滴滴出行

滴滴一下美好出行

MySql杂谈

刘寅

2016-06-12

大纲

滴滴出行
滴滴一下美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

大纲

滴滴出行
滴滴一下美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

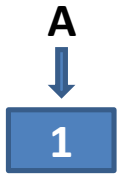
原子(A)

滴滴出行
滴滴一下 美好出行

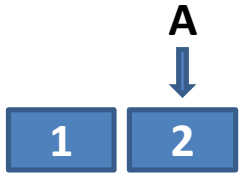
原子 == 保留旧值

A = 1;

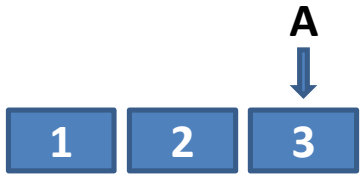
Begin transaction



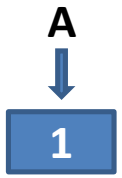
A = 2;



A = 3;



Rollback



大纲

滴滴出行
滴滴一下 美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

并发控制(I)

滴滴出行
滴滴一下美好出行

- 锁

  read uncommitted

  read commit

  repeatable read

  serializable

- MVCC

  undo log

  read view

并发控制(I)(锁)

滴滴出行
滴滴一下美好出行

- read uncommitted

  X锁遵守严格两阶段封锁；不加S锁
- read commit

  X锁遵守两阶段封锁；S锁在单行sql执行前获得，执行后释放
- repeatable read

  X锁和S锁均遵守严格两阶段封锁；为避免幻读，加next key锁
- Serializable

  单一事务队列。。。

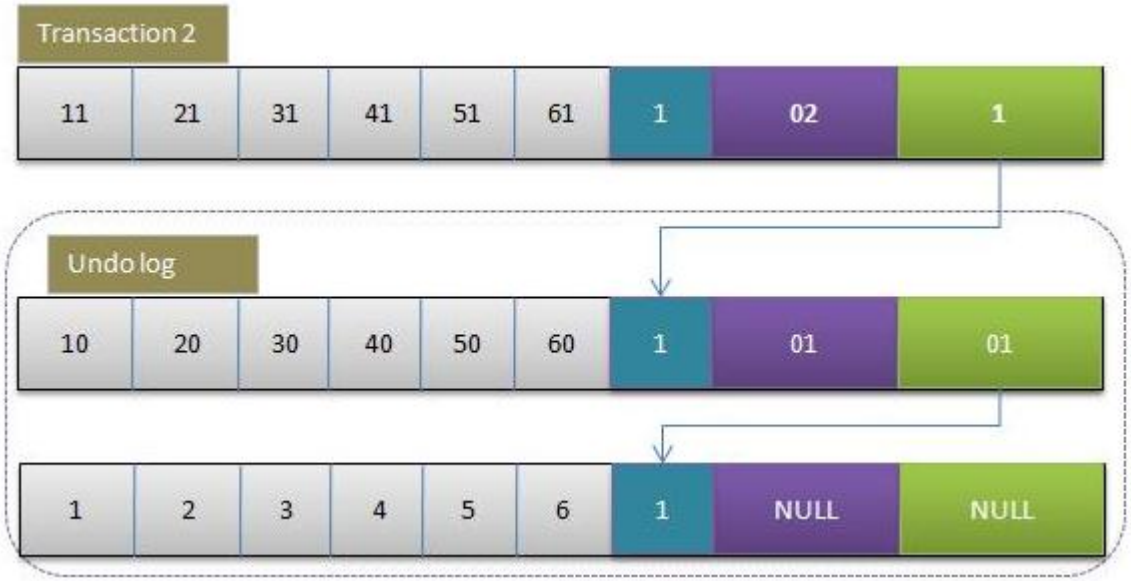
并发控制(I)(MVCC)

滴滴出行
滴滴一下 美好出行

MVCC解决什么问题?

- 读写并发/读写互不阻塞。
- 同时隔离级别较高，数据一致性较lock方式高

Undo Log



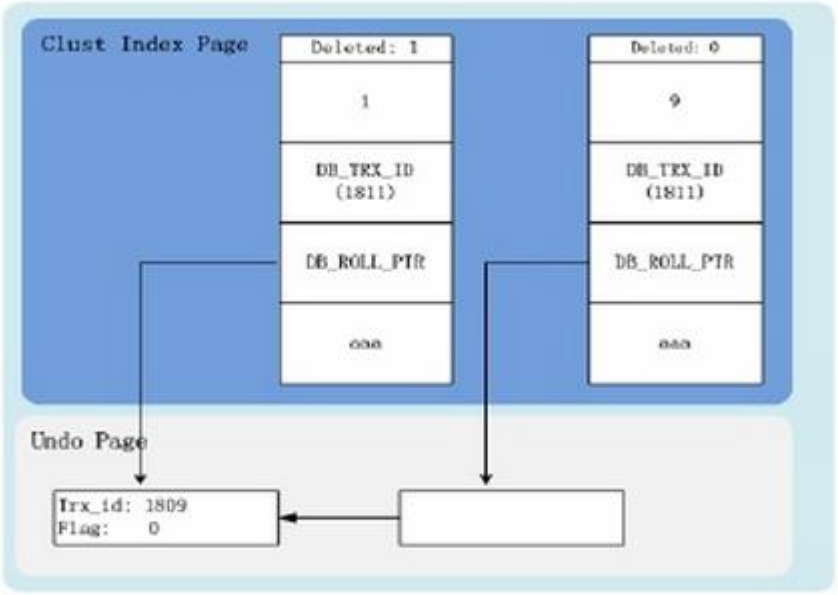
并发控制(I)(MVCC)

滴滴出行
滴滴一下美好出行

Undo Log

- **更新主键**

update test set id = 9 where id = 1;



- 旧记录标识为删除
- 插入一条新纪录
- 新旧记录前项均进入回滚段

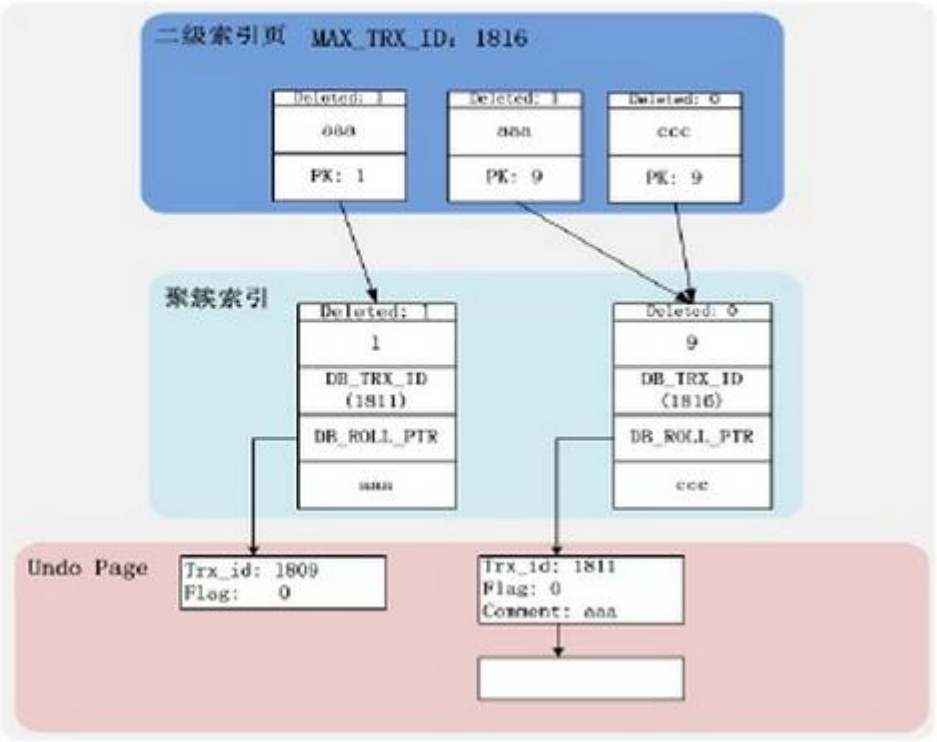
并发控制(I)(MVCC)

滴滴出行
滴滴一下 美好出行

Undo Log

- 更新非主键

update test set comment = 'ccc' where id = 9;

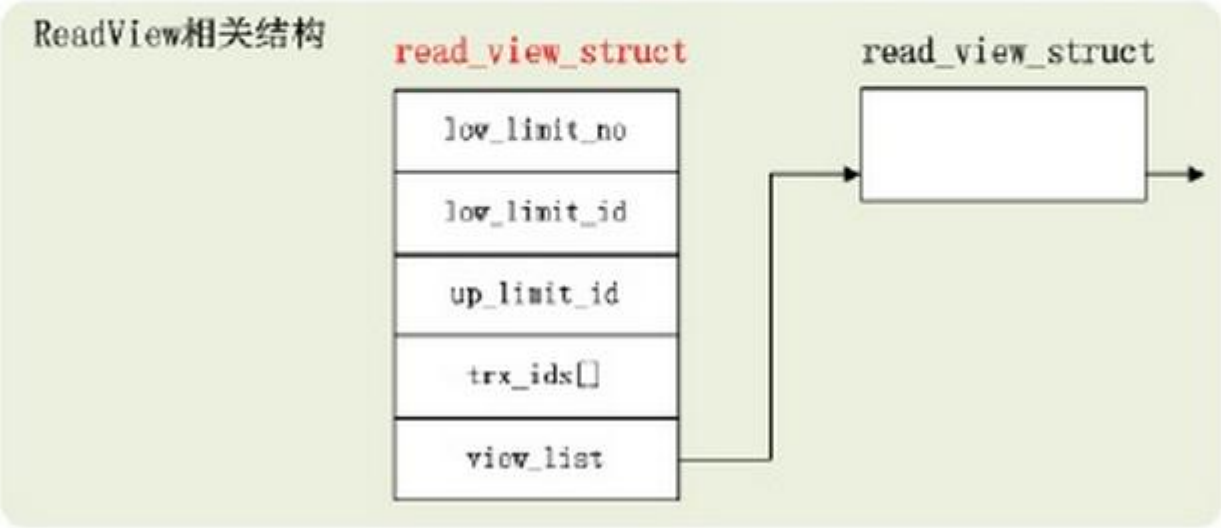


并发控制(I)(MVCC)

滴滴出行
滴滴一下 美好出行

Read View

所谓ReadView，是一个事务的集合，这些事务在ReadView创建时是活跃的(未提交/回滚)



并发控制(I)(MVCC)

滴滴出行
滴滴一下美好出行

Read View

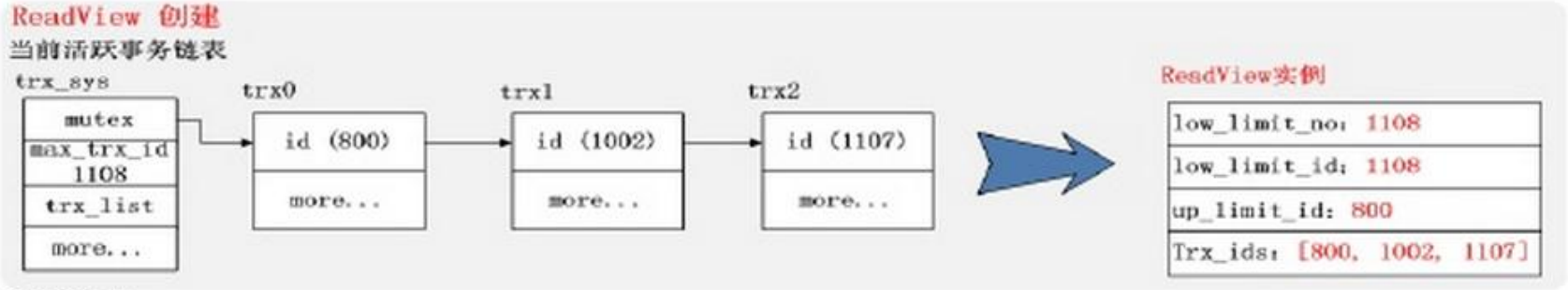
- read_view_struct
  - **low_limit_no**
    - 提交时间早于此值(trx->no < low_limit_no)的事务，可以被purge线程回收
    - low_limit_no = trx_sys->max_trx_id
  - **low_limit_id**
    - >= 此值(trx->id >= low_limit_id)的事务，当前ReadView均不可见
    - low_limit_id = trx_sys->max_trx_id
  - **up_limit_id**
    - < 此值(trx->id < up_limit_id)的事务，当前ReadView一定可见
    - up_limit_id = ReadView创建时系统中最小活跃事务ID
  - **trx_ids[]**
    - 系统中所有活跃事务id组成的数组
- 创建ReadView
  - 获取kernel_mutex
    - 遍历trx_sys的trx_list链表，获取所有活跃事务，创建ReadView
  - Read Committed
    - 语句开始，创建ReadView
  - Repeatable Read
    - 事务开始，创建ReadView

并发控制(I)(MVCC)

滴滴出行
滴滴一下美好出行

Read View

- ReadView创建



- RC VS RR

```
Read Committed

Begin;

Create ReadView1;

Statement 1;

Drop ReadView1;
Create ReadView2;

Statement 2;
...
Drop ReadView2;
Commit;
```

```
Repeatable Read

Begin;

Create ReadView1;

Statement 1;

Statement 2;

...

Drop ReadView1;
Commit;
```

大纲

滴滴出行
滴滴一下 美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

持久(D)

滴滴出行
滴滴一下 美好出行

WAL

- 持久化 == 新值写盘(fsync)

- 如何写盘?

  Fsync的性能， 随机写 vs 顺序写

- Write Ahead Log(WAL)

  写两份，事务提交时，先写redo log，真实数据保存在内存中(Flush List)，后期批量刷数据到盘。延迟数据新值落盘，便于新值落盘merge。

  故障恢复时，重放redo log，将redo log中的内容加载到内存。

持久(D)

滴滴出行
滴滴一下美好出行

脏页刷盘

- 长期不刷脏页到磁盘会咋样?

1.故障恢复阶段时间太长。

2.redo log文件中脏页数据对应的日志大小超过限制，可用日志空间减少

3.可用内存减少

- 怎么办?

问题1和2，Flush List + check point

工作期间将内存中的数据刷到硬盘，并在redo log中插一个断点（checkpoint）；故障恢复阶段从checkpoint重放redo log

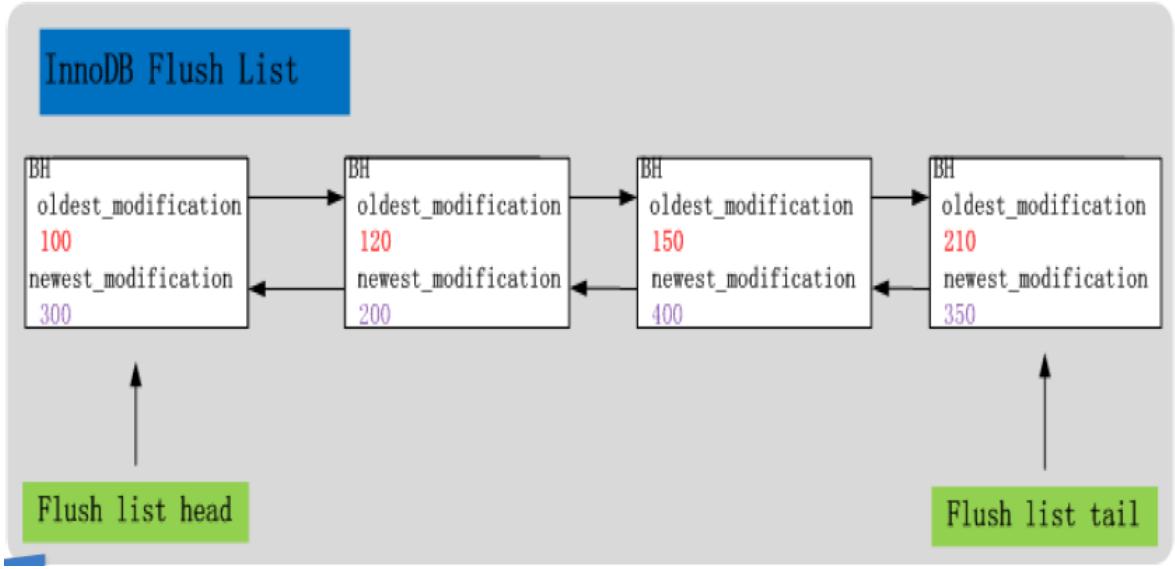
问题3，LRU

两级LRU队列，尾部淘汰。

持久(D)

滴滴出行
滴滴一下美好出行

fuzzy checkpoint(Flush List)



Check point 相关参数

- 可以配置的参数:

innodb_log_file_size，nnodb_max_dirty_pages_pct，innodb_io_capacity

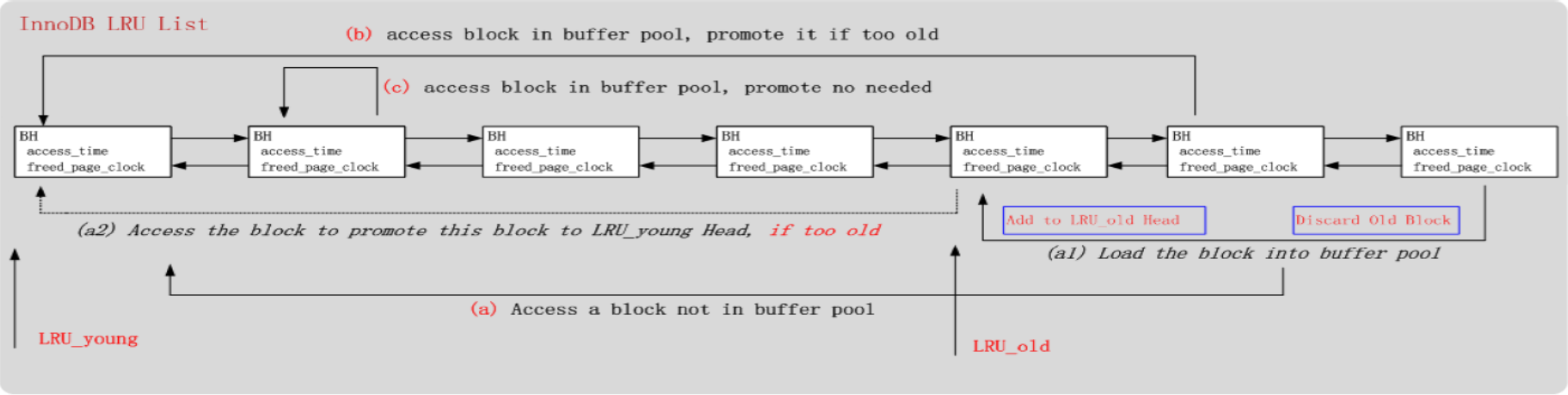
- 貌似写死在代码中了,无法配置

max_modified_age_async, max_modified_age_sync, max_checkpoint_age_async, max_checkpoint_age

持久(D)

滴滴出行
滴滴一下 美好出行

LRU



- 两级队列,LRU_YOUNG与LRU_OLD

  如果在LRU_YOUNG呆久了(资历够了)，就再下次访问时提升到LRU_OLD

  如何衡量资历：1.access_time够老；2.期间evit的page的个数够多，达到LRU_YOUNG长度的1/4

  参数：innodb_old_blocks_time

- 空间不足时，尾部淘汰

持久(D)

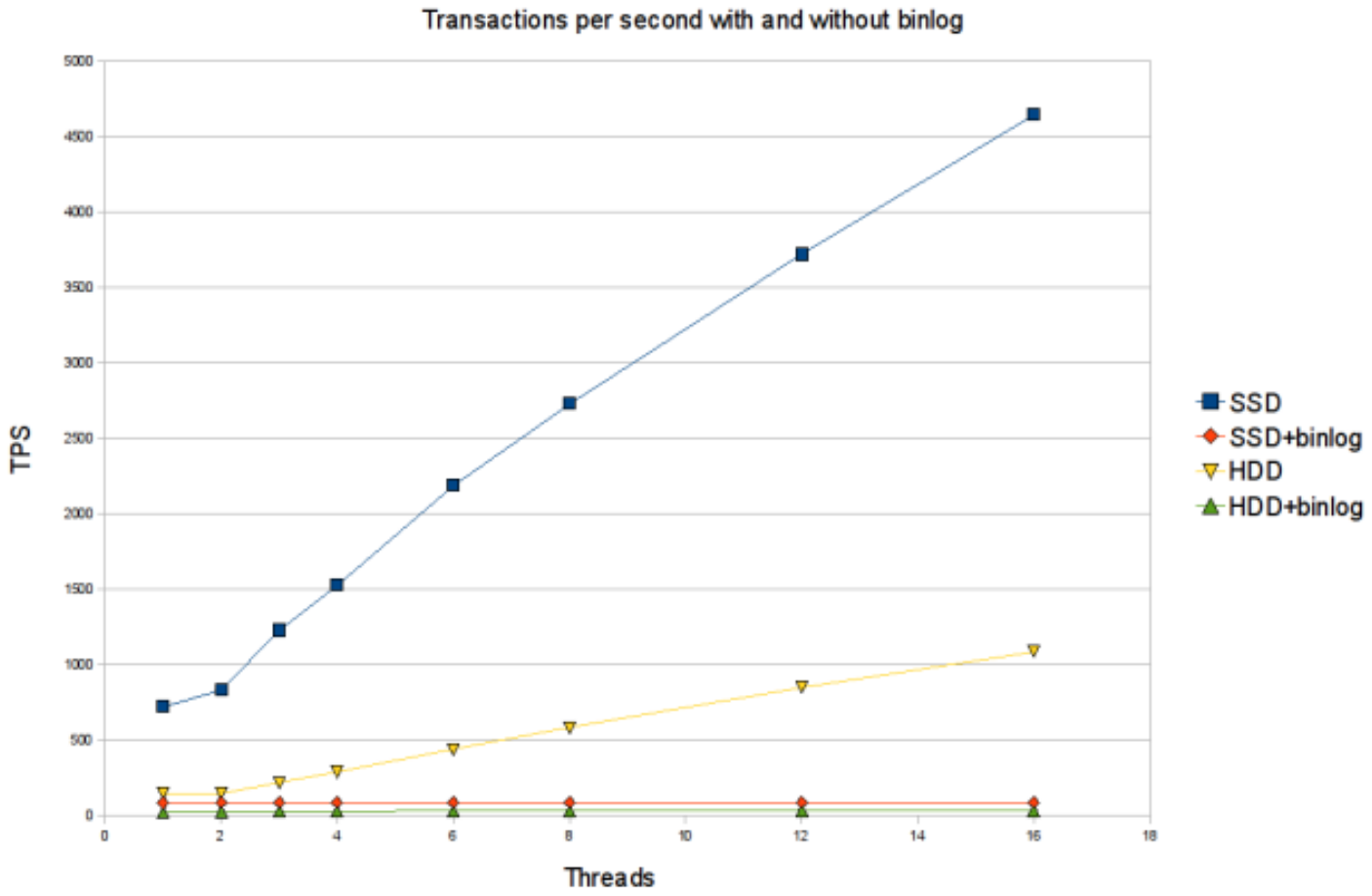
组提交(group commit)

- 组提交之前，多个commit，多次fsync。
- 组提交之后，多个commit，一次fsync。(Binlog_sync=0 或者mysql版本大于5.7)

持久(D)

滴滴出行
滴滴一下美好出行

组提交性能 测试实验



大纲

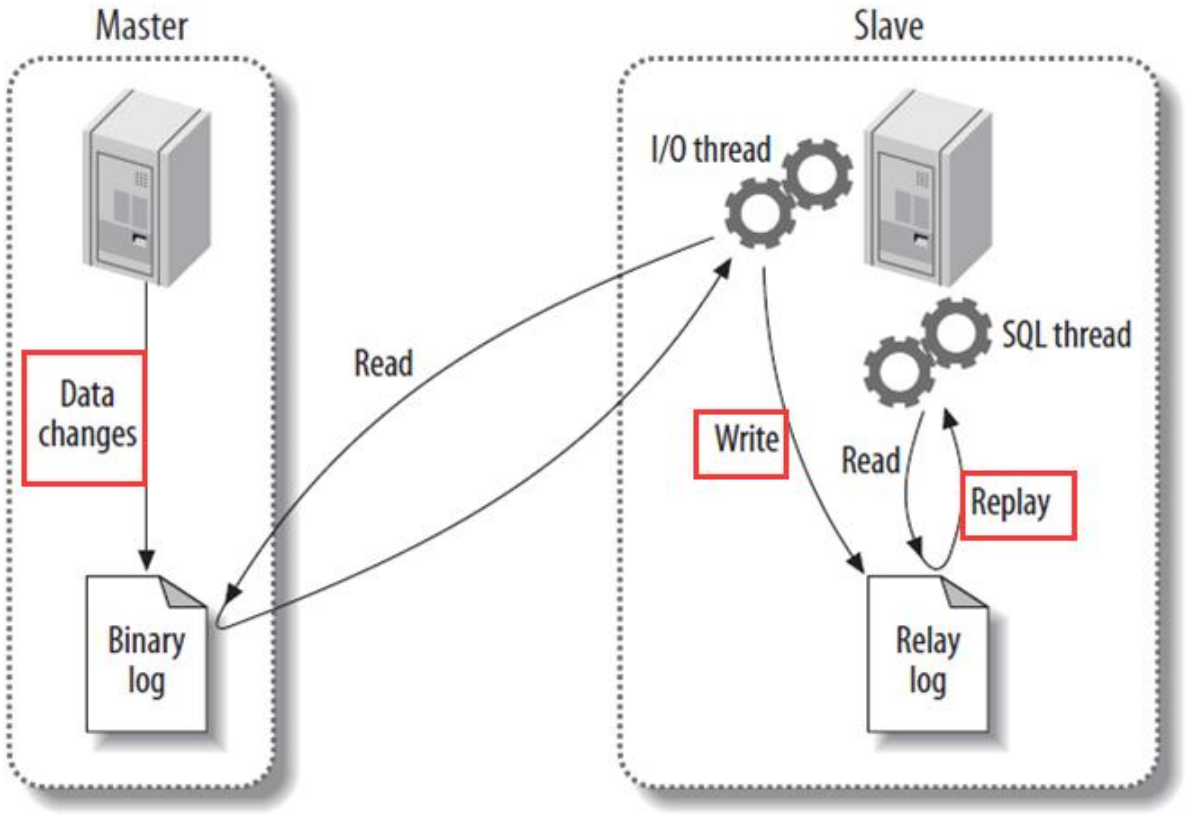
滴滴出行
滴滴一下美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

主从复制

滴滴出行
滴滴一下 美好出行

概览



大纲

滴滴出行
滴滴一下 美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

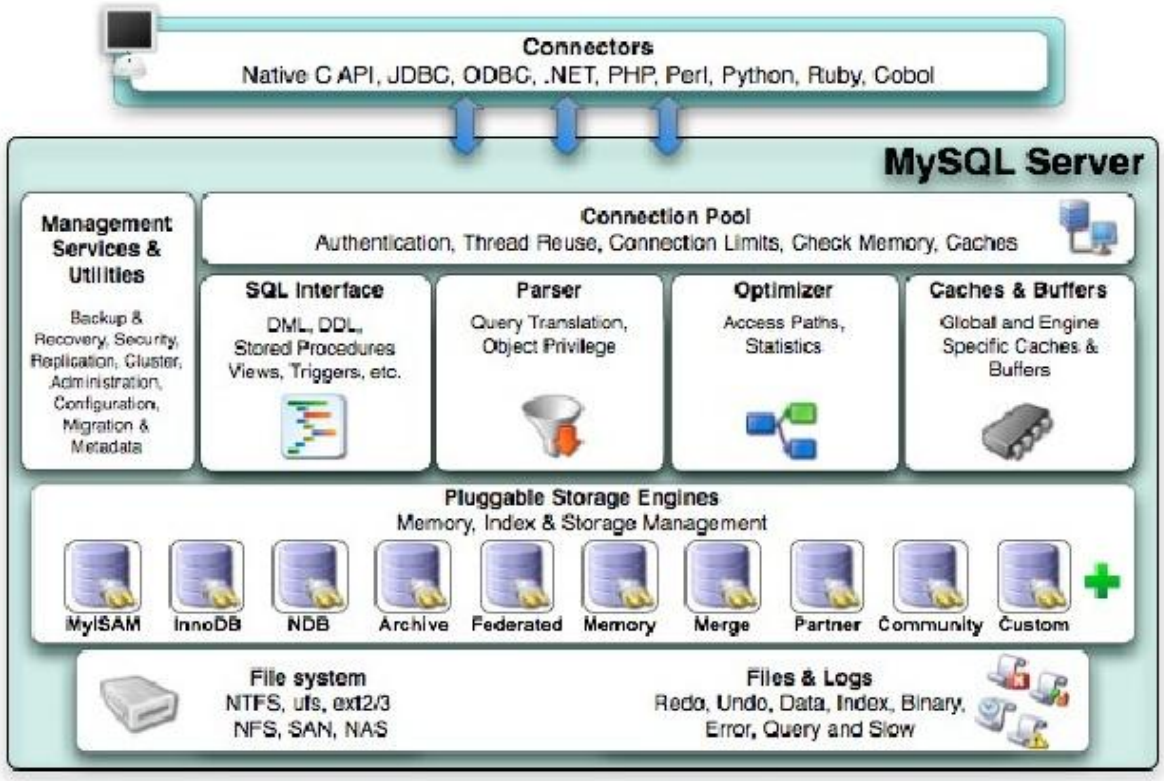
Master组提交(内部XA)

滴滴出行
滴滴一下美好出行

MySql的存储引擎插件式架构

UCONN
CSE 255

MySQL Architecture



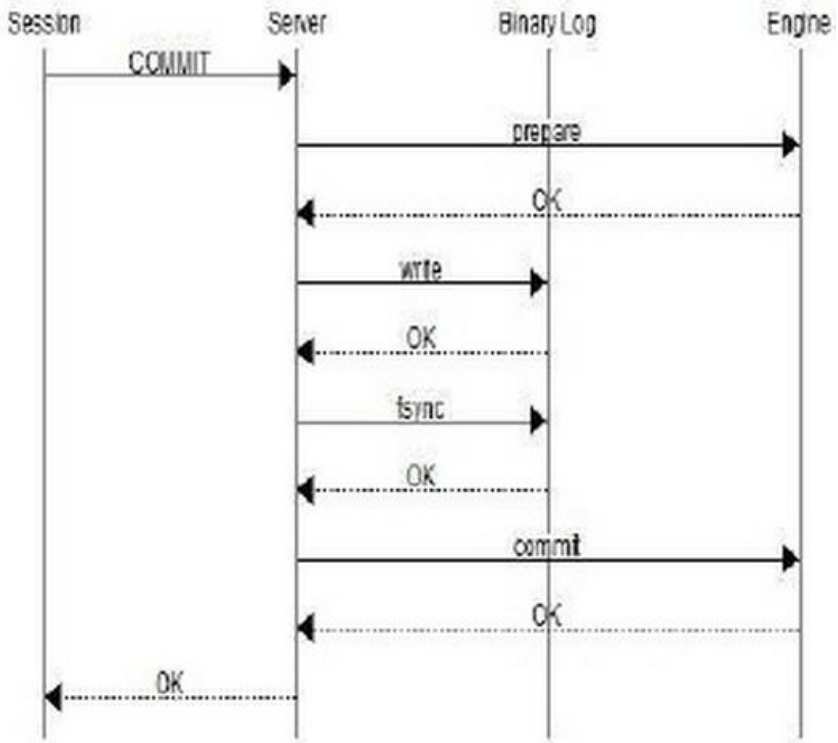
http://dev.mysql.com/doc/refman/5.1/en/images/PSEArch.png

Master组提交(内部XA)

滴滴出行
滴滴一下美好出行

为啥引入XA?

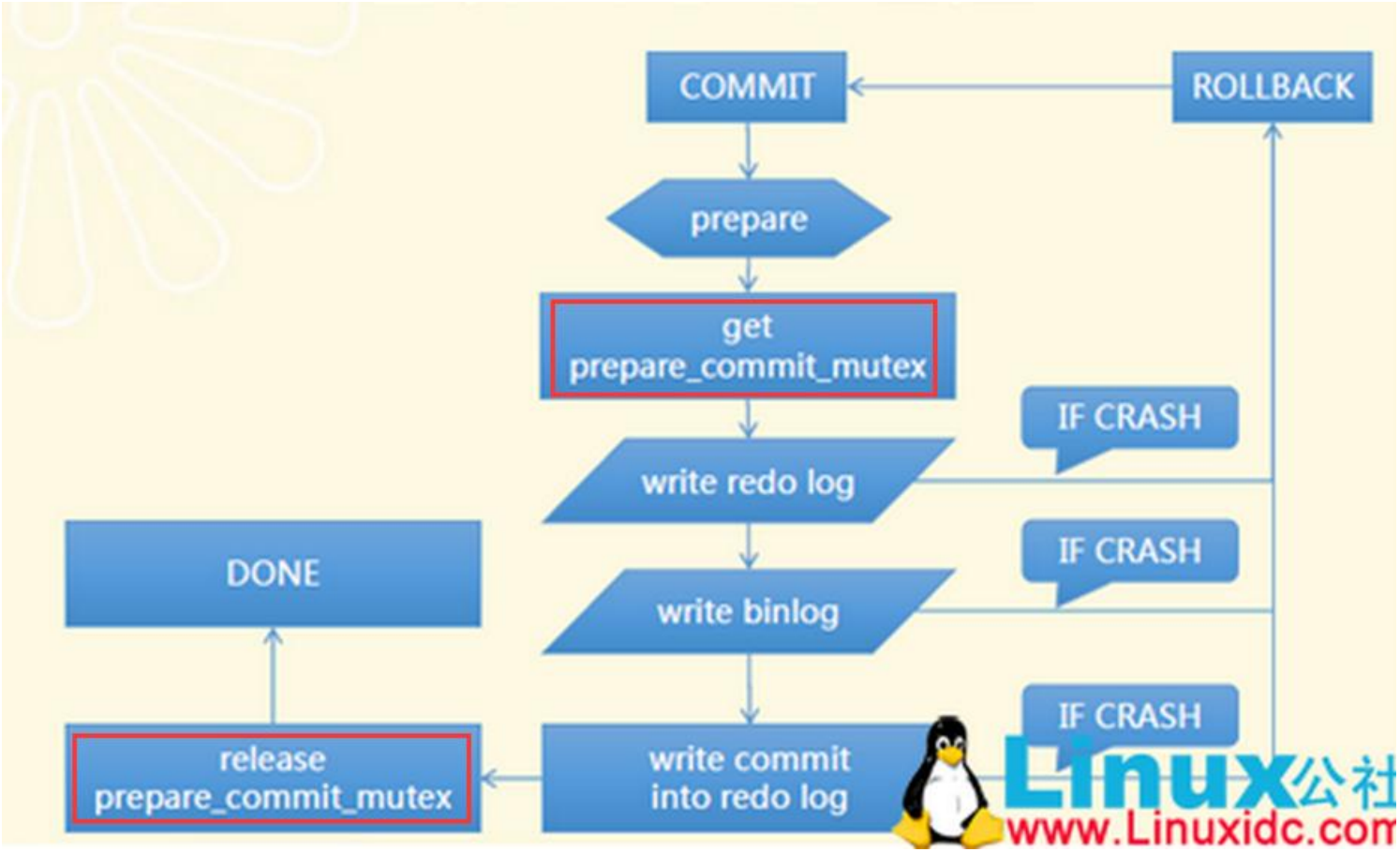
- MySql的存储引擎插件式架构造成binlog和redo log的割裂
- XA保证binlog和redo log之间的一致性，也就是保证master和slave之间的一致性



Master组提交(内部XA)

滴滴出行
滴滴一下美好出行

XA的实现(Mysql5.6之前)



binlog:Binlog_sync=1

Innodb:innodb_flush_log_at_trx_commit=1

Master组提交

滴滴出行
滴滴一下美好出行

XA的实现导致了什么问题（Mysql5.6之前）？

- 不能Group commit了...

  innodb线程争抢锁prepare_commit_mutex，无法并发，只能一个事务一个事务地fsync，性能急剧下降

- 每个事务3次fsync

怎么办？去掉这个prepare_commit_mutex...（Mysql5.7.6/ Mariadb5.3 ）

- Mysql

  5.6：binlog group commit；

  5.7.6 redo log也group commit了,只fsync两次

- Mariadb

  此外，#164 试图只fsync一次，而不是3次

Master组提交

滴滴出行
滴滴一下 美好出行

- Mariadb #164 保持binlog顺序和redo log的提交顺序，然后replay binlog（可能此思路废弃了，因为bin log不是可重复执行的）

WL#164

在Group Commit的基础上，MariaDB再接再厉，推出WL#164。将一次事务提交需要的3次fsync，降低为1次– binlog fsync。用户可以在将参数innodb-flush-log-at-trx-commit设置为{0,2}时，达到与该参数为1时同样的可靠性，不会丢失已提交更新。实现方案也较为简洁，在原有XA recover的基础上，新增了一个处理【binlog存在，但是InnoDB prepare log不存在】的情况，此时需要根据binlog重做(re-play)一遍即可(类似于slave根据binlog恢复的情形)。

在WL#164之后，binlog与InnoDB redo log之间的关系，存在以下几种组合：

- Binlog与commit log同时存在 ——》 no operation in crash recovery
- Binlog与prepare log同时存在 ——》 commit
- Binlog存在，prepare log不存在 ——》 re-play binlog
- Binlog不存在，prepare log存在 ——》 rollback

(Link)

Probably not 100% trustworthy.

I'm actually a bit scared of this approach. I might be over-reacting, but I distrust binary logs as a way to reliably replay what happened on a system. There are so many edge cases that I've observed in production causing different results.

I would be more in favor of something like this: if a transaction isn't in InnoDB's transaction log, discard it and roll back the state of the whole system to the last committed transaction inside InnoDB. Forget about what's in the binary log (or just truncate it).

I know this is not the right solution, and the binary log replay idea is conceptually the right approach for group commit across all engines. But the binary log is not ACID, results aren't always repeatable, and I trust InnoDB a lot more than I trust the binary log. The binary log, whether row-based or statement-based, is logical. InnoDB's redo logs are physical. I trust that.

Maybe the user can be offered the choice in a configuration option.
(Reply) (Thread)

大纲

滴滴出行
滴滴一下 美好出行

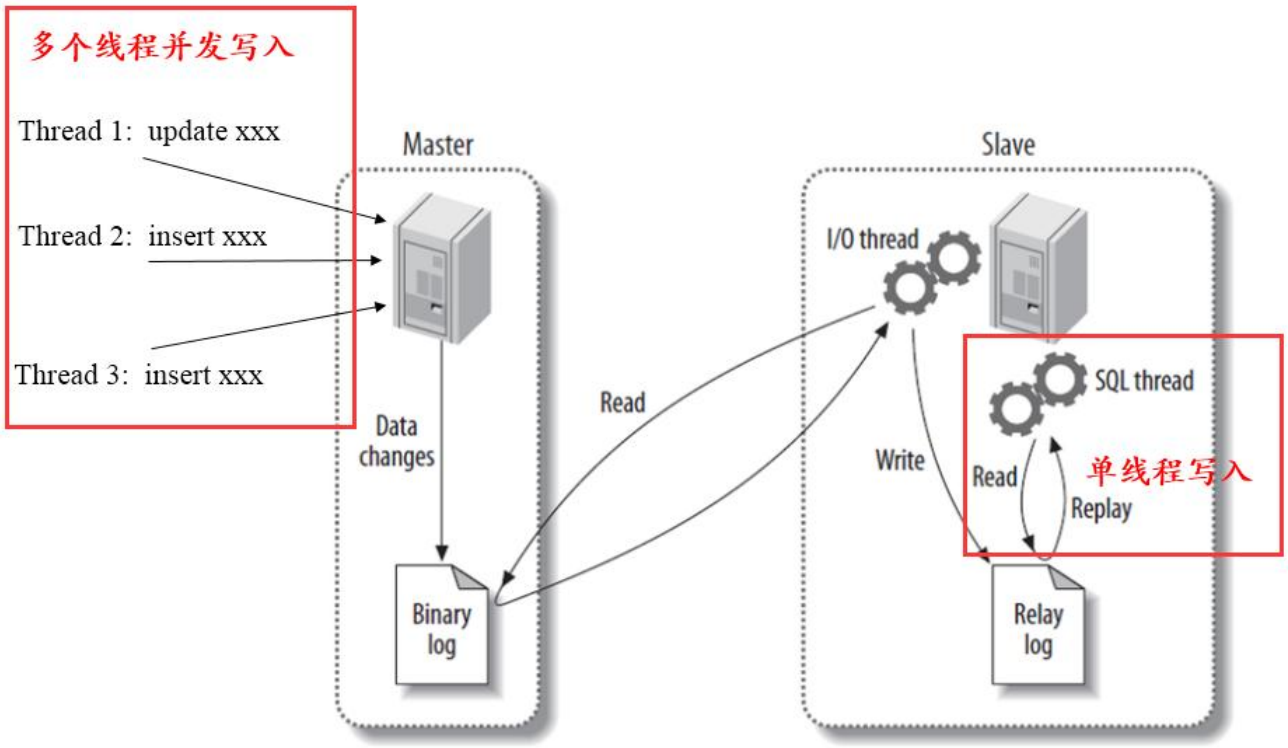
- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

Slave并行复制

滴滴出行
滴滴一下 美好出行

为啥需要并行?

- Slave跟不上master：SQL thread是单线程的



Slave并行复制

滴滴出行
滴滴一下 美好出行

为啥之前版本不引入并行?

- 需要保证互相冲突的事务的commit顺序在master和slave之间是一致的

怎样引入并行?

- 保守方式：不冲突事务可以并发执行。

  Mysql 5.6：假定不同库之间独立，可以并发。没细看，总感觉哪儿不对。。。

  Mysql 5.7/Mariadb 10：主库上可以group commit，所以这一个group的事务是不冲突的，可以并发执行。

- 激进方式：先执行，但保证commit顺序相同，如果两个事务冲突，就回滚commit序号比较大的那个事务。

  Mariadb 10独有

Slave并行复制(保守模式)

滴滴出行
滴滴一下 美好出行

- Binlog内容

```
root@localhost:~# mysqlbinlog mysql-bin.0000006 | grep last_committed
#150520 14:23:11 server id 88 end_log_pos 259 CRC32 0x4ead9ad6 GTID last_committed=0 sequence_number=1
#150520 14:23:11 server id 88 end_log_pos 1483 CRC32 0xdf94bc85 GTID last_committed=0 sequence_number=2
#150520 14:23:11 server id 88 end_log_pos 2708 CRC32 0x0914697b GTID last_committed=0 sequence_number=3
#150520 14:23:11 server id 88 end_log_pos 3934 CRC32 0xd9cb4a43 GTID last_committed=0 sequence_number=4
#150520 14:23:11 server id 88 end_log_pos 5159 CRC32 0x06a6f531 GTID last_committed=0 sequence_number=5
#150520 14:23:11 server id 88 end_log_pos 6386 CRC32 0xd6cae930 GTID last_committed=0 sequence_number=6
#150520 14:23:11 server id 88 end_log_pos 7610 CRC32 0xa1ea531c GTID last_committed=6 sequence_number=7
...
```

Slave并行复制(保守模式)

滴滴出行
滴滴一下 美好出行

Group commit逐层分裂

- 复制路径 M→S1→S2

M

```
   ----Time---->
T1:       B---C
T2:       B---C
T3:   B-------C
T4:   B-------C
```

S1

```
   ----Time---->
T1:   B---C
T2:   B---C
T3:   B-------C
T4:   B-------C
```

S2

```
   ----Time--------->
T1:   B---C
T2:   B---C
T3:       B-------C
T4:       B-------C
```

Slave并行复制(保守模式)

滴滴出行
滴滴一下 美好出行

延迟提交

```
   ------Time----->
T1:  B----C
T2:        B----C
```

```
   -----Time----->
T1:  B---- . . C
T2:        B----C
```

- Mysql 5.7：

  binlog_group_commit_sync_no_delay_count

  binlog_group_commit_sync_delay

- Mariadb 10：

  binlog_commit_wait_count

  binlog_commit_wait_usec

大纲

滴滴出行

滴滴一下美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

SQL线程同步点

滴滴出行
滴滴一下 美好出行

crash-safe slaves

- Mysql5.6 之前

  slave是无法保证同步点和实际执行的sql的点一致的。因为同步点信息是保存在文件relay-log.info中；实际执行结果是保存在redo log中的。

- Mysql5.6 之后

  可以保证一致。

  sync_relay_log_info = 1， relay_log_info_repository = TABLE，Table是事务引擎（innodb）

SQL线程同步点

滴滴出行
滴滴一下 美好出行

crash-safe slaves

In pseudo-code, instead of having:

```
1  START TRANSACTION;
2  -- Statement 1
3  -- ...
4  -- Statement N
5  COMMIT;
6
7  -- Update replication info files
```

The server now behaves as if we had:

```
1  START TRANSACTION;
2  -- Statement 1
3  -- ...
4  -- Statement N
5  -- Update replication info
6  COMMIT;
```

单点，group commit又挂了。。。

大纲

滴滴出行
滴滴一下 美好出行

- 单机事务
  - 原子(A)
  - 并发控制(I)
  - 持久(D)
- 主从复制
  - master组提交
  - slave并行复制
  - 同步点
  - 半同步

半同步

滴滴出行
滴滴一下 美好出行

半同步是啥?

- Master等待直至至少一个slave io线程接收到binlog后，向客户端返回commit成功。

解决了啥问题?

- 方便DBA操作Failover/主从切换

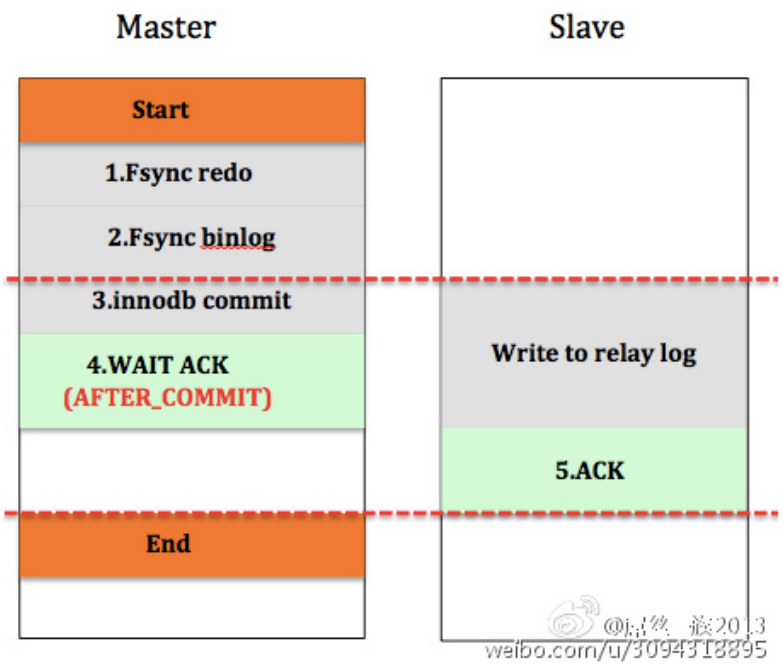
半同步

滴滴出行
滴滴一下 美好出行

参数rpl_semi_sync_master_wait_point

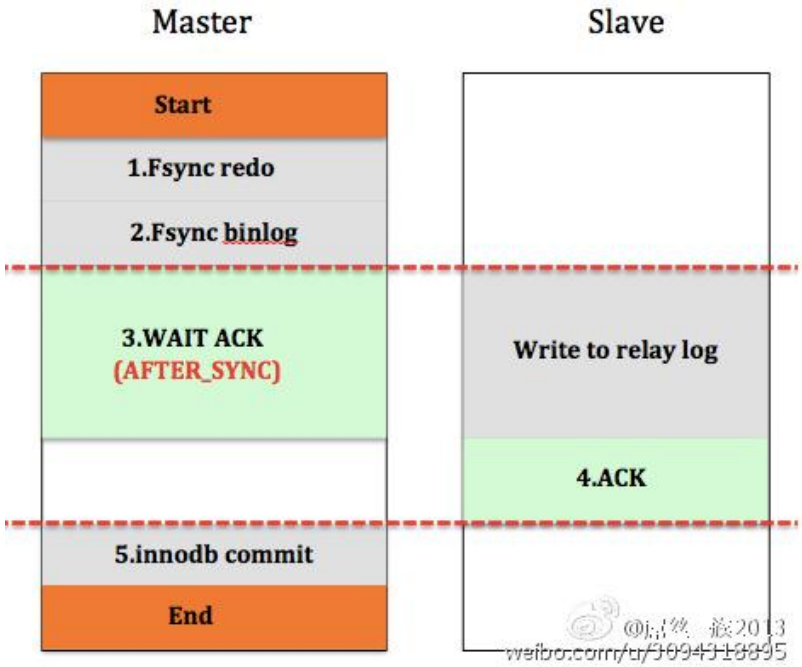
AFTER_COMMIT模式

Mysql 5.7之前



AFTER_SYNC模式

Mysql 5.7



主从切换后，其他client可能看到不一致

滴滴出行
滴滴一下美好出行

Q & A

刘寅

2016-06-14