# 概述

腾讯分布式数据库包括：TDSQL2.0（MySQL）+TDSQL3.0（RocksDB）+TDSQL-A（又称TBase，PostgreSQL系）

云原生数据库包括：CDB（企业级数据库，单机MySQL）+TDSQL-C（又称CynosDB，MySQL系）

参考：<https://zhuanlan.zhihu.com/p/210549705>

## 简介

CynosDB已经更名为TDSQL-C。云原生数据库TDSQL-C（Cloud Native Database TDSQL-C）简称TDSQL-C。

TDSQL-C是腾讯云自研的新一代高性能高可用的企业级分布式云数据库。融合了传统数据库、云计算与新硬件技术的优势，100%兼容MySQL和PostgreSQL，实现超百万级QPS的高吞吐，128TB海量分布式智能存储，保障数据安全可靠。

### CDB/RDB/TXSQL

企业级MySQL（TDSQL-C/CynosDB底层使用的存储）。

TXSQL是Tencent MySQL的简称，是TEG基础架构部CDB (Cloud DataBase) 团队在近十年发展过程中衍生出来的一个对MySQL内核源码深度定制、对官方MySQL版本进行二次开发的项目。其主要目的是在保证线上稳定性的同时，满足业务对数据库的各种需求。

说明：由于历史发展的原因，首先是CDB，然后基于CDB开发的存算分离的CynosDB。

#### DDL等待超时

#### 异步删除大表

**适用场景：**

该功能主要针对需要删除的表数据很大时

**功能介绍：**

主要用于删除数据文件很大的表，避免IO的抖动。

DROP TABLE会将原ibd文件重命名为一个新的临时文件并返回成功，并在后台分批次truncate

**使用说明：**

将innodb\_table\_drop\_mode设成SRV\_ASYNC\_DROP，DROP TABLE即会变成异步模式。

临时文件存放在innodb-async-drop-tmp-dir指定的目录下，每次truncate的文件大小由innodb-async-truncate-size控制

#### 热点更新

**功能介绍：**

针对于秒杀场景，大幅度优化对于热点行数据的update操作的性能。当开启热点更新自动探测时，系统会自动探测是否有单行的热点更新，如果有，则会让大量的并发update排队执行，以减少大量行锁造成的并发性能下降。

**适用场景：**

适用于单点或多点带主键的更新压力非常大的场景（秒杀场景）

**性能数据：**

高并发下，带主键条件的单点并发update有10倍以上的性能提升

**使用说明：**

增加参数innodb\_hot\_update\_detect用于开启热点更新自动探测。

增加参数innodb\_hot\_update\_wait\_timeout来控制事务等待热点更新的超时值，默认值为1000000，单位为微秒，取值范围为1000微秒~100秒(100000000)，大于等于100秒表示永远等待。

新增information\_schema表： INNODB\_HOT\_ROW\_UPDATE\_STATS，输出当前系统中存在的热点行信息

#### 秒级加列

**功能介绍：**

快速加列功能是通过只修改数据字典的方法来实现大表快速加列，避免之前加列操作必须做的数据拷贝，从而的大幅缩小大表加列所需的时间，减少对系统的影响。

**适用场景：**

适用于需要对数据量大的表进行增加列操作的场景

**性能数据：**

5G数据量的大表，增加一列操作从40秒降到1秒以内

**使用说明：**

ALTER TABLE 新增algorithm子句，可以通过如下语句实现快速加列： ALTER TABLE t1 ADD COLUMN c1 int，algorithm=instant;

相关参数：innodb\_alter\_table\_default\_algorithm， 指定缺省的alter table算法，可以是INPLACE和INSTANT，缺省值是INPLACE.

ALTER TABLE 将根据缺省算法设置来采用是用inplace还是instant。

#### OUTLINE

#### SQL限流

**适用场景：**

由于SQL语句使用不当所导致的系统资源占用过大、系统整体性能下降或假死的情况发生

**功能介绍：**

该项功能主要通过限制某些SQL语句的操作，来避免由于SQL语句使用不当所导致的系统资源占用过大、系统整体性能下降或假死的情况发生

**使用说明：**

增加参数cdb\_sql\_filter\_enable用于sql限流开关。

增加参数cdb\_sql\_filter用于修改sql限流规则。

增加参数cdb\_sql\_filter\_seperator来控制sql限流规则的分隔符，默认为','。

使用方法举例如下：

--设置需要限流的语句和限流方式：

set global cdb\_sql\_filter = '+,[select | update | delete | insert | replace], expire\_time, concurrence,key1,key2,key3'

set global cdb\_sql\_filter = 'reset\_all'

set global cdb\_sql\_filter = '-id'

set global cdb\_sql\_filter = '+,select,1,2,key1,key2,key3'

--查询限流设置

show cdb\_sql\_filters;

select \* from information\_schema.cdb\_sql\_filter\_info;

#### 加密

**适用场景：**

该功能主要针对需要对用户数据进行加密时

**功能介绍：**

企业级加密函数（Enterprise Encryption Functions）是一组能够有效管控明文数据的MySQL函数，其通过基于openSSL的RSA/DH/DSA等加密算法进行数据保护。主要功能如下：

创建RSA/DSA/DH等算法等非对称密钥

推导DH对称密钥

利用MD5/SHA1/SHA224/SHA256/SHA384/SHA512生成摘要

RSA加密与解压数据

RSA/DSA数字签名与验证

密钥文件管理

加密过程CPU使用率限制

**使用说明：**

新增参数cdb\_enterprise\_encryption\_rsa\_bits\_threshold来控制RSA key length in bits最大值

新增参数cdb\_enterprise\_encryption\_dh\_bits\_threshold来控制DH key length in bits最大值

新增参数cdb\_enterprise\_encryption\_dsa\_bits\_threshold来控制DSA key length in bits最大值

#### 审计

**适用场景：**

该功能主要针对需要对用户数据进行审计时

**功能介绍：**

该审计功能主要用于记录用户各种指定类型的操作，拥有业界领先的高性能，并支持用户可以根据不通业务需求进行选择模式，同时也支持灵活的审计规则。

**使用说明：**

1）通过配置audit\_cdb\_audit\_reload\_config可以重新加载规则文件，audit\_cdb\_config\_json指定规则文件位置

2）特权用户可通过cdb\_audit\_action\_enabled控制审计开关，通过cdb\_audit\_truncat\_sql\_len设置sql截断长度，只记录感兴趣的内容

3）通过新增状态audit\_cdb\_ignore\_actions统计忽略的审计记录条数，audit\_cdb\_truncat\_counts统计被截断的sql记录条数

4）新增全审计开关：audit\_cdb\_full\_audit

5）支持同步审计和异步审计两种审计模式，通过参数audit\_cdb\_audit\_mode来配置

6）同步审计性能在复杂正则审计规则下性能提升13%，正常规则下平均性能影响为5%; 最高每秒能处理50+w条审计日志

7）异步审计模式性能不受规则影响，平均性能影响为3%; 最高每秒能处理40+w条审计日志

8）系统压力过大时，会有部分审计日志不被记录，可以通过audit\_cdb\_ignore\_push\_queue\_counts和audit\_cdb\_ignore\_actions查看日志丢失数量

#### 线程池

**适用场景：**

该功能主要针对连接数大（2000以上）情况下，性能要求平稳，并且是强同步（线程处理异步化）的基础。

**功能介绍：**

MySQL默认采用的是一个用户连接一个线程的方式，当用户连接过大，性能会下降，因此引入线程池，可以支撑上万的并发连接，并保持稳定的性能。

**使用说明：**

启动的时候设置参数thread\_handling为2即可，此参数只读，因此要求启动之前设置；

show threadpool status \G;查询线程池的工作状况

show global variables like 'thread\_pool%'; 线程池的参数配置，

+------------------------------------------+--------------+

| thread\_pool\_eager\_mode | OFF |

| thread\_pool\_high\_prio\_mode | transactions |

| thread\_pool\_high\_prio\_tickets | 4294967295 |

| thread\_pool\_idle\_timeout | 60 |

| thread\_pool\_listen\_eager\_mode | ON |

| thread\_pool\_max\_threads | 2000 |

| thread\_pool\_oversubscribe | 10 |

| thread\_pool\_oversubscribe\_parall | ON |

| thread\_pool\_oversubscribe\_parall\_num | 3 |

| thread\_pool\_oversubscribe\_parall\_timeout | 20 |

| thread\_pool\_queue\_congest\_threshold | 13 |

| thread\_pool\_size | 40 | #最高的性能则建议vcpu数目一致，当然云上购买实例比较小CPU，可以按购买的实例CPU分配

| thread\_pool\_stall\_limit | 50 |

#### 并行复制

**适用场景：**

该功能主要针对部分负载能提升slave机重放binlog速度，减少主从的delay。

**功能介绍：**

传统的官方MySQL版本是slave去回放，master同步Binlog过来的时候，都是单线程的模式，到5.6以后就变成了并行。

但官方的并行是基于database和logical clock的，并行粒度太大，导致很多情况下并行不起来。该功能做了一些的优化，也就是支持按table并行，这样相当于把它的粒度拆分到表，提升了并行度，从而减少主从延迟。

**使用说明：**

通过将参数slave\_parallel\_type设置为新增加的值TABLE打开这个功能。

另外information\_schema下新增加了状态表cdb\_slave\_thread\_status，用以展示状态信息。

#### GTID复制功能扩展

#### fdatasync同步redo

**功能介绍：**

redo日志文件目前采用fsync系统调用来落盘，这包括文件元数据落盘和文件数据落盘。文件元数据信息包括最后修改时间等不是非常重要的信息，在一些redo落盘场景可以避免总是刷文件元数据到存储设备上（通过fdatasync系统调用)，来减少开销。

**适用场景：**

主要适用于写入压力比较大的场景

**性能数据：**

在sysbench-write-only高并发持续写入场景下，TPS性能有10%左右提升

**使用说明：**

通过设置参数innodb\_flush\_redo\_using\_fdatasync=true/false来控制是否用fdatasync来避免redo日志文件元数据实时落盘。缺省为false。

#### CATS事务调度

**功能介绍：**

增加一种新的并发事务调度算法（CATS：Contention-Aware Transaction Scheduling），自动感知事务直接锁冲突并根据事务的优先级来调度事务执行。

MySQL原有的并发事务是采用First-In-First-Out (FIFO)规则来决定事务执行的顺序的，而CATS事务调度算法的主要原理是根据事务持有锁的情况来判断并发事务的冲突情况，并决定事务执行的优先级，而后，根据优先级来安排事务的执行顺序，从而提升系统事务处理的吞吐量。

**适用场景：**

主要适用于高并发并且锁冲突比较严重的场景

**性能数据：**

在sysbench-oltp高并发，锁冲突严重的场景下有50%以上的TPS性能提升

**使用说明：**

MySQL 5.7 版本可以通过全局参数innodb\_trx\_schedule\_algorithm来指定事务调度算法，该参数缺省值是auto。

其中，算法有三种：

auto：自动，根据当前系统状况自动调整。当锁等待线程数超过32个时采用CATS调度算法，否则采用FCFS算法。

fcfs：先来先服务算法。

cats: 冲突感知调度算法。

MySQL8.0版本固定采用auto算法，不可设置。

#### 自动kill空闲事务

**适用场景：**

该功能主要针对存在一些持有的锁等资源而没有响应释放的事务，影响线上 DDL & 备份操作

**功能介绍：**

通过杀死指定时间内没有活跃的空闲事务，释放其所占用的相关锁资源，提升系统稳定性。

**使用说明：**

设置 cdb\_kill\_idle\_trans\_timeout 值覆盖 wait\_timeout 的值。当一个空闲事务超时，它将会被杀死。

## 特点

### 全面兼容

100%兼容开源数据库引擎MySQL 5.7和PostgreSQL 10。 几乎无需改动代码，即可完成现有数据库的查询、应用和工具平滑迁移。

### 超高性能

具有商用数据库的强劲性能， 最高性能是MySQL数据库八倍、PostgreSQL数据库的四倍。

### 海量存储

最高128TB的海量存储，无服务器Serverless架构，自动扩缩容，轻松应对业务数据量动态变化和持续增长。

### 快速恢复

计算节点实现无状态，支持本地和跨设备的秒级故障切换和恢复；支持基于快照的秒级备份和回档。

### 数据高可靠

集群支持安全组和VPC网络隔离。自动维护数据和备份的多个副本，保障数据安全可靠，可靠性达99.9999999%。

### 弹性扩展

计算节点可根据业务需要快速升降配，秒级完成扩容，结合弹性存储，实现计算资源的成本最优。

# 架构

# TXSQL

## 概述

TXSQL 是 Tencent MySQL 的简称，是 TEG 基础架构部CDB (Cloud DataBase) 团队在近十年发展过程中衍生出来的一个对 MySQL 内核源码深度定制、对官方 MySQL 版本进行二次开发的项目。其主要目的是在保证线上稳定性的同时，满足业务对数据库的各种需求。TXSQL 的服务对象是公司内部用户和腾讯云上小至数G大至数百T的外部客户。TXSQL 是支撑这些业务平稳运行的关键基石，在服务用户的同时，以打造腾讯自己的MySQL分支为己任。

    MySQL 作为最受欢迎的开源数据库，也是云上使用最多的数据库，不同的业务场景对数据库有不同的功能需求与性能需求，这样就决定了数据库本身的多样化需求，同时 MySQL 在不同使用场景下所衍生出来的各种问题，也影响着线上的稳定性，TXSQL 主要包括以下工作：

数据库性能调优；

对内核进行深度改造，满足应用需要的性能要求，在活动促销时需要保证数据库的平稳；

提供业务需要的各种功能需求；

实现资源管理需要内核支持的功能；

MySQL DBA 在运维过程中需要的诊断相关的功能；

MySQL 在运行过程中存在的问题定位并修复影响系统运行的 Kernel Bug；

保证数据库系统的安全，保证云上用户数据的安全；

    做为云上客户的服务者，我们必须有及时响应用户并解决各种问题的能力，也应该满足用户的各种合理要求，对内支持业务的快速发展，满足业务要求；对外为腾讯云提供强有力的竟争力，这就是 TXSQL 存在与发展的目的。

## 性能改进

### read view优化

    read view 又称读视图，用于存储事务创建时的活跃事务集合。当事务创建时，线程会对 trx\_sys 上全局锁，然后遍历当前活跃事务列表，将当前活跃事务的ID存储在数组中的同时，记录最大事务 low\_limit\_id &　最小事务 high\_limit\_id & 最小序列化事务 low\_limit\_no。 当事务执行时，凡是大于low\_limit\_id 的数据对于事务是不可见的，凡是事务小于 high\_limit\_id 的数据都是可见的，事务 ID 是 read\_view 数组中的某一个时也是不可见的；Purge thread 在执行 Purge 操作时，凡是小于 low\_limit\_no 的数据，都是可以被 Purge 的，read view 是 MySQL MVCC 实现的基础；

    由于read\_view 的创建和销毁都需要获取 trx\_sys->mutex, 当并发量很大的时候，事务链表会比较长，又由于遍历本身也是一个费时的工作，所以此处便成为了性能瓶颈，详情可以参考 bug#49169，为了解决这个问题 TXSQL 做了以下事情：

在 trx\_sys下维护一个全局的事务ID的有序集合，事务的 创建 & 销毁 的同时将事务的 ID 从这个集合中移除；

在 trx\_sys下维护一个有序的已分配序列号的事务列表，已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，为每一个 trx\_t 缓存一个 read\_view，read\_view 数组的大小根据创建时的活跃全局事务 ID 集合做必要的调整；

参照 5.7 的实现，在 5.6 中将 ro\_trx\_list 移除，减少只读视图的长度；

    经过上述修改，读写性能都有很大的提升。

### redo log group commit

    MySQL 有两种很重要的 Log，分别为 redo log & binlog，前者是保证事务原子性操作所产生的日志，后者是主备数据同步所产生的同步日志。 其中 binlog 在 ordered\_commit 时进行 group commit, 而 redo log 则是在事务提交的时候分别调用 trx\_prepare 使 redo log 落地，导致 log\_sys->mutex 竟争较为严重；从 crash recovery 的逻辑来看，只要 redo log 早于 binlog 落地，就不会有数据问题，因此在 ordered\_commit 的第一阶段时，TXSQL 会收集各种引擎的最大的 redo log LSN，然后将小于该 LSN 的 redo log 落盘，从而提升写性能。更详细的分析与测试，可以参考 bug#73202。

### redo log双缓冲区

    MySQL redo log 是一个顺序写的单缓冲区，log\_sys->mutex 锁资源竟争激烈，在事务落盘的过程中对 LSN 相关的读、写都被阻塞，为了解决 log\_sys->mutex 的锁竟争问题，引入双缓冲区机制 & w\_mutex 锁，在 flush redo log 的过程中释放 log\_sys->mutex，继续持有 log\_sys->w\_mutex，从而阻塞写，不阻塞 LSN 相关的读操作，flush 完成后释放 w\_mutex；从而提升并发性，提升性能。

### adaptive hash index锁拆分

    MySQL 一次定位 cursor 的过程是从根结点到叶子结点的路径，时间复杂度为：height(index) + [CPU cost time]，上述的两个优化过程无法省略定位 cursor 的中间结点，因此需要引入一种可以从 search info 定位到叶子结点的方法，从而省略根结点到叶子结点的路径上所消耗的时间，而这种方法即是 自适应索引(Adaptive hash index, AHI)。查询语句使用 AHI 的时候有以下优点：

1、可以直接通过从查询条件直接定位到叶子结点，减少一次定位所需要的时间；

2、在 buffer pool 不足的情况下，可以只针对热点数据页建立缓存，从而避免数据页频繁的 LRU；

    AHI 的 btr\_search\_latch (bug#62018) & index lock 是 MySQL 中两个比较大的锁，详情可以参考 Index lock and adaptive search – next two biggest InnoDB problems，5.7 通过对 AHI 锁拆分 (5.7 commit id: ab17ab91) 以及引入不同的索引锁协议 (WL#6326) 解决了这两个问题。但是 AHI 并不总能提升性能，在多表Join & 模糊查询 & 查询条件经常变化的情况下，此时系统监控 AHI 使用的资源大于上述的好处时，不仅不能发挥 AHI 的优点，还会为系统带来额外的 CPU 消耗，此时需要将 AHI 关闭来避免不必要的系统资源浪费，关于 AHI 的适应场景可以参考：mysql\_adaptive\_hash\_index\_implementation。

    TXSQL 则是和 5.7 的解决思路类似，只是使用的分区算法不同，关于自适应索引的详细说明可以参考：MySQL AHI 实现解析。

## 功能添加

### TXSQL 支持多种工作模式

    TXSQL 根据配置的不同，支持不同的工作模式，分别为只读、读写、不可服务，在不同的模式下实例表现形式不同：

只读模式：实例只接受读请求，不接受非管理用户的写请求操作；

读写模式：实例可以接受读写请求，但系统管理的相关权限被禁用；

不可服务模式：此模式下，MySQL 正常运行但不接受除管理账号以外的其它操作；

    OSS 在实现跨园区容灾的时候会根据结点的角色设置不同的工作模式，从而避免双写的问题；

### alter table … nowait功能

    DDL在执行过程中需要获取对应的表锁，然后进行操作，如果此时有事务获取表锁，则会造成此语句的阻塞，而后绪操作此表的请求也会被此DDL阻塞，因此我们引入了Oracle中的alter table超时失败的功能（DDL等待超时），即alter table …. [nowait | wait for n]的功能以降低DDL对线上的影响，oracle中select for update [nowait|wait for n]则可以通过设置参数innodb\_lock\_wait\_timeout来实现。

**适用场景：**

该功能主要针对线上ALTER TABLE操作可能会系统正常运行的场景

**功能介绍：**

支持在获得MDL元数据锁前DDL操作等待超时

**使用说明：**

可以通过在操作语句时加关键字参数来实现功能，支持以下DDL操作：

\* ALTER TABLE table [NO\_WAIT | WAIT [n]] command;

\* DROP TABLE table [NO\_WAIT | WAIT [n]];

\* TRUNCATE TABLE table [NO\_WAIT | WAIT [n]];

\* OPTIMIZE TABLE table [NO\_WAIT | WAIT [n]];

\* RENAME TABLE table\_src [NO\_WAIT | WAIT [n]] TO table\_dst;

\* CREATE INDEX index ON table.columns [NO\_WAIT | WAIT [n]];

\* CREATE FULLTEXT INDEX index ON table.columns [NO\_WAIT | WAIT [n]];

\* CREATE SPATIAL INDEX index ON table.columns [NO\_WAIT | WAIT [n]];

\* DROP INDEX index ON table [NO\_WAIT | WAIT [n]];

NO\_WAIT: the DDL would return immediately, and print

"ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction;"

if it has to wait for a metadata lock;

WAIT [n]: n is the wait time in seconds for a metadata lock before the DDL

returns and raises error message;

### TXSQL 锁系统的扩展

MySQL 锁系统有两个特征：

1、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；

2、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 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，提供接口 cdb\_lock, cdb\_unlock 用于上锁与解锁请求，从而满足业务需求，详细接口介绍如下：

int cdb\_lock（string key, int type, uint64\_t wait\_timeout, uint64\_t hold\_timeout）;

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。

锁类型目前支持两种，共享锁和排它锁。共享锁是读锁，允许多个客户端对锁key加共享锁。排它锁是写锁，同一时刻只允许一个请求占有排它锁，其他请求需要等待锁释放。排它锁比共享锁有更高优先级，防止过多的共享锁等待情况下，排它锁长时间无法获取。

同一个锁持有者对同一锁key发起多次加锁，会认为重复加锁，相同锁发起者同一时刻只能对同一锁key成功发起一次。

获取的锁，不会因为连接断开而释放。锁持有时间是有指定的，超过持有时间以后锁会自动释放。

int cdb\_unlock(string key);

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。

### 基于 semi-sync 的强同步

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；

…

为了解决上面的问题并对强同步进行优化及显示状态信息，TXSQL 做了以下事情：

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；

优化dump线程判断自身是否为半同步的方法：在线程自身变量上记录；

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)；

    这些只是我们暂时做的，在以后的版本中，我们将会引入 thread pool、更完善的审记、防火墙等内部用户 & 外部云用户都需要的功能，随着时间的积累与版本的推移，TXSQL的功能将会越来越丰富。

### GTID复制功能扩展

**适用场景：**

该功能主要实现了需要对 GTID 的功能限制进行适当破除需求

**功能介绍：**

gtid模式下复制支持create as select,create/drop temporary table, 同时更新事务表和非事务表等语句。

**使用说明：**

增加参数cdb\_more\_gtid\_feature\_supported用于该功能开关，默认为off

当gtid\_mode=ON、enforce\_gtid\_consistency=ON 复制支持如下用法：

create table t2 select \* from t;

begin;create temporary table xx (id int); insert into xx select from t2;insert into t1 select from xx;commit;

### 稳定性改进与测试

    稳定性是数据库服务首先要解决的问题，原生 MySQL 在稳定性方面存在着以下问题：

在压力持续增大的过程时，会存在着内存分配的问题，原生 GCC 在内存分配的过程中会调用 \_\_lll\_lock\_wait\_private，在并发量加大且线程数量变化较大时，内存分配会造成系统性能的抖动，详情可以参考： Impact of memory allocators on MySQL performance 、MySQL performance: Impact of memory allocators (Part 2);

随着并发的加大，由于原生 MySQL 没有流量控制或者 Server 层线程调度的功能，导致线程之间资源争夺比较剧烈，直接导致 MySQL 性能的不稳定，性能曲线先升后降；

innodb buffer pool 刷脏算法在 IO bound 类型时，很容易成为瓶颈；

主备同步时备库性能过低导致的延迟问题等；

MySQL 各个版本在实现的过程中存在的 bug, 如 redo log 死锁, 典型的字符集问题、MySQLD normal shutdown 等;

…

    TXSQL 在服务业务的过程中不断的自我完善和发展，解决在发展过程中遇到的各种问题，如：

解决在线上遇到的各种 内核bug;

引入 jemalloc 解决内存分配的效率问题；

对复制线程进行改进，减缓主备延迟；

引入 thread pool 在 server 层进行线程高度，从而避免过多资源浪费在资源争夺等（内测中…）；

实现 OSS 需要的功能，结合 zookeeper 实现强一致功能等；

…

    为了保证版本稳定性，TXSQL 对每一个 Patch 都有对应的 testcase 文件和覆盖率测试，对于比较重要的改动，不仅要有理有据，还会邀请 Oracle 官方人员进行必要的 Code Review； 每个小版本发布之前，都会进行回归测试、覆盖率测试、稳定性测试、性能测试以及压力测试，然后才会进行灰度发布；当然，对于每一次的版本发布，都有详细的 release notes 以供参考。

### 金融业务支持

    数据的可靠性是金融业务必然的需求，由于 MySQL 自身存在的问题，很难保证数据的强一致性，这些问题包括：

MySQL 是典型的两段提交，在 Crash 的过程中依赖 binlog 是否存在 决定事务的 commit 或 rollback，对于已经写 binlog 的事务，一定会提交事务，这样在主备切换后容易出现双写的问题；

当 rpl\_semi\_sync\_master\_wait\_point=after\_commit 时，主备切换时存在着幻读问题，当 rpl\_semi\_sync\_master\_wait\_point=after\_sync，如果用户线程在等待备库 ACK 的过程中被 kill, 那么其它用户线程可以看到被 kill 线程的提交，此时主备切换，仍然存在着幻读问题；

当 rpl\_semi\_sync\_master\_wait\_point=after\_sync 时，用户线程在 binlog 切换时容易出现死锁问题 (bug#68250), 详情如下：

T1 (user thread)

.........rotate(holding Lock\_log, waiting for xids decrease to be zero)

T2 (user thread)

.........after\_sync, wait for ACK from slave

T3 (binlog sender thread)

......... requesting for Lock log to send binlog

    为了解决上述问题并实现主备数据的强一致，TXSQL 结合 CDB 管控平台针对金融业务的需求，实现了以下特性：

为了规避幻读，强同步插件在线程等待 ack 的过程中，禁止用户 kill 线程，同时本地 agent 会通过实例状态变量 rpl\_semi\_sync\_master\_tx\_cur\_wait\_time 监控是否出现长时间 hang 问题，触发告警和 HA 操作，减少 hang 住对实例的影响；

本地 agent 在主库 crash 后，会向 OSS 获取故障切换时的 binlog 位点信息，并以这个位点启动实例，TXSQL 将截断多余的 binlog，并回滚相应事务，与新主库建立主从，确保可以快速补充节点；

跨园区高可用的支持。MySQL 是常见的 HA 架构：由本地 agent 负责监听 MySQL 实例，如果实例出现故障，则上报总控节点 scheduler，scheduler 选择拥有最多数据的 slave，将其提升为 master，并更新接入层的 RS 为新 master。在跨园区环境中，如果主库所在园区故障，OSS HA 时可能无法更新该园区的接入层，而该园区的 client 依然能够操作原主库，导致出现双写的问题，为了规避网络分区情下的双写问题，TXSQL实现了租约功能，租约过期的实例将拒绝服务。本地 agent 只有成功向scheduler续租后，才会更新实例的租约，确保实例的租约小于scheduler，从而解决双写问题；

对主库 Lock\_log 进行拆分，解决死锁的同时提升主库写性能；

    除了数据强一制的支持，TXSQL还实现了一套与连接无关的分布式锁来支持业务部门实现分布式事务，后绪随着对业务的深入，更多更好的金融特性将会在TXSQL中一一实现，比如热点更新等；

## 发展方向

功能开发，以业务需求为出发点，实现业务所需要的各种功能，如审记、防火墙等；

性能优化，在保证稳定性的前提下不断提升单机性能，并根据业务特性有针对性的进行优化，如热点更新、多语句提交等；

提高内核的抗风险能力，提升对烂SQL、索引异常、流量异常等异常的防范能力，在不修改应用的前提下，保证系统的稳定运行。

# 扩展性

# 高并发

# 高可靠