

DTCC

2017第八届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2017

腾讯CDB的核弹头-TXSQL

张青林

自我介绍

姓名：张青林

公司：Tencent

工作：MySQL kernel dev @ TEG.CDB

职责：性能优化、功能开发、线上问题定位及解决

邮箱：musazhang@tencent.com

Agenda

- ✓ TXSQL 概览
- ✓ TXSQL 内核研发
- ✓ TXSQL 云上实践
- ✓ TXSQL 未来发展发向

TXSQL 概览

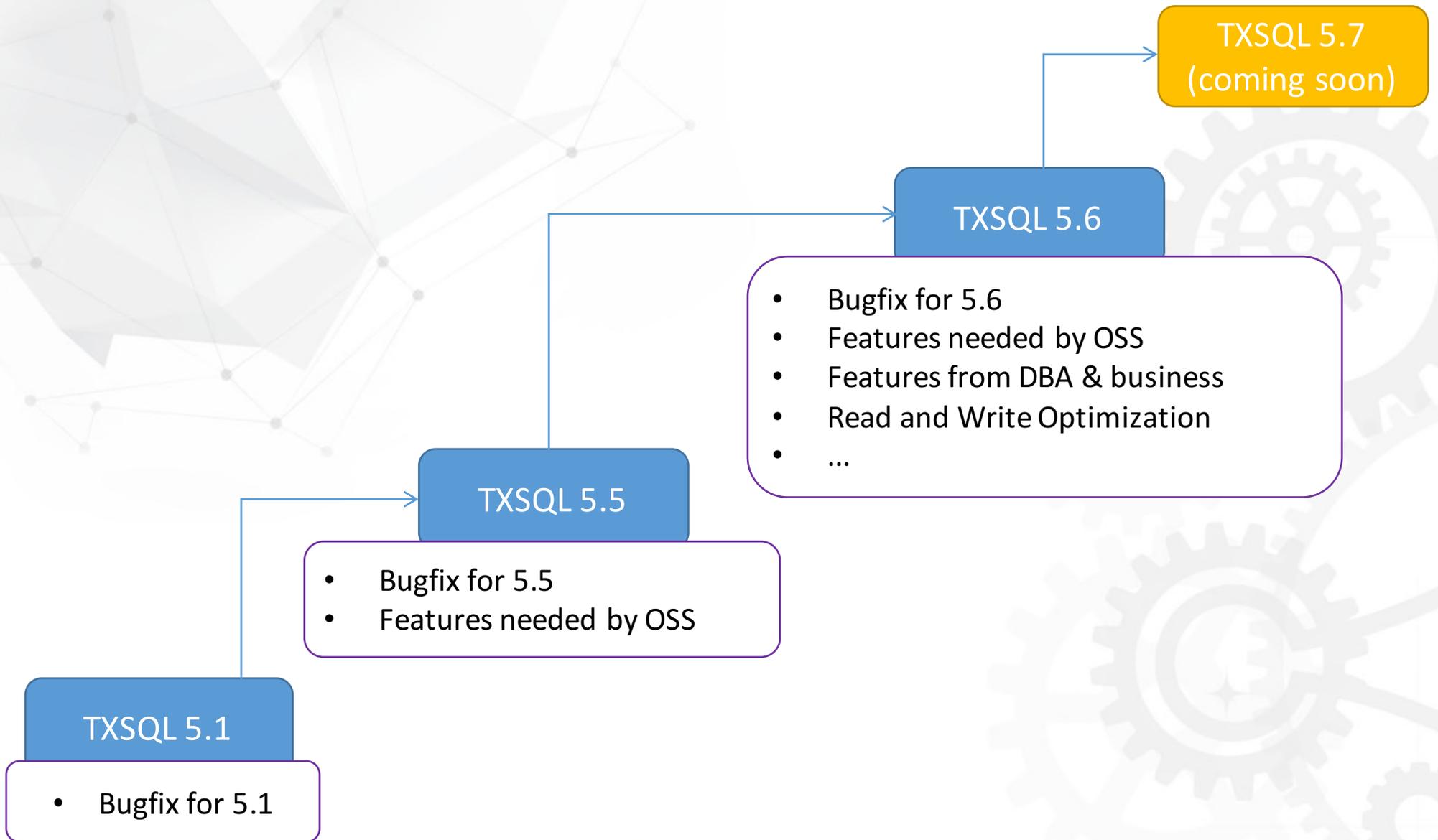
✓ 什么是TXSQL

- ✓ TXSQL = Tencent MySQL
- ✓ 腾讯-TEG-基础架构部-CDB (Cloud DataBase)团队自研 MySQL 分支
- ✓ 腾讯云以及腾讯内部云DBaaS平台官方使用最多的MySQL版本

✓ 为什么有TXSQL

- ✓ MySQL- The most popular database
- ✓ 海量运营的挑战：超大规模、海量开发商、多种业务场景
- ✓ 促进开源数据库技术发展

TXSQL 概览(Cont.1)



TXSQL 内核研发

- ✓ TXSQL 性能改进
- ✓ TXSQL 性能对比
- ✓ TXSQL 功能开发
- ✓ TXSQL 金融特性

TXSQL 性能改进

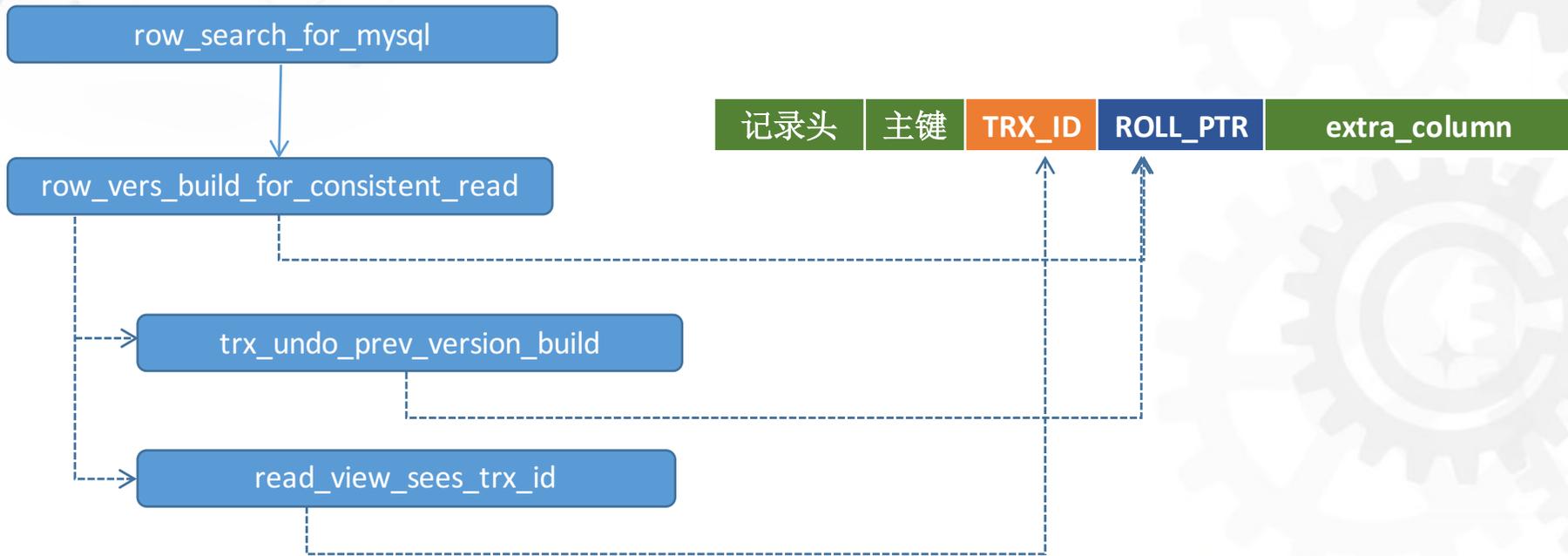
- ✓ TXSQL read view 优化
- ✓ TXSQL redo log 写优化
- ✓ TXSQL redo log 双缓冲区
- ✓ 隐式锁转化
- ✓ 主库 Lock_log 锁拆分

TXSQL read view 优化

read view 是什么

- low_limit_no, used for purge
- low_limit_id, trx whose trx_id >= low_limit_id is invisible
- high_limit_id, trx whose trx_id < high_limit_id is visible

MySQL MVCC



TXSQL read view 优化(Cont.1)

事务创建时的步骤如下:

1. 对 `trx_sys->mutex` 全局上锁;
2. 顺序扫描 `trx_sys->rw_trx_list`, 对 `read_view` 中的元素分配内存并进行赋值;
3. 将新创建的 `read_view` 添加到有序列表 `trx_sys->view_list` 中;
4. 释放 `trx_sys->mutex` 锁;

read view 创建存在的问题?

- ✓ 整个创建过程一直持有 `trx_sys->mutex` 锁;
- ✓ 需要遍历 `trx_sys->trx_list` (5.5) 或 `trx_sys->rw_list` (5.6);
- ✓ `read_view` 的内存在每次创建中被分配, 事务提交后被释放;
- ✓ 并发较大, 活跃事务链表过长时, 会在 `trx_sys->mutex` 上有较大的消耗;

TXSQL read view 优化(Cont.2)

TXSQL 解决上述问题的方法

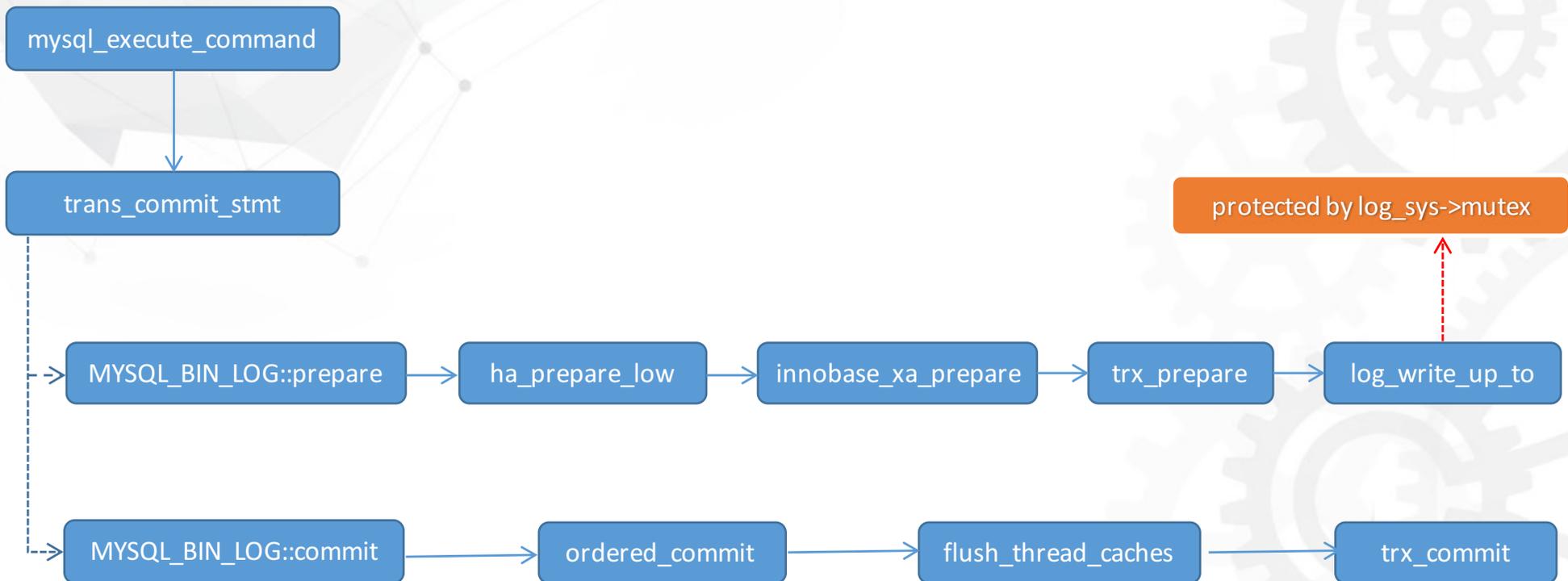
TXSQL backport percona read view 相关修改到 TXSQL 中，并参照 5.7 的实现，在 5.6 中将 ro_trx_list 移除，主要包括以下修改：

- ✓ 在 trx_sys 下维护一个全局的事务ID的有序集合；
- ✓ 在 trx_sys 下维护一个有序的已分配序列号的事务列表，记录拥有最小序列号的事务，供 purge 时使用；
- ✓ 减少不必要的内存分配；
- ✓ 参照 5.7 的实现，在 5.6 中将 ro_trx_list 移除；

Redo log 优化背景

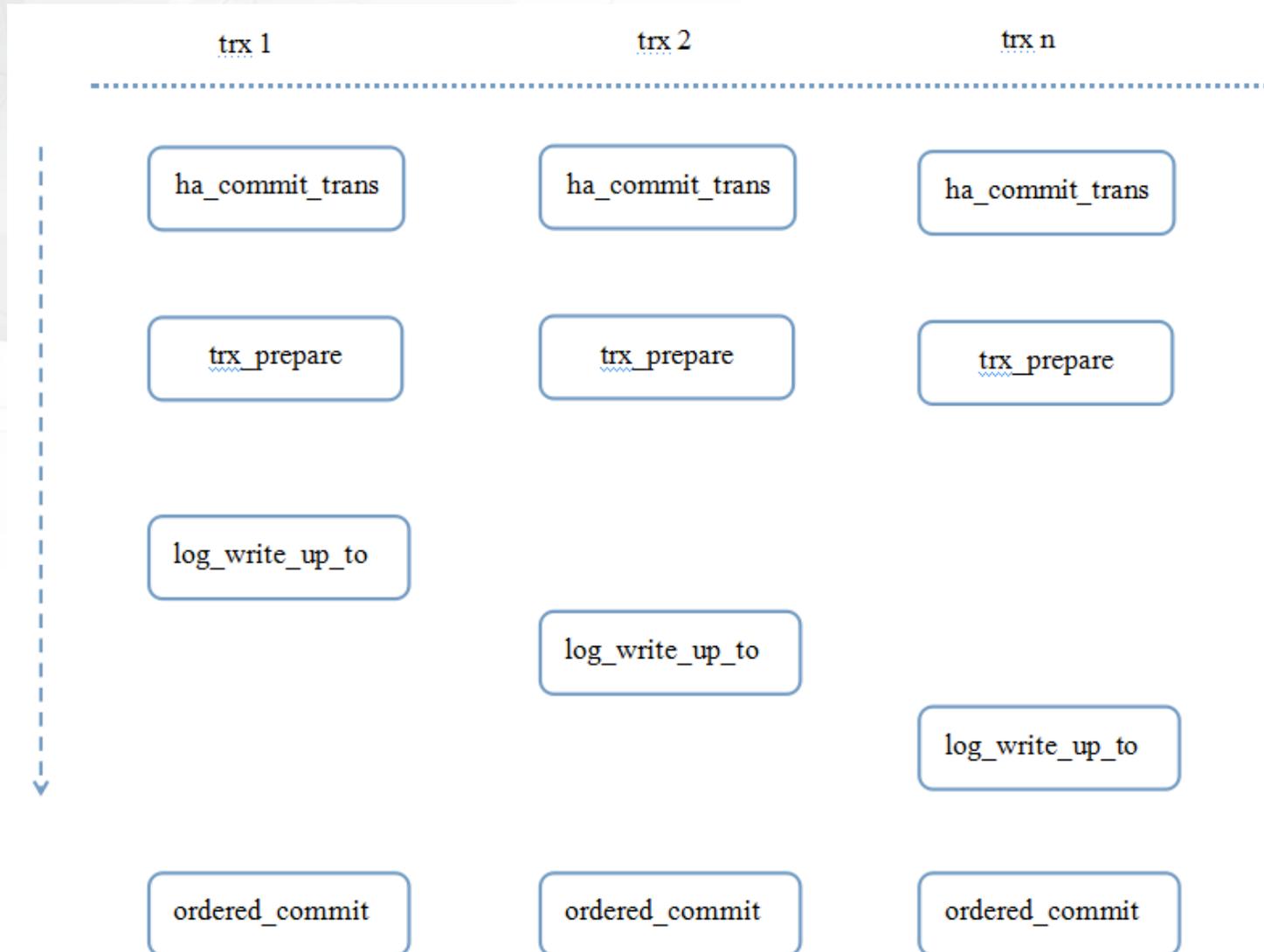
事务提交过程分析

MySQL 是典型的两阶段提交，事务在提交时，会首先将 redo log 落盘，然后落盘 binlog，其过程如下所示：



Redo log 优化背景(Cont.1)

TXSQL redo log 过程分析

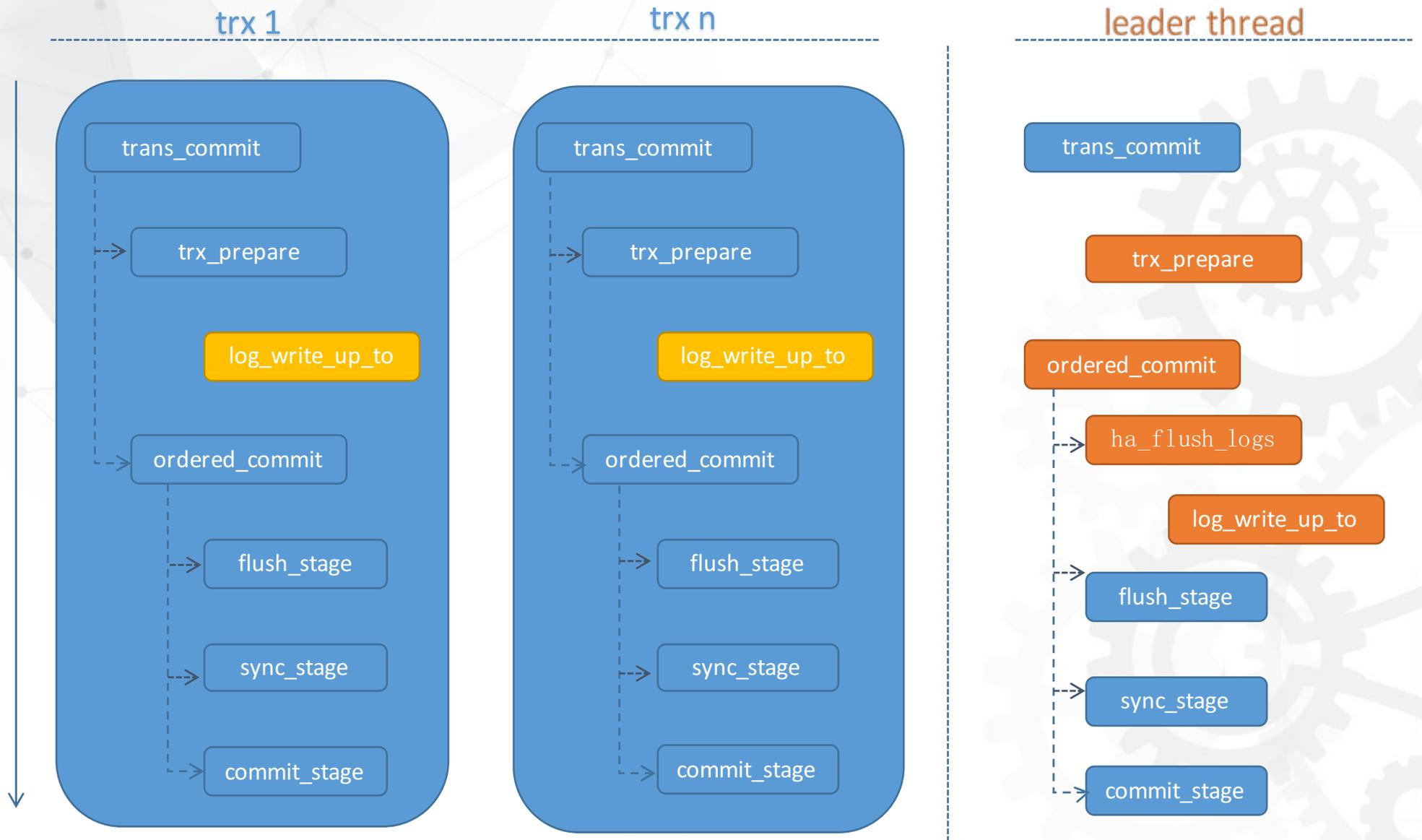


Redo log 优化背景(Cont.2)

MySQL 在 crash recovery 的过程如下:

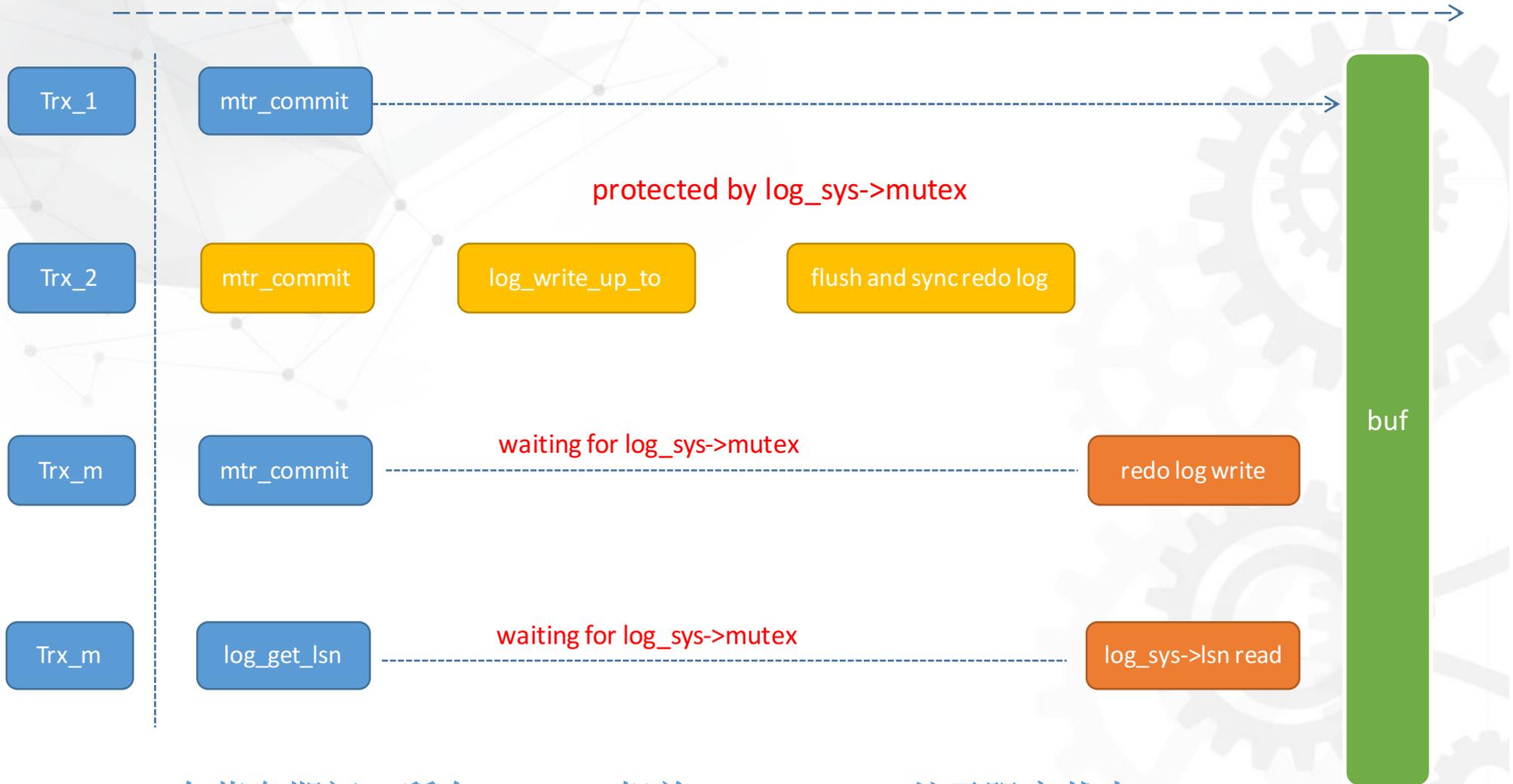
1. 从 logfile 中读取最后的 checkpoint 位点信息, 然后从该位置开始应用 redo 日志;
2. 从 undo 中读取回滚段并初始化读写事务链表 rw_trx_list, 对 active 的事务进行回滚, 详情可参考: `trx_sys_init_at_db_start()` 函数, 对 prepared 事务进行保留, 以在 binlog 文件读取后再决定是否 commit 或者 rollback;
3. 扫描最后的一个 binlog 文件, 内部 XA 事务构建 xids 的集合 commit_list, 外部 XA 事务插入 `xid_cache`, 详情可以参考: `MYSQL_BIN_LOG::recover()`;
4. 对于内部 XA 事务, 如果其 `xid` 存在于内部读写事务链表 `commit_list` 中则进行提交, 否则进行回滚; 对于外部事务, 则需要打印出相关日志, 并等待命令以进行提交或回滚操作, 其中内部 XA 事物与外部 XA 事务的区别为 `xid` 的名称; 详情可参考 `xarecover_handerton()`、`get_my_xid()` 函数。

TXSQL redo log 写优化 (bug#73202)



TXSQL redo log 双缓冲区

redo log 落盘的过程



redo log 在落盘期间，所有redo log 相关 read & write 处于阻塞状态

TXSQL redo log 双缓冲区 (Cont. 1)

redo log 落盘的过程的优化



redo log 在落盘期间，原有 redo log 相关 read & write 不受影响

TXSQL 隐式锁转化 ([worklog#6899](#))

隐式锁转化的过程

1. 获取锁系统 `lock_sys->mutex`;
2. 获取事务系统 `trx_sys->mutex`;
3. 查找相关事务
4. 释放事务系统 `trx_sys->mutex`
5. 隐式锁转换
6. 释放锁系统 `lock_sys->mutex`;

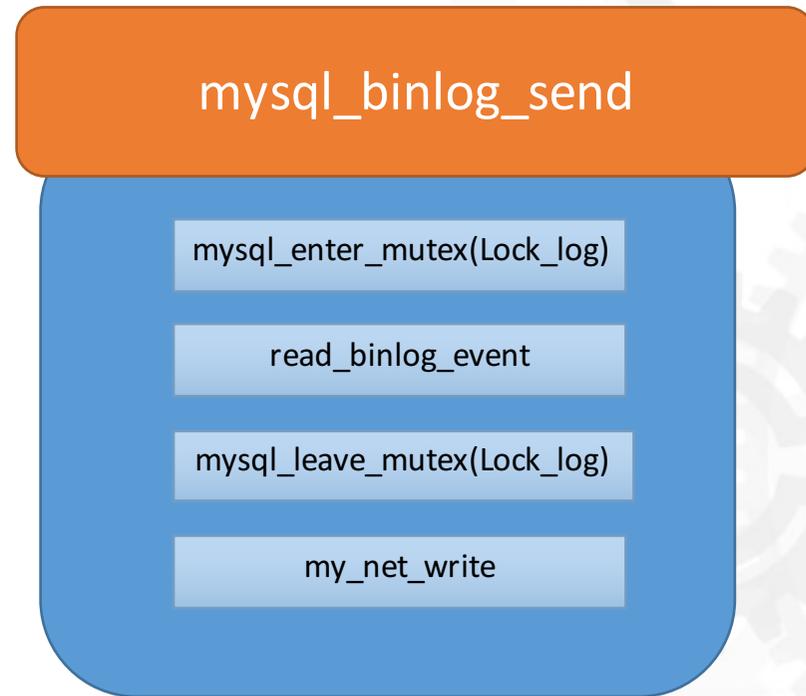
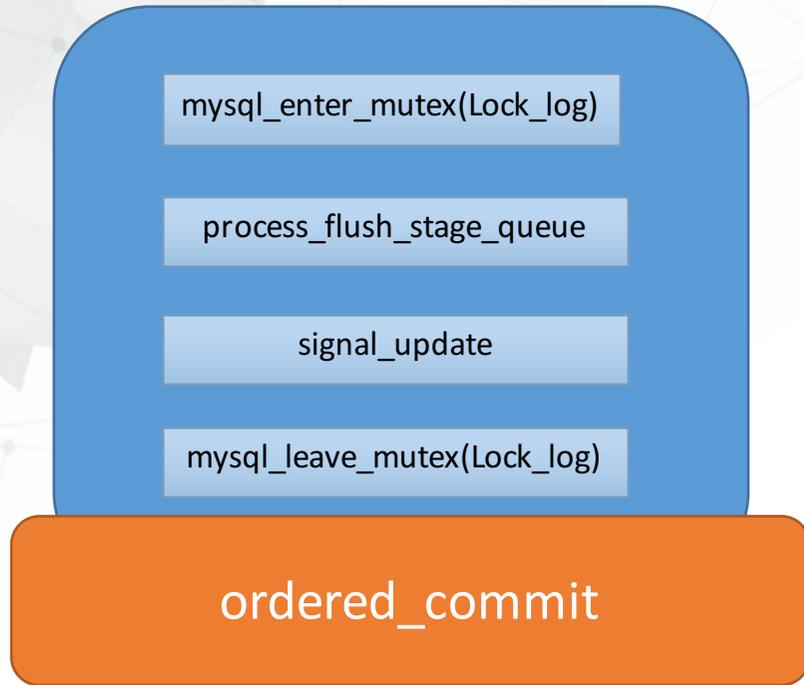
隐式锁转化的优化

1. 获取事务系统 `trx_sys->mutex`;
2. 查找相关事务并进行引用计数
3. 释放事务系统 `trx_sys->mutex`
4. 获取锁系统 `lock_sys->mutex`;
5. 隐式锁转换并进行引用计数的调整
6. 释放锁系统 `lock_sys->mutex`;

其中，涉及的事务在引用计数不为零时，是不允许被销毁的，通过减少 `lock_sys->mutex` 的占用时间来提升性能；

主库 Lock_log 锁拆

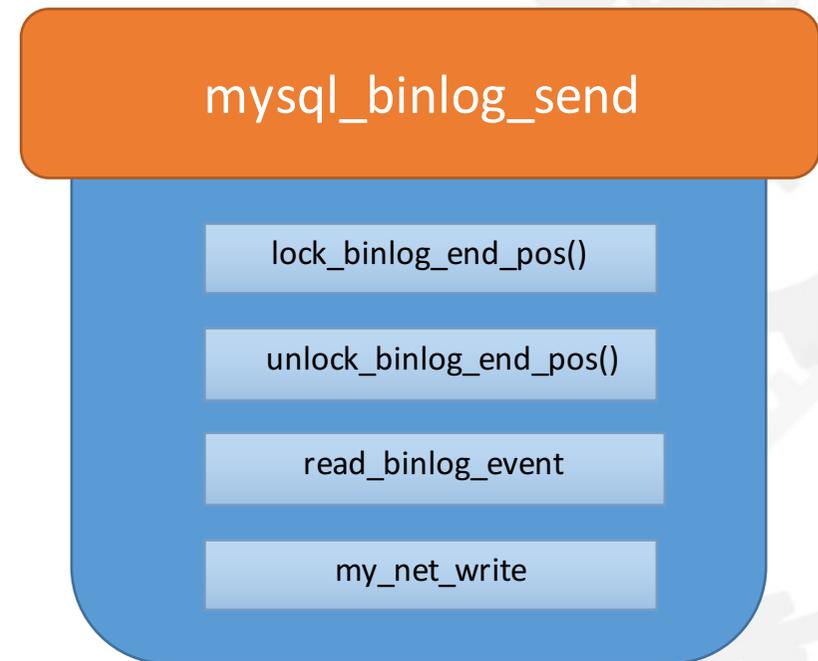
binlog 写线程 & binlog_sender 线程中的性能分析



问题：读写不能同时进行，影响性能

主库 Lock_log 锁拆分(Cont.1)

binlog 写线程 & binlog_sender 线程中的优化方式

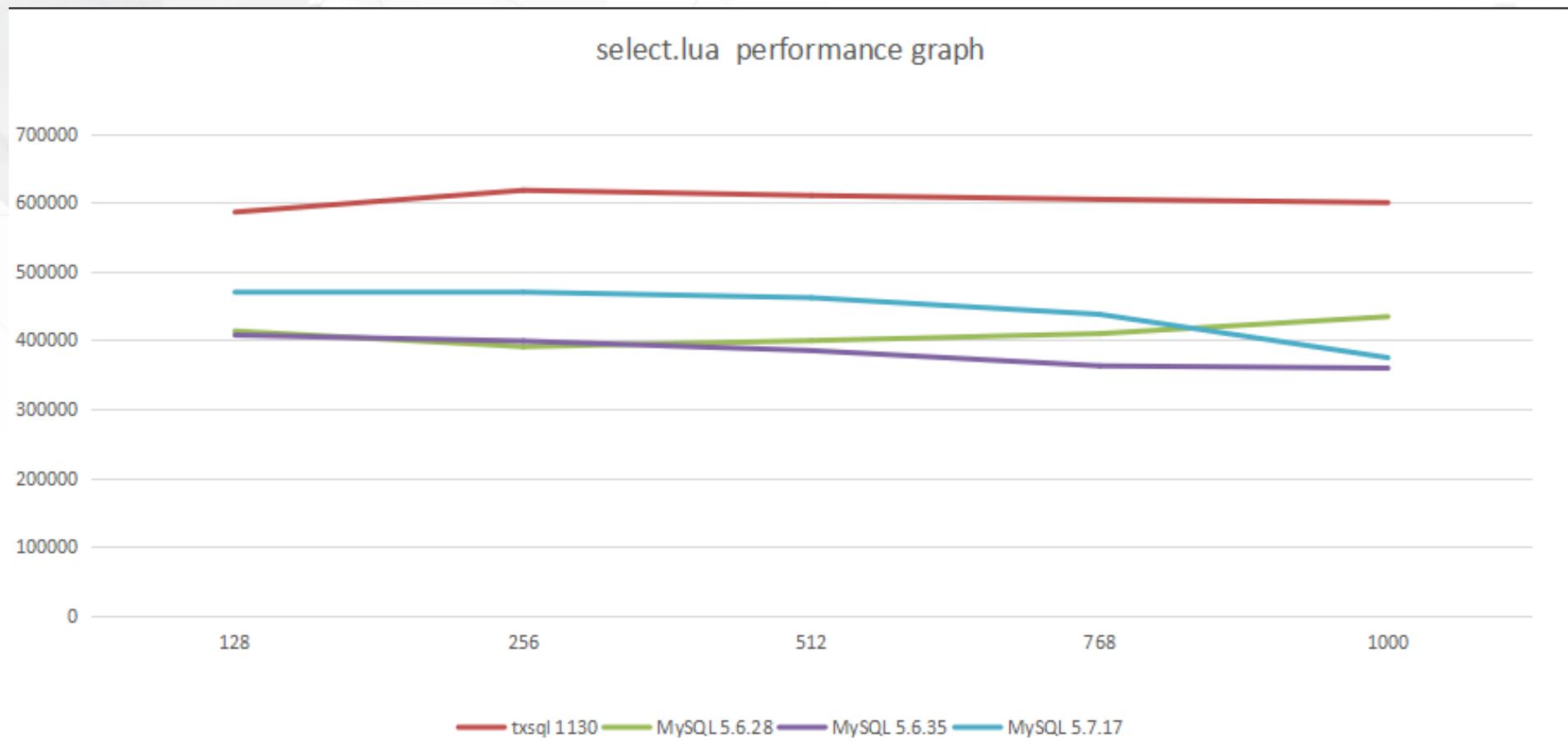


优点：读写同时进行，降低 dump 线程影响

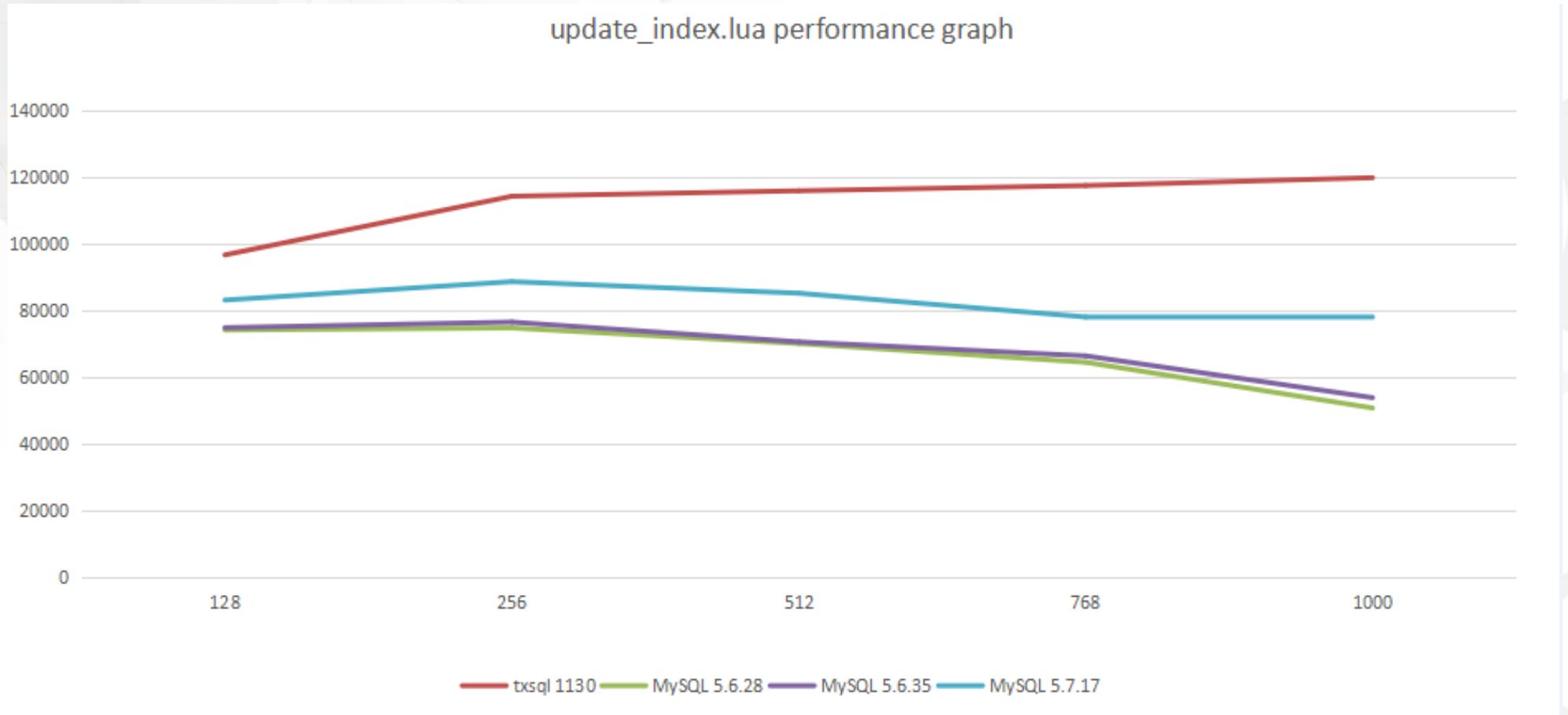
TXSQL 性能数据对比

- ✓ 读性能数据对比
- ✓ 写性能数据对比
- ✓ 读写混合数据对比

读性能数据对比

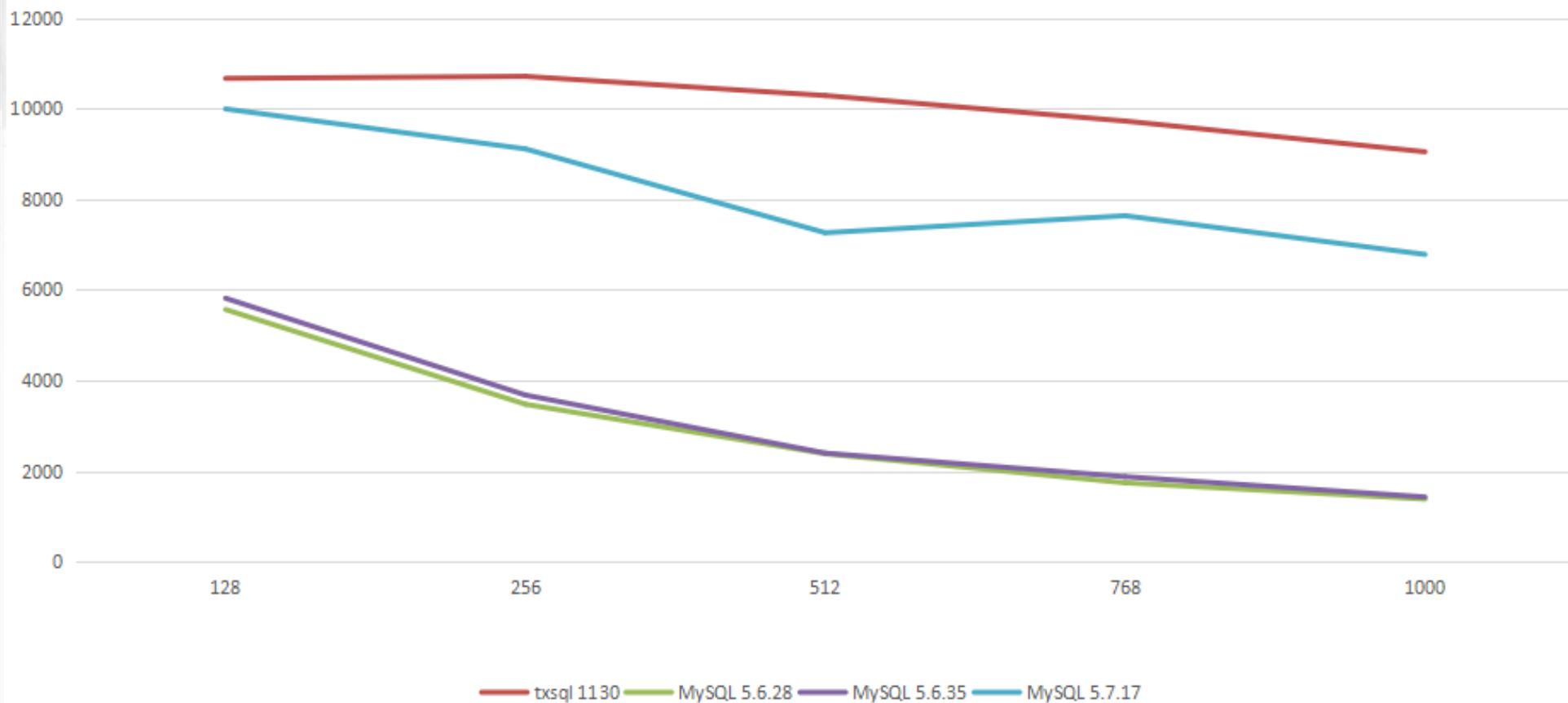


写性能数据对比



读写混合数据对比

oltp.lua performance graph



TXSQL 功能开发

- ✓ TXSQL alter table ... nowait
- ✓ TXSQL 并行复制
- ✓ TXSQL thread pool support

TXSQL alter table .. nowait

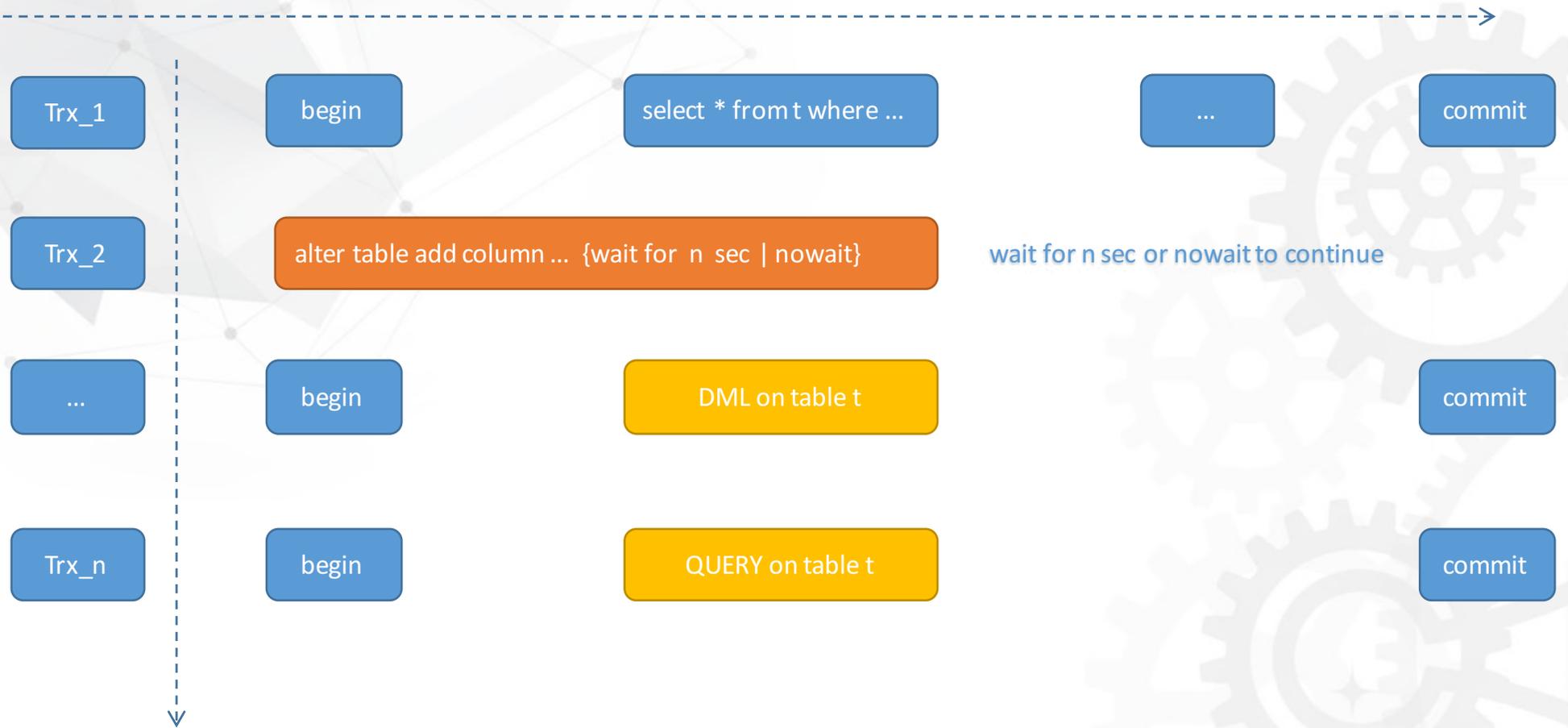
DDL 在运维过程中存在的问题



此时，系统相关线程处于waiting 状态，影响业务的后续访问；

TXSQL alter table .. nowait (con1.)

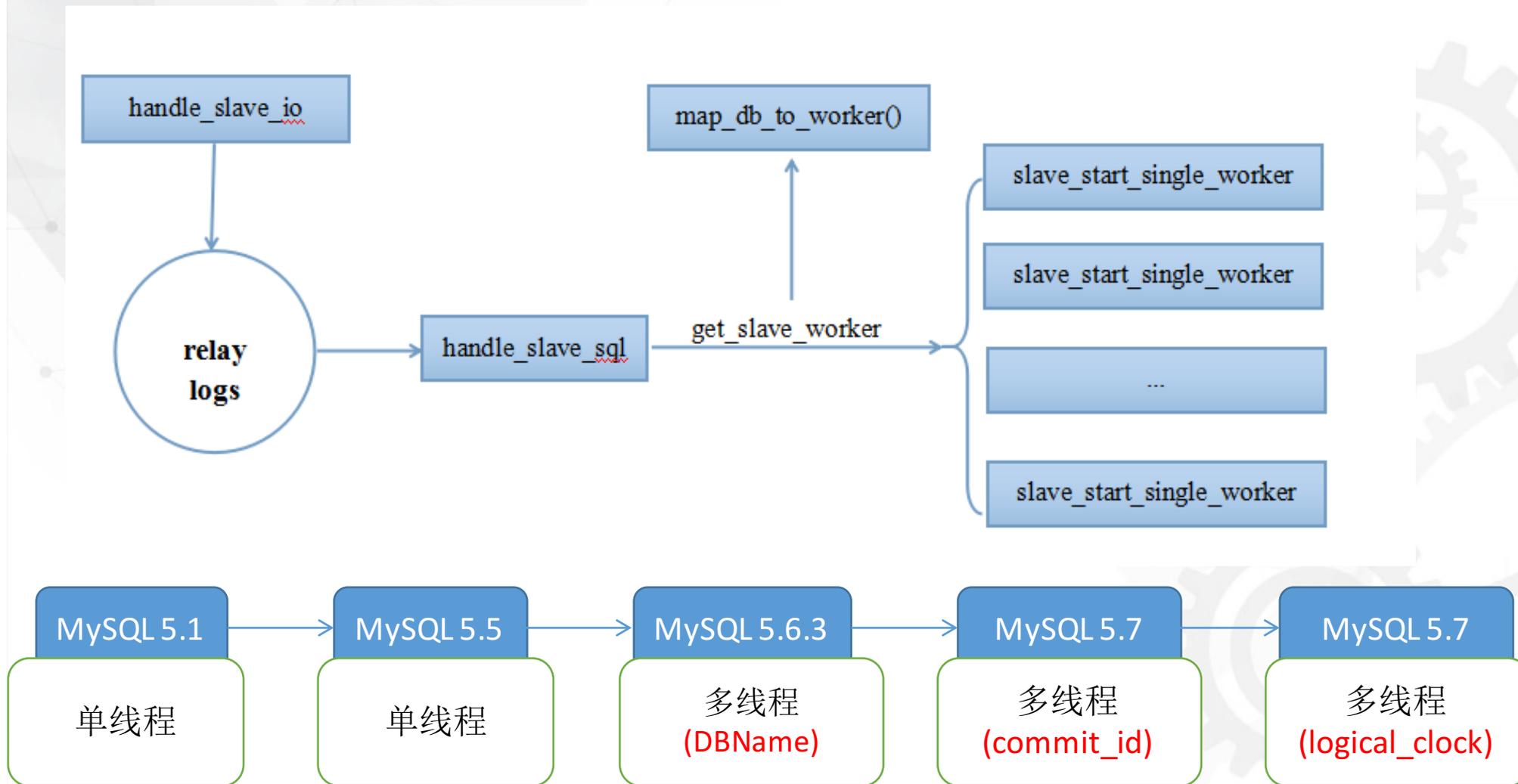
解决问题的方法:



提交新的语法支持以避免上述问题的发生

TXSQL 并行复制

MySQL 复制的原理及发展历程



TXSQL 并行复制(Cont.1)

MySQL 并行复制存在的问题

在实际的应用环境中，实例中往往只有一个 Database，导致 relay log 中的事务大部分会分到同一个 worker 线程中，造成备库的性能低下，当主库的性能超过备库的单线程执行的性能时，就会出现延迟，对只读实例产生影响；

TXSQL 并行复制存在的优化

为了解决上述问题，TXSQL 添加了另外一种分发方式，即基于表粒度的分发，为了实现基于表粒度的分发，TXSQL 对于不同的实现，进行了不同的处理：

- ✓ 当 binlog_row_format=ROW 时，调用 get_slave_worker 直接进行分发；
- ✓ 当 binlog_row_format=statement 时，则需要对语句先进行调用 mysql_parse 对语句进行解析，然后再做分发；

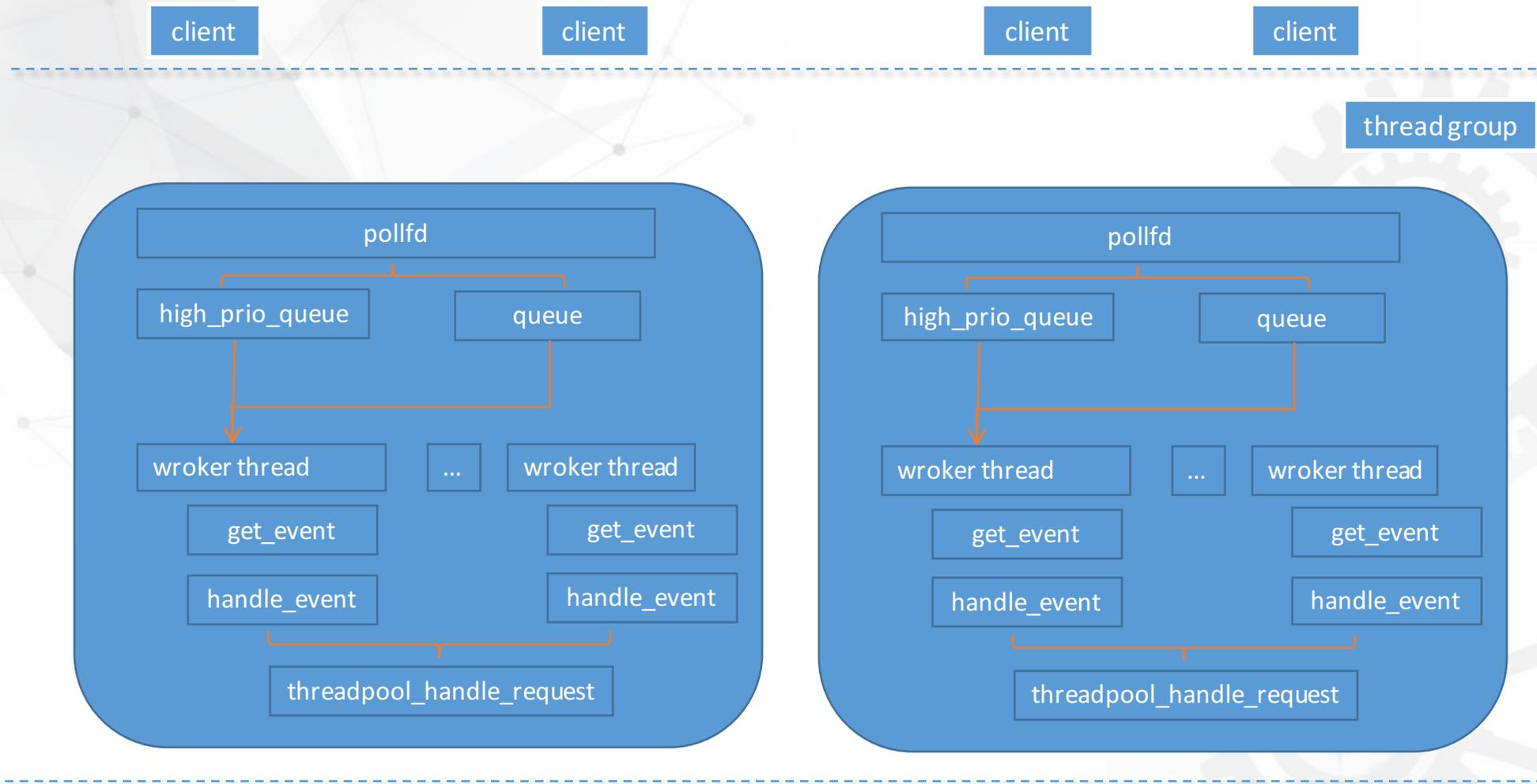
TXSQL thread pool 的支持

为什么需要 thread pool 功能

innodb_thread_cocurrency 可以控制进行 innodb 工作的线程数，以降低大并发对系统的负载影响，但是原生 MySQL 的 Server 层并不能控制 Server 层的线程数，当并发较大、活跃线程较多的时候会有以下问题：

- ✓ MDL lock 锁竞争 & table lock 竞争激烈；
- ✓ Thread running 较高，系统负载较大；
- ✓ Server 层的线程切换较为频繁；
- ✓ Server 层的其它锁资源竞争激烈；

thread pool 的工作原理(Cont.1)



thread_group_size

thread_group_oversubscribe

TXSQL 金融特性

- ✓ TXSQL multi work mode support
- ✓ TXSQL 强同步支持
- ✓ TXSQL 分布式锁支持
- ✓ 其它金融特性相关功能

TXSQL multi work mode support

READ
WRITE

实例可以接受读写请求，但系统管理的相关权限被禁用；

READ
ONLY

实例只接受读请求，不接受非管理用户的写请求操作；

OFFLINE

MySQL 正常运行但不接受除管理账号以外的其它用户操作；

TXSQL 强同步支持

原生 semi-sync 存在着以下问题:

- ✓ semi-sync 在时间超过 `rpl_semi_sync_master_timeout` 会退化为异步;
- ✓ 采用 `select` 进行监听, 当句柄值大于 1024 时则会出现异常, 详情可参考 `bug#79865`;
- ✓ 在 `after commit` 后等待 `ACK` 容易出现幻读的问题;

TXSQL 强同步支持:

- ✓ 优化半同步, 增加ack线程, 收发并行化;
- ✓ 修正select时fd超过1024导致异常的bug, 改为poll;
- ✓ 在半同步基础上实现强同步, 一直hold住直到收到ack;
- ✓ 修改同步方式时, 唤醒正在等待的用户线程, 继续等待或者退出;
- ✓ 增加一些状态, 用于展示当前等待的情况(正在等待的binlog位点, 已等待时间);
- ✓ 对于主多 binlog 备少 binlog 的情况进行特殊的处理, 以保证双写的情况不会发生;

TXSQL 分布式锁支持

MySQL 锁系统有两个特征:

- ✓ MySQL 中的锁与连接强依赖，在连接断开之后便会释放其占有的锁资源，包括 server 层 & engine 层的所有锁资源;
- ✓ 用户线程获取锁之后，如果没有显示释放锁资源，连接没有断开亦或事物没有提交，则会一直占有锁资源;

TXSQL 分布式锁

TXSQL 对 MySQL 的锁系统进行了扩展，实现了一种跨事务的、与连接无关的租约读写锁，用于应用层实现分布式事务;

`cdb_lock()` : 多用户对同一个锁对象key加锁，会在加锁超时前等待。直到获取到锁或者超时返回;

`cdb_unlock()`: 主动锁释放。会对指定锁key的持有者释放锁，只有在锁持有情况下才会成功;

TXSQL 云上优化实践

- ✓ XX 游戏数据库优化案例
- ✓ XX 优化案例

XX 游戏数据库优化案例

问题现象

- ✓ 性能不能满足业务要求，游戏业务逻辑 TPS 不达标；
- ✓ 在压力达到一定程度时，CPU 不能充分利用，idle 较高；
- ✓ 性能抖动较为明显；
- ✓ thread running 过高，系统负载较高；
- ✓ 系统 IO 压力较小，IO 没有问题；

问题排查

pt-pmp & pstack & mysql 命令进行问题排查，发现以下问题：

1. 应用在执行SQL语句的过程中，table_cache_manager 中的锁冲突比较严重；
2. MySQL Server 层中的 MDL_lock 冲突比较重；
3. 实例开启了 Performance_schema 功能；
4. 事务锁 trx_sys->mutx 冲突较高；

XX 游戏数据库优化案例(Cont.1)

调优过程

根据已经查找出来的问题，调整相应参数与版本并重启，效果如下图所示：

1. table_open_cache_instances= 32
2. metadata_locks_hash_instances= 32
3. performance_schema= OFF
4. 其它

dttime	load_1	idlecpu	com_insert	com_update	com_delete	com_select	com_tps	threads_running	threads_conns
2016-07-03 22:21:31	1.54	37.00	1995	8194	0	125782	10189	18	1012
2016-07-03 22:21:32	1.54	37.00	2205	8016	0	125974	10221	17	1012
2016-07-03 22:21:33	1.54	49.00	2061	5758	0	106469	7819	25	1012
2016-07-03 22:21:34	1.54	38.00	2450	7565	0	127511	10015	18	1012
2016-07-03 22:21:35	3.66	39.00	2121	6644	0	128277	8765	27	1012
2016-07-03 22:21:36	3.66	41.00	2617	5966	0	127987	8583	22	1012
2016-07-03 22:21:37	3.66	43.00	2009	6564	0	124135	8573	16	1012
2016-07-03 22:21:38	3.66	43.00	2294	5783	0	123519	8077	15	1012
2016-07-03 22:21:39	4.65	45.00	2050	6931	0	123719	8981	13	1012
2016-07-03 22:21:40	4.65	51.00	2039	5028	0	107993	7067	14	1012
2016-07-03 22:21:41	4.65	49.00	2041	5153	0	110077	7194	23	1012
2016-07-03 22:21:42	4.65	49.00	2215	5347	0	108539	7562	24	1012
2016-07-03 22:21:43	4.65	40.00	2000	7564	0	128957	9564	21	1012

XX 优化案例

问题背景

- ✓ 在压力达到一定程度时，CPU 不能充分利用，idle 较高；
- ✓ 性能抖动较为明显；
- ✓ thread running 过高，系统负载较高；
- ✓ IO 无压力
- ✓ 网络成为系统瓶颈

问题排查及解决过程

- ✓ 使用 pt-pmp 获取压测时 mysqld 的运行时信息，发现 lock_private 时的锁冲突较大；
- ✓ 使用 perf record -g -a -p {pidof mysqld} -F99 --sleep {t} 进行采集，使用 perf report --stdio 进行统计，查找使用 cpu 较多的地方，然后调整相关参数；
- ✓ 查看网卡与 CPU 的绑定情况，然后进行相应的调整，解决网络拥堵的问题；
- ✓ 替换最新版本，引入 redo log 的写优化，提升整体的吞吐量；

数据库问题总结

- ✓ 在压力达到一定程度时，CPU 不能充分利用，idle 较高；
- ✓ 性能抖动明显；
- ✓ 并发过大引起的 thread running 过高，系统负载较高；
- ✓ IO 问题引起的性能抖动；
- ✓ 锁问题导致的性能抖动；
- ✓ 压力不够大或者压力不均匀；
- ✓ 优化器问题引起的执行计划出错；
- ✓ SQL 语句引起的异常；
- ✓ 参数配置的不合理；
- ✓ 内核 Bug；
- ✓ 网络问题；

TXSQL 未来发展方向

- ✓ 批量计算
- ✓ 执行计划缓存
- ✓ XA 三阶段支持
- ✓ 基于 binlog 的深度优化
- ✓ Innodb 的持续优化
- ✓ 引入 oracle 企业级特性

Welcome to join us!



DTCC

2017年第八届中国数据库技术大会
DATABASE TECHNOLOGY CONFERENCE CHINA 2017

SequeMedia
媒体传媒

IT168

ITPUB

ChinaUnix



THANKS

SequeMedia
盛拓传媒

IT168.com

ITPUB

ChinaUnix