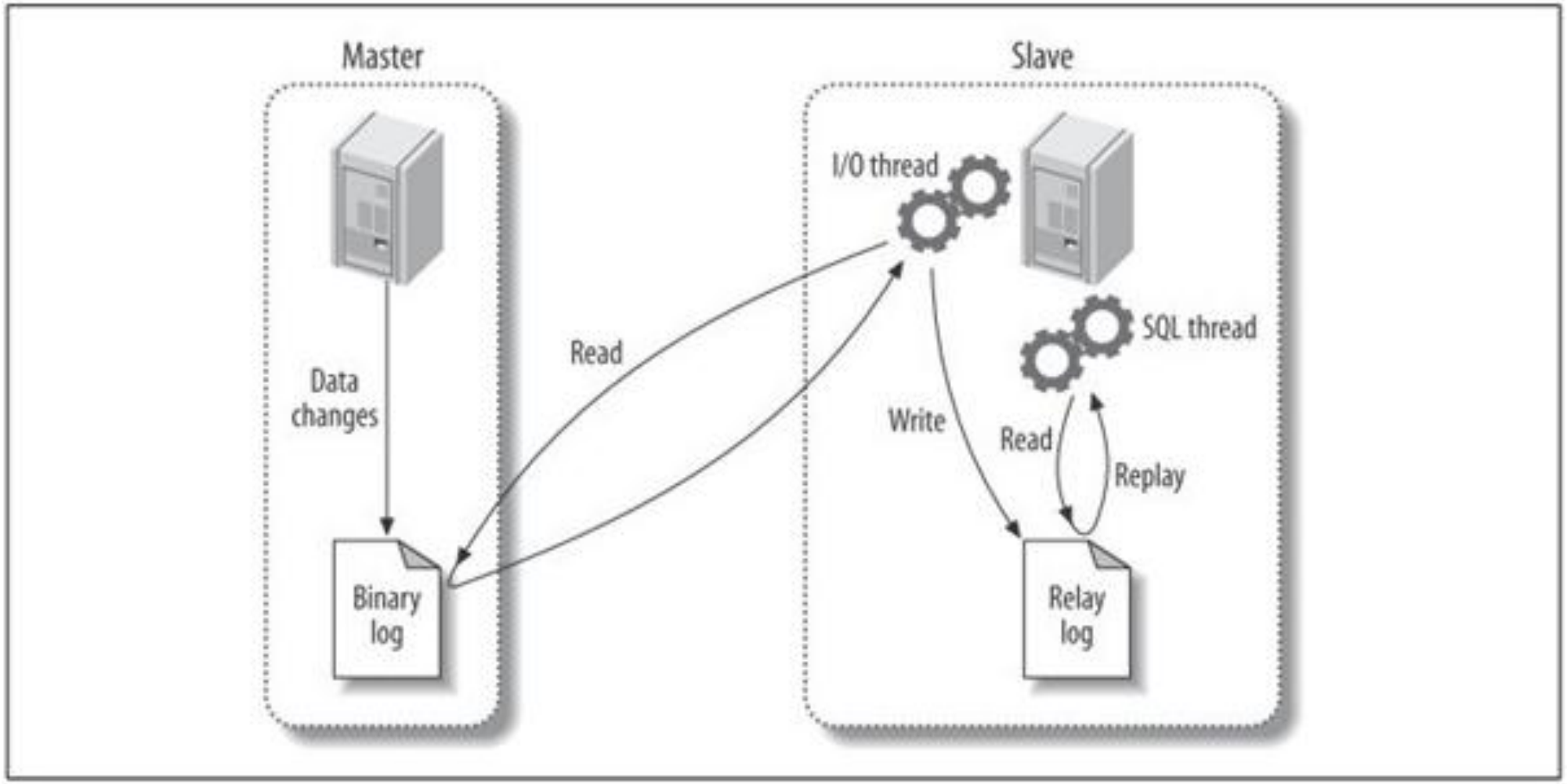
浅析 MySQL Replication

卢飞

复制的用途

- 读写分离
- 灾备/高可用
- 统计
- 备份

复制如何工作


Master
Slave
I/O thread
SQL thread
Data changes
Read
Write
Read
Replay
Binary log
Relay log

主要步骤

- Master将改变记录到二进制日志Binary log中
- Slave 将主库上的日志复制到自己的Relay log中
- Slave上SQL线程回放中继日志的内容，将其Slave上的数据与Master一致

Binary log 格式

- STATEMENT
  --记录操作的SQL语句

- ROW
  --记录操作的每一行数据的变化信息

- MIXED
  --混合模式
  -- STATEMENT--》ROW

STATEMENT

- 优点
  - 减少了binlog日志量，节约IO，提高性能

- 缺点
  - 不是所有的DML语句都能被复制
  - UUID()，FOUND_ROWS()， USER()

```
BJ_128_60 [test] [11:24:51]> insert into test1(name) select 'aaa';
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

```
#150824 11:24:54 server id 1  end_log_pos 309 CRC32 0x3c73a211  Query   thread_id=2     exec_time=0     error_code=0
use `test`/*!*/;
SET TIMESTAMP=1440386694/*!*/;
insert into test1(name) select 'aaa'
/*!*/;
# at 309
#150824 11:24:54 server id 1  end_log_pos 340 CRC32 0xe0f6a137  Xid = 3
COMMIT/*!*/;
```

ROW (推荐使用)

- 优点
  - 任何情况都可以被复制，ROW模式是最安全可靠的

- 缺点
  - 产生大量的日志，copy data 的DDL会让日志暴涨

```
BJ_128_60 [test] [11:28:38]> insert into test1(name) select 'bbb';
Query OK, 1 row affected (0.00 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

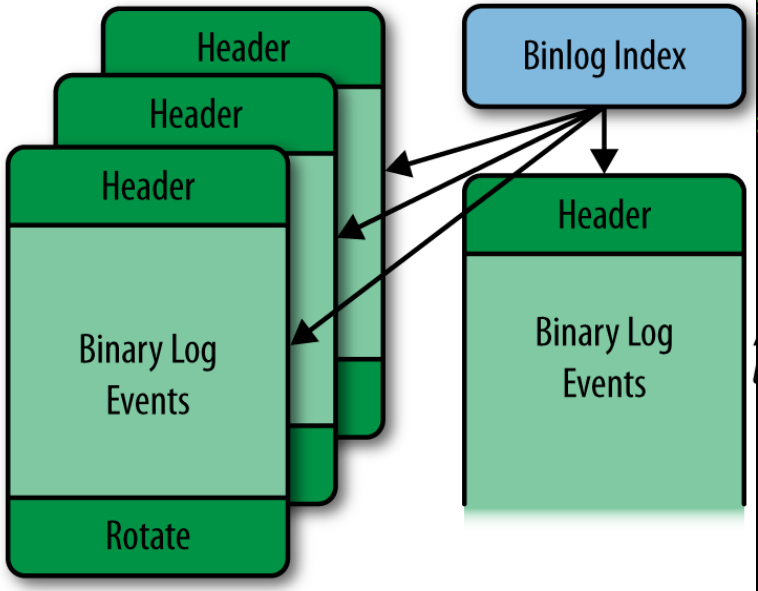
```
# at 661
#150824 11:28:44 server id 1  end_log_pos 705 CRC32 0x56f992b9  Write_rows: table id 70 flags: STMT_END_F

BINLOG '
bI/aVRMBAAAAMwAAAJUCAAAAAEYAAAAAAAEABHR1c3QABXR1c3QxAAIDDwIKAAIkue25
bI/aVR4BAAAALAAAAMECAAAAAEYAAAAAAAEAAgAC//wAAAAAA2JiYrmS+VY=
'/*!*/;
# at 705
#150824 11:28:44 server id 1  end_log_pos 736 CRC32 0x014410cd  Xid = 13
COMMIT/*!*/;
```

MIXED

- 默认会先使用STATEMENT模式保存binlog，对于STATEMENT模式无法复制的操作使用ROW模式保存binlog，MySQL会根据执行的SQL语句选择日志保存方式。

- Bug较多，不建议使用

Binlog Event

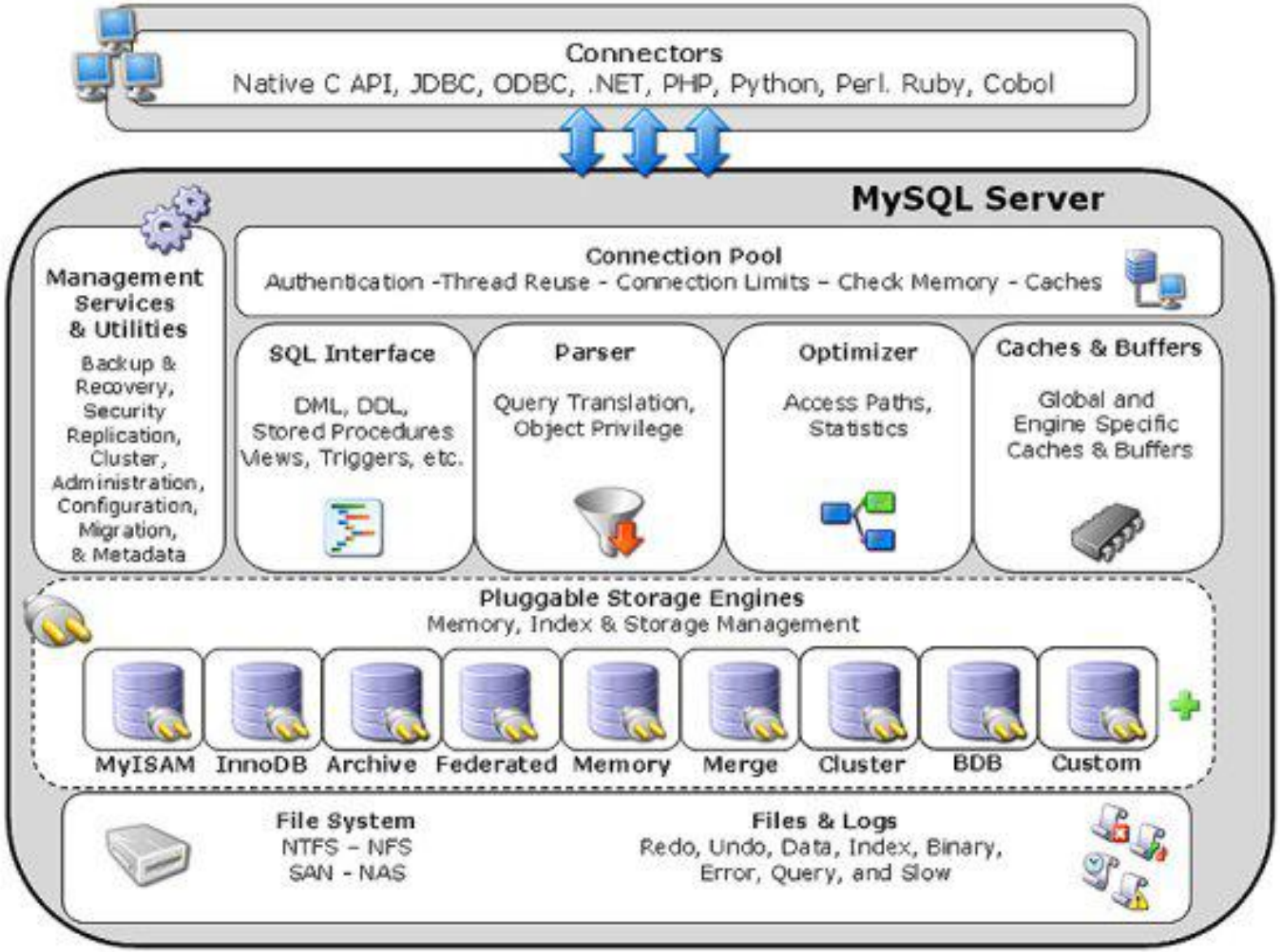



```
mysql> flush logs;
Query OK, 0 rows affected (0.00 sec)

mysql> show binlog events in 'bin.000007';
+------------+-----+----------------+-----------+-------------+-----------------------------------------------------------------------+
| Log_name   | Pos | Event_type     | Server_id | End_log_pos | Info                                                                  |
+------------+-----+----------------+-----------+-------------+-----------------------------------------------------------------------+
| bin.000007 |   4 | Format_desc    |        11 |         123 | Server ver: 5.7.9-log, Binlog ver: 4                                  |
| bin.000007 | 123 | Previous_gtids |        11 |         194 | cac25bc1-934b-11e5-bcd3-c81f66f10582:1-156                            |
| bin.000007 | 194 | Gtid           |        11 |         259 | SET @@SESSION.GTID_NEXT= 'cac25bc1-934b-11e5-bcd3-c81f66f10582:157'   |
| bin.000007 | 259 | Query          |        11 |         333 | BEGIN                                                                 |
| bin.000007 | 333 | Table_map      |        11 |         382 | table_id: 116 (mytest.t)                                              |
| bin.000007 | 382 | Delete_rows    |        11 |         424 | table_id: 116 flags: STMT_END_F                                       |
| bin.000007 | 424 | Xid            |        11 |         455 | COMMIT /* xid=255 */                                                  |
| bin.000007 | 455 | Gtid           |        11 |         520 | SET @@SESSION.GTID_NEXT= 'cac25bc1-934b-11e5-bcd3-c81f66f10582:158'   |
| bin.000007 | 520 | Query          |        11 |         594 | BEGIN                                                                 |
| bin.000007 | 594 | Table_map      |        11 |         643 | table_id: 116 (mytest.t)                                              |
| bin.000007 | 643 | Update_rows    |        11 |         695 | table_id: 116 flags: STMT_END_F                                       |
| bin.000007 | 695 | Xid            |        11 |         726 | COMMIT /* xid=262 */                                                  |
| bin.000007 | 726 | Rotate         |        11 |         767 | bin.000008;pos=4                                                      |
+------------+-----+----------------+-----------+-------------+-----------------------------------------------------------------------+
13 rows in set (0.00 sec)
```

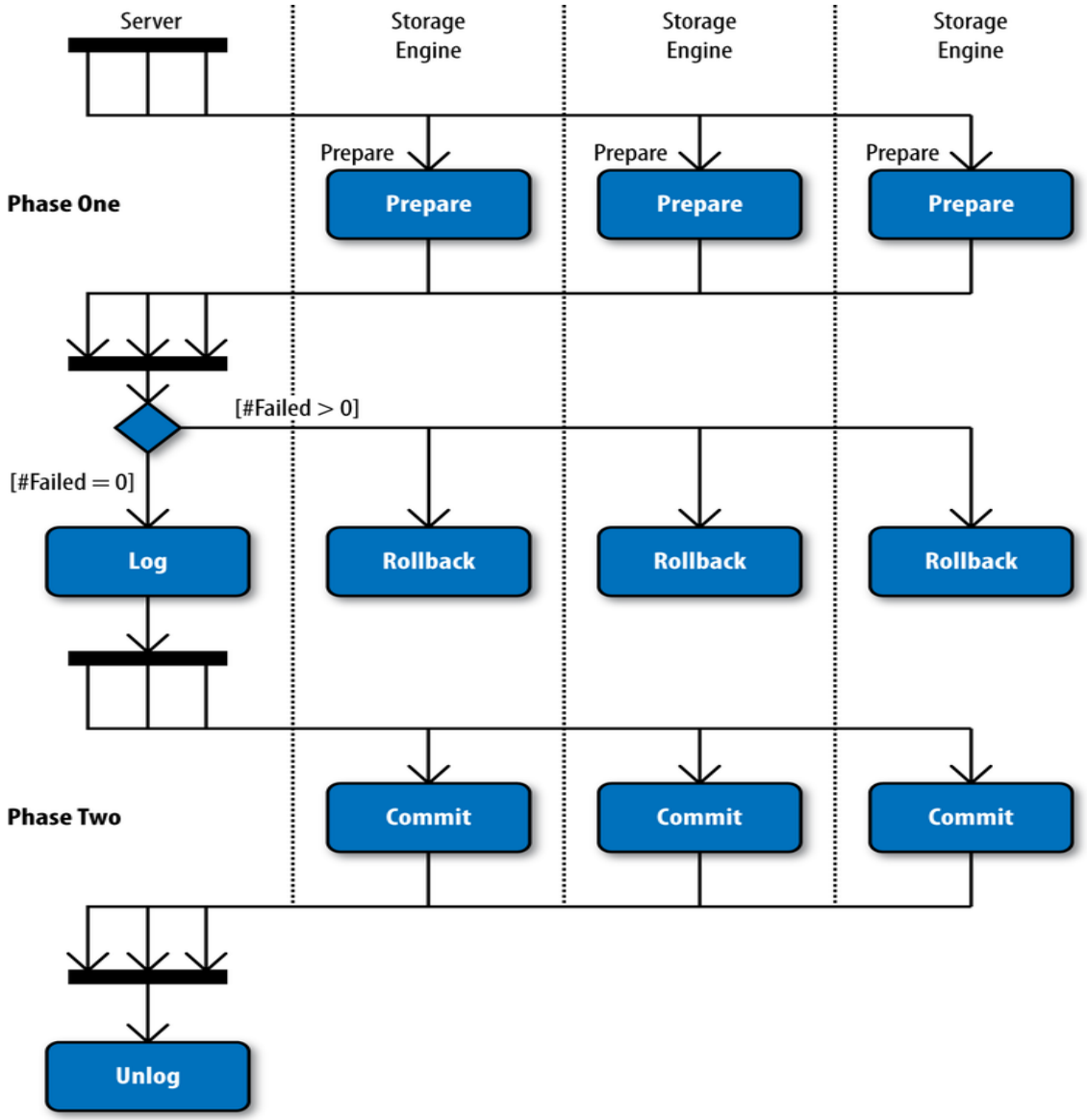
疑问

- 为什么MySQL 有二进制日志，InnoDB还有redo日志?
- 事务是如何提交的？
- 事务提交先写二进制日志还是重做日志？
- 如何保证这两部分的日志做到一致性？

体系架构


Connectors
Native C API, JDBC, ODBC, .NET, PHP, Python, Perl. Ruby, Cobol
MySQL Server
Management Services & Utilities
Backup & Recovery, Security Replication, Cluster, Administration, Configuration, Migration, & Metadata
Connection Pool
Authentication -Thread Reuse - Connection Limits - Check Memory - Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
Archive
Federated
Memory
Merge
Cluster
BDB
Custom
File System
NTFS - NFS
SAN - NAS
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query, and Slow

事务是怎样提交的



1）SQL语句已经成功执行并生成redo和undo的内存日志

2）Binary log write()--》fsync()

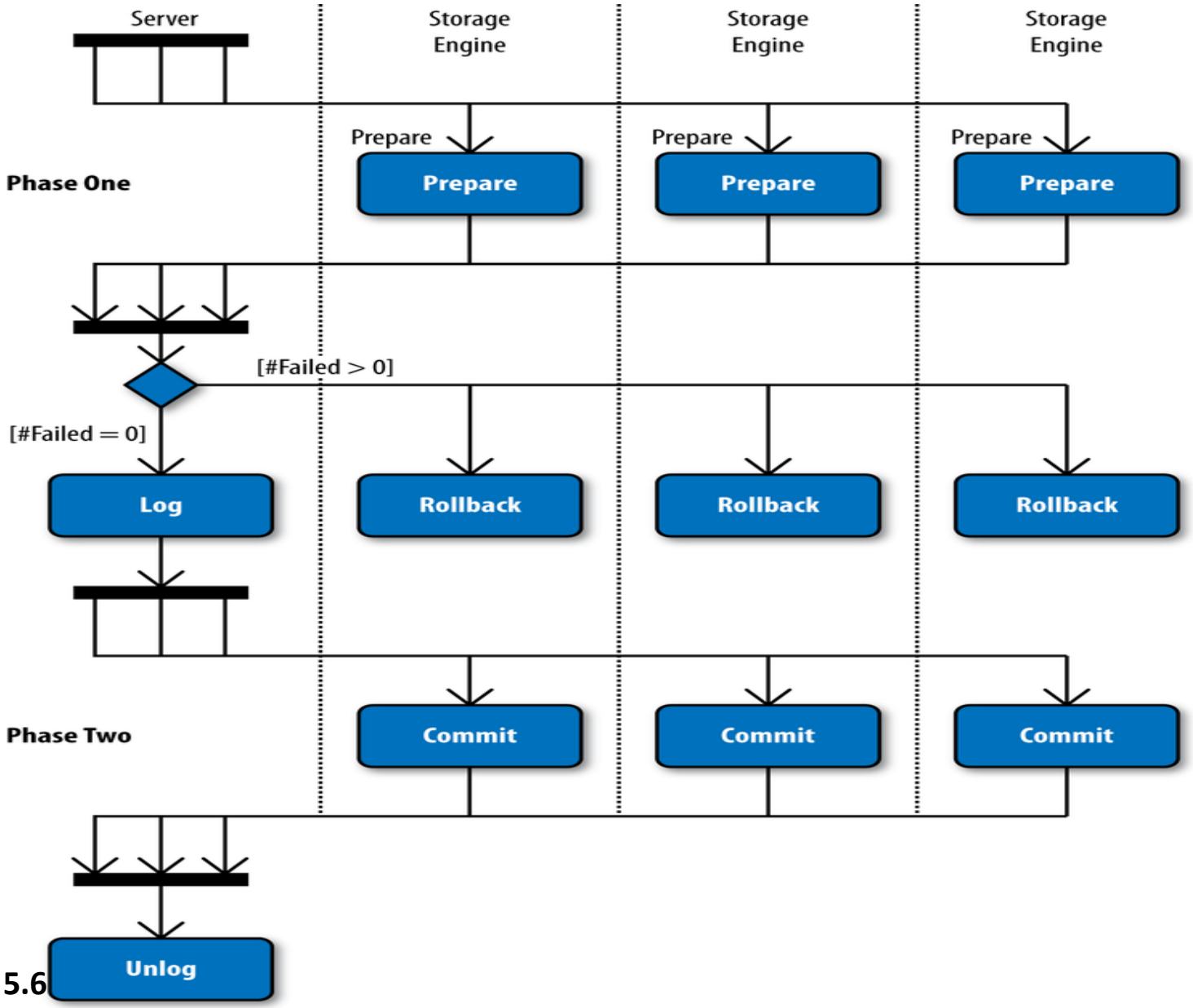
3）存储引擎内提交使undo和redo永久写入磁盘

双1保证

控制MySQL 磁盘写入策略以及数据安全性的关键参数，MySQL为了保证主库和从库的数据一致性，就必须保证Binlog和InnoDB redo日志的一致性

- **innodb_flush_log_at_trx_commit（redo）**
  - **--0** log buffer将每秒一次地写入log file中，并且flush操作同时进行
  - **--1** 每次事务提交时都会把log buffer的数据写入log file，并且flush到磁盘
  - **--2** 每次事务提交时MySQL都会把log buffer的数据写入log file，不flush
- **sync_binlog （binlog）**
  - **--0** 刷新binlog_cache中的信息到磁盘 由os决定
  - **--1** 每次事务提交刷新binlog_cache中的信息到磁盘

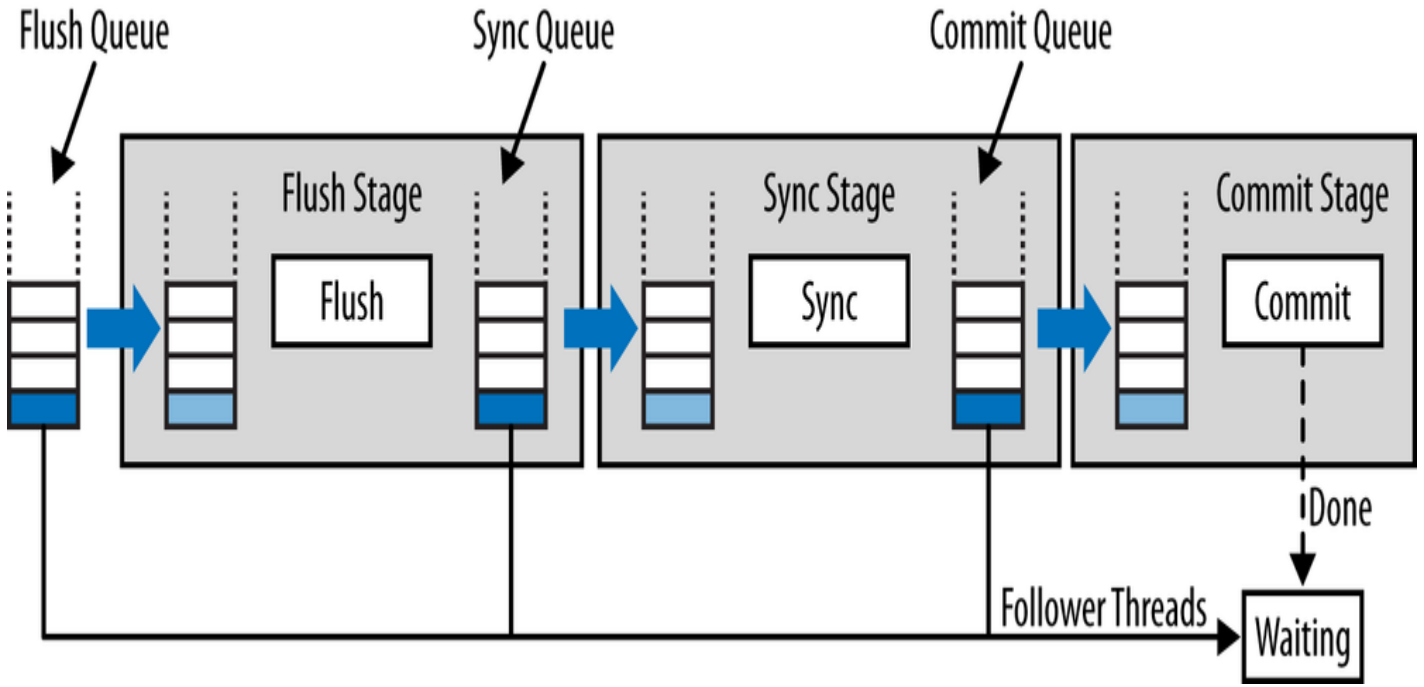
sync_binlog



- prepare_commit_mutex

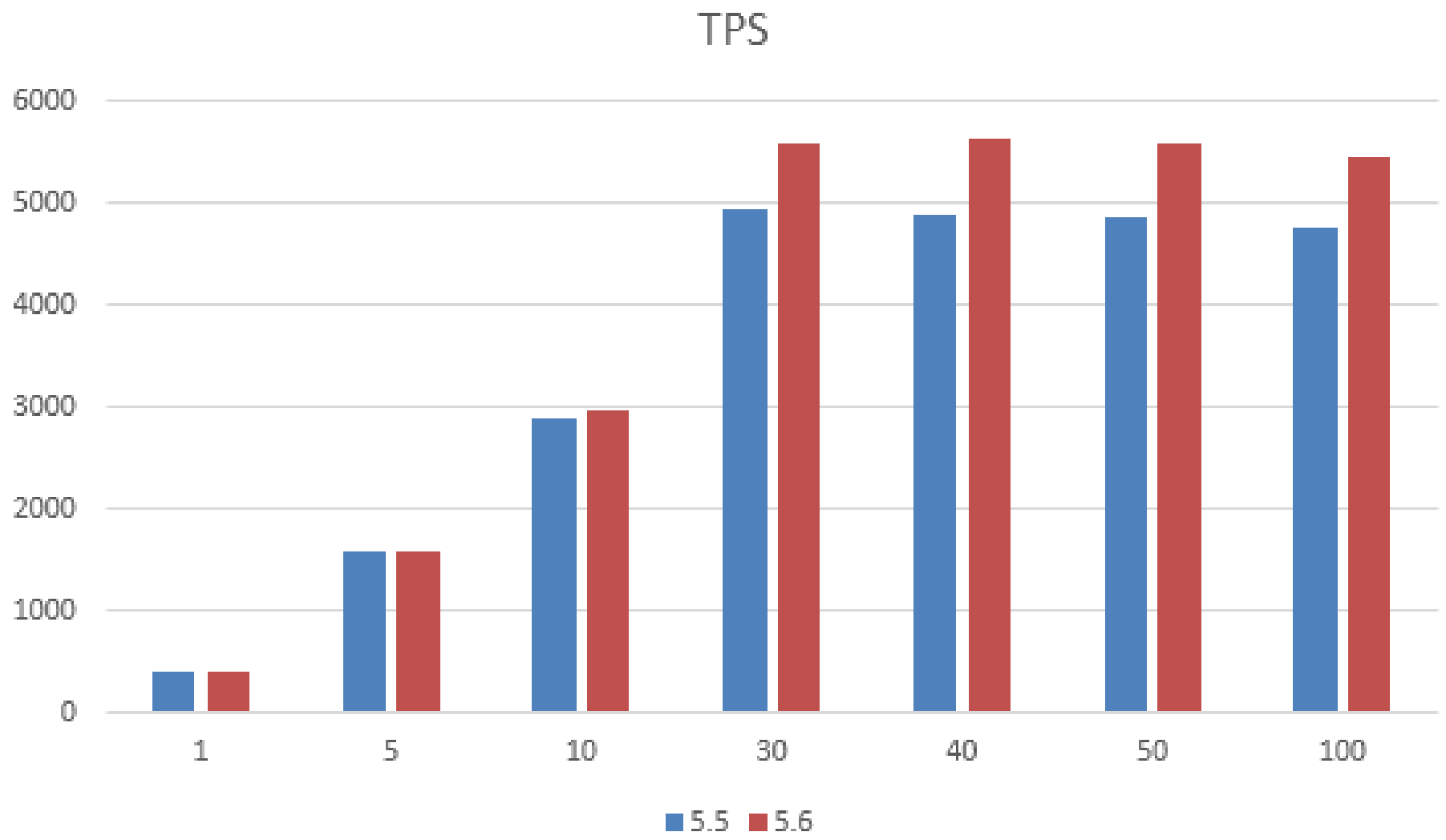
<MySQL 5.6

Binary Log Group Commit



- Flush stage： Leader线程遍历FLUSH_STAGE链表，写入binary log缓存
- Sync stage ： 将binlog缓存sync到磁盘，当sync_binlog=1时所有该队列事务的二进制日志缓存永久写入磁盘
- Commit stage: leader根据顺序让InnoDB存储引擎完成Commit

BLGC TPS性能



场景分析

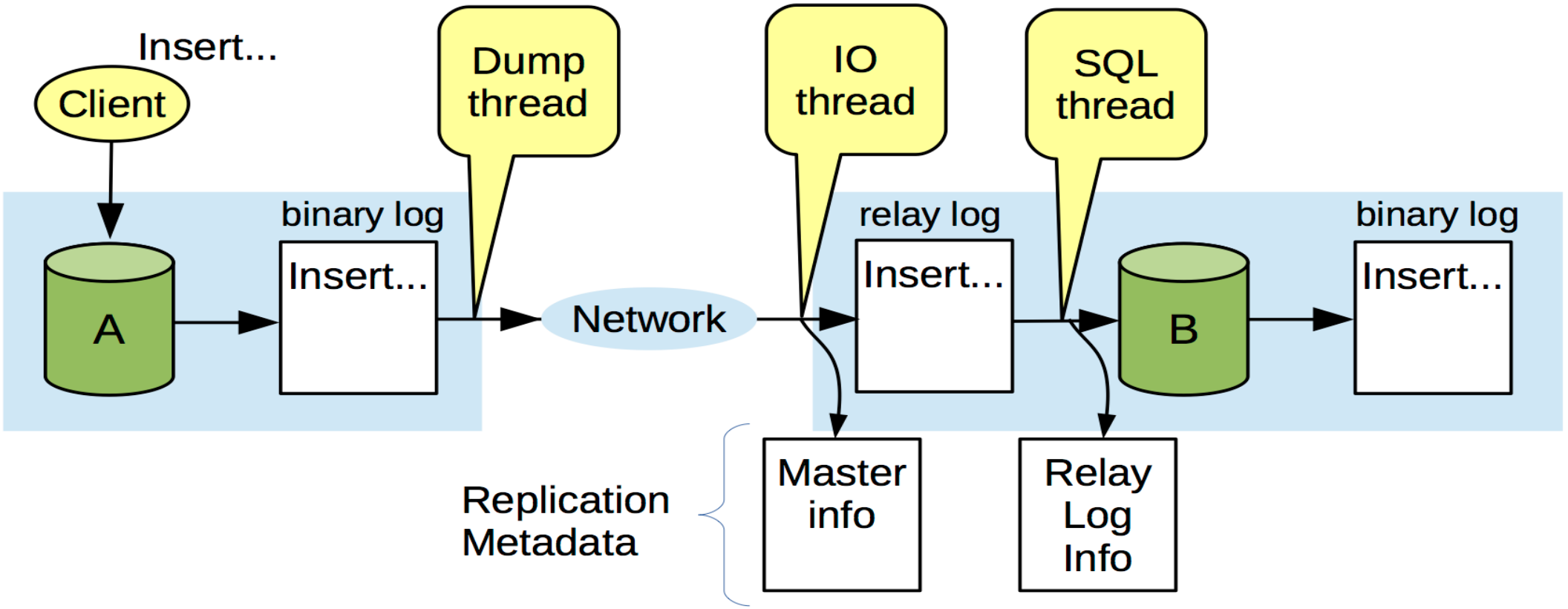
- slave 宕掉重启后，复制报1062 主键冲突，可能按以下方法，跳过这个错误

  SET GLOBAL SQL_SLAVE_SKIP_COUNTER = 1

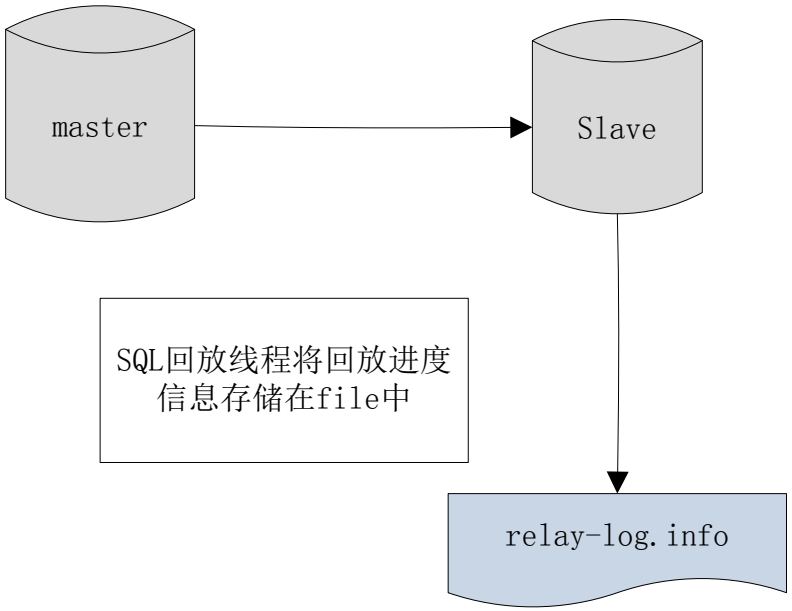
  GTID 通过空事务方式

- 为什么数据会冲突呢？

分解复制步骤


Insert...
Client
Dump thread
IO thread
SQL thread
binary log
relay log
binary log
Insert...
Insert...
Insert...
A
Network
B
Replication Metadata
Master info
Relay Log Info

SQL crash-safe




```
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| sync_relay_log_info  | 10000 |
+----------------------+-------+
```

代表每回放10000个event，写一次relay-info.log

```
[root@zzbdb mysql_3306]# cat relay-log.info
7
/home/mysql_3306/mysql-relay.000003
52604841
mysql-bin.000507
97464033
0
0
1
```

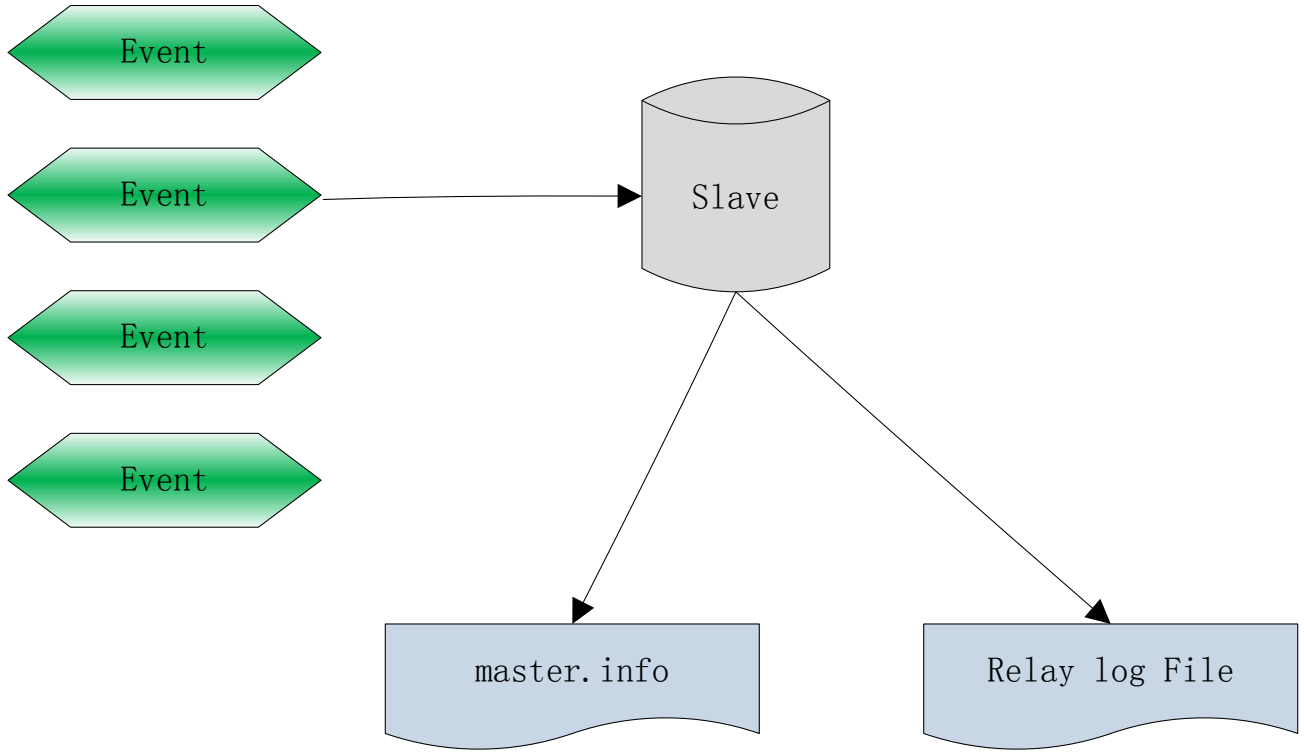
SQL thread crash-safe

START TRANSACTION;
Statement 1
...
Statement N
COMMIT;
Update replication info files

START TRANSACTION;
Statement 1
...
Statement N
Update replication info
COMMIT

relay_log_info_repository=TABLE

IO thread crash-safe



relay-log-recovery = 1

master_info_repository=TABLE

slave不根据master-info.log的信息进行重连，而是根据relay-info中执行到master的位置信息重新开始拉master上的日志数据

复制最优的参数配置

Master

binlog_format = ROW

expire_logs_days = 15

server-id = 327

sync_binlog = 1

innodb_flush_log_at_trx_commit = 1

innodb_support_xa = 1

Slave

log_slave_updates=1

server-id = 328

Relay_log_recover = 1

relay_log_info_repository = TABLE

master_info_repository = TABLE

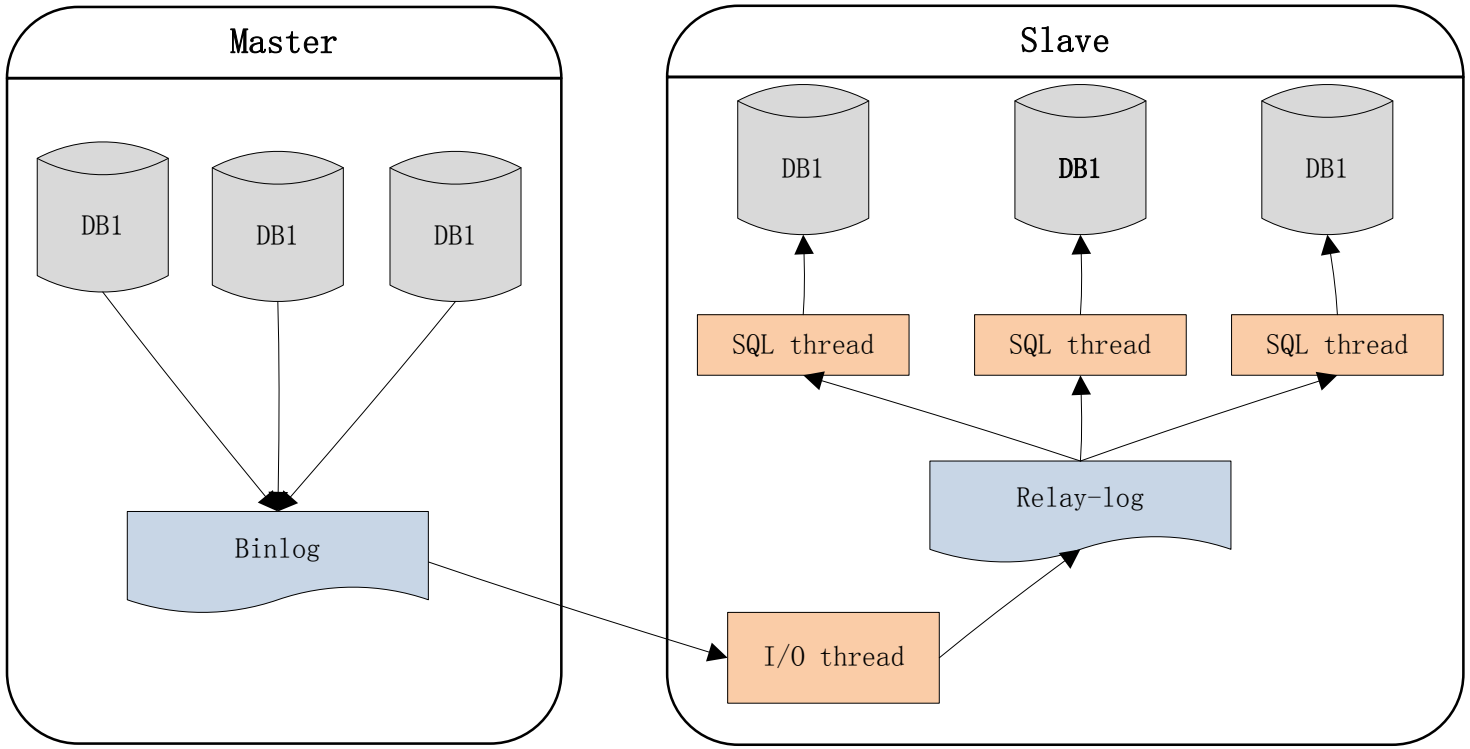
read_only = 1

复制效率

如何提高复制效率？

MySQL 5.6提供了多线程复制……

Multi-Threaded Slave



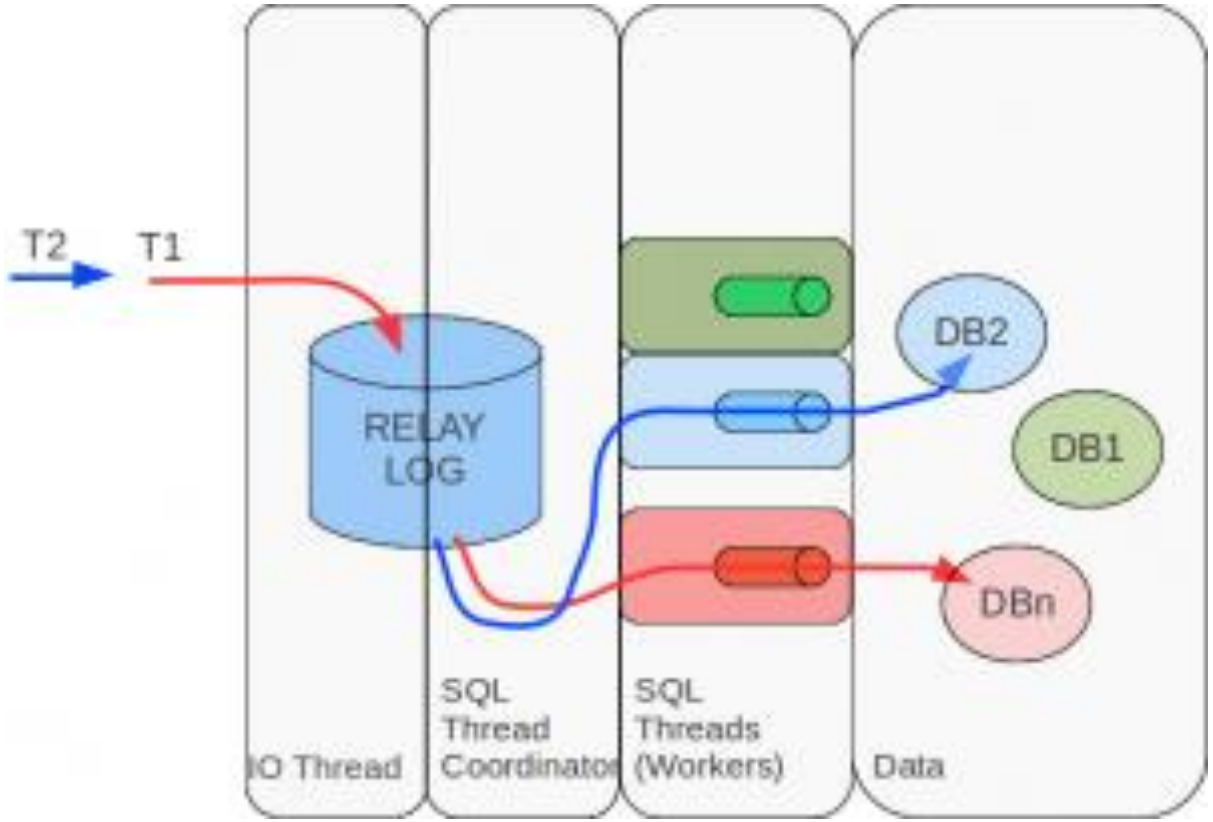
并行只是基于database的

slave_parallel_workers=3

如果是基于单database的依然无法做到真正的并行回放，

延迟依然无法解决

基于组提交的并行复制



一组中的事务 可以并行回放

GTID

- **Global Transaction Identifier**
- **uuid + transactionid**

更容易的进行**failover**操作

用来替换(filename,position)

gtid_mode=ON
log_slave_updates=1
enforce-gtid-consistency=1

基于组提交的并行复制

```
#160330 17:30:24 server id 11  end_log_pos 259 CRC32 0x9ed4d9ba         GTID    last_committed=1        sequence_number=1
#160330 17:30:24 server id 11  end_log_pos 520 CRC32 0xbbc4fb0a         GTID    last_committed=1        sequence_number=2
#160330 17:30:24 server id 11  end_log_pos 781 CRC32 0x59228ba1         GTID    last_committed=1        sequence_number=3
#160330 17:30:24 server id 11  end_log_pos 1042 CRC32 0xe66bb4f7        GTID    last_committed=1        sequence_number=4
#160330 17:30:24 server id 11  end_log_pos 1303 CRC32 0xd0bc88db        GTID    last_committed=4        sequence_number=5
```

MySQL 5.7 引入Anonymous_Gtid的二进制日志event类型

Multi-Threaded Slave

slave-parallel-type= DATABASE /LOGICAL_CLOCK

-- DATABASE -- 基于库级别的并行复制 与5.6相同

-- 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。

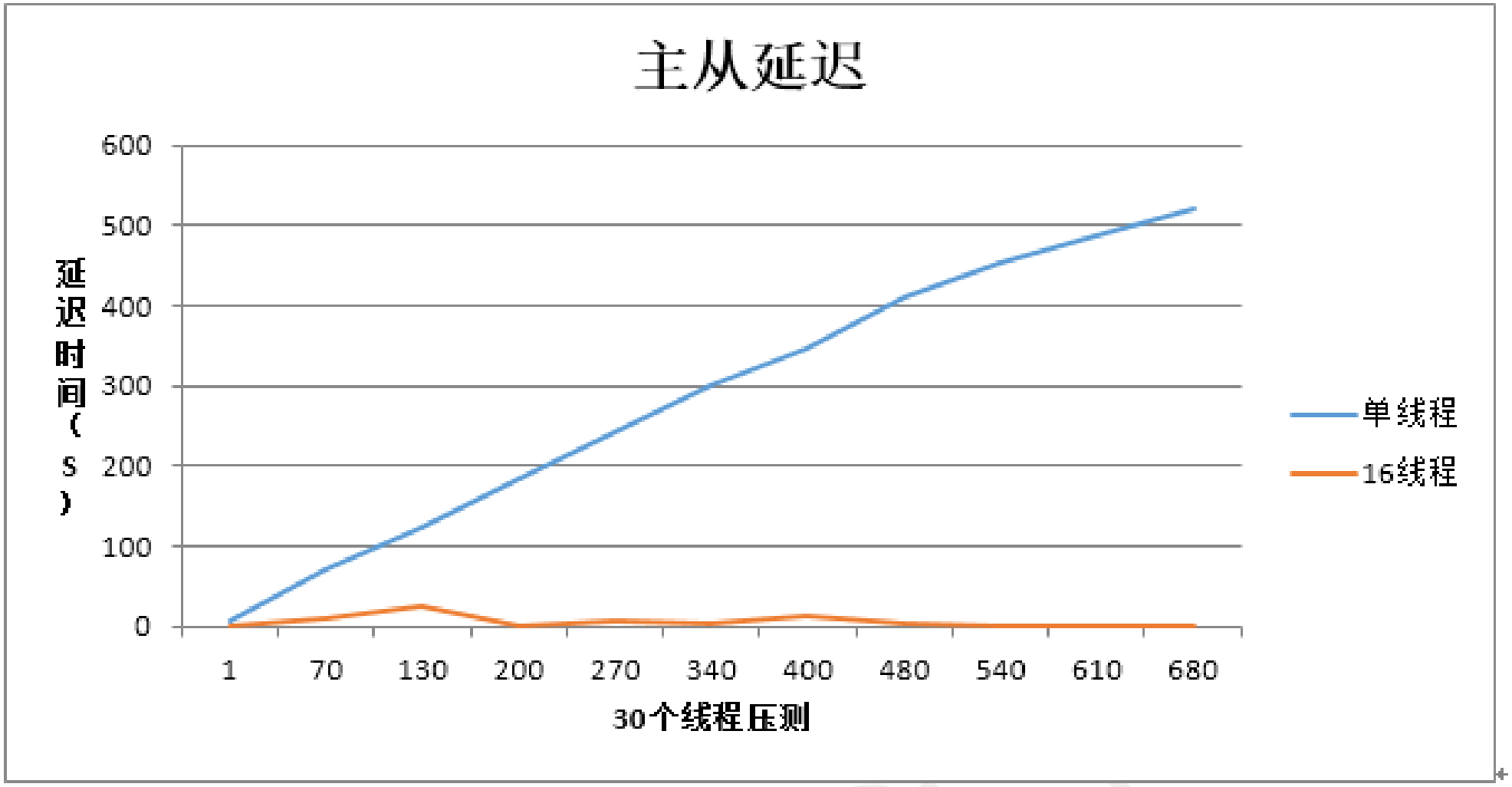
slave-parallel-workers=16

-- 并行复制的线程数

slave_preserve_commit_order=1

--Slave上commit的顺序保持一致

多线程复制性能


主从延迟
600
500
400
300
200
100
0
延迟时间（s）
1
70
130
200
270
340
400
480
540
610
680
30个线程压测
单线程
16线程

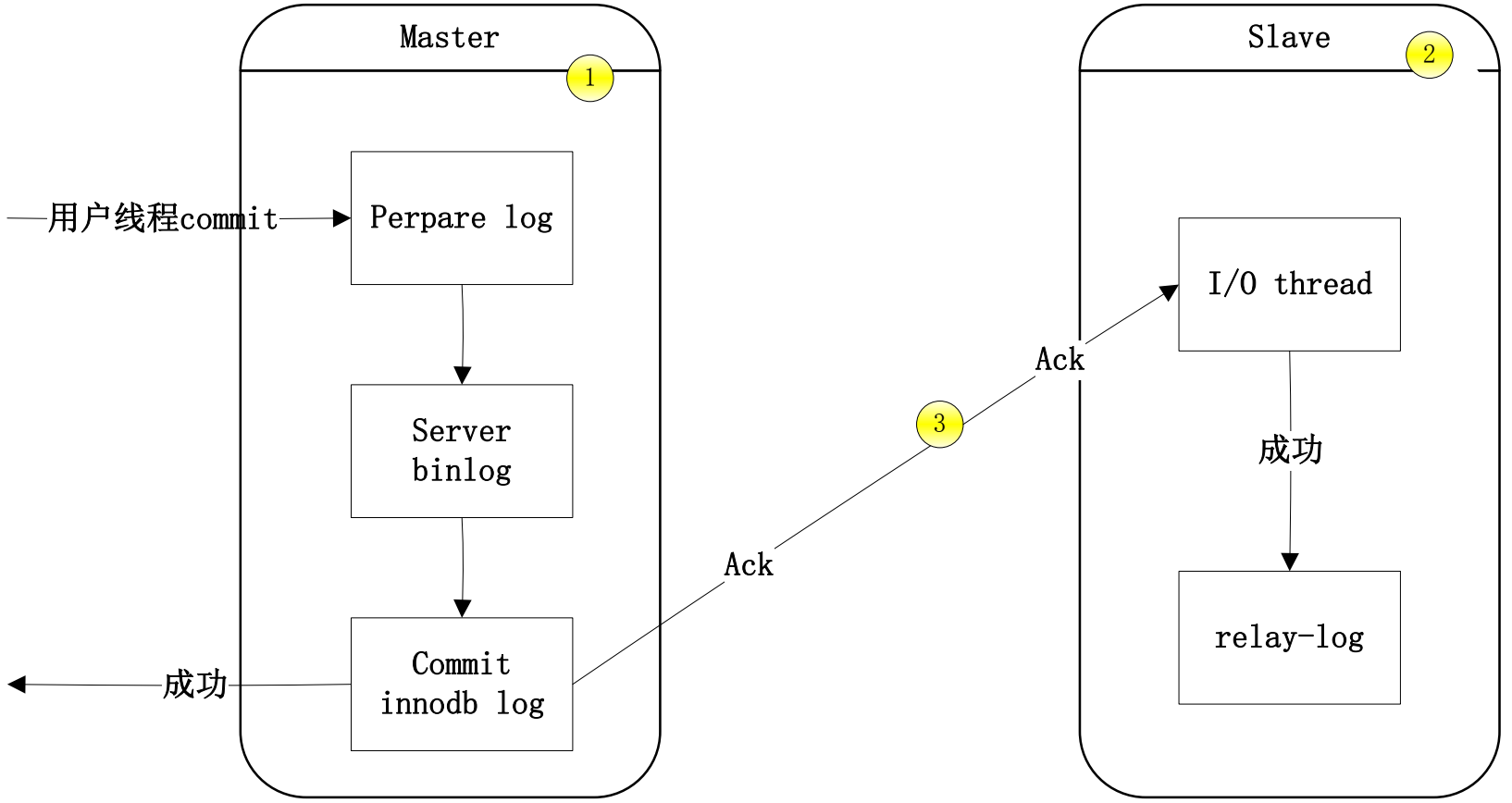
数据丢失

默认复制为异步模式

MySQL 5.5 semi-sync replication
减少数据丢失风险，不能完全避免数据丢失

MySQL 5.7 lossless semi-sync replication
可保证数据完全不丢失

semi-sync replication



至少有一个Slave节点收到binlog后再返回

不能完全避免数据丢失

减少数据丢失风险

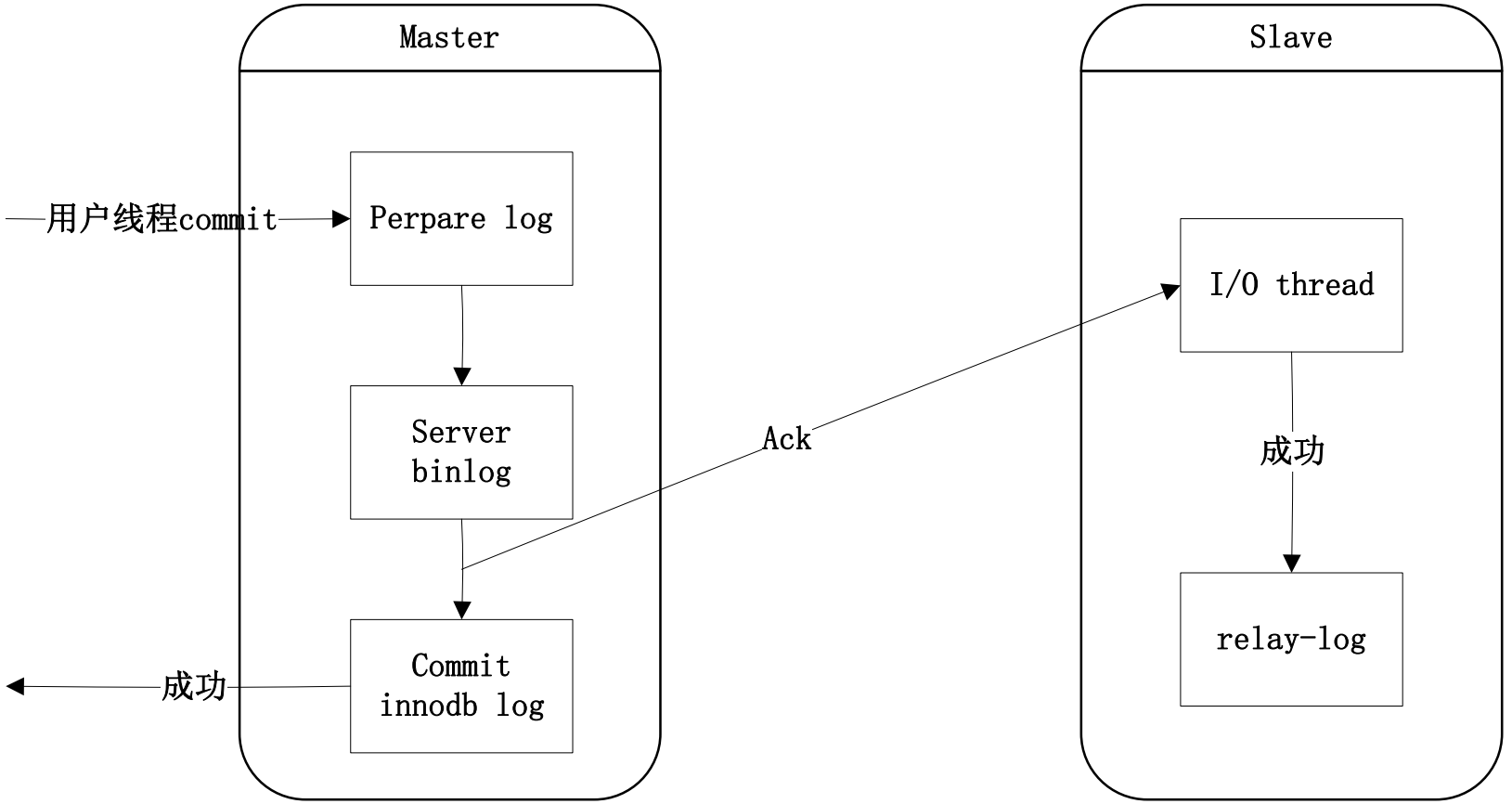
超时后，切回异步复制

1.rpl_semi_sync_master_enabled=1

3.rpl_semi_sync_master_timeout

2.rpl_semi_sync_slave_enabled=1

loss less semi-sync replication



可保证数据完全不丢失

二进制日志先写远程

rpl_semi_sync_master_wait_point=AFTER_SYNC/AFTER_COMMIT

rpl_semi_sync_master_wait_slave_count

MySQL 5.7

半同步复制与无损复制的对比

ACK的时间点 不同

--半同步复制在InnoDB层的Commit Log后，等待ACK

--无损复制在MySQL Server层的Write binlog后，等待ACK

半同步复制与无损复制的对比

主从数据一致性

--半同步复制意味着在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

--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

半同步相关事件统计

- Rpl_semi_sync_master_tx_avg_wait_time

  --开启Semi_sync，平均需要额外等待的时间

- Rpl_semi_sync_master_net_avg_wait_time

  --事务进入等待队列后，到网络平均等待时间

  Semi-sync的网络消耗有多大，给某个事务带来的额外的消耗有多大。

- Rpl_semi_sync_master_status

  -- 则表示当前Semi-sync是否正常工作。

- Rpl_semi_sync_master_no_times

  --可以知道一段时间内，Semi-sync是否有超时失败过，记录了失败次数。

半同步参数

rpl_semi_sync_master_enabled=1

rpl_semi_sync_slave_enabled=1

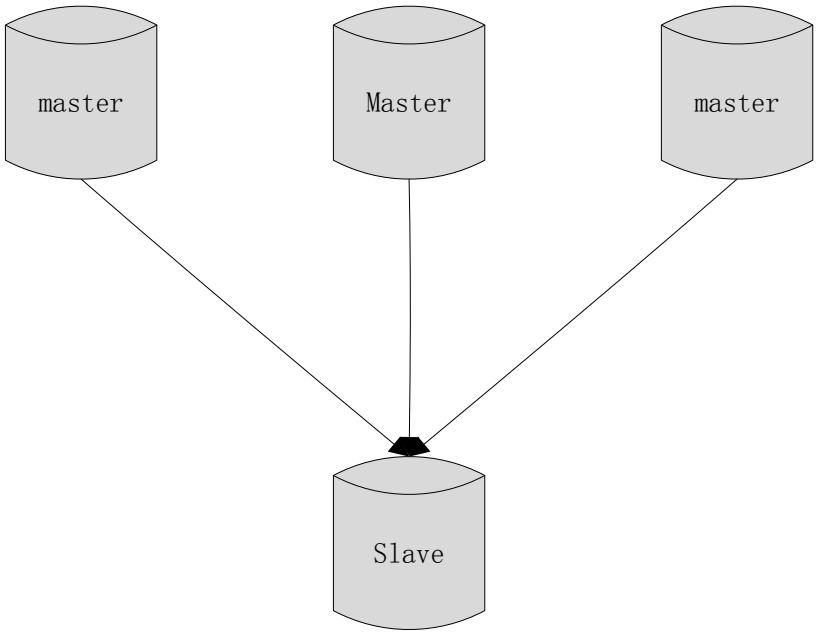
rpl_semi_sync_master_timeout=1000

rpl_semi_sync_master_wait_for_slave_count=1

rpl_semi_sync_master_wait_point=AFTER_SYNC

rpl_semi_sync_master_wait_for_slave_count=1

multi-source

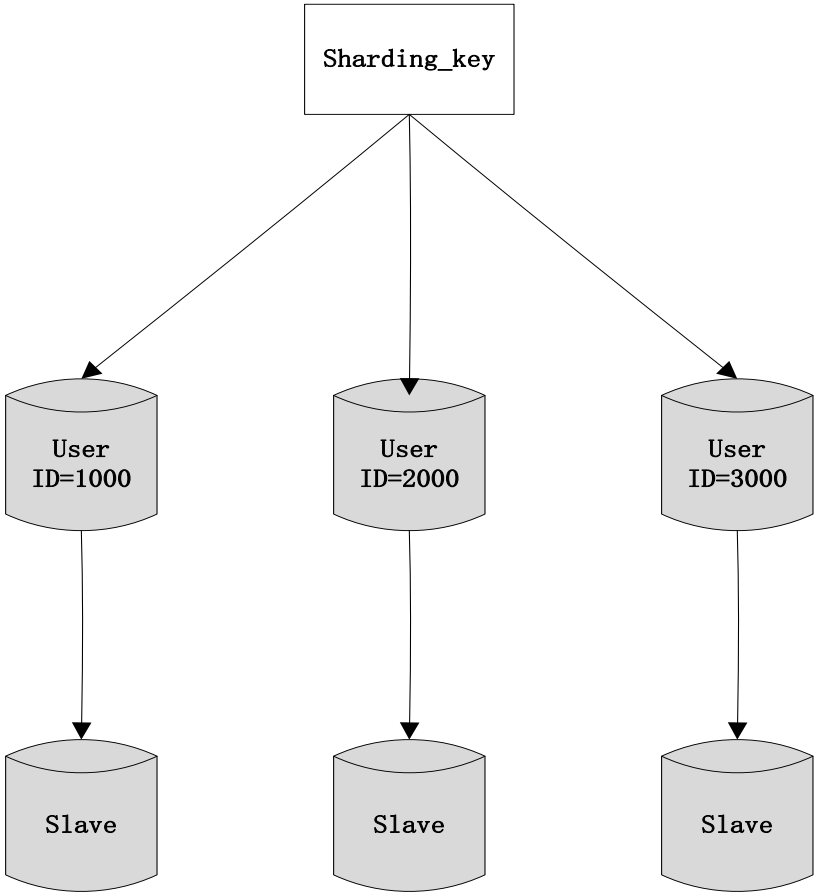


- 多源复制采用多通道的模式

CHANGE MASTER TO .... FOR CHANNEL ‘m1';

CHANGE MASTER TO .... FOR CHANNEL ‘m2';

multi-source

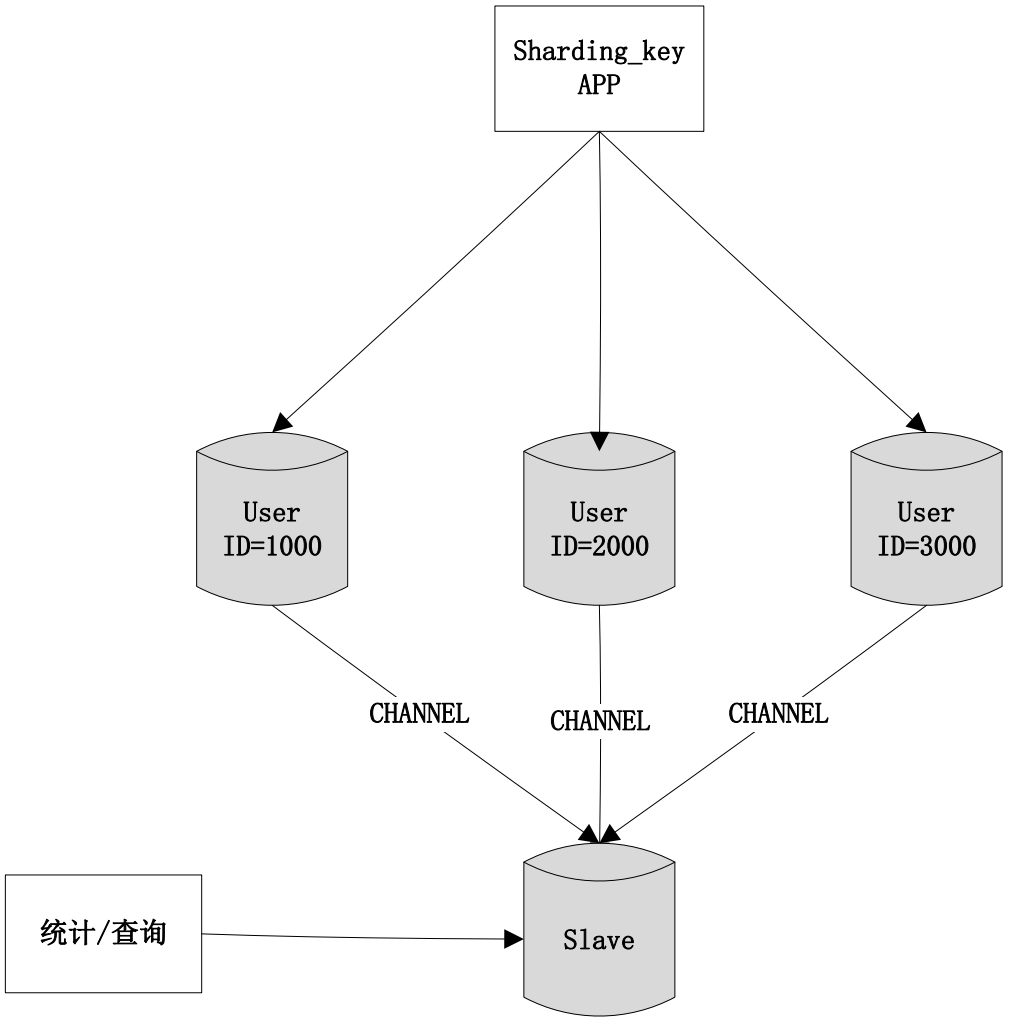


join 绝对是关系型数据库中最常用一个特性，然而在分布式的环境中，join是最难解决的一个问题。

update user set name=.. where id in(1000,2000)

或是不根据分区键查询→反查

multi-source



数据库，表名一致如何进行？

实现奇偶方法

auto_increment_offset=1…n

auto_increment_increment=n

Thanks