 前提
  - 主机保证可用性5个9
  - 网络保证可用性5个9
  - 宕机瞬时没有发生网络超时
- 目标
  - 备库随时可以得知自己的状态（跟主库同步 或 没有跟主库同步）
  - 在确认跟主库不同步时，通知应用参与数据补偿，并且告知所缺数据范围
  - 在确认跟主库同步时，可以保证备库执行到跟主库一致状态再提供服务

- **核心：避免备库状态不可知！**

攻破SemiSync的缺点

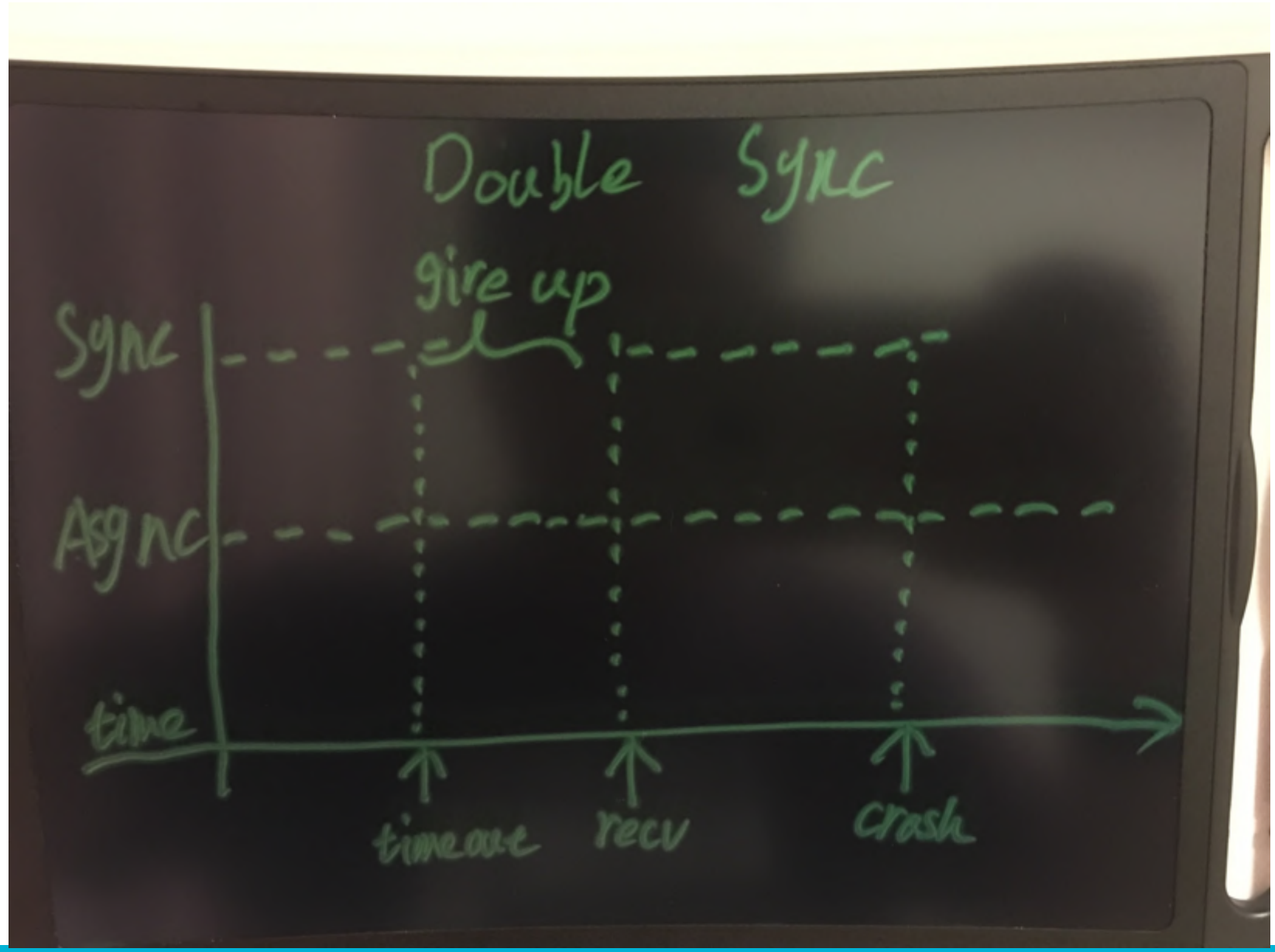
- SemiSync一旦超时断开，即使网络恢复，依然需要补偿拖取断开期间的日志
  - 如果SemiSync超时断开，网络恢复后不再补偿数据，只发最新日志，如何？
  - 只要宕机时网络正常，备库始终会知道主库最新位点
  - 依此可以判断备库是否跟主库日志有差异

- 备库如果只接收最新数据，那么中断期间的数据如何处理？
  - 异步复制可以在不影响主库提交的情况下拖取日志
  - 利用异步复制的日志可以进行完整的日志回放

结合两种复制

- 异步复制（Async_Channel）
  - 拖取连续日志，保证备库接收的日志不中断
  - 接收到日志后直接执行
- 半同步复制（Sync_Channel）
  - 拖取最新日志，保证备库始终知道最新的日志位置
  - 接收到日志后并不执行，只保留位置
- 一致性判断
  - 比较异步复制和半同步复制的日志段，可以判断备库日志可否连续接上

结合两种复制



两个通道如何做到（1）

- 多源复制可以在一个Slave上创建多个独立通道分别进行复制

- 问题1：同一个ServerID发起两个通道到Master，Master会认为是原Slave断开没有主动发起close连接，从而会踢掉先连上的通道
- 解决：可以将SemiSync通道伪装一个ServerID，避免被踢

```
root@127.0.0.1 : mysql 11:20:36> select Host,Port,Master_log_name,Master_log_pos,Channel_name from slave_master_info;
+-----------+------+------------------+---------------+--------------------------------+
| Host      | Port | Master_log_name  | Master_log_pos | Channel_name                   |
+-----------+------+------------------+---------------+--------------------------------+
| 127.0.0.1 | 13000 | master-bin.000001 |           309 |                                |
| 127.0.0.1 | 13000 | master-bin.000001 |           309 | __#alibaba_rds_sync_channel#__ |
+-----------+------+------------------+---------------+--------------------------------+
2 rows in set (0.00 sec)
```

两个通道如何做到（2）

- 问题2：一个Slave同时有一个非SemiSync通道和一个SemiSync通道，而SemiSync设置是保存在全局的
- 解决：把SemiSync改为Per-Channel的设置，将SemiSyncSlave类转移到Master_info结构体中

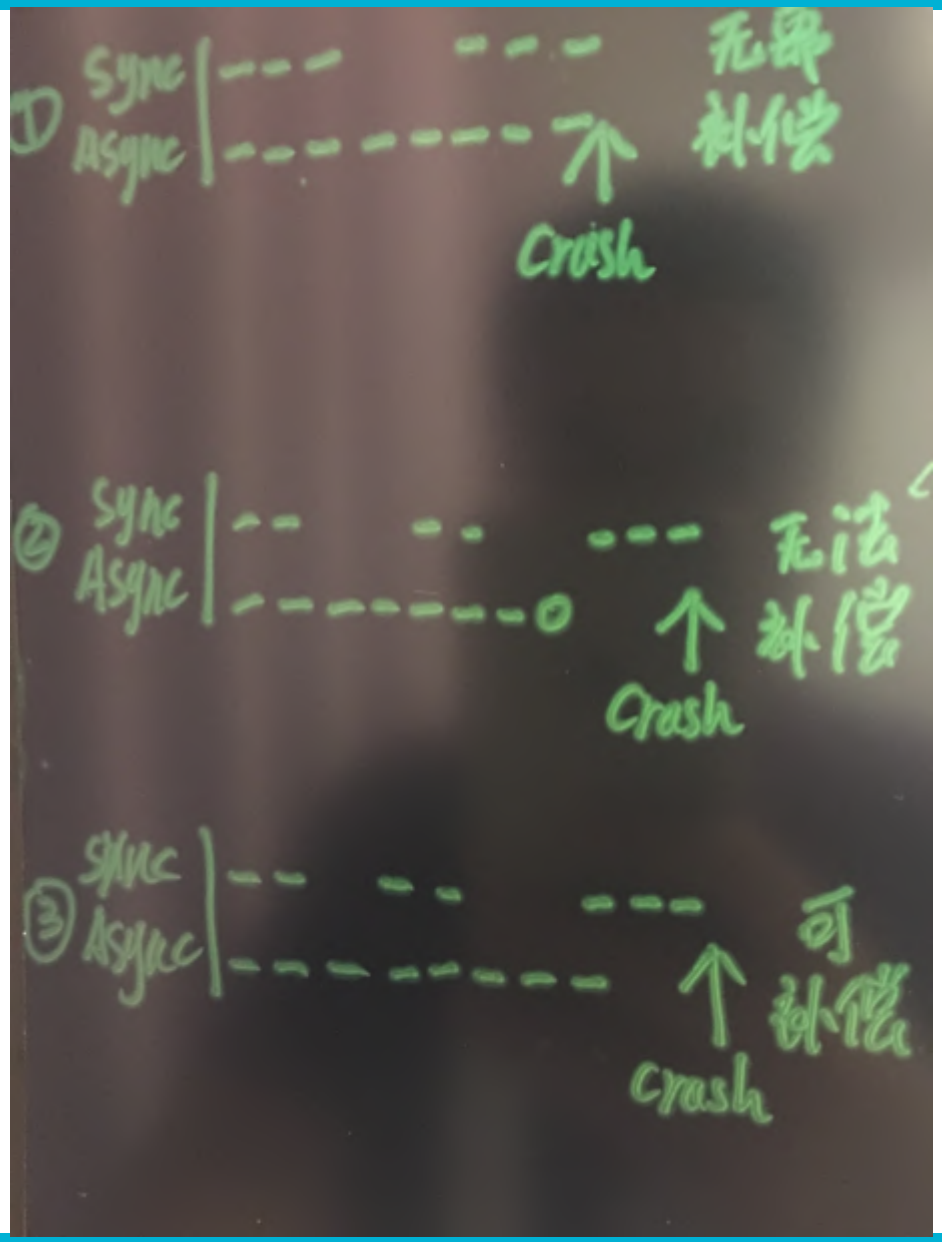
```
root@127.0.0.1 : mysql 11:27:10> show full semisync status\G
*************************** 1. row ***************************
                        Channel_Name:
          rpl_semi_sync_slave_enabled: 0
           rpl_semi_sync_slave_status: 0
         rpl_semi_sync_slave_send_ack: 0
      rpl_semi_sync_slave_trace_level: 32
     rpl_semi_sync_slave_delay_master: 0
rpl_semi_sync_slave_kill_conn_timeout: 5
*************************** 2. row ***************************
                        Channel_Name: __#alibaba_rds_sync_channel#__
          rpl_semi_sync_slave_enabled: 1
           rpl_semi_sync_slave_status: 1
         rpl_semi_sync_slave_send_ack: 0
      rpl_semi_sync_slave_trace_level: 32
     rpl_semi_sync_slave_delay_master: 0
rpl_semi_sync_slave_kill_conn_timeout: 5
2 rows in set (0.00 sec)
```

如何判断两个通道日志是否连续

- 利用两个通道收到的GTID序号作对比
- 利用两个通道收到日志的Log_file_name和Log_file_pos

- 如果半同步通道的日志起始点小于等于异步通道结束点，那么备库其实有完整的日志，反之备库无法跟上主库

如何判断两个通道日志是否连续



CASE 1: 无需补偿

- 备库两通道数据结束点完全一致

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:2-3:7-8
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-8
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File: master-bin.000001
Last_Read_Master_Log_Pos: 1496
        Is_Sync_Channel: 1
      Is_Purge_Relay_Log: 1
            Channel_Name: __#alibaba_rds_sync_channel#__
```

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-8
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-8
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File:
Last_Read_Master_Log_Pos: 0
        Is_Sync_Channel: 0
      Is_Purge_Relay_Log: 0
            Channel_Name:
```

```
root@127.0.0.1 : mysql 11:44:38> repair slave;
+---------+-----------------------------------------------------------+
| Result  | Message                                                   |
+---------+-----------------------------------------------------------+
| Success | Sync_Channel and Async_Channel are the same, needn't repair. |
+---------+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

CASE 2: 无法补偿

- 备库两通道数据合集存在断点

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-6:12-14
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-9
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File: master-bin.000001
Last_Read_Master_Log_Pos: 2681
         Is_Sync_Channel: 1
      Is_Purge_Relay_Log: 0
            Channel_Name: __#alibaba_rds_sync_channel#__
```

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-9
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-9
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File:
Last_Read_Master_Log_Pos: 0
         Is_Sync_Channel: 0
      Is_Purge_Relay_Log: 0
            Channel_Name:
```

```
root@127.0.0.1 : mysql 11:51:53> repair slave;
+--------+-----------------------------------------------+
| Result | Message                                       |
+--------+-----------------------------------------------+
| Fail   | Sync_Channel has no enough binlog, can't repair. |
+--------+-----------------------------------------------+
1 row in set (0.01 sec)
```

CASE 3: 可以补偿

- 备库两通道数据合集没有断点

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-4:8-11:16-25
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-18
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File: master-bin.000001
Last_Read_Master_Log_Pos: 3629
        Is_Sync_Channel: 1
      Is_Purge_Relay_Log: 0
            Channel_Name: __#alibaba_rds_sync_channel#__
```

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-18
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-18
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File:
Last_Read_Master_Log_Pos: 0
        Is_Sync_Channel: 0
      Is_Purge_Relay_Log: 0
            Channel_Name:
```

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-4:8-11:16-25
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-25
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File: master-bin.000001
Last_Read_Master_Log_Pos: 3629
        Is_Sync_Channel: 1
      Is_Purge_Relay_Log: 1
            Channel_Name: __#alibaba_rds_sync_channel#__
```

```
      Retrieved_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-18
       Executed_Gtid_Set: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1-25
           Auto_Position: 1
    Replicate_Rewrite_DB:
    Last_Master_Log_File:
Last_Read_Master_Log_Pos: 0
        Is_Sync_Channel: 0
      Is_Purge_Relay_Log: 0
            Channel_Name:
```

```
root@127.0.0.1 : mysql 12:05:01> repair slave;
+---------+-----------------------------------------------------+
| Result  | Message                                             |
+---------+-----------------------------------------------------+
| Success | Reparied Slave, the Master and Slave are the same! |
+---------+-----------------------------------------------------+
1 row in set (1.01 sec)
```

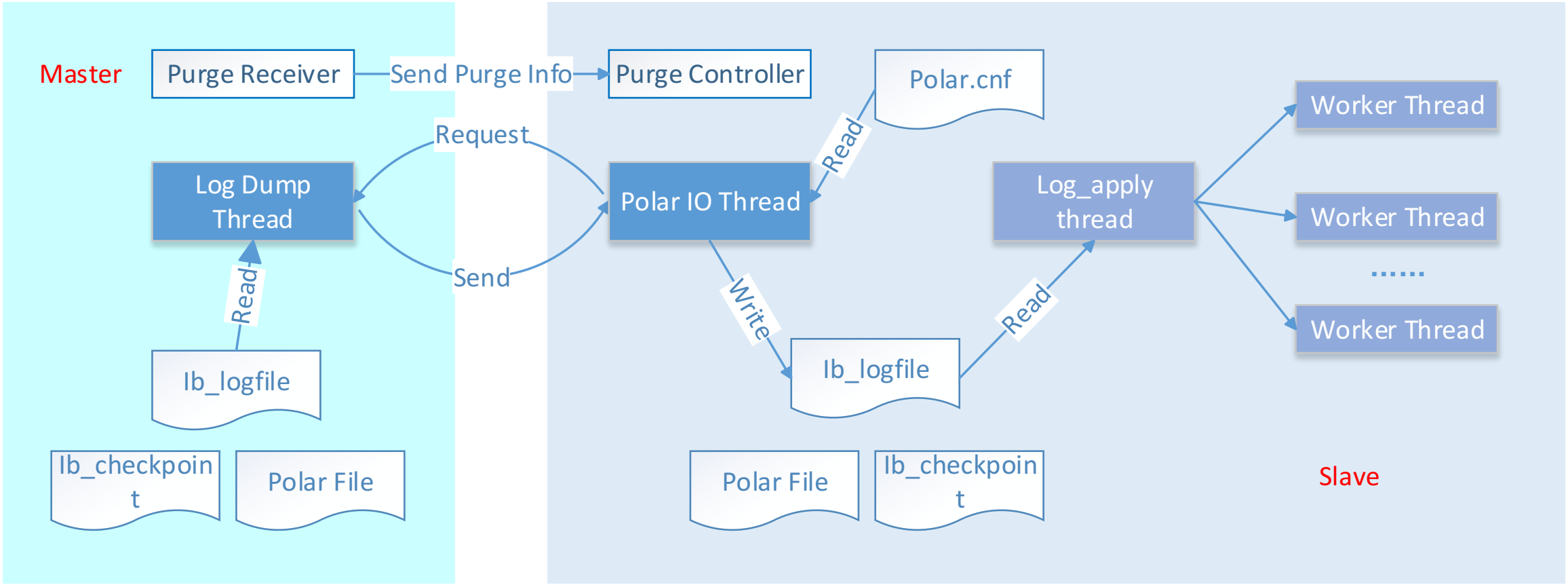
如何补偿数据

- 利用半同步通道收到的日志，在异步通道应用完日志后，启用半同步通道应用日志
- 利用GTID来过滤重复Event
- 提供 REPAIR SLAVE 命令来尝试补偿数据并返回备库状态，根据Result列的结果判断备库是否跟主库一致

InnoDB Redo Replication

——完全实现物理层的复制

复制架构


Master
Purge Receiver
Send Purge Info
Purge Controller
Request
Log Dump Thread
Send
Read
Ib_logfile
Ib_checkpoint
Polar File
Polar IO Thread
Read
Polar.cnf
Write
Ib_logfile
Read
Log_apply thread
Worker Thread
Worker Thread
.....
Worker Thread
Polar File
Ib_checkpoint
Slave

Show Polar Status

```
-----------------
Polar Role
-----------------
Slave
-----------------
Connection Configuration
-----------------
master_host(127.0.0.1), master_port(3001), master_user(repl)
Master server_uuid   (12411695503500972517, 70334969080646222O2)
Current server_uuid  (12544637746473341413, 27612697115502854 34)
-----------------
Connected slave
-----------------
0 slave(s) connected
-----------------
Thread State
-----------------
log_dump thread: Running
log_apply thread: Running
-----------------
Log State
-----------------
Lsn write up to             2723444
Lsn stored up to            2723444
Lsn parsed up to            2723444
Lsn applied up to           2723444
Lsn safe to make checkpoint 2723444
Last checkpoint at          2723417
Seconds behind master:      0
-----------------
Ibuf State
-----------------
Insert Ibuf(0), Merge(Start: 0, End: 0, Active: 0)
```

```
-----------------
Polar Role
-----------------
Master
-----------------
Connection Configuration
-----------------
master_host(NULL), master_port(0), master_user(NULL)
Master server_uuid   (12411695503500972517, 70334969080646222O2)
Current server_uuid  (12411695503500972517, 70334969080646222O2)
-----------------
Connected slave
-----------------
1 slave(s) connected
# slave server_id 3002
read up to lsn(2723525) in file(ib_logfile0, offset 2717696)
Purge limit up to trx_id(0), applied lsn(2723498)
-----------------
Thread State
-----------------
log_dump thread: Not running
log_apply thread: Stopped
-----------------
Log State
-----------------
Lsn write up to             2723525
Lsn stored up to            2722423
Lsn parsed up to            2722423
Lsn applied up to           2722423
Lsn safe to make checkpoint 2722423
Last checkpoint at          2723516
Seconds behind master:      NULL
-----------------
```

On Master

```
mysql> create table t (id int, a char(1)) engine = innodb;
Query OK, 0 rows affected (0.19 sec)

mysql> insert into t values (1, 'a');
Query OK, 1 row affected (0.02 sec)

mysql> insert into t values (2, 'b');
Query OK, 1 row affected (0.02 sec)
```

```
---------------
Log State
---------------
Lsn write up to             2757465
Lsn stored up to            2722423
Lsn parsed up to            2722423
Lsn applied up to           2722423
Lsn safe to make checkpoint 2722423
Last checkpoint at          2757465
Seconds behind master:      NULL
---------------
Ibuf State
---------------
Insert Ibuf(0), Merge(Start: 0, End: 0, Active: 0)
Shadow page: alloc(0), free(0)
---------------
Purge State
---------------
Purge view: up_limit_id(4408), low_limit_id(4408)
Max_trx_id of trx_sys: 4408
Length of global trx array: 0
Status: Normal Purge State
---------------
Log File Info
---------------
1 active ib_logfiles
The oldest log file number: 0, start_lsn: 8192
The newest log file number: 0, start_lsn: 8192
Log purge up to file number: 0
0 free files for reallocation
Lastest(Doing) checkpoint at lsn 2757465(ib_logfile0, offset 2751321)
```

On Slave

```
mysql> select * from t;
+------+------+
| id   | a    |
+------+------+
|    1 | a    |
|    2 | b    |
+------+------+
2 rows in set (0.04 sec)
```

```
---------------
Thread State
---------------
log_dump thread: Running
log_apply thread: Running
---------------
Log State
---------------
Lsn write up to             2757609
Lsn stored up to            2757609
Lsn parsed up to            2757609
Lsn applied up to           2757609
Lsn safe to make checkpoint 2757609
Last checkpoint at          2757591
Seconds behind master:      0
---------------
Ibuf State
---------------
Insert Ibuf(0), Merge(Start: 0, End: 0, Active: 0)
Shadow page: alloc(0), free(0)
---------------
Purge State
---------------
Purge view: up_limit_id(4408), low_limit_id(4408)
Max_trx_id of trx_sys: 4408
Length of global trx array: 0
Purge thread is not active on slave
---------------
Log File Info
---------------
1 active ib_logfiles
The oldest log file number: 0, start_lsn: 8192
The newest log file number: 0, start_lsn: 8192
```

Statement/Transaction Timeout

——避免语句/事务长时间占用资源

无限制执行Query的危害

- 执行时间过长的SELECT可能导致占用大量CPU/IO资源，拖慢整个服务器
- UPDATE/DELETE语句不提交，可能导致长时间持有锁资源，而且不易从PROCESSLIST中察觉

语句级超时（MAX_STATEMENT_TIME）

```
mysql> SELECT MAX_STATEMENT_TIME=1000 * FROM t;
ERROR 3006 (HY000): Query execution was interrupted, max_statement_time exceeded
mysql>
```

```
mysql> set MAX_STATEMENT_TIME =1000;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t;
ERROR 3006 (HY000): Query execution was interrupted, max_statement_time exceeded
mysql>
```

事务级超时（rds_trx_idle_timeout）

- 可区分只读事务（rds_trx_readonly_idle_timeout），读写事务分别设置（rds_trx_changes_idle_timeout），也可以统一设置。

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

mysql> set rds_trx_idle_timeout = 5;
Query OK, 0 rows affected (0.00 sec)

mysql> update t set a = 'g';
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
mysql>
```

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

mysql> update t set a = 'a';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2016-03-15 16:42:18 |
+---------------------+
1 row in set (0.00 sec)

mysql> select * from t;
ERROR 2013 (HY000): Lost connection to MySQL server during query
mysql>
```

InnoDB Asynchronous Optimization

——全异步整理InnoDB空间（From FB）

InnoDB表空间碎片整理

- 有些用户可能会使用 OPTIMIZE TABLE 或者 ALTER TABLE <table> ENGINE=InnoDB 来重建做过大量删除操作的表，但是这样会导致表的拷贝，如果临时空间不足甚至不足以进行一次 OPTIMIZE TABLE 操作。并且如果你用的是共享表空间方式，OPTIMIZE TABLE 会导致你的共享表空间文件持续增大，因为整理的索引和数据都追加在数据文件的末尾。

构造数据

```
mysql> CREATE TABLE tb_defragment (
    -> pk1 bigint(20) NOT NULL,
    -> pk2 bigint(20) NOT NULL,
    -> fd4 text,
    -> fd5 varchar(50) DEFAULT NULL,
    -> PRIMARY KEY (pk1),
    -> KEY ix1 (pk2)
    -> ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.20 sec)
```

执行：**call innodb_insert_proc(50000);**

```
mysql> SELECT table_name,
    ->        data_free / 1024 / 1024 AS data_free_MB,
    ->        table_rows
    -> FROM   information_schema.tables
    -> WHERE  engine LIKE 'InnoDB'
    ->        AND table_name LIKE '%tb_defragment%';
+---------------+--------------+------------+
| table_name    | data_free_MB | table_rows |
+---------------+--------------+------------+
| tb_defragment |   4.00000000 |      49513 |
+---------------+--------------+------------+
1 row in set (0.00 sec)
```

```
mysql> create procedure innodb_insert_proc (repeat_count int)
    -> begin
    ->   declare current_num int;
    ->   set current_num = 0;
    ->   while current_num < repeat_count do
    ->     INSERT INTO tb_defragment VALUES (current_num, 1, REPEAT('Abcdefg', 20), REPEAT('12345',5));
    ->     INSERT INTO tb_defragment VALUES (current_num+1, 2, REPEAT('HIJKLM', 20), REPEAT('67890',5));
    ->     INSERT INTO tb_defragment VALUES (current_num+2, 3, REPEAT('HIJKLM', 20), REPEAT('67890',5));
    ->     INSERT INTO tb_defragment VALUES (current_num+3, 4, REPEAT('HIJKLM', 20), REPEAT('67890',5));
    ->     set current_num = current_num + 4;
    ->   end while;
    -> end//
Query OK, 0 rows affected (0.09 sec)
```

OPTIMIZE TABLE ASYNC

mysql> delete from tb_defragment where pk2 between 2 and 4;
Query OK, 37500 rows affected (38.87 sec)

```
mysql> SELECT table_name,
    ->        data_free / 1024 / 1024 AS data_free_MB,
    ->        table_rows
    -> FROM   information_schema.tables
    -> WHERE  engine LIKE 'InnoDB'
    ->        AND table_name LIKE '%tb_defragment%';
+---------------+--------------+------------+
| table_name    | data_free_MB | table_rows |
+---------------+--------------+------------+
| tb_defragment |   4.00000000 |      11927 |
+---------------+--------------+------------+
1 row in set (0.01 sec)
```

```
mysql> OPTIMIZE TABLE tb_defragment ASYNC;
+--------------------+----------+----------+----------+
| Table              | Op       | Msg_type | Msg_text |
+--------------------+----------+----------+----------+
| test.tb_defragment | optimize | status   | OK       |
+--------------------+----------+----------+----------+
1 row in set (3.73 sec)

mysql> show status like '%innodb_def%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Innodb_defragment_compression_failures | 0     |
| Innodb_defragment_failures            | 1     |
| Innodb_defragment_count               | 5     |
+---------------------------------------+-------+
3 rows in set (0.02 sec)
```

```
mysql> SELECT stat_name,
    ->        Count(stat_value)
    -> FROM   mysql.innodb_index_stats
    -> WHERE  table_name LIKE '%tb_defragment%'
    ->        AND stat_name IN ( 'n_pages_freed', 'n_page_split', 'n_leaf_pages_defrag' )
    -> GROUP  BY stat_name;
+--------------------+-------------------+
| stat_name          | Count(stat_value) |
+--------------------+-------------------+
| n_leaf_pages_defrag |                 2 |
| n_page_split       |                 2 |
| n_pages_freed      |                 2 |
+--------------------+-------------------+
3 rows in set (0.01 sec)
```

后台合并压缩页面数

```
mysql> SELECT table_name,
    ->        index_name,
    ->        Sum(number_records),
    ->        Sum(data_size)
    -> FROM   information_schema.innodb_buffer_page
    -> WHERE  table_name LIKE '%tb_defragment%'
    ->        AND index_name IN ( 'PRIMARY', 'ix1' )
    -> GROUP  BY index_name;
+-----------------------+------------+---------------------+-----------------+
| table_name            | index_name | Sum(number_records) | Sum(data_size) |
+-----------------------+------------+---------------------+-----------------+
| `test`.`tb_defragment` | ix1        |               50071 |         1051775 |
| `test`.`tb_defragment` | PRIMARY    |               50618 |         9373624 |
+-----------------------+------------+---------------------+-----------------+
2 rows in set (2.86 sec)
```

```
mysql> SELECT table_name,
    ->        index_name,
    ->        Sum(number_records),
    ->        Sum(data_size)
    -> FROM   information_schema.innodb_buffer_page
    -> WHERE  table_name LIKE '%tb_defragment%'
    ->        AND index_name IN ( 'PRIMARY', 'ix1' )
    -> GROUP  BY index_name;
+-----------------------+------------+---------------------+-----------------+
| table_name            | index_name | Sum(number_records) | Sum(data_size) |
+-----------------------+------------+---------------------+-----------------+
| `test`.`tb_defragment` | ix1        |               41683 |          875551 |
| `test`.`tb_defragment` | PRIMARY    |               42851 |         8050407 |
+-----------------------+------------+---------------------+-----------------+
2 rows in set (4.08 sec)
```

```
mysql> SELECT table_name,
    ->        index_name,
    ->        Sum(number_records),
    ->        Sum(data_size)
    -> FROM   information_schema.innodb_buffer_page
    -> WHERE  table_name LIKE '%tb_defragment%'
    ->        AND index_name IN ( 'PRIMARY', 'ix1' )
    -> GROUP  BY index_name;
+-----------------------+------------+---------------------+-----------------+
| table_name            | index_name | Sum(number_records) | Sum(data_size) |
+-----------------------+------------+---------------------+-----------------+
| `test`.`tb_defragment` | ix1        |               28386 |          596174 |
| `test`.`tb_defragment` | PRIMARY    |               29824 |         5831744 |
+-----------------------+------------+---------------------+-----------------+
2 rows in set (2.88 sec)
```

THANKS!