MySQL复制与数据一致性

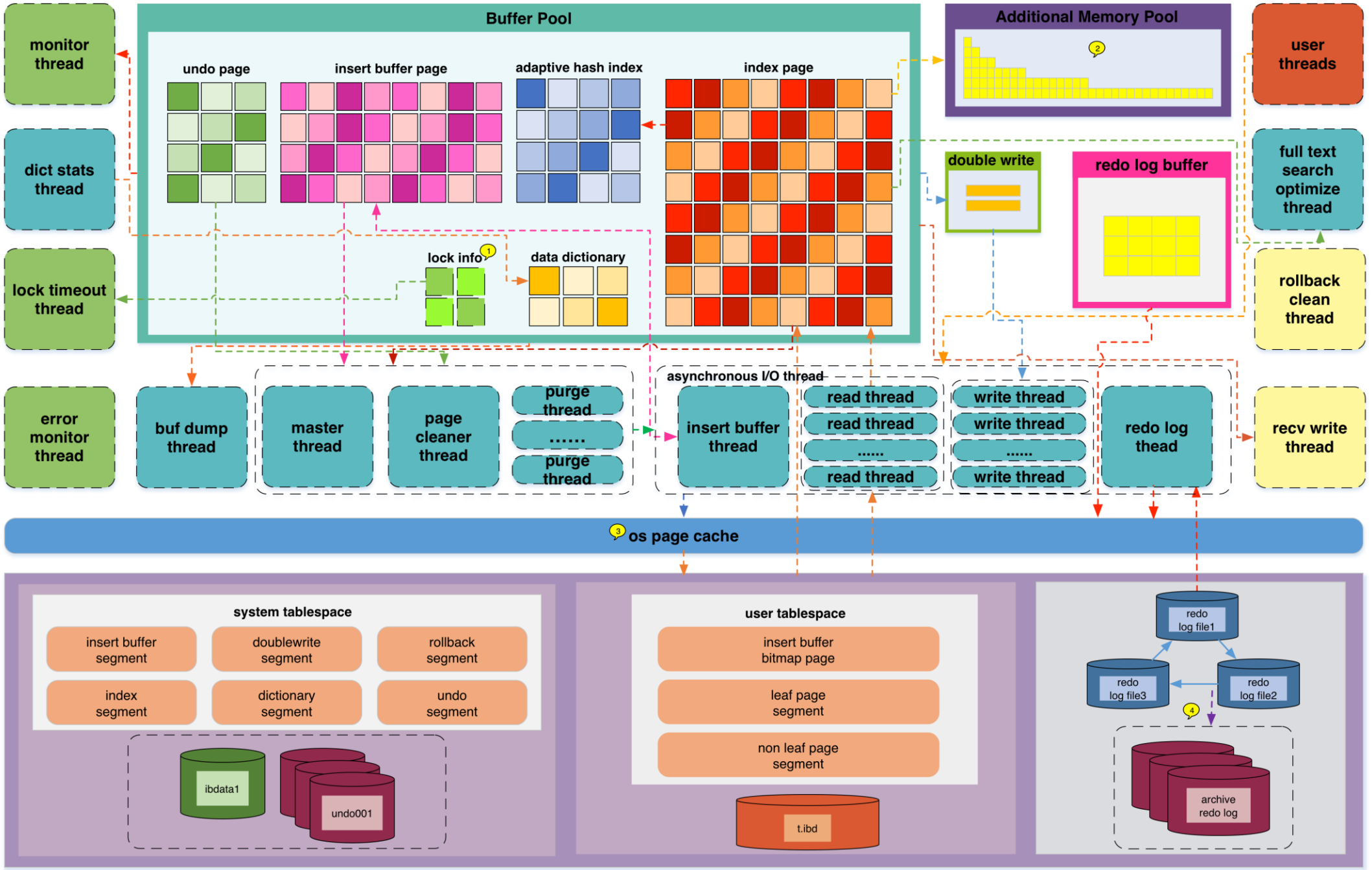
六度人和-研发部-周晓

2018-03-22

Agenda

1. MySQL Binlog
2. 事务提交过程(2PC，分布式事务CAP)
3. 组提交原理(Group Commit)
4. Replication流程（异步，半同步，并行、延迟的原因）
5. Crash Safe(master/slave，物理备份原理)
6. 高可用类型

MySQL 5.6 | InnoDB Storage Engine Architecture



1. MySQL Binlog

- Binlog格式
  - Statement (SBR)
  - ROW (RBR)
  - Mixed

- 开启二进制日志(部分参数)
  - server_id=12456
    binlog_format=ROW
    binlog_row_image=FULL|MINIMAL
    binlog_rows_query_log_events=OFF
    log_bin=ON
    expire_logs_days=3
    max_binlog_size=500M
    binlog_order_commits=OFF
    gtid_mode=ON

思考：

- 什么情况下会出现某binlog的开始时间会明显小于上一个binlog的结束时间？

```
# at 372
#180302 10:16:44 server id 11101  end_log_pos 451 CRC32 0x58beaa31      Query   thread_id=2032484       exec_time=0     error_code=0
SET TIMESTAMP=1519957004/*!*/;
BEGIN
/*!*/;
# at 451
#180302 10:16:44 server id 11101  end_log_pos 522 CRC32 0xea91a834      Table_map: `d_ec_crmlog`.`positions` mapped to number 328043
# at 522
#180302 10:16:44 server id 11101  end_log_pos 644 CRC32 0x03f75485      Update_rows: table id 328043 flags: STMT_END_F

BINLOG '
DLSYWhNdKwAARwAAAAoCAAAAAGsBBQAAAAEAC2RfZWNfY3JtbG9nAAlwb3NpdGlvbnMABwMPAw8P
CAgG/AMAQP8AbjSokeo=
DLSYWh9dKwAAegAAAIQCAAAAAGsBBQAAAAEAAgAH//+oXSsAABAAbXlzcWwtYmluLjAwMDAwN/vD
/B8HbWF4d2VsbGtJf+RhAQAAqF0rAAAQAG15c3FsLWJpbi4wMDAwMDhVAQAAB21heHdlbGxFUX/k
YQEAAIVU9wM=
'/*!*/;
### UPDATE `d_ec_crmlog`.`positions`
### WHERE
###   @1=11101 /* INT meta=0 nullable=0 is_null=0 */
###   @2='mysql-bin.000007' /* VARSTRING(1020) meta=1020 nullable=1 is_null=0 */
###   @3=536658939 /* INT meta=0 nullable=1 is_null=0 */
###   @4=NULL /* INT meta=16384 nullable=1 is_null=1 */
###   @5='maxwell' /* VARSTRING(255) meta=255 nullable=0 is_null=0 */
###   @6=NULL /* VARSTRING(255) meta=0 nullable=1 is_null=1 */
###   @7=1519957002603 /* LONGINT meta=0 nullable=1 is_null=0 */
### SET
###   @1=11101 /* INT meta=0 nullable=0 is_null=0 */
###   @2='mysql-bin.000008' /* VARSTRING(1020) meta=1020 nullable=1 is_null=0 */
###   @3=341 /* INT meta=0 nullable=1 is_null=0 */
###   @4=NULL /* INT meta=16384 nullable=1 is_null=1 */
###   @5='maxwell' /* VARSTRING(255) meta=255 nullable=0 is_null=0 */
###   @6=NULL /* VARSTRING(255) meta=0 nullable=1 is_null=1 */
###   @7=1519957004613 /* LONGINT meta=0 nullable=1 is_null=0 */
# at 644
#180302 10:16:44 server id 11101  end_log_pos 675 CRC32 0xe84ebf40      Xid = 609662496
COMMIT/*!*/;
# at 675
```

1. MySQL Binlog

- 查看binlog：
  - `show binary logs;` -- 查看当前在线的binlog列表
  - `show master status;` -- 查看当前binlog position
  - `show binlog events in 'mysql-bin.000493' from 4 limit 10;`
  - `purge master logs to 'binlog.000493';` -- 删除binlog
  - `flush logs;` -- 滚动当前binlog，需要reload权限

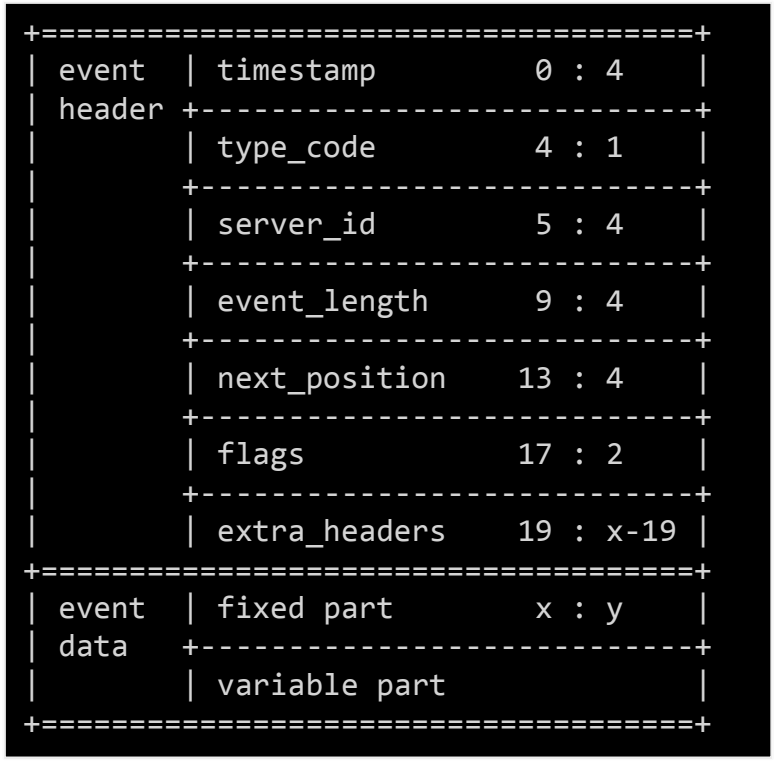
```
mysql> show binlog events in 'mysql-bin.000016';
+------------------+-----+-------------+-----------+-------------+-----------------------------------------+
| Log_name         | Pos | Event_type  | Server_id | End_log_pos | Info                                    |
+------------------+-----+-------------+-----------+-------------+-----------------------------------------+
| mysql-bin.000016 |   4 | Format_desc |      1951 |         120 | Server ver: 5.6.27-log, Binlog ver: 4   |
| mysql-bin.000016 | 120 | Query       |      1951 |         199 | BEGIN                                   |
| mysql-bin.000016 | 199 | Table_map   |      1951 |         264 | table_id: 456 (d_ _ og.t_repl_test)     |
| mysql-bin.000016 | 264 | Write_rows  |      1951 |         332 | table_id: 456 flags: STMT_END_F         |
| mysql-bin.000016 | 332 | Xid         |      1951 |         363 | COMMIT /* xid=149043 */                 |
| mysql-bin.000016 | 363 | Query       |      1951 |         442 | BEGIN                                   |
| mysql-bin.000016 | 442 | Table_map   |      1951 |         507 | table_id: 456 (d_ _ og.t_repl_test)     |
| mysql-bin.000016 | 507 | Delete_rows |      1951 |         553 | table_id: 456 flags: STMT_END_F         |
| mysql-bin.000016 | 553 | Table_map   |      1951 |         618 | table_id: 456 (d_ og.t_repl_test)       |
| mysql-bin.000016 | 618 | Delete_rows |      1951 |         675 | table_id: 456 flags: STMT_END_F         |
| mysql-bin.000016 | 675 | Table_map   |      1951 |         740 | table_id: 456 (d_ og.t_repl_test)       |
| mysql-bin.000016 | 740 | Write_rows  |      1951 |         808 | table_id: 456 flags: STMT_END_F         |
| mysql-bin.000016 | 808 | Xid         |      1951 |         839 | COMMIT /* xid=149300 */                 |
+------------------+-----+-------------+-----------+-------------+-----------------------------------------+
13 rows in set (0.00 sec)
```

思考：

- 图中有多少个事务？
  有多少语句？影响多少行？

1. MySQL Binlog

- Binlog结构



- FORMAT_DESCRIPTION_EVENT = 15
  - GTID_EVENT
  - QUERY_EVENT = 2
    - TABLE_MAP_EVENT = 19
    - ROW_LOG_EVENT

      (WRITE_ROWS, UPDATE_ROWS, DELETE_ROWS)
  - XID_EVENT = 16
- ROTATE_EVENT = 4

Event Type	Cols_before_image	Cols_after_image
DELETE	Deleted row	NULL
INSERT	NULL	Inserted row
UPDATE	Old row	Updated row

- https://dev.mysql.com/doc/internals/en/event-structure.html
- https://dev.mysql.com/doc/internals/en/event-data-for-specific-event-types.html

思考：

1. 为什么不直接把字段名存入binlog?
2. table_map_event里面有什么？table_id什么情况下会增长？

1. MySQL Binlog

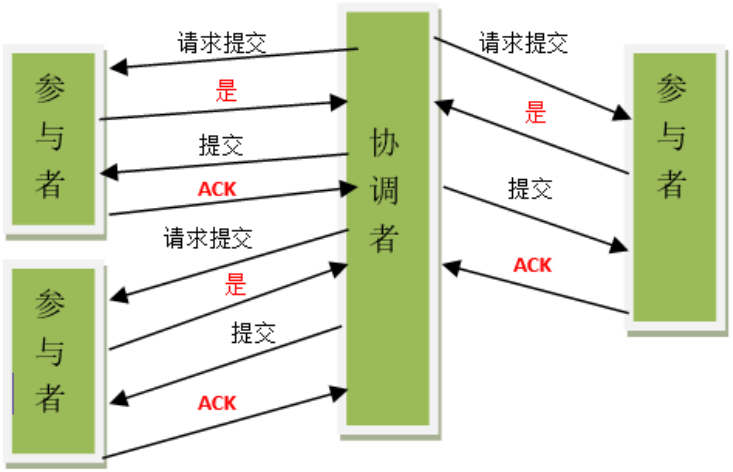
- GTID: Global Transaction Identifier (5.6)
- 在binlog中唯一标识一个事务，在记录binlog之前生成，放在gtid_next中
- 作用
  - 可以追踪事务在哪一个实例提交的
  - 搭建主从或者Failover时，自动找点
  - 帮助实现多线程复制
- 表示
  - source_server_uuid : transaction_id
  - set集合

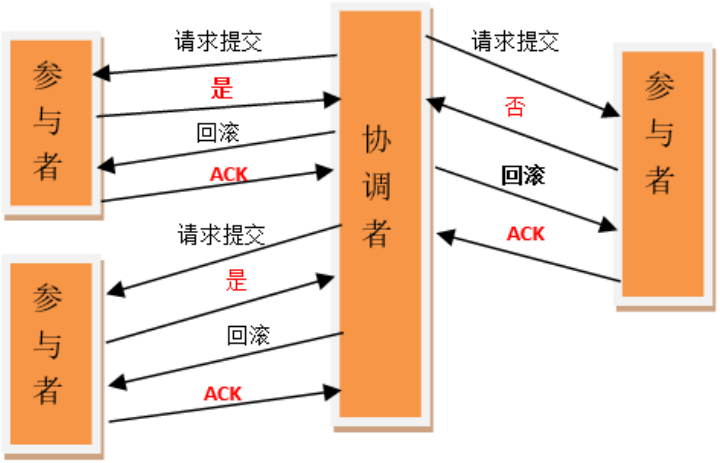
```
mysql> select @@server_uuid;
+--------------------------------------+
| @@server_uuid                        |
+--------------------------------------+
| 21a239ec-ed3f-11e7-a33c-7cd30ad3aab0 |
+--------------------------------------+

mysql> show master status\G
*********************** 1. row ***********************
             File: mysql-bin.003036
         Position: 353239290
     Binlog_Do_DB:
 Binlog_Ignore_DB:
Executed_Gtid_Set:
03513914-d8bf-11e7-9d8e-7cd30ae07f2c:1-34996457,
21a239ec-ed3f-11e7-a33c-7cd30ad3aab0:1-281678164,
e6bfce35-d8be-11e7-9d8d-7cd30ae081b8:1-6847,
fe20b59d-ed3e-11e7-a33b-7cd30abc971a:1-500001758
```

提示：MySQL官方与MariaDB Server实现gtid的方式不兼容

2. 分布式事务

- 分布式事务
  - 解决多节点数据一致性的问题
  - XA角色有 事务管理器(TM) 和 资源管理器(RM)
  - 两阶段提交(2PC): Prepare → Commit





- 3PC
- CAP理论、BASE理论

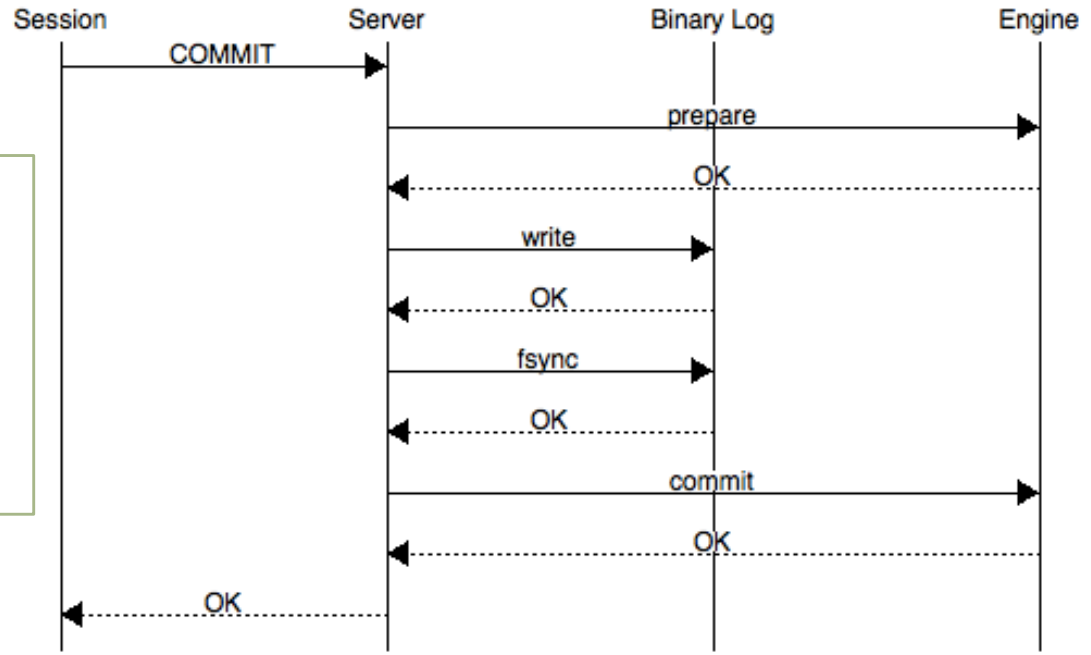
思考：

- 二阶段提交有哪些问题？

2. InnoDB两阶段提交

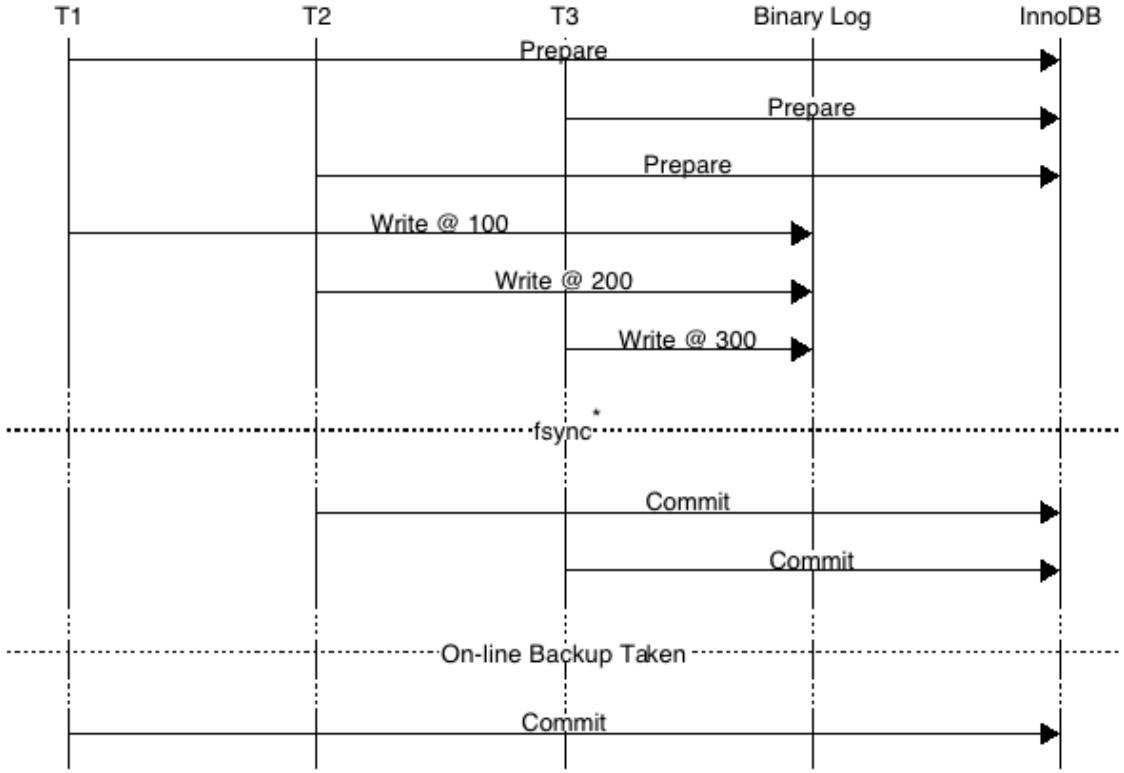
- 内部XA
  - 解决binlog日志与redo log 的一致性
  - Server层的binlog作为2阶段提交的事务协调者
  - binlog刷磁盘成功，相当于commit ok

- sync_binlog=0，二进制日志fsync()的操作基于操作系统
- sync_binlog=1，每一个tx commit都会调用一次fsync()，
  此时能保证数据最安全但是性能影响较大
- sync_binlog=N，每N个事务调用一次fsync
  当数据库crash的时候会丢失N-1个事务



3. 组提交(Binlog Group Commit)

- 为什么会有组提交
  - 多个事务一起提交
    binlog记录顺序不一致



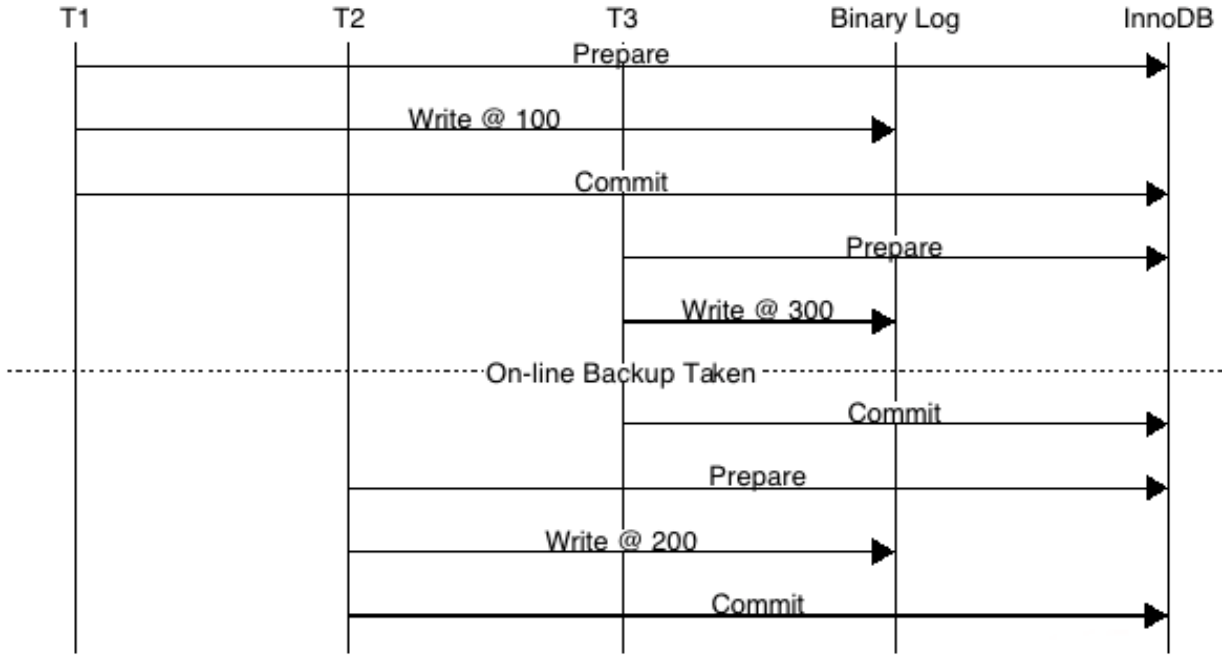
思考：

- binlog写入顺序与引擎提交顺序一致，
  就能保证从库不丢数据吗？

- binlog写: T1,T2,T3
- 提交顺序: T2,T3,<backup>,T1
- 物理备份会记录当前binlog最后事务标记，
  认为T3之前的事务都已提交，但备份数据文件中没有T1数据

3. 组提交(Binlog Group Commit)

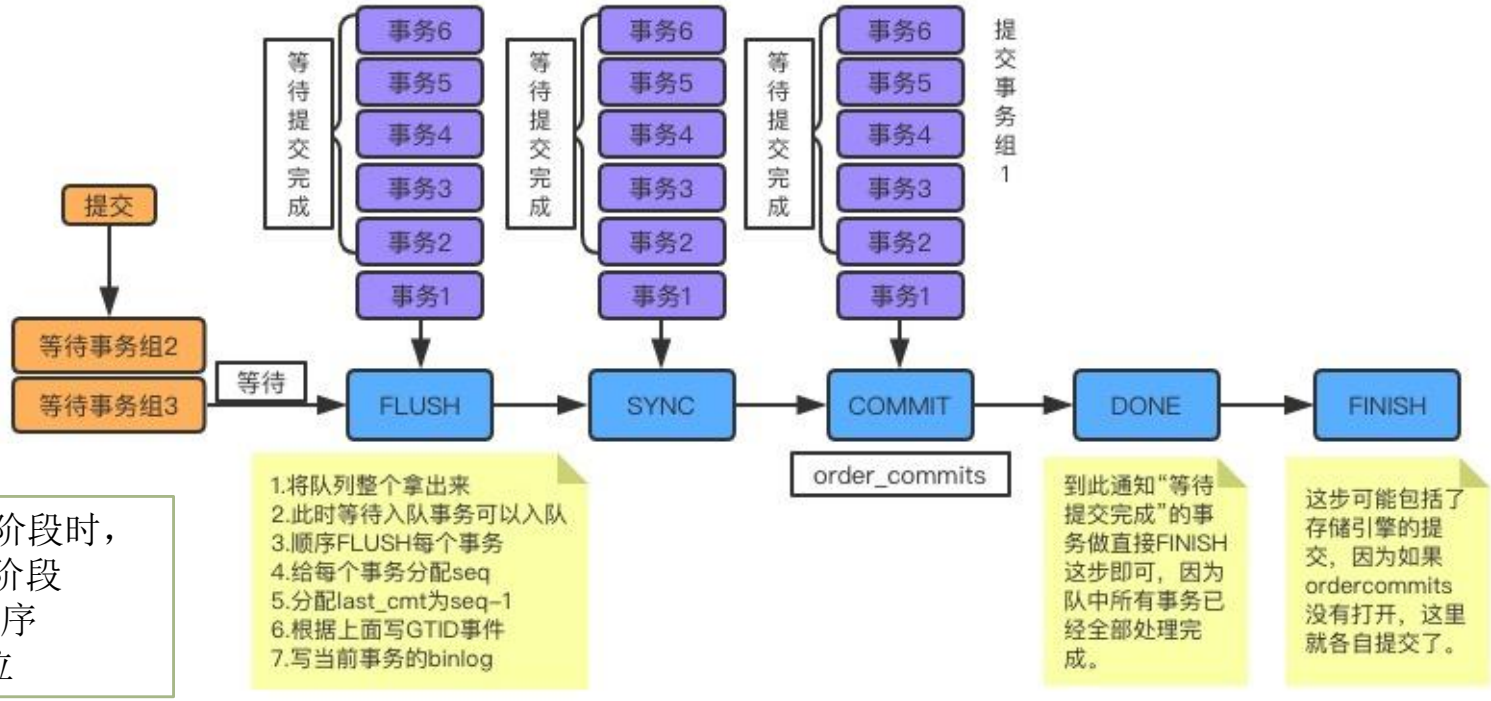
- 为什么会有组提交
  - 提高写入性能
  - 二进制日志写入顺序和存储引擎提交顺序保持一致
  - 5.7并行复制的基础



prepare_commit_mutex

3. 组提交(Binlog Group Commit)

- Flush stage: 将事务的日志刷入binlog文件的buffer中
- Sync stage: 将binlog文件缓存写入磁盘
- Commit stage: 根据顺序调用存储引擎提交事务



- 当一组事务在进行Commit阶段时，他新的事务可以进行Flush阶段
- 每个阶段都控制了事务的顺序
- sync_binlog变成以组为单位

3. 组提交(Binlog Group Commit)

- 参数控制
  - Ver < 5.7.9:
  - `binlog_max_flush_queue_time=N`
  - `Ver > 5.7.9`
  - `binlog_group_commit_sync_delay=N`
  - `binlog_group_commit_sync_no_delay_count`

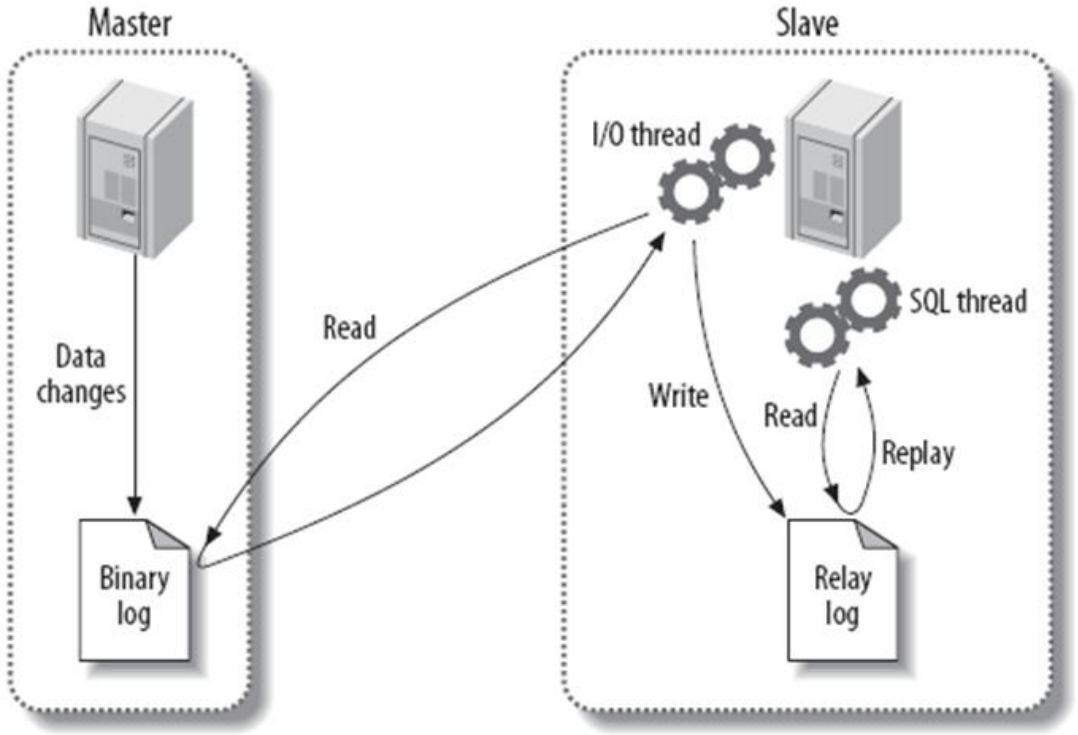
MySQL 5.7 Parallel replication

- 实现主备多线程复制基于主库Binary Log Group Commit, 并在Binary log日志中标识同一组事务的last_commited=N和该组事务内所有的事务提交顺序

- 事务提交过程
  1. InnoDB 的事务 进入Prepare 阶段，即 SQL 已经成功执行并生成 redo 和 undo 的内存日志；
  2. binlog 提交，flush - sync - done
  3. InnoDB 内部提交，commit 阶段在存储引擎内提交，通过 innodb_flush_log_at_trx_commit 参数控制，使 undo 和 redo 永久写入磁盘

4. MySQL复制流程

5.1：异步复制
5.5 ： 半同步复制
5.7 ： 增强半同步



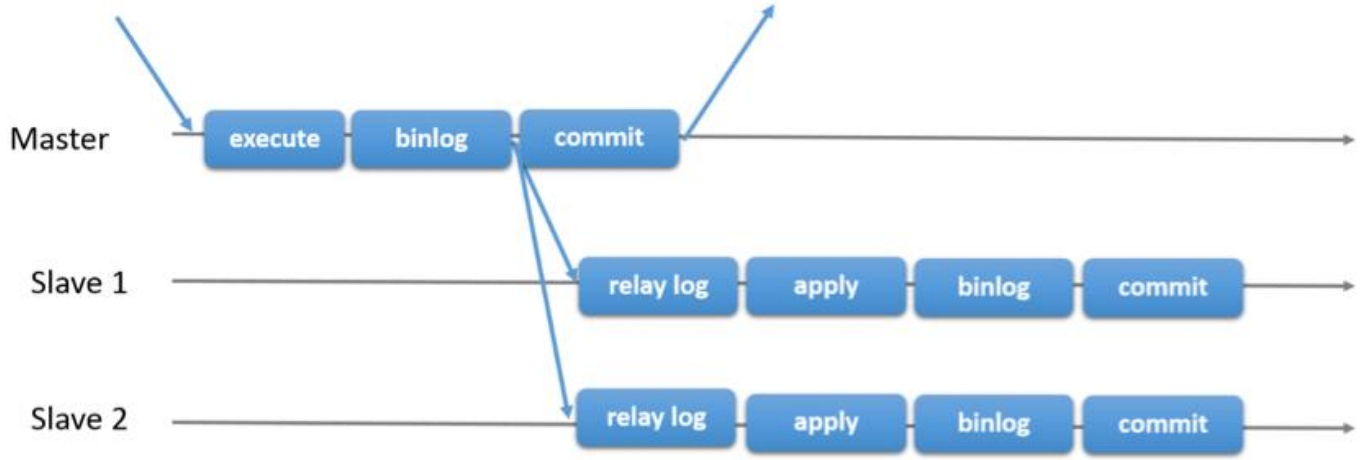
思考：

- binlog传输是“推”还是“拉”？

4.1 MySQL复制流程—半同步复制

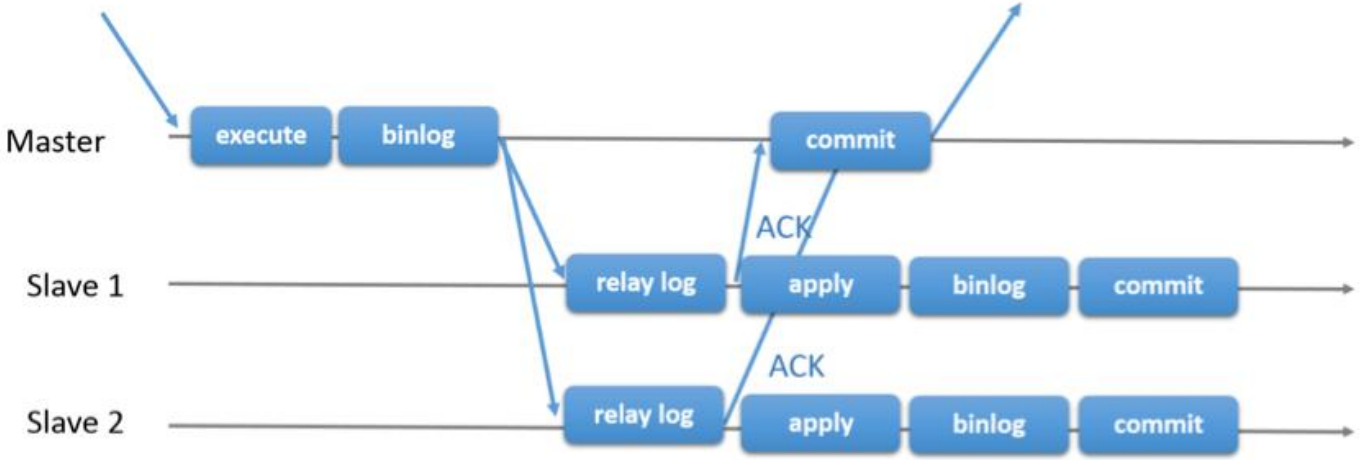
async replication

- 主库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
- 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



semi-sync replication

- 主库提交事务后不立刻返回给客户端，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
- 如果等待超时，没有任何一个从节点通知当前事务，自动转为异步(`rpl_semi_sync_master_timeout`)
- 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



增强半同步复制

- 5.7 loss-less半同步复制
  - 独立ack collector
  - `rpl_semi_sync_master_wait_point`

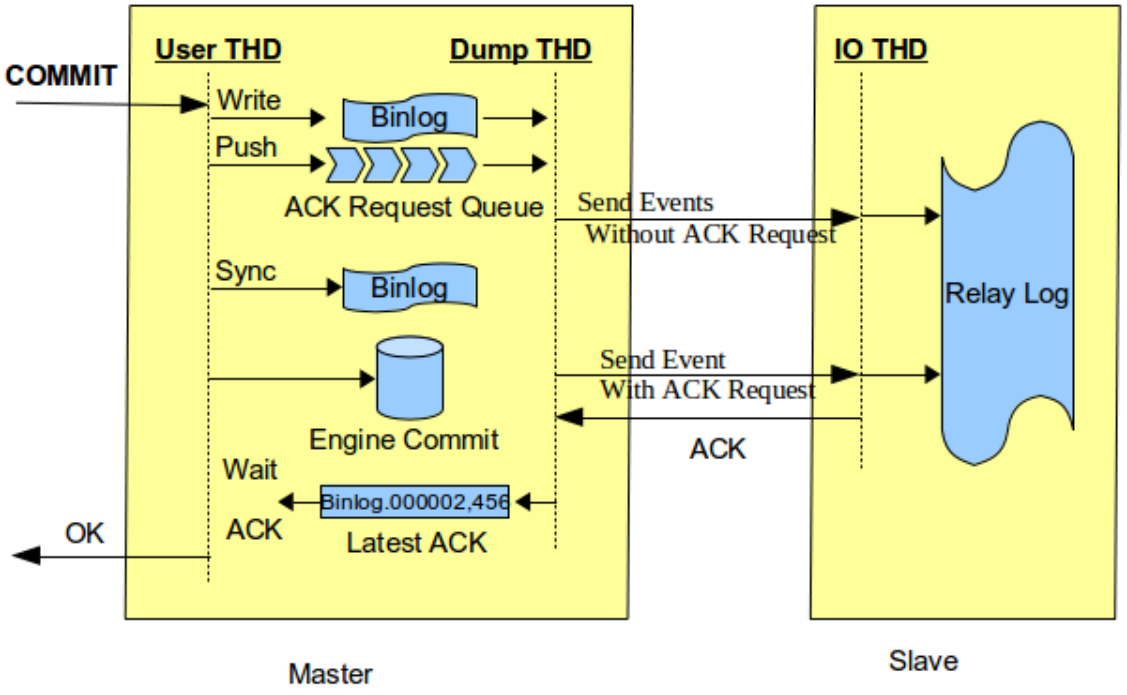
CAP理论

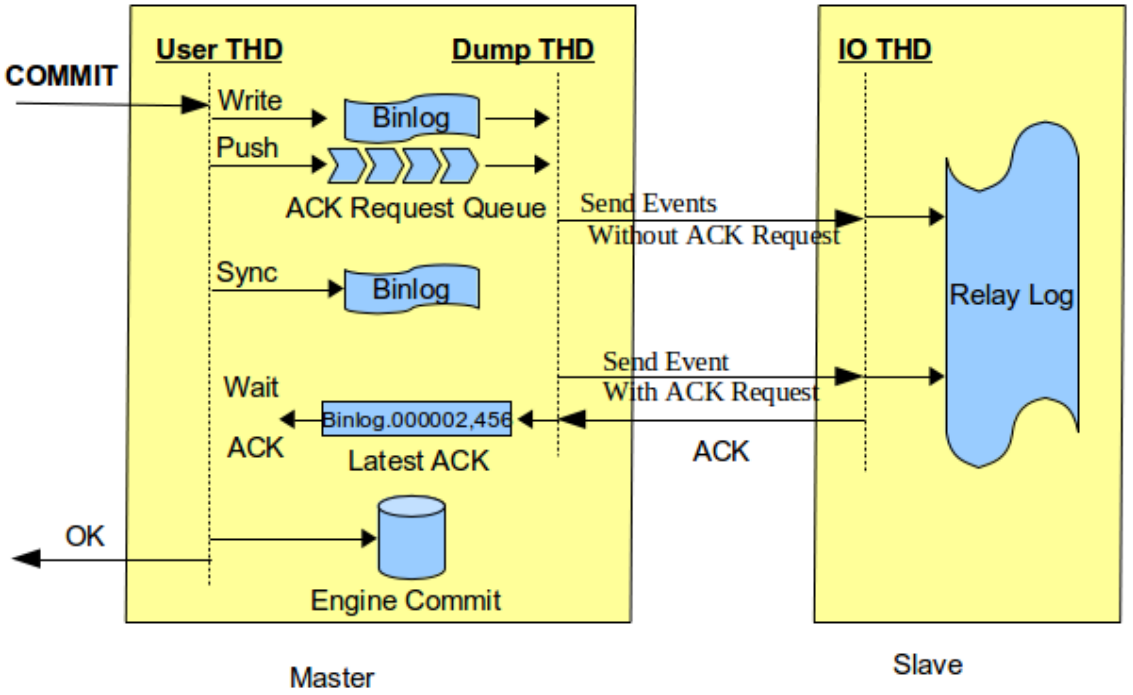
  - `AFTER_COMMIT`

`AFTER_SYNC`

思考：

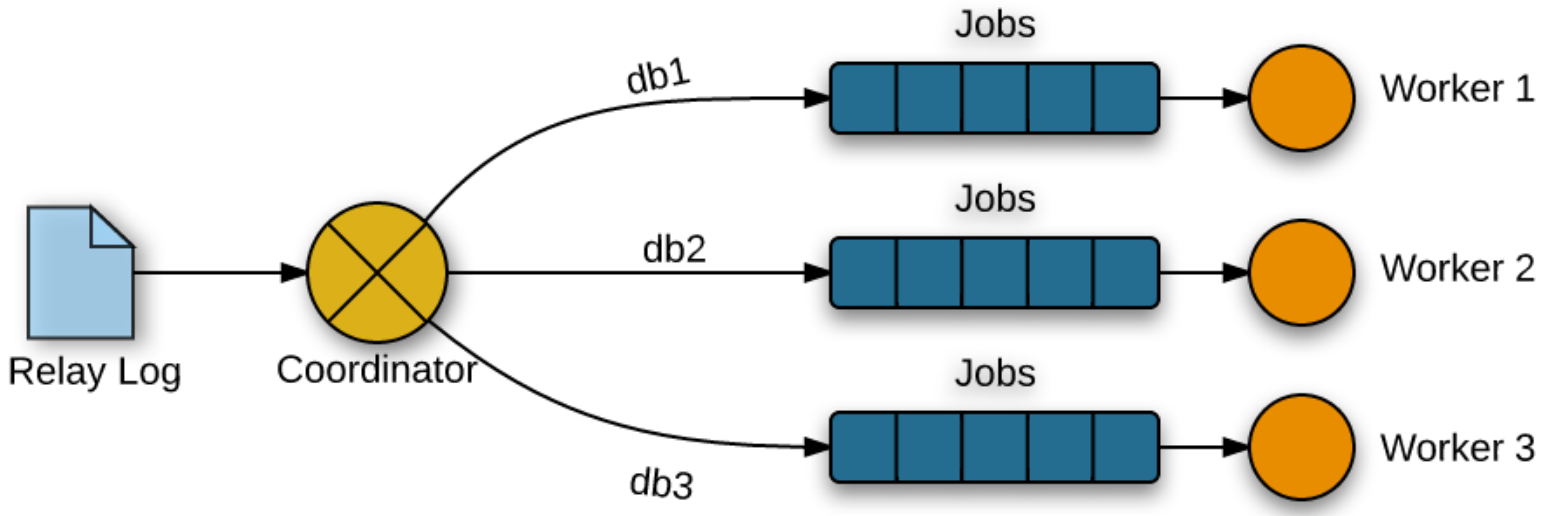
- 半同步复制下，发生主备切换，为什么旧主库同步出现了1062异常？





4.2 MySQL复制流程—并行复制

- 5.6并行复制



4.2 MySQL复制流程—并行复制

- 5.7 多线程复制(MTS)
  - GTID (Anonymous_Gtid when OFF)
  - Binlog Group Commit
  - last_committed

一个组提交的事务都是可以并行回放，因为这些事务都已进入到事务的prepare阶段，则说明事务之间没有任何冲突（否则就不可能提交）

- slave参数

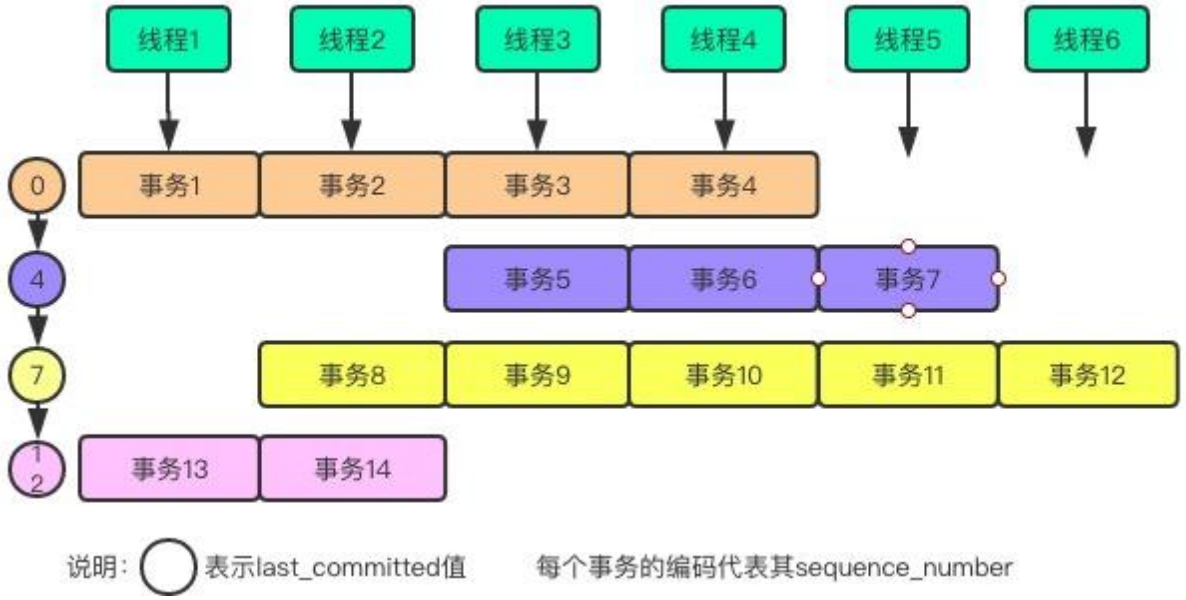
```
slave-parallel-type=DATABASE|LOGICAL_CLOCK
slave-parallel-workers=16
master_info_repository=TABLE
relay_log_info_repository=TABLE
relay_log_recovery=ON
```

提示：

1. 启动多线程复制后(wokers>1)，不能使用pt-slave-restart去修正主从复制错误
2. 阿里RDS5.6，支持表级别并行复制并且默认，slave_pr_mode=TABLE
   - 思考：它是怎么实现的？
3. MySQL 8.0 在binary log中记录writeset信息，可以做到行级别并发复制

4.2 MySQL复制流程—并行复制

- 5.7 多线程复制(MTS)
  - slave-parallel-type=LOGICAL_CLOCK
  - slave_preserve_commit_order=OFF -- (BASE理论：最终一致)




```
TID last_committed=0 sequence_number=1
GTID last_committed=0 sequence_number=2
GTID last_committed=0 sequence_number=3
GTID last_committed=0 sequence_number=4
GTID last_committed=4 sequence_number=5
GTID last_committed=4 sequence_number=6
GTID last_committed=4 sequence_number=7
GTID last_committed=7 sequence_number=8
 GTID last_committed=7 sequence_number=9
 GTID last_committed=7 sequence_number=10
 GTID last_committed=7 sequence_number=11
 GTID last_committed=7 sequence_number=12
 GTID last_committed=12 sequence_number=13
 GTID last_committed=12 sequence_number=14
```

4.3 MySQL常见同步延迟原因

- 常见延迟原因
  - 从机配置低，如IO、多实例
  - 主库有大量更新，TPS高，而从库SQL Thread串行回放
  - 主库有大事务，如insert ... select ...数据迁移动作
  - 主库在做DDL，如修改索引、添加字段
  - 表上无主键: slave_rows_search_algorithms
  - 其它：
    - 网络延迟、从库在做备份、从库复制异常、同步参数未优化、数据库版本低

- 思考
  - 执行stop slave卡住是什么原因，怎么处理？
  - 常见处理？参考复制异常
  - **怎么看是否有**复制错误有哪些，怎么**延迟？**

4.3 MySQL常见同步延迟原因

- Seconds_Behind_Master

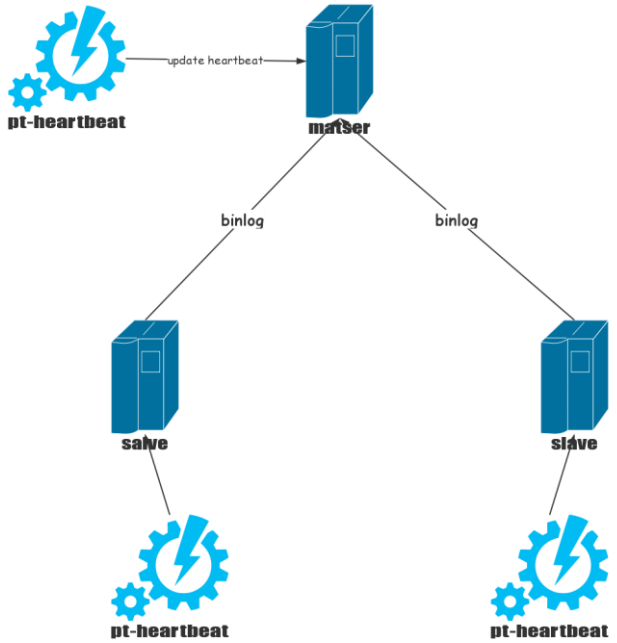
```
mysql> show slave status\G
*********************** 1. row ***********************
             Slave_IO_State: Waiting for master to send event
            Master_Log_File: mysql-bin.014670
        Read_Master_Log_Pos: 181716556
             Relay_Log_File: slave-relay.028871
              Relay_Log_Pos: 166693104
      Relay_Master_Log_File: mysql-bin.014670
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
        Exec_Master_Log_Pos: 166692941
      Seconds_Behind_Master: 0
```

Seconds_Behind_Master的计算规则：

- 从库当前的时间戳 - 从库SQL线程当前正在执行的事件记录的主库的时间戳
- 当从库sql线程上没有事件被执行时，SBM的值为0

http://mysql.taobao.org/monthly/2016/03/09/

- pt-heartbeat



工作原理：

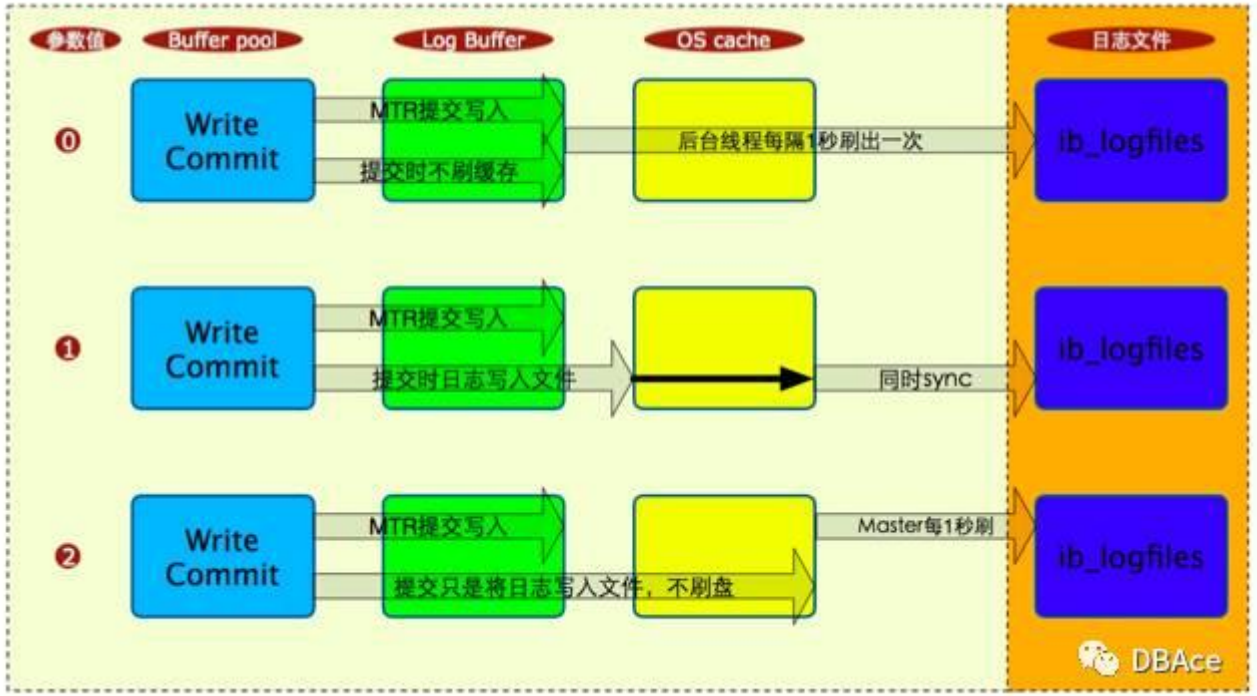
在主库上创建一张表，然后由一个进程间隔一定时间不断地去更新这张表记录的当前主库的时间戳
由另一个进程间隔一定时间不断地去从库查询这张表记录的主库时间戳，再跟当前时间戳相减得到主从复制的延迟

5. Crash Safe(InnoDB)

- 崩溃恢复机制
  - innodb_fast_shutdown=0|1|2
  - innodb_force_recovery=[0-6]
  - checkpoint: lsn
- 在恢复开始前事务有以下几种状态　　Xid
  - InnoDB中已经提交
    根据前面2PC的过程，可知Binlog中也一定记录了该事务的的Events。所以这种事务是一致的不需要处理
  - InnoDB中是prepared状态，Binlog中有该事务的Events
    需要通知InnoDB**提交**这些事务
  - InnoDB中是prepared状态，Binlog中没有该事务的Events
    因为Binlog还没记录，需要通知InnoDB**回滚**这些事务
  - Before InnoDB Prepare
    事务可能还没执行完，因此InnoDB中的状态还没有prepare。根据2PC的过程，Binlog中也没有该事务的events。 需要通知InnoDB回滚这些事务

5.1 Crash Safe Master

- 崩溃恢复机制
  - Innodb_flush_log_at_trx_commit

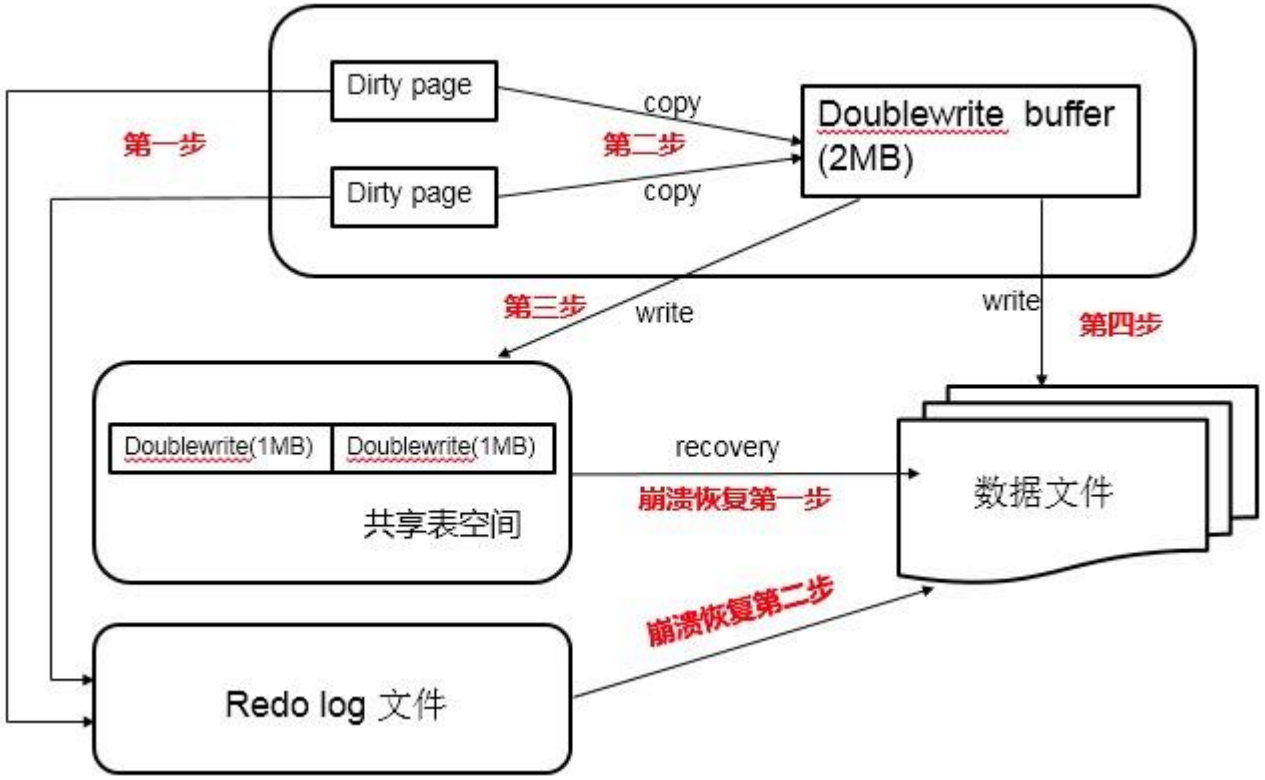


思考：

1. 什么情况下会出现binlog写入了，但是实际这条数据不存在库中？
2. 开启增强半同步之后，innodb_flush_log_at_trx_commit=2 不会丢失数据。为什么？

5.1 Crash Safe Master

- 崩溃恢复机制
  - 刷脏页出现 partial write问题
  - Double write



思考：DW什么情况下可以关掉？

5.2 Crash Safe slave

- slave crash safe

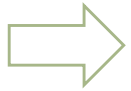
```
mysql> show slave status\G
*********************** 1. row ***********************
         Slave_IO_State: Waiting for master to send event
        Master_Log_File: mysql-bin.014670    -- IO-thread读取主库binlog的文件名
    Read_Master_Log_Pos: 181716556           -- IO-thread读取主库binlog的文件偏移
         Relay_Log_File: slave-relay.028871  -- SQL-thread读取relay-log的文件名
          Relay_Log_Pos: 166693104           -- SQL-thread读取relay-log的文件偏移
  Relay_Master_Log_File: mysql-bin.014670    -- SQL-thread读取的事件对应主库的文件名
       Slave_IO_Running: Yes
      Slave_SQL_Running: Yes
    Exec_Master_Log_Pos: 166692941           -- SQL-thread读取的事件对应主库的文件偏移
  Seconds_Behind_Master: 0
...
```

- 写文件pos与sql apply不是一个原子操作
  - sync_master_info=10000
  - sync_relay_log_info=10000

5.2 Crash Safe slave

- slave crash safe
  - master_info_repository=TABLE
  - relay_log_info_repository=TABLE -- 表myysql.slave_relay_log_info
  - relay_log_recovery=ON
- SQL-Thread

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



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

- IO-Thread
  - IO 线程启动时，会从 slave_relay_log_info 表中读取需要读取的位点信息，然后从主库拉取数据
  - 无需设置 master_info_repository=TABLE

MySQL高可用方案—阿里RDS

- 阿里云RDS
  - 5.6高可用版: 优化半同步复制