ALIBABA CLOUD

阿里云

云数据库RDS

自研内核 AliSQL

文档版本：20220117

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.AliSQL 功能概览

本文介绍AliSQL和其他版本的功能对比。

AliSQL介绍

AliSQL是阿里云深度定制的独立MySQL分支，除了社区版的所有功能外，AliSQL提供了类似于MySQL企业版的诸多功能，如企业级备份恢复、线程池、并行查询等，并且AliSQL还提供兼容Oracle的能力，如sequence引擎等。RDS MySQL使用AliSQL内核，为用户提供了MySQL所有的功能，同时提供了企业级的安全、备份、恢复、监控、性能优化、只读实例等高级特性。

版本支持情况

类别	功能	描述	MySQL 8.0	MySQL 5.7	MySQL 5.6
功能	Native Flashback	通过SQL语句查询或恢复指定时间点的数据，保证在误操作后可以快速获取历史数据。	支持	不支持	不支持
	Thread Pool	提供线程池（Thread Pool）功能，将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。	支持	支持	支持
	Statement Outline	利用Optimizer Hint和Index Hint让MySQL稳定执行计划，该方法称为Statement Outline，并提供了工具包（DBMS_OUTLN）便于您快捷使用。	支持	支持	不支持
	Sequence Engine	提供Sequence Engine，简化获取序列值的复杂度。	支持	支持	支持
	Returning	支持DML语句返回Resultset，同时提供了工具包（DBMS_TRANS）便于您快捷使用。	支持	不支持	不支持
	Lizard事务系统	Lizard事务系统能够更好地提升RDS MySQL数据库的吞吐能力，并支持分布式事务和全局一致性。	支持	不支持	不支持
性能	Fast Query Cache	针对原生MySQL Query Cache的不足，阿里云进行重新设计和全新实现，推出Fast Query Cache，能够有效提高数据库查询性能。	不支持	支持	不支持
	Binlog in Redo	在事务提交时将Binlog内容同步写入到Redo Log中，减少对磁盘的操作，提高数据库性能。	支持	不支持	不支持
	Statement Queue	针对语句的排队机制，将语句进行分桶排队，尽量把可能具有相同冲突的语句（例如操作相同行）放在一个桶内排队，减少冲突的开销。	支持	支持	不支持

类别	功能	描述	MySQL 8.0	MySQL 5.7	MySQL 5.6
	Inventory Hint	快速提交、回滚事务，配合Returning和Statement Queue，能有效提高业务吞吐能力。	支持	支持	支持
稳定	Faster DDL	优化DDL操作过程中的Buffer Pool管理机制，降低DDL操作带来的性能影响，提升在线DDL操作的并发数。	支持	支持	支持
	Statement Concurrency Control	提供基于语句规则的并发控制CCL（Concurrency Control），并提供了工具包（DBMS_CCL）便于您快捷使用。	支持	支持	不支持
	Performance Agent	便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。	支持	支持	支持
	Purge Large File Asynchronously	通过异步删除大文件的方式保证系统稳定性。	支持	支持	支持
	Performance Insight	是专注于实例负载监控、关联分析、性能调优的利器，帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。	支持	支持	不支持
安全	Recycle Bin	支持回收站（Recycle Bin）功能，临时将删除的表转移到回收站，还可以设置保留的时间，方便您找回数据，同时提供了工具包（DBMS_RECYCLE）便于您快捷使用。	支持	不支持	不支持

功能列表

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
企业增值服务	24*7 支持	未提供	√	√	√
	紧急故障救援	未提供	√	√	√
	专家服务顾问支持	未提供	√	√	√
	MySQL Database Server	√	√	√	√
	MySQL Document Store	√	√	MySQL 8.0支持	MySQL 8.0支持
	MySQL Connectors	√	√	支持公开发行版	支持公开发行版

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
MySQL Features	MySQL Replication	√	√	√	√
	MySQL Router	√	√	MaxScale（MySQL 8.0支持）	数据库单租户代理
	MySQL Partitioning	√	√	√	√
	Storage Engine	InnoDB MyISAM NDB	InnoDB MyISAM NDB	InnoDB X-Engine	InnoDB X-Engine
Oracle Compatibility	Sequence Engine	未提供	未提供	MySQL 8.0支持	MySQL 8.0支持
MySQL Enterprise Monitor	Enterprise Dashboard	未提供	√	开发中	Enhanced Monitor
	Enterprise Advisors	未提供	√	开发中	CloudDBA
	Query Analyzer	未提供	√	开发中	Performance Insight
	Replication Monitor	未提供	√	开发中	√
	Enhanced OS Metrics	未提供	未提供	未提供	Enhanced Monitor
MySQL Enterprise Backup	Hot backup for InnoDB	未提供	√	√	√
	Full, Incremental, Partial, Optimistic Backups	未提供	√	√	库表级备份
	Full, Partial, Selective, Hot Selective restore	未提供	√	√	库表级恢复
	Point-In-Time-Recovery	未提供	√	√	√
	Cross-Region Backup	未提供	未提供	未提供	跨地域备份

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
	Recycle bin	未提供	未提供	MySQL 8.0支持	MySQL 8.0支持
	Flashback	未提供	未提供	√	√
MySQL Enterprise Security	Enterprise TDE	本地密钥替换	√	BYOK TDE，Key Rotating	BYOK TDE，Key Rotating
	Enterprise Disk Data Encryption at Rest	未提供	未提供	未提供	BYOK 落盘加密
	Enterprise Encryption	SSL	√	SSL	SSL
	SQL Explorer	未提供	√	SQL洞察	SQL洞察
	安全加密算法SM4	未提供	未提供	√	√
MySQL Enterprise Scalability	Thread Pool	未提供	√	MySQL 8.0支持	MySQL 8.0支持
	Enterprise Readonly Request Extention	未提供	未提供	√	只读实例
MySQL Enterprise Reliability	Zero Data Loss	未提供	未提供	√	三节点企业版
	Statement Outline	未提供	未提供	√	√
	Inventory Hint	未提供	未提供	√	√
	Statement Concurrency Control	未提供	未提供	√	√
	Hot SQL Firewall	未提供	未提供	√	√
MySQL Enterprise High-Availability	Enterprise Automatic Failover Switch	未提供	未提供	需要第三方HA机制	高可用版
	InnoDB Cluster	√	√	√	三节点企业版
	Multi-Source Replication	√	√	√	只读实例高可用

分类	功能	社区版	官方企业版	AliSQL内核（5.7&8.0）	阿里云 RDS MySQL
	Cross-Region Standby	未提供	未提供	未提供	灾备实例

2.AliSQL 小版本 Release Notes

AliSQL是RDS MySQL的内核，除了为用户提供MySQL社区版的所有功能外，还提供了企业级备份恢复、线程池、并行查询等类似于MySQL企业版的诸多功能，赋予了RDS MySQL安全、备份、恢复、监控、性能优化、只读实例等各项能力。本文介绍AliSQL的内核版本更新说明。

> **说明**
>
> - AliSQL内核小版本过低可能会导致任务中断。建议您定期，或在收到阿里云的运维通知后升级您的内核小版本。
> - 本文全量列举AliSQL的小版本，升级小版本时，可能会存在部分小版本维护中，无法选取的情况，请以控制台可选小版本为准。
> - 关于RDS MySQL独享代理的小版本说明请参见数据库代理小版本Release Notes。

MySQL 8.0基础版或高可用版

小版本	说明
20210930（暂未正式发布）	• 新特性 ◦ 合并官方8.0.25变更。 ◦ 引入 `recovery_apply_binlog` 机制，使XA事务支持Crash Safe。 ◦ 审计日志支持V3版本，增加单独一列事务ID，增加ext字段以存放SQL被截断的标识。 ◦ 允许将新的SEQUENCE类型定义为TIMESTAMP SEQUENCE，相比DIGITAL SEQUENCE原始序列，构建的格式不同。 语法： `CREATE SEQUENCE seq CACHE [缓存大小] TIMESTAMP;` ◦ 对高权限账号放开 `mysql.slow_log` 和 `mysql.general_log` 表的 `truncate` 权限。 ◦ 增加Native Falshback Query功能，支持直接通过SQL语句进行回滚查询和数据恢复。 ◦ 增加自研X-Tree作为X-Engine MemTable的索引，同时支持高性能的写入，点查询和范围查询。 ◦ 支持自由调整Buffer Pool大小，优化调整过程，避免影响实例性能。 ◦ 支持异步多块读（Multi Blocks Read）功能。在SQL语句中添加HINT `/*+ MULTI_BLOCKS(n) */` 可预读多个数据页。 ◦ 优化扫描Bufeer Pool LRU链表获取空闲页的逻辑。 ◦ 支持带主键或唯一键的 `UPDATE` 和 `DELETE` 语句自动进入CCL队列。 ◦ 在performance_schema.events_statements_summary_by_digest_supplement表中增加TCP写入等待时长。 ◦ 允许DDL语句以In Place方式执行时记录redo日志。 • Bug修复 ◦ 修复 `SHOW GLOBAL STATUS` 的执行结果出现异常的问题。 ◦ 修复Jemalloc Profiling功能开启导致实例无法启动的问题。 ◦ 修复 `INSERT INTO SELECT` 语句使用SEQUENCE导致实例崩溃的问题。 ◦ 回收 `dbms_recycle.restore_table` 权限，如有特殊情况可以提交工单申请恢复。 ◦ `I_S.INDEX_STATISTICS` 表新增索引扫描次数。

小版本	说明
	○ `I_S.PERF_STATISTICS` 表增加更多性能数据指标。 ○ 优化用户自定义函数（`nextval` 和 `currval`）与 `SEQUENCE` 系统函数发生冲突时的调用性能。 ○ X-Engine支持即时响应用户发起的 `KILL SESSION` 请求。 ○ X-Engine修复在极端场景下，部分DDL操作失败后处于不一致状态的问题。 ○ 增加栈溢出保护机制。 ○ 修复 `persist_binlog_to_redo` 和 `X-Engine` 同时开启时，mysqld启动失败的问题。 ○ 修复 `pthread_getattr_np` 导致的内存泄漏问题。 ○ 自动清理 `Performance Agent` 文件产生的Page Cache。 ○ 修复Outline bug。 ○ 修复语句执行错误时session tracker重置的问题。 ○ 调整 `ccl_wait_timeout` 变量名称。 ○ 修复由于进程退出时未清理socket文件导致重启失败的问题。 ○ 调整用户自定义函数（nextval或currval）的优先级。 ○ 修复 `net_length_size` 返回的251错误。
20201031	• 新特性 ○ 支持从Recycle Bin还原表。 ○ 实例初始化时自动恢复Slow Log文件。 • 性能优化 使用X-Engine引擎时不支持开启Binlog in Redo。 • Bug修复 ○ 修复唯一索引键值过大导致ASSERT异常的问题。 ○ 修复无法终止COM_DAEMON守护进程的问题。 ○ 修复FTS查询导致缓存溢出的问题。 ○ 修复Instant-DDL崩溃后回滚出错的问题。

小版本	说明

20200831	• 新特性
	  ○ 新增是否允许 `count(*)` 函数执行并行扫描的选项，将innodb_parallel_read_threads参数设置为 `0` 可禁止该函数并行扫描。更多信息，请参见设置实例参数。
	  ○ MySQL Binlog工具增加开始GTID（start gtid）和结束GTID（stop gtid）功能。
	  ○ 支持输出Redo Log的各个LSN值：
	  ■ innodb_lsn：重做日志的lsn编号。
	  ■ innodb_log_checkpoint_lsn：最后检查点的lsn。
	  ■ innodb_log_write_lsn：log写入的lsn。
	  ■ innodb_log_ready_for_write_lsn：log buffer完成时间的lsn。
	  ■ innodb_log_flush_lsn：磁盘上刷新redo log的lsn。
	  ■ innodb_log_dirty_pages_added_up_to_lsn：添加脏页的lsn。
	  ■ innodb_log_oldest_lsn：页面刷新的lsn。
	• 性能优化
	  ○ 优化CCL（Concurrency Control）的等待与并发机制。
	  ○ 调整Concurrency Control在存储过程中的执行优先级。
	• Bug修复
	  ○ 修复解析器递归时缺少堆内存大小检查的问题。
	  ○ 修复TDE打开时无法修改表定义的问题。
	  ○ 修复事件调度程序内存泄露的问题。

20200630	• 新特性
	  ○ Faster DDL：优化DDL操作过程中的Buffer Pool管理机制，降低DDL操作带来的性能影响，提升在线DDL操作的并发数。
	  ○ 增加连接数上限，最大支持500,000连接。
	• 性能优化
	  ○ 线程池内部优化。
	  ○ 根据实例规格设置Performance Schema占用内存的上限。
	  ○ 不再检测审计日志文件。
	  ○ TDE会缓存KMS服务提供的密钥。
	  ○ 修改在Statement Concurrency Control中运行的线程状态。
	• Bug修复
	  ○ 修复Outline计算摘要时将分号（;）视为输入查询的其中一部分的问题。
	  ○ 修复更改表导致服务器崩溃的问题。
	  ○ 修复关键字member和array与旧版本不兼容的问题。
	  ○ 修复读取客户端命令时的等待计数不正确的问题。
	  ○ 修复内核小版本升级失败的问题。

小版本	说明

20200430	• 新特性
	  ◦ Binlog in Redo：通过将Binlog写入Redo Log来优化事务落盘机制，提高数据库性能。
	  ◦ 重构X-Engine引擎的行缓存代码。
	  ◦ 开放XA_RECOVER_ADMIN权限。
	• 性能优化
	  ◦ 在操作InnoDB临时表时仅扫描脏页列表，而不是扫描整个Buffer Pool列表。
	  ◦ 兼容MySQL 5.6，将全局参数opt_readonly_trans_implicit_commit重命名为rds_disable_explicit_trans。
	  ◦ 在实例升级期间，不记录升级相关日志到审计日志。
	  ◦ 降低在X-Engine引擎表上执行DDL操作消耗的内存。
	• Bug修复
	  ◦ 修复磁盘中实际X-Engine引擎表大小与IS表中的统计信息不一致的问题。
	  ◦ 修复重新打开错误日志会导致X-Engine日志初始化的问题。

20200331	• 新特性
	Recycle Bin：新增支持 `TRUNCATE TABLE` 命令，执行时将原始表移动到专门的recycle bin目录中，并使用相同的结构创建新表。
	• 性能优化
	  ◦ 默认关闭TCP错误的输出。
	  ◦ 提高线程池默认配置下的性能。
	• Bug修复
	  ◦ 修复因为 `#p` 分割分区文件名导致的数据库、表无效问题。
	  ◦ 修复CCL匹配时区分大小写问题，即不再区分大小写。
	• 合并官方8.0.17、8.0.18变更，更多信息，请参见Changes in MySQL 8.0.17和Changes in MySQL 8.0.18。

小版本	说明

20200229	• 新特性
	  ◦ Performance Agent：更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。
	  ◦ 在半同步模式下添加网络往返时间，并记录到性能数据。
	  ◦ X-Engine引擎支持在线DDL功能。
	• 性能优化
	  ◦ 允许在只读实例上进行语句级并发控制（CCL）操作。
	  ◦ 备实例支持Outline。
	  ◦ Proxy短连接优化。
	  ◦ 优化不同CPU架构下的pause指令执行时间。
	  ◦ 添加内存表查看线程池运行情况。
	• Bug修复
	  ◦ 在低于4.9的Linux Kernel中禁用ppoll，使用poll代替。
	  ◦ 修复wrap_sm4_encrypt函数调用错误问题。
	  ◦ 修复在滚动审核日志时持有全局变量锁的问题。
	  ◦ 修复恢复不一致性检查的问题。
	  ◦ 修复io_statistics表出现错误time值的问题。
	  ◦ 修复无效压缩算法导致崩溃的问题。
	  ◦ 修复用户列与5.6不兼容的问题。
	• 修补程序
	  ◦ Faster DDL：优化DDL操作过程中的Buffer Pool管理机制，降低DDL操作带来的性能影响，提升在线DDL操作的并发数。
	  ◦ 线程池性能优化。
	  ◦ 修复缓冲区计数泄漏问题。

小版本	说明
20200110	• 新特性 Inventory Hint：新增了三个hint， 支持SELECT 、UPDATE、INSERT 、DELETE 语句，快速提交/回滚事务，提高业务吞吐能力。 • 性能优化 ○ 启动实例时，先初始化Concurrency Control队列结构，再初始化Concurrency Control规则。 ○ 异步清除文件时取消小文件的链接。 ○ 优化Thread Pool性能。 ○ 默认情况下禁用恢复不一致性检查。 ○ 更改设置变量所需的权限： ■ 设置以下变量所需的权限已更改为普通用户权限： ■ auto_increment_increment ■ auto_increment_offset ■ bulk_insert_buffer_size ■ binlog_rows_query_log_events ■ 设置以下变量所需的权限已更改为超级用户或系统变量管理用户权限： ■ binlog_format ■ binlog_row_image ■ binlog_direct ■ sql_log_off ■ sql_log_bin
20191225	• 新特性 Recycle Bin：临时将删除的表转移到回收站，还可以设置保留的时间，方便您找回数据。 • 性能优化 ○ 提高短连接处理性能。 ○ 使用专用线程为maintain user服务，避免HA失败。 ○ 通过Redo刷新Binlog时出现错误会显式释放文件同步锁。 ○ 删除不必要的TCP错误日志。 ○ 默认情况下启用线程池。 • Bug修复 ○ 修复慢日志刷新的问题。 ○ 修复锁定范围不正确的问题。 ○ 修复TDE的Select函数导致的核心转储问题。

小版本	说明
20191115	新特性 Statement Queue：针对语句的排队机制，将语句进行分桶排队，尽量把可能具有相同冲突的语句放在一个桶内排队，减少冲突的开销。

20191101	• 新特性
	  ◦ 为TDE添加SM4加密算法。
	  ◦ 保护备实例信息：拥有SUPER或REPLICATION_SLAVE_ADMIN权限的用户才能插入/删除/修改表slave_master_info、slave_relay_log_info、slave_worker_info。
	  ◦ 提高自动递增键的优先级：如果表中没有主键或非空唯一键，具有自动增量的非空键将是第一候选项。
	  ◦ 对系统表和处于初始化状态线程用到的表，不进行Memory引擎到MyISAM引擎的自动转换。
	  ◦ Redo Log刷新到磁盘之前先将Binlog文件刷新到磁盘。
	  ◦ 实例被锁定时也会影响临时表。
	  ◦ 添加新的基于LSM树的事务存储引擎X-Engine。
	• 性能优化
	  ◦ Thread Pool：互斥优化。
	  ◦ Performance Insight：性能点支持线程池。
	  ◦ 参数调整：
	  ▪ `primary_fast_lookup`：会话参数，默认值为true。
	  ▪ `thread_pool_enabled`：全局参数，默认值为true。

小版本	说明

20191015	• 新特性
	  ○ TDE：支持透明数据加密TDE（Transparent Data Encryption）功能，可对数据文件执行实时I/O加密和解密，数据在写入磁盘之前进行加密，从磁盘读入内存时进行解密。
	  ○ Returning：Returning功能支持DML语句返回Resultset，同时提供了工具包（DBMS_TRANS）便于您快捷使用。
	  ○ 强制将引擎从MyISAM或MEMORY转换为InnoDB：如果全局变量**force_mysiam_to_innodb**或**force_memory_to_innodb**为**ON**，则创建和修改表时会将表引擎从MyISAM或MEMORY转换为InnoDB。
	  ○ 禁止非高权限账号切换主备实例。
	  ○ 性能代理插件：收集性能数据并保存到本地格式化文本文件，采用文件轮循方式，保留最近的秒级性能数据。
	  ○ Innodb mutex timeout configurable：可配置全局变量**innodb_fatal_semaphore_wait_threshold**，默认值：600。
	  ○ 忽略索引提示错误：可配置全局变量**ignore_index_hint_error**，默认值：false。
	  ○ 可关闭SSL加密功能。
	  ○ TCP错误信息：返回TCP方向（读取、读取等待、写入等待）错误及错误代码到end_connection事件，并且输出错误信息到错误日志。
	• Bug修复
	  ○ 支持本地AIO的Linux系统内，在触发线性预读之前会合并AIO请求。
	  ○ 优化表/索引统计信息。
	  ○ 如果指定了主键，则直接访问主索引。

20190915	Bug修复 修复Cmd_set_current_connection内存泄露问题。

小版本	说明

20190816	• 新特性
	  ○ Thread Pool：将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。
	  ○ Statement Concurrency Control：通过控制并发数应对突发的数据库请求流量、资源消耗过高的语句访问以及SQL访问模型的变化，保证MySQL实例持续稳定运行。
	  ○ Statement Outline：利用Optimizer Hint和Index Hint让MySQL稳定执行计划。
	  ○ Sequence Engine：简化获取序列值的复杂度。
	  ○ Purge Large File Asynchronously：删除单个表空间时，会将表空间文件重命名为临时文件，等待异步清除进程清理临时文件。
	  ○ Performance Insight：专注于实例负载监控、关联分析、性能调优的利器，帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。
	  ○ 优化实例锁状态：实例锁定状态下，可以drop或truncate表。
	• Bug修复
	  ○ 修复文件大小计算错误的问题。
	  ○ 修复偶尔出现的内存空闲后再次使用的问题。
	  ○ 修复主机缓存大小为0时的崩溃问题。
	  ○ 修复隐式主键与CTS语句的冲突问题。
	  ○ 修复慢查询导致的slog出错问题。

20190601	• 性能优化
	  ○ 缩短日志表MDL范围，减少MDL阻塞的可能性。
	  ○ 重构终止选项的代码。
	• Bug修复
	  ○ 修复审计日志中没有记录预编译语句的问题。
	  ○ 屏蔽无效表名的错误日志。

MySQL 8.0三节点企业版

小版本	说明
20210305	Bug修复 修复performance schema的内存泄漏Bug。
20200918	新特性 支持独享的proxy开启SSL加密。
20200805	Bug修复 修复一个元数据Bug。

小版本	说明
20200608	• 新特性 Recycle Bin：新增支持 `TRUNCATE TABLE` 命令，执行时将原始表移动到专门的recycle bin目录中，并使用相同的结构创建新表。 • 性能优化 ○ 默认关闭TCP错误的输出。 ○ 提高线程池默认配置下的性能。 • Bug修复 ○ 修复因为 `#p` 分割分区文件名导致的数据库、表无效问题。 ○ 修复CCL匹配时区分大小写问题，即不再区分大小写。 • 合并官方8.0.17、8.0.18变更，更多信息，请参见Changes in MySQL 8.0.17和Changes in MySQL 8.0.18。
20200317	• 新特性 ○ Performance Agent：更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。 ○ 在半同步模式下添加网络往返时间，并记录到性能数据。 • 性能优化 ○ 允许在只读实例上进行语句级并发控制（CCL）操作。 ○ 备实例支持Outline。 ○ Proxy短连接优化。 ○ 优化不同CPU架构下的pause指令执行时间。 ○ 添加内存表查看线程池运行情况。 • Bug修复 ○ 在低于4.9的Linux Kernel中禁用ppoll，使用poll代替。 ○ 修复wrap_sm4_encrypt函数调用错误问题。 ○ 修复在滚动审核日志时持有全局变量锁的问题。 ○ 修复恢复不一致性检查的问题。 ○ 修复io_statistics表出现错误time值的问题。 ○ 修复无效压缩算法导致崩溃的问题。 ○ 修复用户列与5.6不兼容的问题。 • 修补程序 ○ Faster DDL：优化DDL操作过程中的Buffer Pool管理机制，降低DDL操作带来的性能影响，提升在线DDL操作的并发数。 ○ 线程池性能优化。 ○ 修复缓冲区计数泄漏问题。

MySQL 5.7基础版或高可用版

小版本	说明

20210630	• 新特性
	  ◦ 审计日志支持MYSQL_V3格式。
	  ◦ 允许用户对slow log和general log执行truncate操作。
	  ◦ 增加线程栈内存溢出检查。
	  ◦ 增加参数thread_pool_strict_mode以控制最大worker线程数量。
	• Bug修复
	  ◦ mysqld_safe脚本启动mysqld时删除老的socket lock文件。
	  ◦ 修复recycle_bin内存泄漏问题。

20210430	• 新特性
	  ◦ 支持Recycle Bin。更多信息，请参见Recycle Bin。
	  ◦ 支持SEQUENCE Engine。更多信息，请参见Sequence Engine。
	• 性能优化
	  ◦ `I_S.PERF_STATISTICS` 表中增加更多性能数据指标。
	  ◦ `I_S.INDEX_STATISTICS` 表中新增索引扫描次数。
	  ◦ 优化TDE性能。
	• Bug修复
	  修复社区版本回滚过程中生成列处理异常Bug。

20201031	Bug修复 • 修复并发更新导致ROW_SEARCH_MVCC崩溃的问题。 • 修复更改innodb_undo_tablespaces导致无法启动的问题。 • 修复FTS查询导致缓存溢出的问题。

小版本	说明

20200831	• 新特性
	  ◦ 合并官方5.7.30变更，更多信息，请参见GitHub。
	  ◦ 优化CCL（Concurrency Control）的等待与并发机制。
	  ◦ MySQL Binlog工具增加开始GTID（start gtid）和结束GTID（stop gtid）功能。
	  ◦ 支持输出Redo Log的各个LSN值：
	  ■ innodb_lsn：重做日志的lsn编号。
	  ■ innodb_log_write_lsn：log写入的lsn。
	  ■ innodb_log_checkpoint_lsn：最后检查点的lsn。
	  ■ innodb_log_flushed_lsn：磁盘上刷新redo log的lsn。
	  ■ innodb_log_Pages_flushed：页面刷新的lsn。
	• 性能优化
	调整Concurrency Control在存储过程中的执行优先级。
	• Bug修复
	SQL运行期间使用的临时表Page可能发生引用计数泄漏，这可能导致整个Buffer Pool脏块刷新效率低下，引发Buffer Pool中无可用空闲Page，严重影响数据库的运行效率。更多信息，请参见官方文档。

20200630	• 新特性
	  ◦ Inventory Hint：新增了三个hint， 支持SELECT、UPDATE、INSERT、DELETE 语句，快速提交/回滚事务，提高业务吞吐能力。
	  ◦ Statement Concurrency Control：通过控制并发数应对突发的数据库请求流量、资源消耗过高的语句访问以及SQL访问模型的变化，保证MySQL实例持续稳定运行。
	  ◦ Statement Queue：针对语句的排队机制，将语句进行分桶排队，尽量把可能具有相同冲突的语句放在一个桶内排队，减少冲突的开销。
	  ◦ Statement Outline：利用Optimizer Hint和Index Hint让MySQL稳定执行计划。
	  ◦ Faster DDL：优化DDL操作过程中的Buffer Pool管理机制，降低DDL操作带来的性能影响，提升在线DDL操作的并发数。
	  ◦ 增加连接数上限，最大支持500,000连接。
	• 性能优化
	  ◦ 可通过 `call dbms_admin.show_native_procedure();` 命令查看所有本机过程。
	  ◦ 新增删除孤立表的函数。
	  ◦ 线程池内部优化。
	  ◦ 优化查询缓存。
	  ◦ 根据实例规格设置Performance Schema占用内存的上限。
	• Bug修复
	修复审计刷新线程进入死循环的问题。

小版本	说明
20200430	• 性能优化 QueryCache中删除rwlock，并将默认哈希函数从LF_hash改为murmur3 hash。 • Bug修复 修复在事务隔离（可重复读级别）中命中查询缓存时的两个错误。
20200331	• 新特性 ○ Fast Query Cache：针对原生MySQL Query Cache的不足，阿里云进行重新设计和全新实现，推出RDS Query Cache，能够有效提高数据库查询性能。 ○ 从percona-server 5.7移植两个MDL锁，LOCK TABLES FOR BACKUP （LTFB）和LOCK BINLOG FOR BACKUP （LBFB）。 • 性能优化 ○ 添加线程池对低版本的兼容。 ○ 默认关闭TCP错误的输出。 ○ 提高线程池默认配置下的性能。 • Bug修复 ○ 修复清理大文件时包含临时文件的问题。 ○ 修复线程池转储线程超时的问题。 ○ 修复进程上下文中IPK字段计数错误的问题。 ○ 修复rds_change_user导致的pfs线程泄漏和释放问题。 • 合并官方5.7.28变更，更多信息，请参见GitHub。
20200229	• 新特性 ○ Performance Agent：更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。 ○ 在半同步模式下添加网络往返时间，并记录到性能数据。 • 性能优化 ○ 优化不同CPU架构下的pause指令执行时间。 ○ Proxy短连接优化。 ○ 添加内存表查看线程池运行情况。 • Bug修复 ○ 修复DDL重做日志不安全的问题。 ○ 修复io_statistics表出现错误time值的问题。 ○ 修复更改表导致服务器崩溃的问题。 ○ 修复MySQL测试用例。

小版本	说明
20200110	性能优化 • 异步清除文件时取消小文件的链接。 • 优化Thread Pool性能。 • thread_pool_enabled参数的默认值调整为OFF。
20191225	• 新特性 内部账户管理与防范：调整用户权限保护数据安全。 • 性能优化 ◦ 提高短连接处理性能。 ◦ 使用专用线程为maintain user服务，避免HA失败。 ◦ 删除不必要的TCP错误日志。 ◦ 优化线程池。 • Bug修复 ◦ 修复读写分离时mysqld进程崩溃问题。 ◦ 修复密钥环引起的核心转储问题。
20191115	Bug修复 修复主备切换后审计日志显示变量的问题。
20191101	• 新特性 ◦ 为TDE添加SM4加密算法。 ◦ 如果指定了主键，则直接访问主索引。 ◦ 对系统表和处于初始化状态线程用到的表，不进行Memory引擎到MyISAM引擎的自动转换。 • 性能优化 ◦ Thread Pool：互斥优化。 ◦ 引入审计日志缓冲机制，提高审计日志的性能。 ◦ Performance Insight：性能点支持线程池。 ◦ 默认开启Thread Pool。 • Bug修复 ◦ 在处理维护用户列表时释放锁。 ◦ 补充更多TCP错误信息。

小版本	说明

20191015	• 新特性
	  ◦ 轮换慢日志：为了在收集慢查询日志时保证零数据丢失，轮换日志表会将慢日志表的csv数据文件重命名为唯一名称并创建新文件。您可以使用 `show variables like '%rotate_log_table%';` 查看是否开启轮换慢日志。
	  ◦ 性能代理插件：收集性能数据并保存到本地格式化文本文件，采用文件轮循方式，保留最近的秒级性能数据。
	  ◦ 强制将引擎从MEMORY转换为InnoDB：如果全局变量**rds_force_memory_to_innodb**为**ON**，则创建/修改表时会将表引擎从MEMORY转换为InnoDB。
	  ◦ TDE机制优化：添加keyring-rds插件与管控系统/密钥管理服务进行交互。
	  ◦ TCP错误信息：返回TCP方向（读取、读取等待、写入等待）错误及错误代码到end_connection事件，并且输出错误信息到错误日志。
	• Bug修复
	修复DDL中的意外错误Error 1290。

20190925	参数修改 • 将系统变量auto_generate_certs的默认值由true改为false。 • 增加全局只读变量auto_detact_certs，默认值为false，有效值为[true | false]。 该系统变量在Server端使用OpenSSL编译时可用，用于控制Server端在启动时是否在数据目录下自动查找SSL加密证书和密钥文件，即控制是否开启Server端的证书和密钥的自动查找功能。
20190915	新特性 Thread Pool：将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。

20190815	• 新特性
	  ◦ Purge Large File Asynchronously：删除单个表空间时，会将表空间文件重命名为临时文件，等待异步清除进程清理临时文件。
	  ◦ Performance Insight：专注于实例负载监控、关联分析、性能调优的利器，帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。
	  ◦ 优化实例锁状态：实例锁定状态下，可以drop或truncate表。
	• Bug修复
	  ◦ 禁止在 `set rds_current_connection` 命令中设置rds_prepare_begin_id。
	  ◦ 允许更改已锁定用户的信息。
	  ◦ 禁止用关键字actual作为表名。
	  ◦ 修复慢日志导致时间字段溢出的问题。

20190510	新特性 允许在事务内创建临时表。

小版本	说明
20190319	新特性 支持在handshake报文内代理设置threadID。

20190131	• 性能优化
	  ◦ 升级到官方5.7.25版本。
	  ◦ 关闭内存管理功能jemalloc。
	• Bug修复
	修复内部变量net_lenth_size计算错误问题。

20181226	• 新特性 支持动态修改binlog-row-event-max-size，加速无主键表的复制。 • Bug修复 修复Proxy实例内存申请异常的问题。
20181010	性能优化 • 支持隐式主键。 • 加快无主键表的主备复制。 • 支持Native AIO，提升I/O性能。
20180431	新特性 • 支持高可用版。 • 支持SQL审计。 • 增强对处于快照备份状态的实例的保护。

MySQL 5.7三节点企业版

小版本	说明
20201229	Bug修复 • 修复非预期的数据库崩溃和卡住的问题。 • 修复前缀索引被判定为invisible index索引的问题。 • 修复一致性协议中logger节点的状态异常问题，该问题可能导致集群无法正确选举主节点。

小版本	说明
20191128	• 新特性 支持读写分离。 • Bug修复 ○ 修复部分场景下Follower Second_Behind_Master计算错误问题。 ○ 修复表级并行复制事务重试时死锁问题。 ○ 修复XA相关bug。
20191016	• 新特性 ○ 支持MySQL 5.7高可用版（本地SSD盘）升级到三节点企业版。 ○ 兼容MySQL官方GTID功能，默认不开启。 ○ 合并AliSQL MySQL 5.7基础版/高可用版 20190915版本及之前的自研功能。 • Bug修复 修复重置备实例导致binlog被关闭问题。
20190909	• 新特性 ○ 优化大事务在三节点强一致状态下的执行效率。 ○ 支持从Leader/Follower进行Binlog转储。 ○ 支持创建只读实例。 ○ 系统表默认使用InnoDB引擎。 • Bug修复 ○ 修复Follower日志清理命令失效问题。 ○ 修复参数slave_sql_verify_checksum=OFF和binlog_checksum=crc32时Slave线程异常退出问题。
20190709	新特性 • 支持三节点功能。 • 禁用semi-sync插件。 • 支持表级并行复制、Writeset并行复制。 • 支持pk_access主键查询加速。 • 支持线程池。 • 合并AliSQL MySQL 5.7基础版/高可用版 20190510版本及之前的自研功能。

MySQL 5.6

小版本	说明

小版本	说明
20210630	• 新特性 增加线程栈内存溢出检查。 • Bug修复 ○ 修复MySQL 5.6社区版本的Bug。 ○ 放开mysql库下SHOW CREATE语句的操作限制。
20210430	• 性能优化 `I_S.PERF_STATISTICS` 表中增加更多性能数据指标。 • Bug修复 修复了社区版本中几个内存泄漏的Bug。
20201031	Bug修复 • 修复IN子句内的子查询无效的问题。 • 修复进程权限错误的问题。 • 修复kill_user_list表中用户授权的问题。 • 修复 `DROP DATABASE` 语句出错的问题。 • 修复PREVIOUS_GTID事件导致SECONDS_BEHIND_MASTER计算错误的问题。
20200831	• 新特性 支持输出Redo Log的各个LSN值： ○ innodb_lsn：重做日志的lsn编号。 ○ innodb_log_write_lsn：log写入的lsn。 ○ innodb_log_checkpoint_lsn：最后检查点的lsn。 ○ innodb_log_flushed_lsn：磁盘上刷新redo log的lsn。 ○ innodb_log_Pages_flushed：页面刷新的lsn。 • Bug修复 ○ 修复SHOW_HA_ROWS类型错误的问题。 ○ 修复过程上下文中IPK字段计数错误的问题。 ○ 修复查询INFORMATION_SCHEMA导致服务器崩溃的问题。 ○ 修复审计刷新线程死循环的问题。 ○ 修复备实例不报告主备延迟的问题。

小版本	说明

20200630	• 新特性
	  ◦ Performance Agent：更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。
	  ◦ 增加连接数上限，最大支持500,000连接。
	  ◦ Faster DDL：优化DDL操作过程中的Buffer Pool管理机制，降低DDL操作带来的性能影响，提升在线DDL操作的并发数。
	• 性能优化
	  ◦ 增加全局参数max_execution_time，当SQL语句执行时间超过此参数值时会被中断。
	  ◦ 线程池内部优化。
	• Bug修复
	  ◦ 修复读取客户端命令时的等待计数不正确的问题。
	  ◦ 修复普通账号没有DROP DATABASE命令执行权限的问题。

20200430	• 新特性 增加存储MDL锁信息的表mdl_info。 • Bug修复 解决同时开启线程池和ic_reduce（秒杀）功能的冲突问题。
20200331	性能优化 • 提高线程池默认配置下的性能。 • 默认关闭TCP错误的输出。

20200229	• 新特性
	支持Proxy读写分离功能。
	• 性能优化
	  ◦ 优化线程池功能。
	  ◦ 优化不同CPU架构下的pause指令执行时间。
	• Bug修复
	修复XA事务部分提交的问题。

小版本	说明
20200110	• 新特性 Thread Pool：将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。 • 性能优化 异步清除文件时取消小文件的链接。 • Bug修复 ◦ 修复页面清理程序的睡眠时间计算不正确问题。 ◦ 修复 `SELECT @@global.gtid_executed` 导致的故障转移失败问题。 ◦ 修复IF CLIENT KILLED AFTER ROLLBACK TO SAVEPOINT PREVIOUS STMTS COMMITTED问题。
20191212	性能优化 删除不必要的tcp错误日志
20191115	Bug修复 修复慢日志时间戳溢出问题。
20191101	Bug修复 • 修复刷新日志时切换慢日志的问题，仅在执行刷新慢日志时切换慢日志。 • 修正部分显示错误。
20191015	• 新特性 ◦ 轮换慢日志：为了在收集慢查询日志时保证零数据丢失，轮换日志表会将慢日志表的csv数据文件重命名为唯一名称并创建新文件。您可以使用 `show variables like '%rotate_log_table%';` 查看是否开启轮换慢日志。 ◦ SM4加密算法：添加新的SM4加密算法，取代旧的SM加密算法。 ◦ Purge Large File Asynchronously：删除单个表空间时，会将表空间文件重命名为临时文件，等待异步清除进程清理临时文件。 ◦ TCP错误信息：返回TCP方向（读取、读取等待、写入等待）错误及错误代码到end_connection事件，并且输出错误信息到错误日志。 ◦ 引入审计日志缓冲机制，提高审计日志的性能。 • Bug修复 ◦ 禁用pstack，避免存在大量连接时可能导致pstack无响应。 ◦ 修复隐式主键与 `create table as select` 语句之间的冲突。 ◦ 自动清除由二进制日志创建的临时文件。
20190815	性能优化 优化实例锁状态：实例锁定状态下，可以drop或truncate表。

小版本	说明
20190130	Bug修复 修复部分可能导致系统不稳定的Bug。
20181010	性能优化 添加参数rocksdb_ddl_commit_in_the_middle（MyRocks）。如果这个参数被打开，部分DDL在执行过程中将会执行commit操作。
201806**	新特性 slow log精度提升为微秒。
20180426	• 新特性 引入隐藏索引，支持将索引设置为不可见，更多信息，请参见参考文档。 • Bug修复 ○ 修复备库apply线程的bug。 ○ 修复备库apply分区表更新时性能下降问题。 ○ 修复TokuDB下alter table comment重建整张表问题，更多信息，请参见参考文档。 ○ 修复由show slave status/show status可能触发的死锁问题。
20171205	Bug修复 • 修复OPTIMIZE TABLE和ONLINE ALTER TABLE同时执行时会触发死锁的问题。 • 修复SEQUENCE与隐含主键冲突的问题。 • 修复SHOW CREATE SEQUENCE问题。 • 修复TokuDB引擎的表统计信息错误。 • 修复并行OPTIMIZE表引入的死锁问题。 • 修复QUERY_LOG_EVENT中记录的字符集问题。 • 修复信号处理引起的数据库无法停止问题，更多信息，请参见参考文档。 • 修复RESET MASTER引入的问题。 • 修复备库陷入等待的问题。 • 修复SHOW CREATE TABLE可能触发的进程崩溃问题。
20170927	Bug修复 修复TokuDB表查询时使用错误索引问题。
20170901	• 新特性 ○ 升级SSL加密版本到TLS 1.2，更多信息，请参见参考文档。 ○ 支持SEQUENCE。 • Bug修复 修复NOT IN查询在特定场景下返回结果集有误的问题。

小版本	说明
20170530	新特性 支持高权限账号Kill其他账号下的连接。
20170221	新特性 支持开启代理终端。

MySQL 5.5

小版本	说明
20181212	Bug修复 修复调用系统函数 `gettimeofday(2)` 返回值不准确的问题。该系统函数返回值为时间，常用来计算等待超时，时间不准确时会导致一些操作永不超时。

3.X-Engine引擎

3.1. X-Engine简介

X-Engine是阿里云数据库产品事业部自研的联机事务处理OLTP（On-Line Transaction Processing）数据库存储引擎。作为自研数据库PolarDB的存储引擎之一，已经广泛应用在阿里集团内部诸多业务系统中，包括交易历史库、钉钉历史库等核心应用，大幅缩减了业务成本，同时也作为双十一大促的关键数据库技术，挺过了数百倍平时流量的冲击。

为什么设计一个新的存储引擎

X-Engine的诞生是为了应对阿里内部业务的挑战，早在2010年，阿里内部就大规模部署了MySQL数据库，但是业务量的逐年爆炸式增长，数据库面临着极大的挑战：

- 极高的并发事务处理能力（尤其是双十一的流量突发式暴增）。
- 超大规模的数据存储。

这两个问题虽然可以通过扩展数据库节点的分布式方案解决，但是堆机器不是一个高效的手段，我们更想用技术的手段将数据库性价比提升到极致，实现以少量资源换取性能大幅提高的目的。

传统数据库架构的性能已经被仔细的研究过，数据库领域的泰斗，图灵奖得主Michael Stonebreaker就此写过一篇论文 *OLTP Through the Looking Glass, and What We Found There*，指出传统关系型数据库，仅有不到10%的时间是在做真正有效的数据处理工作，剩下的时间都浪费在其它工作上，例如加锁等待、缓冲管理、日志同步等。

造成这种现象的原因是因为近年来我们所依赖的硬件体系发生了巨大的变化，例如多核（众核）CPU、新的处理器架构（Cache/NUMA）、各种异构计算设备（GPU/FPGA）等，而架构在这些硬件之上的数据库软件却没有太大的改变，例如使用B-Tree索引的固定大小的数据页（Page）、使用ARIES算法的事务处理与数据恢复机制、基于独立锁管理器的并发控制等，这些都是为了慢速磁盘而设计，很难发挥出现有硬件体系应有的性能。

基于以上原因，阿里开发了适合当前硬件体系的存储引擎，即X-Engine。

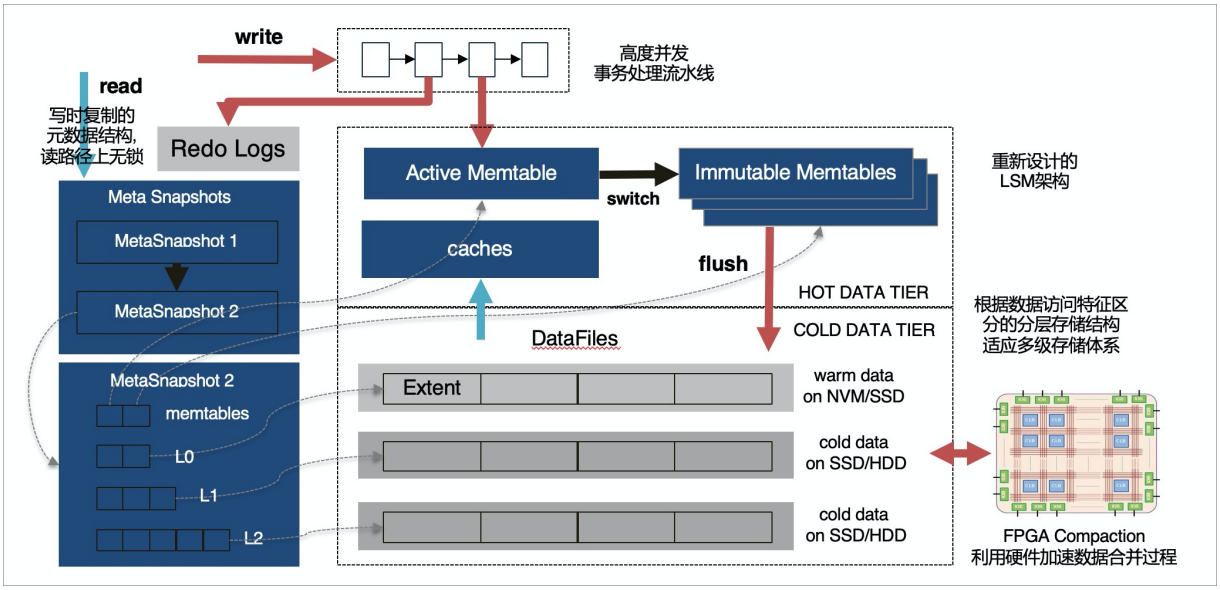
X-Engine架构

全新架构的X-Engine存储引擎不仅可以无缝对接兼容MySQL（得益于MySQL Pluginable Storage Engine特性），同时X-Engine使用分层存储架构。

因为目标是面向大规模的海量数据存储，提供高并发事务处理能力和降低存储成本，在大部分大数据量场景下，数据被访问的机会是不均等的，访问频繁的热数据实际上占比很少，X-Engine根据数据访问频度的不同将数据划分为多个层次，针对每个层次数据的访问特点，设计对应的存储结构，写入合适的存储设备。

X-Engine使用了LSM-Tree作为分层存储的架构基础，并进行了重新设计：

- 热数据层和数据更新使用内存存储，通过内存数据库技术（Lock-Free index structure/append only）提高事务处理的性能。
- 流水线事务处理机制，把事务处理的几个阶段并行起来，极大提升了吞吐。
- 访问频度低的数据逐渐淘汰或是合并到持久化的存储层次中，并结合多层次的存储设备（NVM/SSD/HDD）进行存储。
- 对性能影响比较大的Compaction过程做了大量优化：
  - 拆分数据存储粒度，利用数据更新热点较为集中的特征，尽可能的在合并过程中复用数据。
  - 精细化控制LSM的形状，减少I/O和计算代价，有效缓解了合并过程中的空间增大。
- 同时使用更细粒度的访问控制和缓存机制，优化读的性能。



> **说明** X-Engine的架构和优化技术已经被总结成论文 *X-Engine: An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing*，在数据管理国际会议SIGMOD'19发表，这是中国内地公司首次在国际性学术会议上发表OLTP数据库内核相关的技术成果。

技术特点

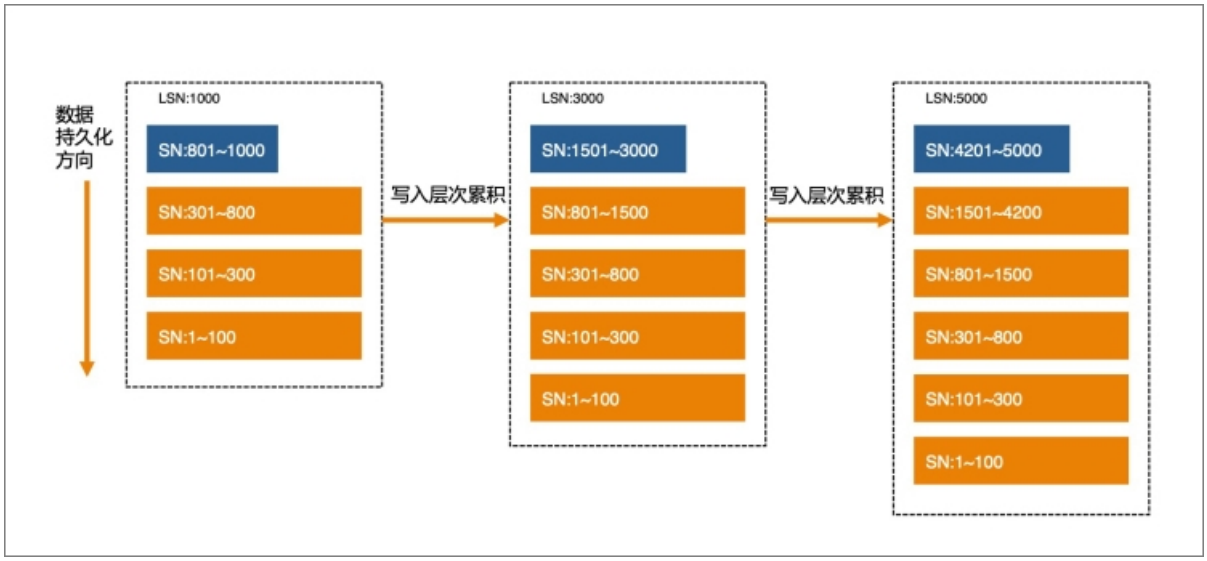
- 利用FPGA硬件加速Compaction过程，使得系统上限进一步提升。这个技术属首次将硬件加速技术应用到在线事务处理数据库存储引擎中，相关论文 *FPGA-Accelerated Compactions for LSM-based Key Value Store*已经被2020年的FAST'20国际会议接收。
- 通过数据复用技术减少数据合并代价，同时减少缓存淘汰带来的性能抖动。
- 使用多事务处理队列和流水线处理技术，减少线程上下文切换代价，并计算每个阶段任务量配比，使整个流水线充分流转，极大提升事务处理性能。相对于其他类似架构的存储引擎（例如RocksDB），X-Engine的事务处理性能有10倍以上提升。
- X-Engine使用的Copy-on-write技术，避免原地更新数据页，从而对只读数据页面进行编码压缩，相对于传统存储引擎（例如InnoDB），使用X-Engine可以将存储空间降低至10%~50%。
- Bloom Filter快速判定数据是否存在，Surf Filter判断范围数据是否存在，Row Cache缓存热点行，加速读取性能。

LSM基本逻辑

LSM的本质是所有写入操作直接以追加的方式写入内存。每次写到一定程度，即冻结为一层（Level），并写入持久化存储。所有写入的行，都以主键（Key）排序好后存放，无论是在内存中，还是持久化存储中。在内存中即为一个排序的内存数据结构（Skiplist、B-Tree、etc），在持久化存储也作为一个只读的全排序持久化存储结构。

普通的存储系统若要支持事务处理，需要加入一个时间维度，为每个事务构造出一个不受并发干扰的独立视域。例如存储引擎会对每个事务定序并赋予一个全局单调递增的事务版本号（SN），每个事务中的记录会存储这个SN以判断独立事务之间的可见性，从而实现事务的隔离机制。

如果LSM存储结构持续写入，不做其他的动作，那么最终会成为如下结构。



这种结构对于写入是非常友好的，只要追加到最新的内存表中即完成，为实现故障恢复，只需记录Redo Log，因为新数据不会覆盖旧版本，追加记录会形成天然的多版本结构。

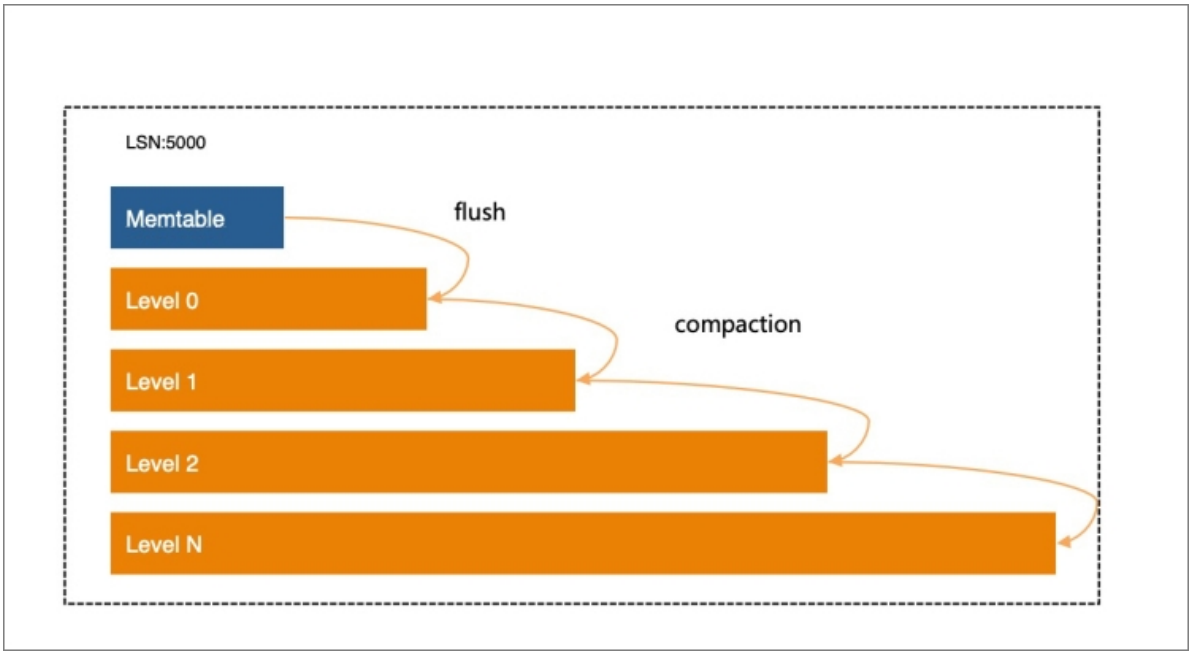
但是如此累积，冻结的持久化层次越来越多，会对查询会产生不利的影响。例如对同一个key，不同事务提交产生的多版本记录会散落在各个层次中；不同的key也会散落在不同层次中。读操作需要查找各个层并合并才能得到最终结果。

因此LSM引入了Compaction操作解决这个问题，Compaction操作有2种作用：

- 控制LSM层次形状

  一般的LSM形状都是层次越低，数据量越大（倍数关系），目的是为了提升读性能。

  通常存储系统的数据访问都有局部性，大量的访问都集中在少部分数据上，这也是缓存系统能有效工作的基本前提。在LSM存储结构中，如果把访问频率高的数据尽可能放在较高的层次上，存放在快速存储设备中（例如NVM、DRAM），而把访问频率低的数据放在较低层次中，存放在廉价慢速存储设备中。这就是X-Engine的冷热分层概念。



- 合并数据

Compaction操作不断的把相邻层次的数据合并，并写入更低层次。合并的过程实际上是把要合并的相邻两层或多层的数据读出来，按key排序，相同的key如果有多个版本，只保留新的版本（比当前正在执行的活跃事务中最小版本号新），丢掉旧版本数据，然后写入新的层，这个操作非常耗费资源。

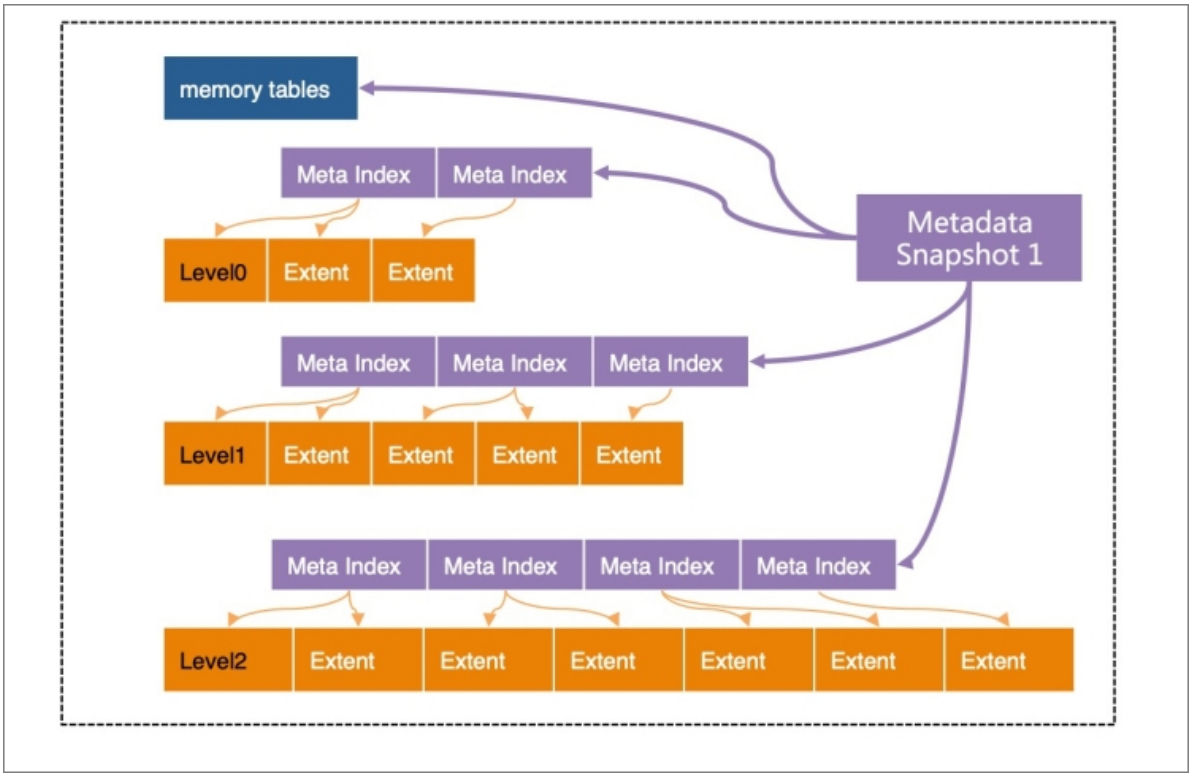
合并数据除了考虑冷热分层以外，还需要考虑其他维度，例如数据的更新频率，大量的多版本数据在查询的时候会浪费更多的I/O和CPU，因此需要优先进行合并以减少记录的版本数量。X-Engine综合考虑了各种策略形成自己的Compaction调度机制。

高度优化的LSM

X-Engine的memory tables使用了无锁跳表（Locked-free SkipList），并发读写的性能较高。在持久化层如何实现高效，就需要讨论每层的细微结构。

- 数据组织

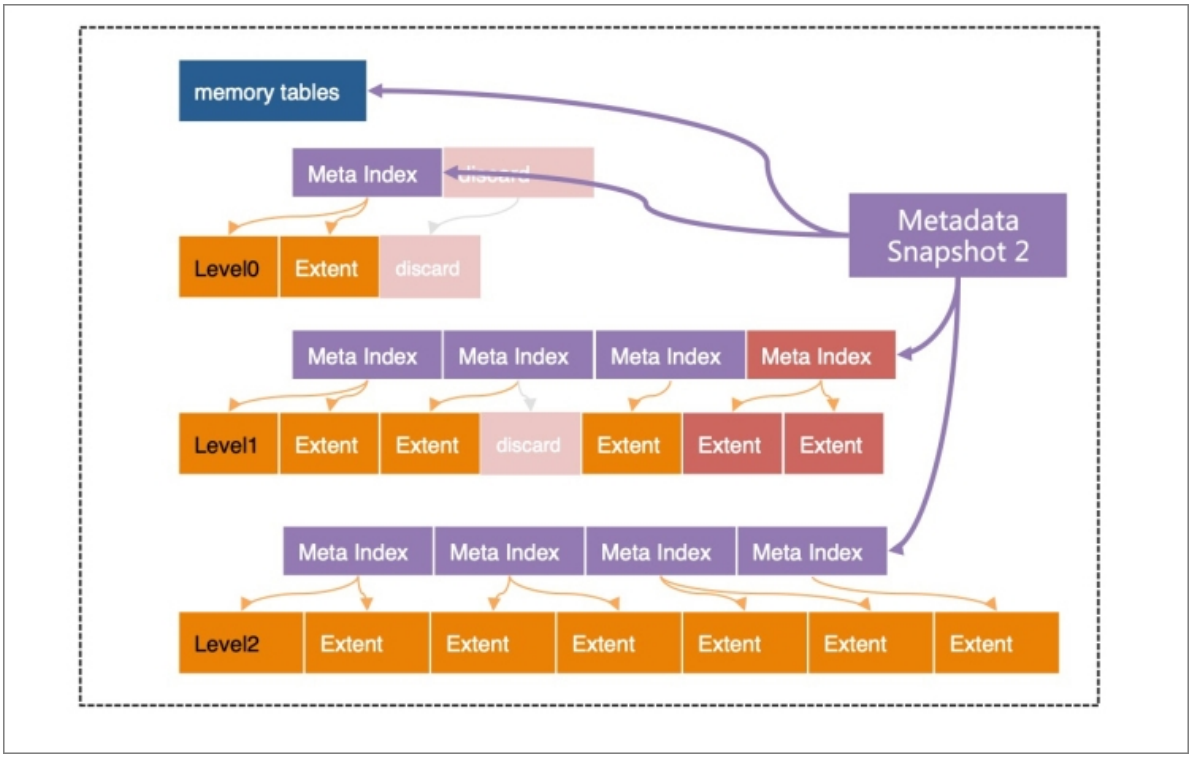
  X-Engine的每层都划分成固定大小的Extent，存放每个层次中的数据的一个连续片段（Key Range）。为了快速定位Extent，为每层Extents建立了一套索引（Meta Index），所有这些索引，加上所有的memory tables（active/immutable）一起组成了一个元数据树（Metadata Tree），root节点为Metadata Snapshot，这个树结构类似于B-Tree。



X-Engine中除了当前的正在写入的active memory tables以外，其他结构都是只读的，不会被修改。给定某个时间点，例如LSN=1000，上图中的Metadata Snapshot 1引用到的结构即包含了LSN=1000时的所有的数据的快照，因此这个结构被称为Snapshot。

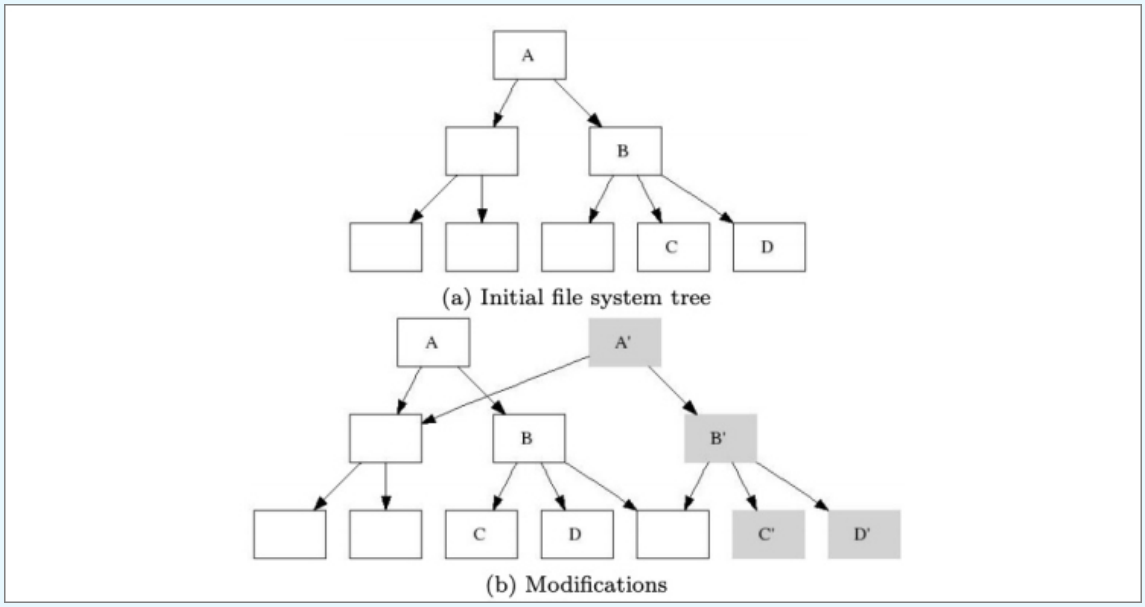
即便是Metadata结构本身，也是一旦生成就不会被修改。所有的读请求都是以Snapshot为入口，这是X-Engine实现Snapshot级别隔离的基础。前文说过随着数据写入，累积数据越多，会执行Compaction操作、冻结memory tables等，这些操作都是用Copy-on-write实现，即每次都将修改产生的结果写入新的Extent，然后生成新的Meta Index结构，最终生成新的Metadata Snapshot。

例如执行一次Compaction操作会生成新的Metadata Snapshot，如下图所示。



可以看到Metadata Snapshot 2相对于Metadata Snapshot 1并没有太多的变化，仅仅修改了发生变更的一些叶子节点和索引节点。

> **说明** 这个技术颇有些类似 B-trees, Shadowing, and Clones，如果您阅读那篇论文，会对理解这个过程有所帮助。



- 事务处理

  得益于LSM的轻量化写机制，写入操作固然是其明显的优势，但是事务处理不只是把更新的数据写入系统那么简单，还要保证ACID（原子性、一致性、隔离性、持久性），涉及到一整套复杂的流程。X-Engine将整个事务处理过程分为两个阶段：

i. 读写阶段

校验事务的冲突（写写冲突、读写冲突），判断事务是否可以执行、回滚重试或者等锁。如果事务冲突校验通过，则把修改的所有数据写入Transaction Buffer。

ii. 提交阶段

写WAL、写内存表，以及提交并返回用户结果，这里面既有I/O操作（写日志、返回消息），也有CPU操作（拷贝日志、写内存表）。

为了提高事务处理吞吐，系统内会有大量事务并发执行，单个I/O操作比较昂贵，大部分存储引擎会倾向于聚集一批事务一起提交，称为Group Commit，能够合并I/O操作。但是一组事务提交的过程中，还是有大量等待过程的，例如写入日志到磁盘过程中，除了等待落盘无所事事。

X-Engine为了进一步提升事务处理的吞吐，使用流水线技术，把提交阶段分为4个独立的更精细的阶段：

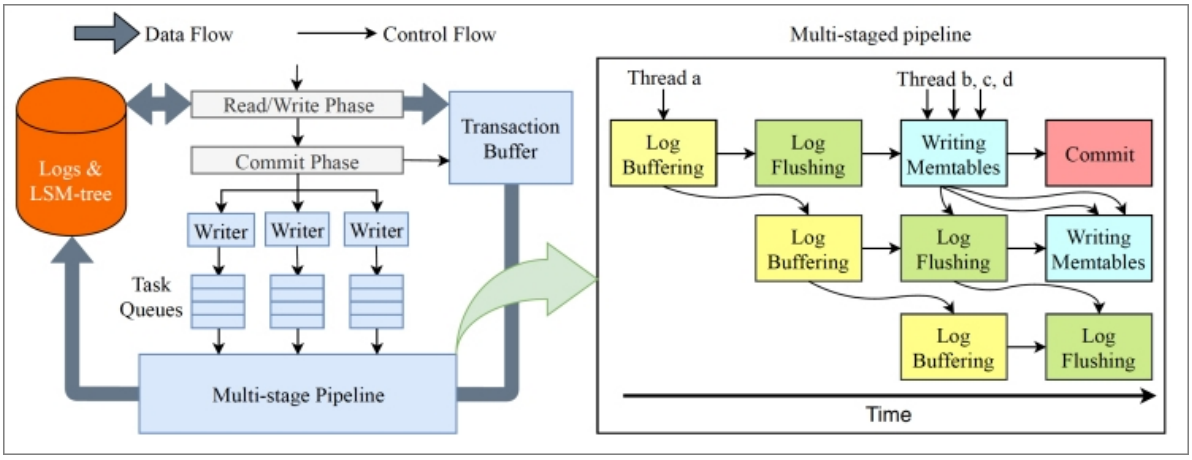
i. 拷贝日志到缓冲区（Log Buffer）

ii. 日志落盘（Log Flush）

iii. 写内存表（Write memory table）

iv. 提交返回（Commit）

事务到了提交阶段，可以自由选择执行流水线中任意一个阶段，只要流水线任务的大小划分得当，就能充分并行起来，流水线处于接近满载状态。另外这里利用的是事务处理的线程，而非后台线程，每个线程在执行的时候，选择流水线中的一个阶段执行任务，或者空闲后处理其他请求，没有等待，也无需切换，充分利用了每个线程的能力。

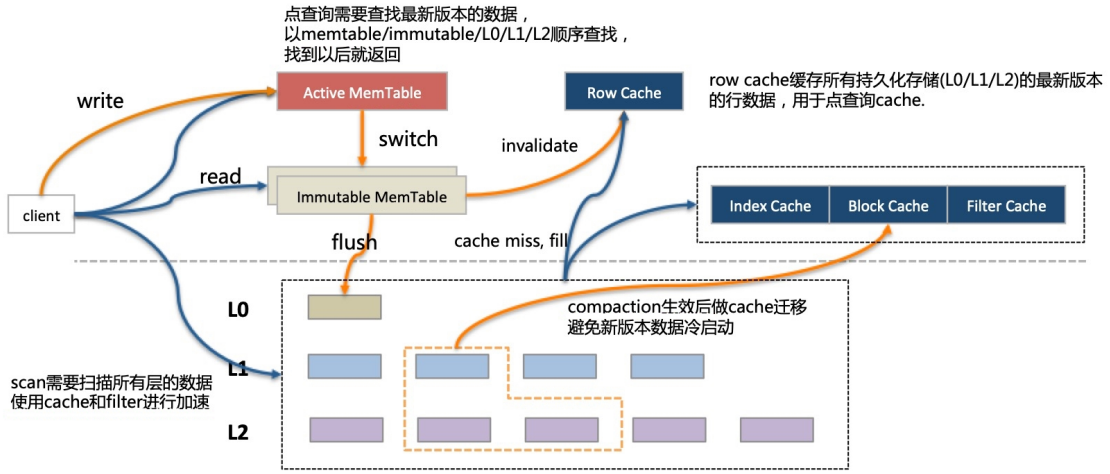


- 读操作

LSM处理多版本数据的方式是新版本数据记录会追加在老版本数据后面，从物理上看，一条记录不同的版本可能存放在不同的层，在查询的时候需要找到合适的版本（根据事务隔离级别定义的可见性规则），一般查询都是查找最新的数据，总是由最高的层次往低层次找。

对于单条记录的查找而言，一旦找到便可以终止，如果记录在比较高的层次，例如memory tables，很快便可以返回；如果记录已经落入了很低的层次，那就得逐层查找，也许Bloom Filter可以跳过某些层次加快这个旅程，但毕竟还是有很多的I/O操作。X-Engine针对单记录查询引入了Row Cache，在所有持久化的层次的数据之上做了一个缓存，在memory tables中没有命中的单行查询，在Row Cache之中也会被捕获。Row Cache需要保证缓存了所有持久化层次中最新版本的记录，而这个记录是可能发生变化的，例如每次flush将只读的memory tables写入持久化层次时，就需要恰当的更新Row Cache中的缓存记录，这个操作比较微妙，需要精心的设计。

对于范围扫描而言，因为没法确定一个范围的key在哪个层次中有数据，只能扫描所有的层次做合并之后才能返回最终的结果。X-Engine采用了一系列的手段，例如SuRF（SIGMOD'18 best paper）提供range scan filter减少扫描层数、异步I/O与预取。

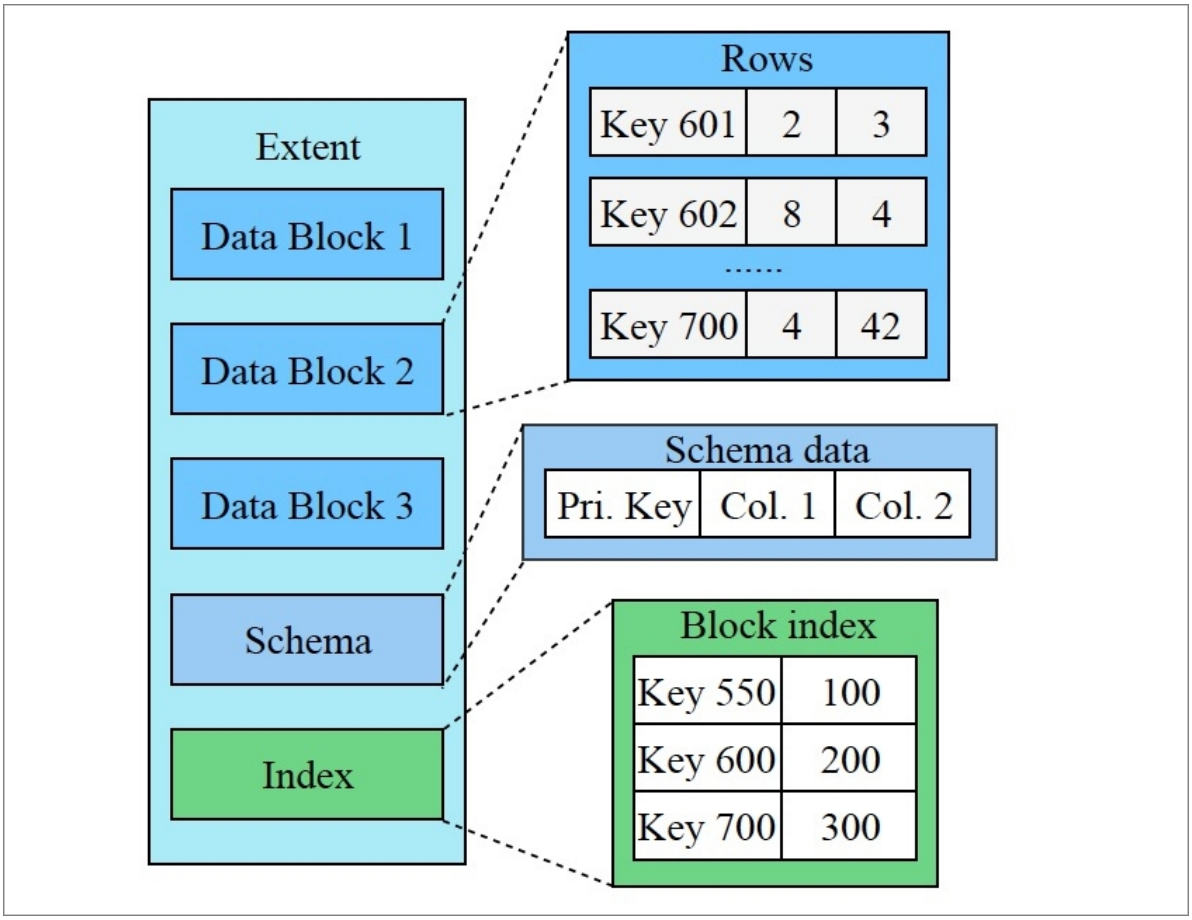


读操作中最核心的是缓存设计，Row Cache负责单行查询，Block Cache负责Row Cache的漏网之鱼，也用来进行范围扫描。由于LSM的Compaction操作会一次更新大量的Data Block，导致Block Cache中大量数据短时间内失效，导致性能的急剧抖动，因此X-Engine做了很多的优化：

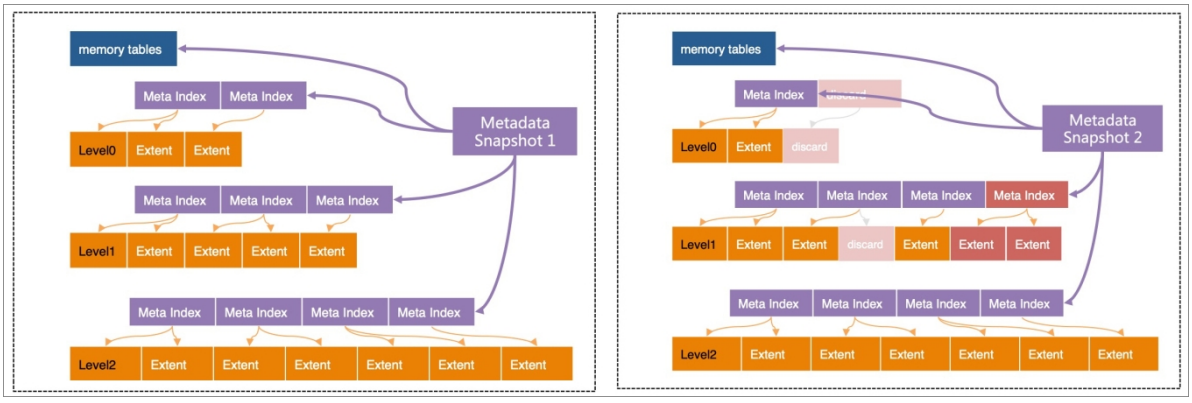
- 减少Compaction的粒度。
- 减少Compaction过程中改动的数据。
- Compaction过程中针对已有的缓存数据做定点更新。

- Compaction

  Compaction操作是比较重要的，需要把相邻层次交叉的Key Range数据读取合并，然后写到新的位置。这是为前面简单的写入操作付出的代价。X-Engine为优化这个操作重新设计了存储结构。



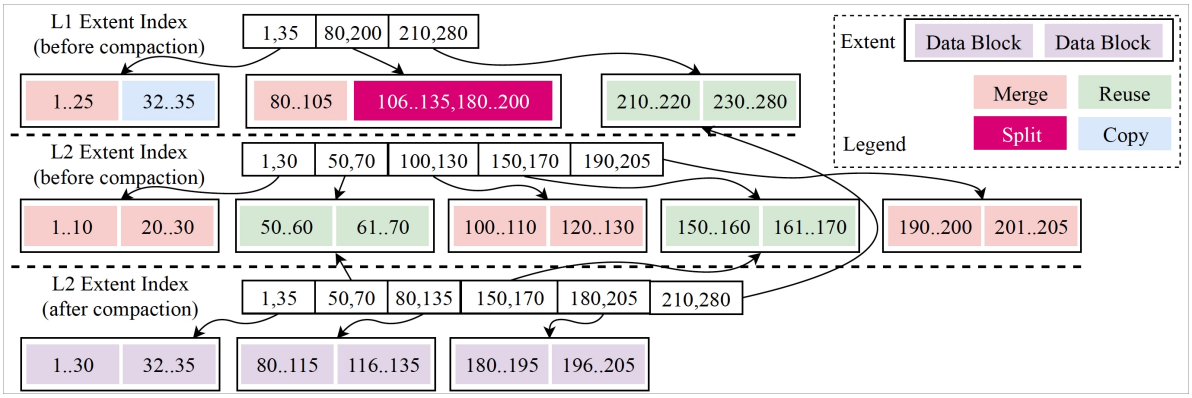
如前文所述，X-Engine将每一层的数据划分为固定大小的Extent，一个Extent相当于一个小而完整的排序字符串表（SSTable），存储了一个层次中的一个连续片段，连续片段又进一步划分为一个个连续的更小的片段Data Block，相当于传统数据库中的Page，只不过Data Block是只读而且不定长的。



回看并对比Metadata Snapshot 1和Metadata Snapshot 2，可以发现Extent的设计意图。每次修改只需要修改少部分有交叠的数据，以及涉及到的Meta Index节点。两个Metadata Snapshot结构实际上共用了大量的数据结构，这被称为数据复用技术（Data Reuse），而Extent大小正是影响数据复用率的关键，Extent作为一个完整的被复用的物理结构，需要尽可能的小，这样与其他Extent数据交叉点会变少，但又不能非常小，否则需要索引过多，管理成本太大。

X-Engine中Compaction的数据复用是非常彻底的，假设选取两个相邻层次（Level1, Level2）中的交叉的Key Range所涵盖的Extents进行合并，合并算法会逐行进行扫描，只要发现任意的物理结构（包括Data Block和Extent）与其他层中的数据没有交叠，则可以进行复用。只不过Extent的复用可以修改Meta Index，而Data Block的复用只能拷贝，即便如此也可以节省大量的CPU。

一个典型的数据复用在Compaction中的过程可以参见下图。



可以看出数据复用的过程是在逐行迭代的过程中完成的，不过这种精细的数据复用带来另一个副作用，即数据的碎片化，所以在实际操作的过程中也需要根据实际情况进行分析。

数据复用不仅给Compaction操作本身带来好处，降低操作过程中的I/O与CPU消耗，更对系统的综合性能产生一系列的影响。例如Compaction过程中数据不用完全重写，大大降低了写入时空间的增大；大部分数据保持原样，数据缓存不会因为数据更新而失效，减少合并过程中因缓存失效带来的读性能抖动。

实际上，优化Compaction的过程只是X-Engine工作的一部分，更重要的是优化Compaction调度的策略，选什么样的Extent、定义compaction任务的粒度、执行的优先级等，都会对整个系统性能产生影响，可惜并不存在什么完美的策略，X-Engine积累了一些经验，定义了很多规则，而探索更合理的调度策略是未来一个重要方向。

适用场景

请参见X-Engine最佳实践。

如何使用X-Engine

请参见X-Engine引擎使用须知。

后续发展

作为MySQL的存储引擎，持续地提升MySQL系统的兼容能力是一个重要目标，后续会根据需求的迫切程度逐步加强原本取消的一些功能，例如外键，以及对一些数据结构、索引类型的支持。

X-Engine作为存储引擎，核心的价值还在于性价比，持续提升性能降低成本，是一个长期的根本目标，X-Engine还在Compaction调度、缓存管理与优化、数据压缩、事务处理等方向上进行深层次的探索。

X-Engine不仅仅局限为一个单机的数据库存储引擎，未来还将作为自研分布式数据库PolarDB分布式版本的核心，提供企业级数据库服务。

3.2. X-Engine引擎使用须知

RDS MySQL提供阿里云自研的X-Engine存储引擎，支持事务并且可以大幅降低磁盘空间占用。

产品介绍

X-Engine是阿里云数据库产品事业部自研的联机事务处理OLTP（On-Line Transaction Processing）数据库存储引擎。作为自研数据库PolarDB的存储引擎之一，已经广泛应用在阿里集团内部诸多业务系统中，包括交易历史库、钉钉历史库等核心应用，大幅缩减了业务成本，同时也作为双十一大促的关键数据库技术，挺过了数百倍平时流量的冲击。

X-Engine是适用于大规模电子商务交易处理的优化存储引擎，X-Engine团队撰写的论文 *X-Engine: An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing*，详细讲述了X-Engine在数据库存储引擎领域所做的原创性工作，2019年被SIGMOD'19 Industrial Track接收。

与传统的InnoDB引擎不同，X-Engine使用分层存储架构（LSM-Tree）。分层存储有两个比较显著的优点：

- 需要索引的热点数据集更小，写入性能更高。
- 底层持久化的数据页是只读的，数据页采用紧凑存储格式，同时默认进行压缩，存储成本更低。

除了LSM-Tree架构自身的优势之外，X-Engine在工程实现上也进行了大量的创新，主要包含如下几个方面：

- 利用先天性的优势，持续优化写入性能，X-Engine相比同为LSM-tree架构的Rocksdb，有超过10倍的性能提升。
- 在存储层引入数据复用技术等，优化Compaction的性能，降低传统LSM-tree架构中Compaction动作对系统资源的冲击，保持系统性能平稳。
- 支持在同一实例中混合部署SSD/HDD等不同IO能力的存储设备， 利用天然分层结构的特点，结合不同存储硬件的IO读写性能，智能地进行数据的冷热分离存储，在不降低性能的前提下，降低综合成本。
- 引入多个层级Cache，同时结合Cach回填和预取机制，利用精细化访问机制和缓存技术，弥补传统LSM-tree引擎的读性能短板。

通过以上多方面的工程优化，X-Engine成为传统InnoDB引擎的一个替代选项，既支持事务，同时又能够显著的降低业务存储成本（依据数据特征，存储空间可降低至10%~50%），特别适合数据容量巨大，同时又要保证一定事务读写性能的业务。

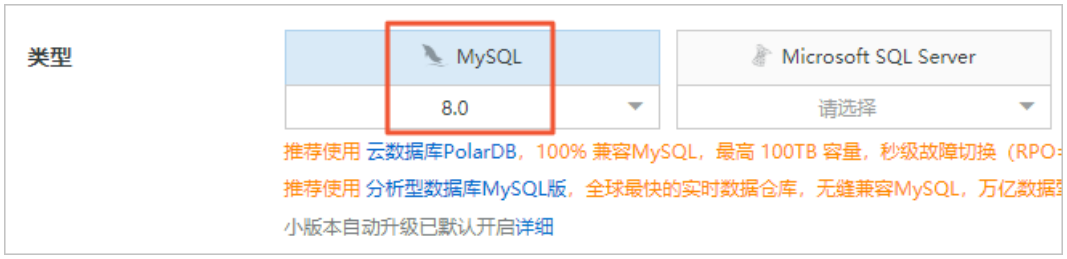
> **说明** 关于X-Engine引擎适用的业务场景请参见X-Engine最佳实践。

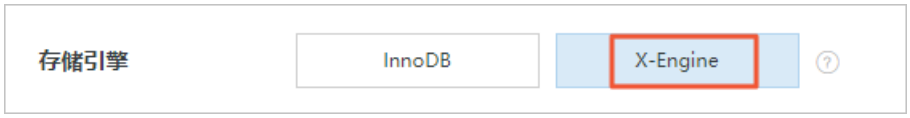
前提条件

实例为RDS MySQL 8.0高可用版或基础版。

购买RDS实例（X-Engine）

如果您需要使用X-Engine引擎，请在购买RDS实例时，**基础资源**页面选择实例类型为MySQL 8.0，然后在**实例配置**页面选择存储引擎为**X-Engine**。其他参数说明请参见创建RDS MySQL实例。





> **说明**
>
> - RDS MySQL 5.5、5.6、5.7的用户如果需要使用X-Engine引擎，请迁移至RDS MySQL 8.0版本实例。详情请参见RDS实例间的数据迁移。
> - 转换引擎请参见InnoDB/TokuDB/Myrocks引擎转换为X-Engine引擎。

创建X-Engine表

如果创建实例时设置了默认引擎为X-Engine，则建表时默认引擎就是X-Engine。您可以通过如下命令查看默认引擎：

```
show variables like '%default_storage_engine%';
```

```
mysql> show variables like '%default_storage_engine%';
+------------------------+---------+
| Variable_name          | Value   |
+------------------------+---------+
| default_storage_engine | XENGINE |
+------------------------+---------+
1 row in set (0.00 sec)
```

当默认引擎是X-Engine时，建表语句无需指定存储引擎。

```
mysql> create table tt(id int primary key, c1 int, c2 varchar(100));
Query OK, 0 rows affected (0.01 sec)

mysql> show create table tt;
+-------+-----------------------------------------------------------------------------------------
| Table | Create Table
+-------+-----------------------------------------------------------------------------------------
| tt    | CREATE TABLE `tt` (
  `id` int(11) NOT NULL,
  `c1` int(11) DEFAULT NULL,
  `c2` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=XENGINE DEFAULT CHARSET=utf8 |
+-------+-----------------------------------------------------------------------------------------
1 row in set (0.01 sec)
```

表创建成功后，后续使用方法与InnoDB一样，数据会存储在X-Engine引擎。

> **说明** 实例上仍然可以创建InnoDB引擎的表，尤其是使用DTS迁移数据时，可能会出现迁移的表引擎仍然为InnoDB。解决方案请参见引擎转换方案二。

使用限制

- 与InnoDB引擎共存时的资源分配限制

  使用X-Engine引擎时，95%的内存会提供给X-Engine引擎用做写入缓存和BlockCache以加速读写速度，留给InnoDB Buffer Pool的内存非常少，所以在X-Engine引擎的实例中尽量避免使用InnoDB引擎表存储太多数据，否则会因为缓存命中率低而导致性能大幅降低。建议使用RDS MySQL 8.0时，所有的表都使用相同的引擎（X-Engine或InnoDB），避免两种引擎混用。

- 引擎功能限制

  X-Engine在引擎功能上有一些限制，其中部分功能尚在开发中。其他未列出的部分，默认其功能特性与InnoDB引擎相同。

分类	功能	X-Engine引擎	备注
	外键	不支持	-
	临时表	不支持	-

分类 SQL功能	功能	X-Engine引擎	备注
	分区表（partition）	不支持（所有partition相关创建及增删改查操作均不支持）	-
	Generated Column	不支持	-
	Handler API	不支持	-
列属性	最大列长度 （longblob/longtext/json）	32MB	-
	GIS地理数据类型	所有GIS相关数据类型均不支持（包含geometry、point、linestring、polygon、multipoint、multilinestring、multipolygon、gemometrycollection）	-
索引	哈希索引	不支持	-
	空间索引	不支持（所有fulltext索引相关的创建，使用均不支持）	-
事务	事务隔离级别	2个隔离级别： ○ 读已提交（RC） ○ 可重复读（RR）	-
	最大事务	32MB	更大事务的支持在开发中
	Savepoint	不支持	-
	XA事务	不支持	功能开发中
锁	锁粒度	○ 支持表级别锁 ○ 支持行级别锁 ○ 不支持GAP锁	-
	Skip Locked Lock Nowait	不支持	-
	非索引列支持的字符格式	非索引列支持所有的字符集（校对规则）	-

分类	功能	X-Engine引擎	备注
字符集支持	索引列支持的字符格式	○ latin1（latin1_bin） ○ gbk（gbk_chinese_ci、gbk_bin） ○ utf8（utf8_general_ci、utf8_bin） ○ utf8mb4（utf8mb4_0900_ai_ci、utf8mb4_general_ci、utf8mb4_bin）	-
主从复制	Binlog格式	stmt/row/mixed **说明** 默认为row，采用stmt/mixed在特定并发场景可能存在数据安全性问题。	-

- 大事务功能限制

  X-Engine目前不支持大事务。当一个事务修改的行数特别多时，X-Engine会使用commit in middle功能。例如用户在一个事务中修改的行数超过10000行，X-Engine会在内部把该事务提交，并且重新开启一个事务继续服务当前用户开启的事务。但是commit in middle并不能遵循严格意义上的ACID，您在使用过程中需要注意。以下举例说明：

  - 用户开启一个事务插入大量数据，在插入的过程中，由于先提交了一部分数据，其它请求就可以访问到插入的数据。
  - 用户开启一个事务修改大量数据，回滚的时候，已经执行了commit in middle的事务无法回滚。

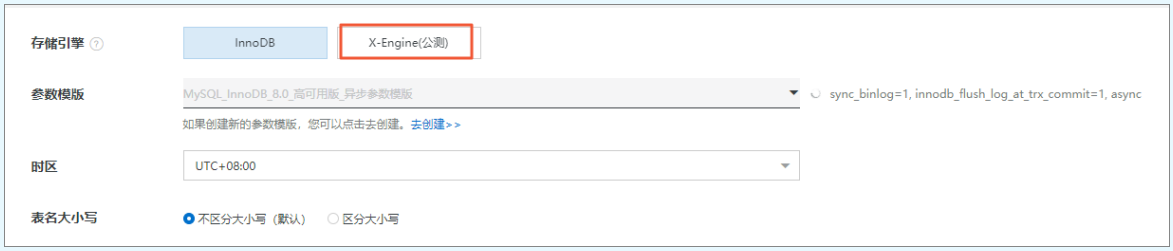
```
drop table t1;
create table t1(c1 int primary key , c2 int)ENGINE=xengine;
begin;
call insert_data(12000); //插入12000行数据，触发commit in middle，前10000行数据已经提交。
rollback;// 回滚只能把最后2000条数据回滚。
select count(*) from t1; // 这里仍然能够查询到10000条数据。
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.00 sec)
```

- 用户开启一个事务删除或者修改大量数据时，会遗漏掉本事务修改的行。

```
drop table t1;
create table t1(c1 int primary key , c2 int)ENGINE=xengine;
call insert_data(10000);
begin;
insert into t1 values(10001,10001), (10002,10002);
delete from t1 where c1 >= 0;// delete操作触发commit in middle，导致delete操作没有读到本事务插入的行。
commit;
select * from t1;
+-------+-------+
| c1    | c2    |
+-------+-------+
| 10001 | 10001 |
| 10002 | 10002 |
+-------+-------+
2 rows in set (0.00 sec)
```

参数说明

说明 在创建RDS MySQL实例时，可以选择X-Engine为默认存储引擎，也可以根据下表的参数说明调整参数模板以便适应自身业务。



类别	参数	说明	备注
性能	xengine_arena_block_size	memtable向操作系统/jemalloc的外部内存管理系统申请新内存分配的单位。	启动后只读
	xengine_batch_group_max_group_size	事务流水线最大分组数。	启动后只读
	xengine_batch_group_max_leader_wait_time_us	事务流水线最大等待时间。	启动后只读
	xengine_batch_group_slot_array_size	事务流水线最大batch大小。	启动后只读
	xengine_block_cache_size	读block缓存的大小。	不可修改
	xengine_row_cache_size	行缓存的大小。	不可修改
	xengine_write_buffer_size	单Memtable的最大大小。	不可修改

类别	参数	说明	备注
内存	xengine_block_size	磁盘上数据block大小。	初始化后只读 启动后只读
	xengine_db_write_buffer_size	所有subtable的Active Memtable的总计大小限制。	不可修改
	xengine_db_total_write_buffer_size	所有subtable的Active Memtable/Immutable memtable的总大小限制。	不可修改
	xengine_scan_add_blocks_limit	每个请求在范围扫描时，可以加到BlockCache中的Block数目。	不可修改
compaction	xengine_flush_delete_percent_trigger	当Memtable中记录数超过此数目时，则xengine_flush_delete_record_trigger参数生效。	-
锁	xengine_max_row_locks	单SQL请求中，最大可以锁定的行数。	不可修改
	xengine_lock_wait_timeout	锁等待超时时间。	不可修改

运行状态指标

下表为X-Engine的运行状态指标。

指标名	含义
xengine_rows_deleted	删除行数。
xengine_rows_inserted	写入行数。
xengine_rows_read	读取行数。
xengine_rows_updated	更新行数。
xengine_system_rows_deleted	对引擎为X-Engine的系统表的删除次数。
xengine_system_rows_inserted	对引擎为X-Engine的系统表的插入次数。
xengine_system_rows_read	对引擎为X-Engine的系统表的读取次数。
xengine_system_rows_updated	对引擎为X-Engine的系统表的更新次数。
xengine_block_cache_add	向Block Cache添加次数。
xengine_block_cache_data_hit	读数据Block命中Cache次数。
xengine_block_cache_data_miss	读数据Block时Miss次数。

指标名	含义
xengine_block_cache_filter_hit	Filter Block的命中次数。
xengine_block_cache_filter_miss	Filter Block的miss次数。
xengine_block_cache_hit	Block Cache的整体命中次数（data_hit + index_hit）。
xengine_block_cache_index_hit	索引Block命中次数。
xengine_block_cache_index_miss	索引Block miss次数。
xengine_block_cache_miss	Block Cache整体Miss次数（data_miss + index_miss）。
xengine_block_cachecompressed_miss	压缩的Block Cache Miss次数。
xengine_bytes_read	读物理磁盘的字节数。
xengine_bytes_written	加入Block Cache的字节数。
xengine_memtable_hit	Memtable命中次数。
xengine_memtable_miss	Memtable Miss次数。
xengine_number_block_not_compressed	未压缩的Block数目。
xengine_number_keys_read	Key的读取次数。
xengine_number_keys_updated	Key的更新次数。
xengine_number_keys_written	Key的写入次数。
xengine_number_superversion_acquires	Superversion引用的申请次数统计。
xengine_number_superversion_cleanups	Superversion的清理次数。当一个Superversion无人再引用时则被清理。
xengine_number_superversion_releases	Superversion的引用释放次数，当一个Superversion的引用次数为0时则被清理。
xengine_snapshot_conflict_errors	在RR隔离级别下，因为Snapshot版本冲突而报错的次数。
xengine_wal_bytes	Redo落盘字节数。
xengine_wal_group_syncs	Redo执行GroupCommit的次数。
xengine_wal_synced	Redo日志Sync的次数。
xengine_write_other	在事务流水线中，作为Follower完成提交的次数。
xengine_write_self	在事务流水线中，作为Leader完成提交的次数。

指标名	含义
xengine_write_wal	写Redo日志的次数。

3.3. InnoDB/TokuDB/Myrocks引擎转换为X-Engine引擎

RDS MySQL 8.0支持X-Engine引擎，X-Engine可以提供更好的数据压缩能力，降低磁盘空间成本。本文介绍如何将InnoDB/TokuDB/Myrocks引擎转换为X-Engine引擎。

背景信息

X-Engine是阿里云自研的联机事务处理OLTP（On-Line Transaction Processing）数据库存储引擎。作为自研数据库PolarDB的存储引擎之一，X-Engine已经广泛应用在阿里集团内部诸多业务系统中，包括交易历史库、钉钉历史库等，大幅缩减了业务成本；同时也作为双十一大促的关键数据库技术，挺过了数百倍平时流量的冲击。

更多详情请参见：

- X-Engine引擎使用须知
- X-Engine最佳实践

> **说明** 本文旨在指导您如何将存量InnoDB/TokuDB/Myrocks引擎转换为X-Engine引擎。如果是一个新业务，建议您在购买RDS MySQL 8.0实例时直接指定默认存储引擎为X-Engine，或建表语句指定engine=xengine。更多信息，请参见X-Engine引擎使用须知。

注意事项

- 如果原表为InnoDB引擎，转换前请确保实例的剩余磁盘空间是现有数据量的2倍。转换为X-Engine引擎后空间占用会减小至原数据大小的10%~50%。
- 使用方案一进行转换时，需要更新配置参数并重启实例，请在停止业务之后进行操作。
- 使用方案二进行全库迁移时，由于需要切换连接地址，请在业务低峰期进行操作。
- 在转换线上业务引擎之前，请提前进行SQL兼容性测试，确认正常后再转换线上业务引擎。
- 转换引擎后请修改实例参数**default_storage_engine**为**xengine**，这样后续创建的表的引擎即为X-Engine。

方案建议

- 实例为RDS MySQL 8.0（内核小版本20200229或以上）时，建议使用方案一，无需配置各种工具。

  > **说明** 内核小版本过低时，您可以在**基本信息**页面单击**升级内核小版本**按钮进行升级。如果没有该按钮，表示当前小版本已经是最新版。详情请参见升级内核小版本。

- 其他情况时（例如实例为MySQL 5.6/5.7），建议您使用方案二。

方案一

此方案为修改参数模板启用X-Engine引擎，然后使用DDL语句转换引擎，最简单直接，但此过程会重启实例，并且全程阻塞DML操作，大表转换时间比较久。

1. 访问RDS实例列表，在上方选择地域，然后单击目标实例ID。
2. 在左侧导航栏中单击**参数设置**。
3. 在左上角单击**应用模板**，选择**MySQL_8.0_X-Engine_高可用版_默认参数模板**，然后单击**确定**。

> **说明** 此操作会重启实例，重启后会将95%的内存资源分配给X-Engine引擎使用，请不要混用InnoDB引擎。

4. 通过DMS登录RDS数据库。
5. 在页面上方选择**SQL操作 > SQL窗口**。

6. 执行如下命令进行转换：

```
alter table <数据库名>.<表名> engine xengine;
```

示例

```
alter table test.sbtest1 engine xengine;
```

方案二

此方案为使用阿里云的数据传输服务DTS（Data Transmission Service）实时同步原表数据到新实例，然后将业务切换到新实例。

> **说明** 通过DTS迁移数据时默认会继承源实例的引擎类型。所以需要先单独导出建表语句的SQL，修改Create语句中的引擎类型为X-Engine引擎. 然后再迁移数据到新建的X-Engine表中。

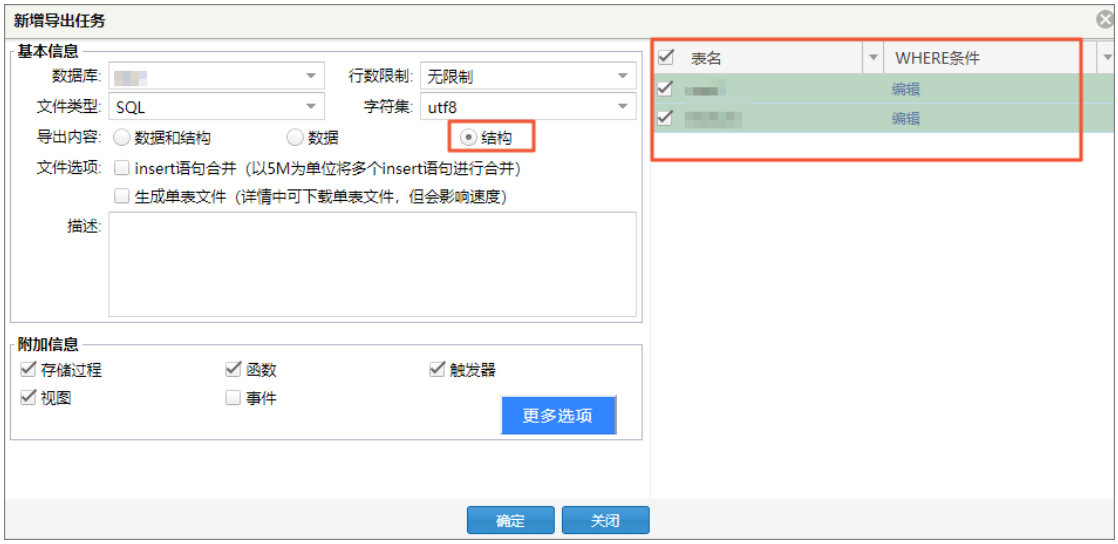
1. 执行如下步骤导出原实例的所有表结构脚本。
   i. 通过DMS登录原实例。

ii. 在页面上方选择**数据方案 > 导出**。

iii. 选择**新增导出任务 > 导出数据库**。

iv. 按下图设置导出数据库的所有表结构脚本，单击**确定**，在弹出的提示框中单击**YES**。

> **说明** 弹出提示框是因为勾选了**更多选项**中的**SQL脚本扩展选项**，忽略即可。



2. 解压缩后修改表结构脚本，将InnoDB或TokuDB修改为xengine。

```
SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS `sbtest1`;
CREATE TABLE `sbtest1` (
  `id` int(11) NOT NULL,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `sbtest10`;
CREATE TABLE `sbtest10` (
  `id` int(11) NOT NULL,
  `k` int(11) NOT NULL DEFAULT '0',
```

改为 xengine

3. 新购一个与原实例规格相同的RDS MySQL 8.0实例（购买时选择X-Engine参数模板）。

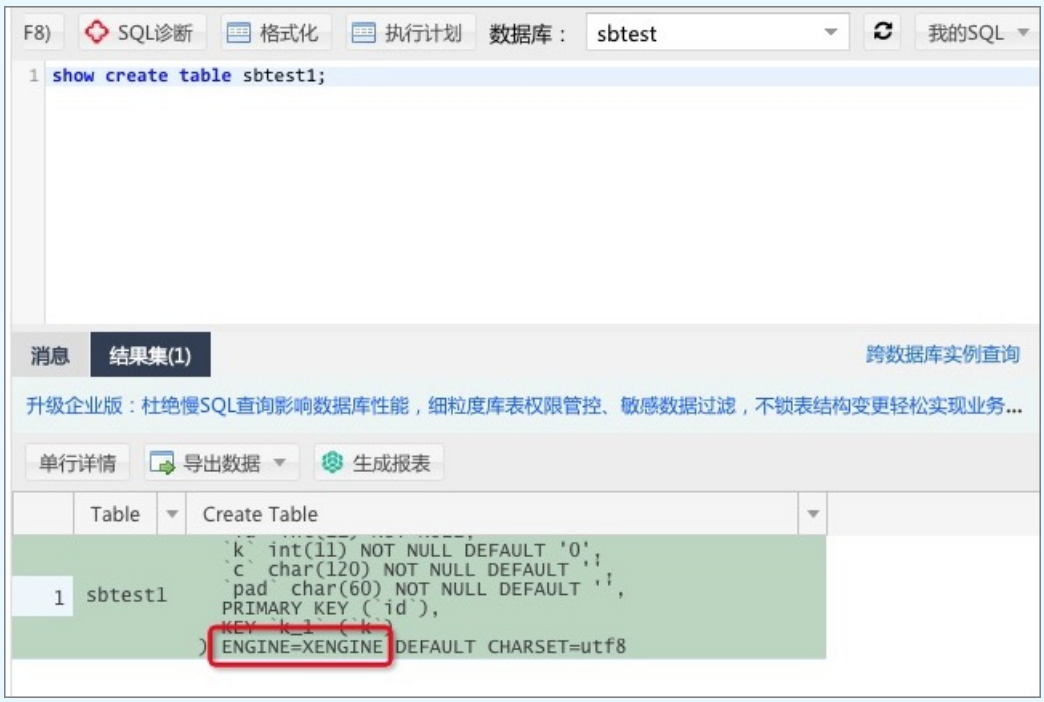
> **说明** 在创建RDS MySQL实例时，可以选择应用默认的X-Engine参数模板来设置X-Engine为默认存储引擎。

4. 执行如下步骤将表结构脚本导入新实例。
   i. 通过DMS登录新实例。
   ii. 在页面上方选择**数据方案 > 导入**。

   iii. 单击**新增任务**。

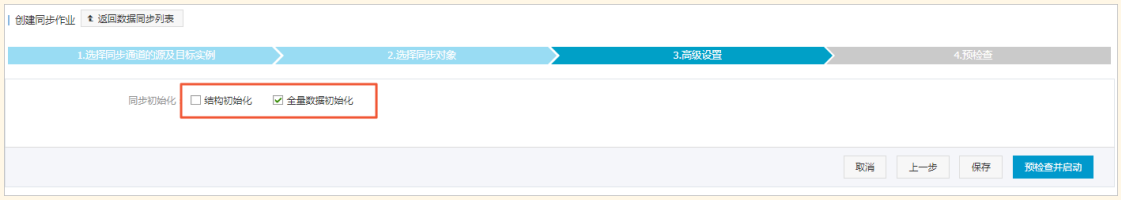
iv. 按下图设置导入表结构脚本，单击**开始**。

说明 导入成功后可以通过 `show create table <表名>;` 命令确认表引擎为X-Engine。



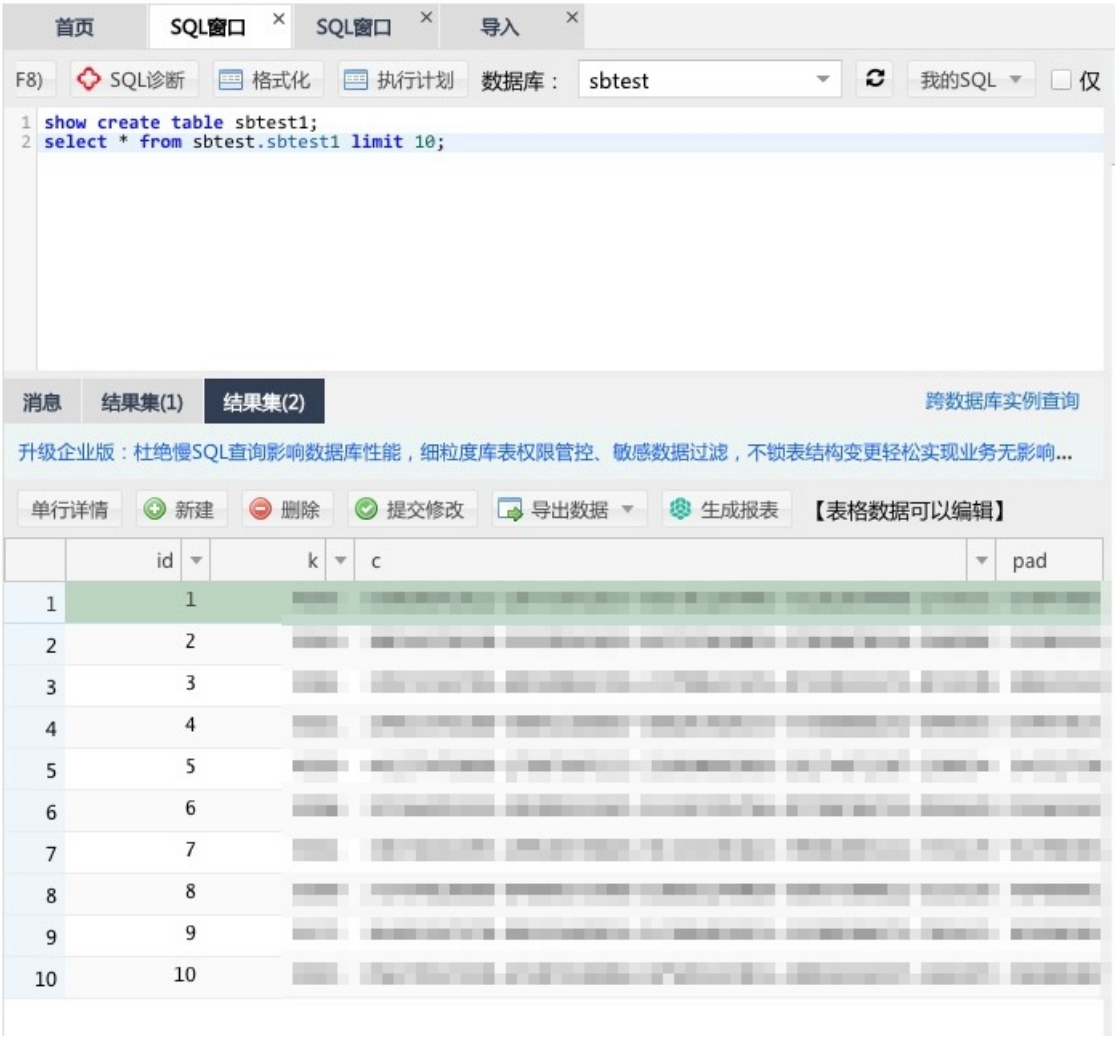
5. 参考RDS MySQL实例间的双向同步将原实例的数据同步至新实例。

警告 在同步初始化时，请勿勾选**结构初始化**。



执行结果

同步完成后，您可以查询数据是否同步成功，然后进行SQL兼容性测试，测试正常再转换线上业务引擎。



3.4. X-Engine性价比优势

X-Engine的性能与InnoDB相似，但是存储成本远低于InnoDB，因此拥有极高的性价比。

背景信息

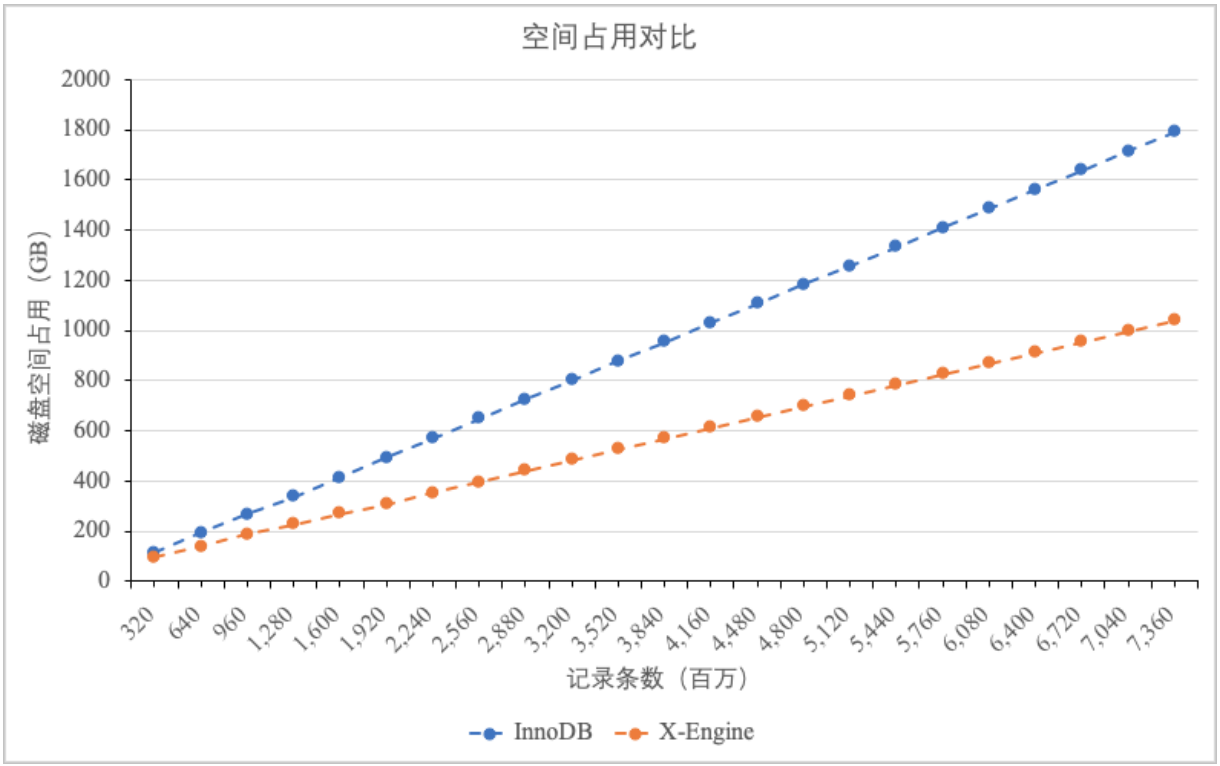
RDS MySQL提供阿里云自研存储引擎X-Engine，相比InnoDB，X-Engine在磁盘空间占用和数据库整体成本上有比较明显的优势，X-Engine使用了层次化的存储结构，并结合ZSTD压缩算法实现了更高的数据压缩率。下文将详细对比X-Engine、InnoDB、TokuDB的存储开销和性能。

> **说明** X-Engine的主要技术创新已经发表在国际数据库领域学术会议ACM SIGMOD 2019、VLDB2020和存储领域学术会议USENIX FAST 2020。

测试环境

用于测试的RDS实例规格为rds.mysql.s3.large（4核CPU、8 GB内存），存储空间为2 TB。

X-Engine存储成本约为InnoDB一半



上图为分别使用InnoDB和X-Engine存储引擎时的磁盘空间使用情况。

两种存储引擎均使用默认配置，使用SysBench的默认表结构，每张表包含1千万条记录，表总数从32张逐渐增长到736张。实测数据显示，随着数据量的逐渐增长，X-Engine的空间占用的增长更慢，节省的空间越多，最多时仅为InnoDB的58%。对于单条记录长度更长的场景，X-Engine有更大的存储空间优势。例如淘宝图片空间库从InnoDB迁移到X-Engine后，存储空间仅为InnoDB的14%。

由于绝大部分的InnoDB业务场景中未使用数据压缩，如果开启压缩，InnoDB的存储空间会压缩为之前的67%左右，且查询性能会大幅下降，严重影响业务，以主键更新为例，其吞吐性能仅为压缩前的10%。相比开启压缩后性能过低的InnoDB，X-Engine是一个兼顾存储成本和性能的高性价比存储引擎。

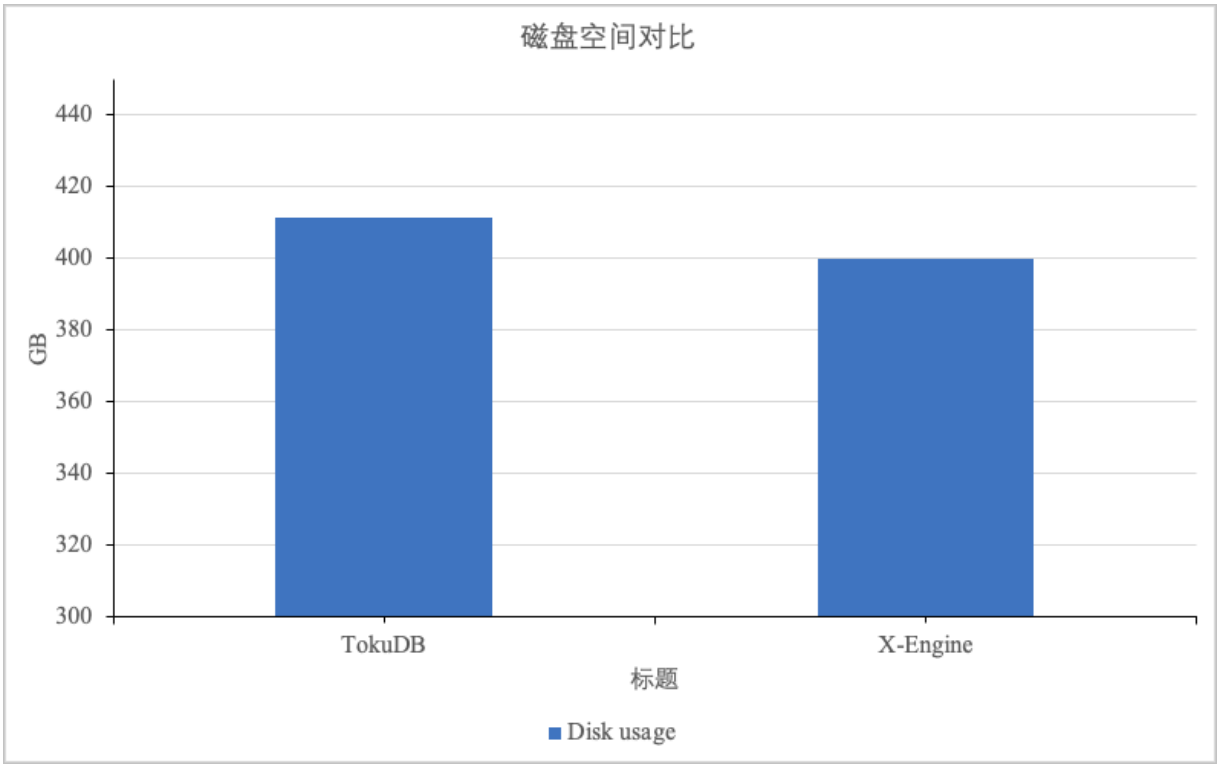
Sysbench测试命令：

```
#InnoDB prepare数据库
sysbench /usr/share/sysbench/oltp_update_index.lua\
    --mysql-host=[RDS实例连接串]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --mysql-db=sbtest\
    --threads=32\
    --tables=[32-736]\
    --table_size=10000000\
    --mysql-storage_engine=INNODB\
    prepare
#X-Engine prepare数据库
sysbench /usr/share/sysbench/oltp_update_index.lua\
    --mysql-host=[RDS实例连接串]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --mysql-db=sbtest\
    --threads=32\
    --tables=[32-736]\
    --table_size=10000000\
    --mysql-storage_engine=XENGINE\
    prepare
```

X-Engine存储开销比TokuDB更低

TokuDB曾经也是提供低存储开销的数据库引擎，但其开发者Percona已经停止TokuDB的维护，而且X-Engine与TokuDB相比拥有更低的存储开销，所以阿里云建议将TokuDB引擎的数据库迁移至X-Engine引擎。

TokuDB采用分形树（Fractal Tree），较InnoDB使用的B+-tree而言拥有更多充满数据的叶子节点及相应的数据块，能够实现比InnoDB更高的压缩率。但TokuDB没有X-Engine的分层存储设计，而X-Engine同样拥有充满记录的数据块这一优势，结合其它空间优化，X-Engine实现了比TokuDB更低的存储开销。



上图为分别使用TokuDB和X-Engine存储引擎时的磁盘空间使用情况。

实例中创建32张表，每张表1亿条记录，最终TokuDB和X-Engine分别占用411 GB和400 GB磁盘空间，由此可见X-Engine可以替代TokuDB，满足您的低成本存储需求。

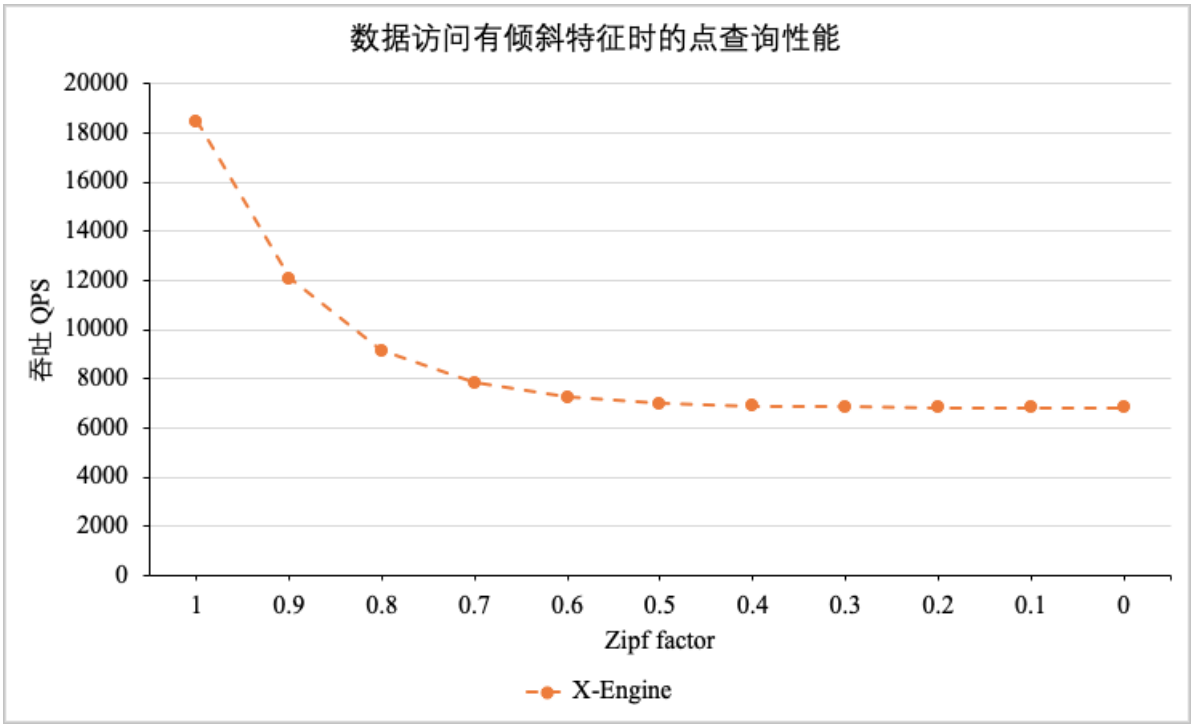
Sysbench测试命令：

```
#TokuDB prepare数据库
sysbench /usr/share/sysbench/oltp_update_index.lua\
    --mysql-host=[RDS实例连接串]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --mysql-db=sbtest\
    --threads=32\
    --tables=[32-736]\
    --table_size=1000000000\
    --mysql-storage_engine=TokuDB\
    prepare
#X-Engine prepare数据库
sysbench /usr/share/sysbench/oltp_update_index.lua\
    --mysql-host=[RDS实例连接串]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --mysql-db=sbtest\
    --threads=32\
    --tables=[32-736]\
    --table_size=1000000000\
    --mysql-storage_engine=XENGINE\
    prepare
```

X-Engine分层存储和分层访问提高QPS

X-Engine可以保证在不影响热数据查询性能的情况下，降低冷数据占用的空间，以实现降低总存储成本。主要原因如下：

- 采用层次化的存储结构，将热数据与冷数据分别存放在不同的层次中，并默认对冷数据所在层次进行压缩。
- 对每一条记录都使用了前缀编码等减少存储开销的技术。
- 采用分层访问，结合真实业务场景中广泛存在的局部性和数据访问倾斜现象（热数据量往往远小于冷数据量），提高QPS。



上图为X-Engine处理有倾斜特征的数据访问时点查询的性能情况。

这项测试使用了业界常用的齐夫分布来控制数据访问的倾斜程度，当倾斜程度（Zipf factor）较高时，更多的点查询会命中缓存中的热数据，而不是磁盘中的冷数据，所以访问延迟更低，整体QPS性能更高，此时压缩冷数据对QPS的影响很小。

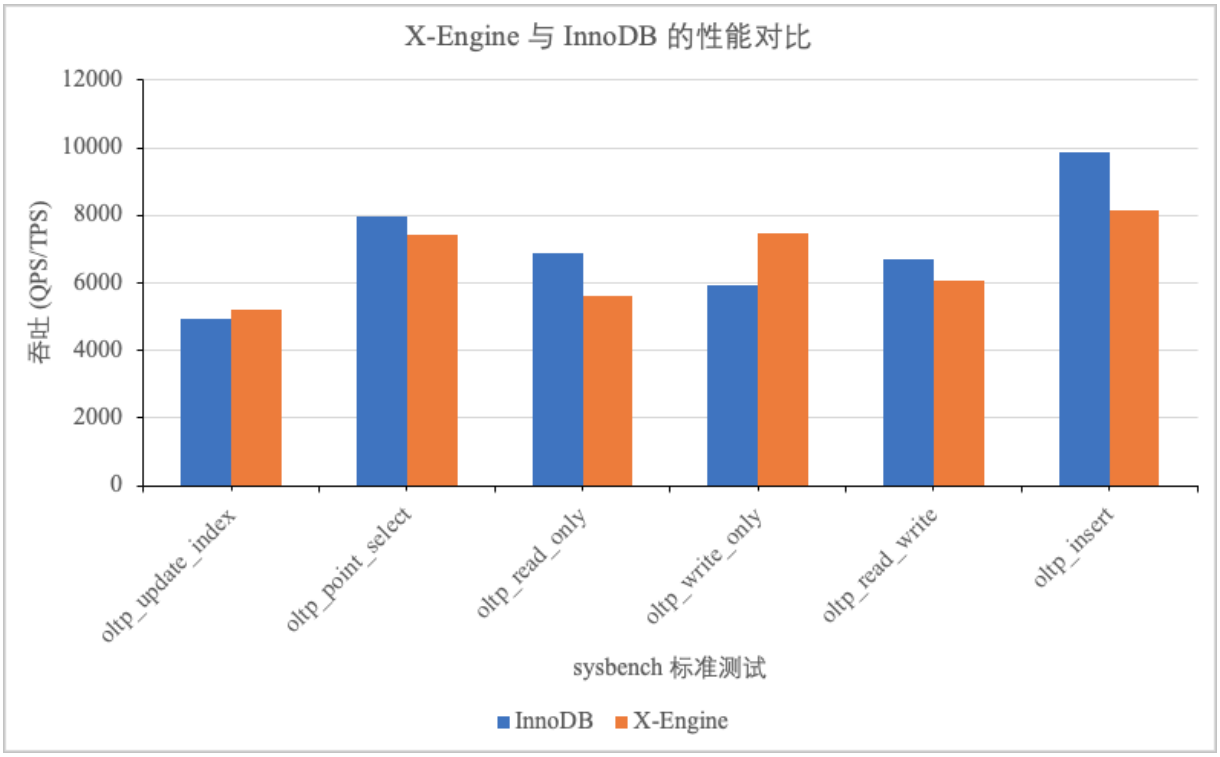
简而言之，X-Engine分层存储、分层访问的方式使得业务中绝大部分访问热数据的SQL可以不受冷数据的影响，QPS比均匀访问所有数据高2.7倍。

Sysbench测试命令：

```
sysbench /usr/share/sysbench/oltp_point_select.lua\
    --mysql-host=[RDS实例连接串]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --time=3600\
    --mysql-db=sbtest\
    --tables=32\
    --threads=512\
    --table_size=10000000\
    --rand-type=zipfian\
    --rand-zipfian-exp=[0-1]\
    --report-interval=1\
    run
```

X-Engine查询存量数据性能与InnoDB基本持平

如果将大量存量数据（尤其是归档和历史类数据）存入X-Engine，查询存量数据时X-Engine的性能（QPS或TPS）整体略低于InnoDB。



上图为各种场景分别使用InnoDB和X-Engine存储引擎时的性能对比，通过对比可以发现X-Engine与InnoDB性能相近。

在大多数的OLTP事务型负载中，更新（oltp_update_index、oltp_write_only）和点查（oltp_point_select）的执行频率较高，X-Engine在这两项上的性能与InnoDB基本持平。

由于X-Engine的分层存储特性，X-Engine在执行范围扫描时或检查一条记录是否唯一时，需要扫描或访问多个层次，所以X-Engine的范围查询（oltp_read_only）和新记录插入（oltp_insert）性能比InnoDB略差。

在混合场景（oltp_read_write），X-Engine与InnoDB性能基本持平。

Sysbench测试命令：

```
#以oltp_read_only为例
sysbench /usr/share/sysbench/oltp_read_only.lua\
    --mysql-host=[RDS实例连接串]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --mysql-db=sbtest\
    --time=3600\
    --tables=32\
    --threads=512\
    --table_size=10000000\
    --rand-type=uniform\
    --report-interval=1\
    run
```

总结

X-Engine是一款性能与InnoDB相似，但是存储成本非常低的RDS MySQL存储引擎产品。目前，X-Engine已经服务于阿里云上承载的钉钉历史库、淘宝图片信息库、淘宝交易历史库等多项阿里集团核心业务，不仅降低了存储成本，还能保证性能满足业务需求。关于X-Engine的详细介绍请参见X-Engine简介。

使用X-Engine

- 如果您是新用户，想使用X-Engine，请在购买RDS MySQL实例时选择X-Engine作为默认引擎，详情请参见创建RDS MySQL实例。
- 如果您已经在使用RDS MySQL实例，想转换为X-Engine，请参见InnoDB/TokuDB/Myrocks引擎转换为X-Engine引擎。

4.功能

4.1. Native Flashback

Native Flashback功能可以通过SQL语句查询或恢复指定时间点的数据，保证在误操作后可以快速获取历史数据。

功能说明

数据库运维过程中的误操作可能会给业务带来严重的影响，常见的恢复手段Binlog Flashback操作较为复杂、容易出错且耗时较长，而通过备份集恢复则需要额外的系统资源，在数据量较大时恢复时间不可控。

AliSQL在InnoDB引擎上设计和实现了Native Flashback功能，无需复杂的恢复操作，通过简单的SQL语句即可查询或恢复误操作前的历史数据，节省了大量宝贵的时间，保证业务平稳运行。

前提条件

- 实例为RDS MySQL 8.0基础版或高可用版。
- 内核小版本为20210930及以上。如何查看或升级内核小版本，请参见升级内核小版本。

注意事项

- 仅支持使用InnoDB引擎的表。
- 需要消耗额外的undo表空间，可以通过INNODB_UNDO_SPACE_SUPREMUM_SIZE参数进行配置。
- Native Flashback的查询结果取最接近指定时间点的数据，不保证查询到的数据与指定时间点的数据完全匹配。
- 暂不支持跨DDL操作的历史版本数据查询和恢复。例如您无法通过Native Flashback查询某个已经被删除的表的内容。

语法

Native Flashback提供了全新的 `AS OF` 语法，通过该语法指定需要回滚的时间。语法规则如下：

```
SELECT ... FROM <表名>
  AS OF TIMESTAMP <表达式>;
```

其中，在表达式中指定回滚的时间，该表达式支持多种形式，示例：

```
SELECT ... FROM tablename
  AS OF TIMESTAMP '2020-11-11 00:00:00';
SELECT ... FROM tablename
  AS OF TIMESTAMP now();
SELECT ... FROM tablename
  AS OF TIMESTAMP (SELECT now());
SELECT ... FROM tablename
  AS OF TIMESTAMP DATE_SUB(now(), INTERVAL 1 minute);
```

参数说明

Native Flashback功能开放了如下可配置参数：

参数名称	说明
INNODB_RDS_FLASHBACK_TASK_ENABLED	• 描述：Native Flashback的功能开关。 • 命令行格式：`--innodb-rds-flashback-task-enabled=#` 。 • 参数范围：全局参数。 • 数据类型：Boolean。 • 默认值：OFF。 • 取值范围：[ON|OFF]。
INNODB_UNDO_RETENTION	• 描述：undo记录的保留时长，超出该时长的undo记录无法查询，单位：秒。 **说明** 该参数的值越大，Native Flashback支持的回档查询时间越长，同时undo表空间占用的存储空间也会上升。 • 命令行格式：`--innodb-undo-retention=#` 。 • 参数范围：全局参数。 • 数据类型：Integer。 • 默认值：0。 • 取值范围：0~4294967295。
INNODB_UNDO_SPACE_SUPREMUM_SIZE	• 描述：undo表空间可占用的最大磁盘空间，单位：MB。占用空间超过这个值时，忽略 `INNODB_UNDO_RETENTION` 参数强制清理undo记录。 • 命令行格式：`--innodb-undo-space-supremum-size=#` 。 • 参数范围：全局参数。 • 数据类型：Integer。 • 默认值：102400。 • 取值范围：0~4294967295。
INNODB_UNDO_SPACE_RESERVED_SIZE	• 描述：预留的undo表空间大小，单位：MB。在 `INNODB_UNDO_RETENTION` 参数非0的情况下，将使用这部分空间尽可能多地保留undo记录。 • 命令行格式：`--innodb-undo-space-reserved-size=#` 。 • 参数范围：全局参数。 • 数据类型：Integer。 • 默认值：0。 • 取值范围：0~4294967295。

使用示例

```
# 获取时间点
MySQL [mytest]> select now();
+---------------------+
| now()               |
+---------------------+
| 2020-10-14 15:44:09 |
```

```
+---------------------+
1 row in set (0.00 sec)
# 查看数据
MySQL [mytest]> select * from mt1;
+----+------+
| id | c1   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
|  5 |    5 |
+----+------+
5 rows in set (0.00 sec)
# 不带WHERE条件的更新操作
MySQL [mytest]> update mt1 set c1 = 100;
Query OK, 5 rows affected (0.00 sec)
Rows matched: 5  Changed: 5  Warnings: 0
MySQL [mytest]> select * from mt1;
+----+------+
| id | c1   |
+----+------+
|  1 |  100 |
|  2 |  100 |
|  3 |  100 |
|  4 |  100 |
|  5 |  100 |
+----+------+
5 rows in set (0.00 sec)
# 查询历史时间点的数据，成功返回结果
MySQL [mytest]> select * from mt1 AS OF timestamp '2020-10-14 15:44:09';
+----+------+
| id | c1   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
|  5 |    5 |
+----+------+
5 rows in set (0.00 sec)
# 如果超出保留的历史数据范围，返回失败
MySQL [mytest]> select * from mt1 AS OF timestamp '2020-10-13 14:44:09';
ERROR 7545 (HY000): The snapshot to find is out of range
# 开始恢复数据
MySQL [mytest]> create table mt1_tmp like mt1; # 创建一个与原表结构相同的临时表
Query OK, 0 rows affected (0.03 sec)
MySQL [mytest]> insert into mt1_tmp
             -> select * from mt1 AS OF
             -> TIMESTAMP '2020-10-14 15:44:09'; # 将原表中的历史数据写入临时表
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
MySQL [mytest]> select * from mt1_tmp; # 确认临时表中的数据是否正确
+----+------+
```

```
| id | c1   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
|  5 |    5 |
+----+------+
5 rows in set (0.00 sec)
MySQL [mytest]> rename table mt1 to mt1_bak,
             -> mt1_tmp to mt1; # （进行本操作需要先停止业务读写）更改原表表名为mt1_bak，并将临时表名改成原表表名，完成数据恢复
Query OK, 0 rows affected (0.02 sec)
MySQL [mytest]> select * from mt1; # 确认恢复完成后的数据
+----+------+
| id | c1   |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
|  5 |    5 |
+----+------+
5 rows in set (0.01 sec)
```

4.2. Thread Pool

为了发挥出RDS的最佳性能，阿里云提供线程池（Thread Pool）功能，将线程和会话分离，在拥有大量会话的同时，只需要少量线程完成活跃会话的任务即可。

优势

MySQL默认的线程使用模式是会话独占模式，每个会话都会创建一个独占的线程。当有大量的会话存在时，会导致大量的资源竞争，大量的系统线程调度和缓存失效也会导致性能急剧下降。

阿里云RDS的线程池实现了不同类型SQL操作的优先级及并发控制机制，将连接数始终控制在最佳连接数附近，使RDS数据库在高连接大并发情况下始终保持高性能。线程池的优势如下：

- 当大量线程并发工作时，线程池会自动调节并发的线程数量在合理的范围内，从而避免线程调度工作过多和大量缓存失效。
- 大量的事务并发执行时，线程池会将语句和事务分为不同的优先级，分别控制语句和事务的并发数量，从而减少资源竞争。
- 线程池给予管理类的SQL语句更高的优先级，保证这些语句优先执行。这样在系统负载很高时，新建连接、管理、监控等操作也能够稳定执行。
- 线程池给予复杂查询SQL语句相对较低的优先级，并且有最大并发数的限制。这样可以避免过多的复杂SQL语句将系统资源耗尽，导致整个数据库服务不可用。

前提条件

实例版本为RDS MySQL 5.6/5.7/8.0。

使用Thread Pool

Thread Pool设计了如下三个参数，您可以在控制台进行修改。详情请参见设置实例参数。

参数	说明
loose_thread_pool_enabled	是否开启线程池功能。取值： • ON • OFF 默认值：ON。 说明 • 开启/关闭线程池功能使用本参数即可，不再使用参数thread_handling进行控制。 • 开启/关闭线程池功能无需重启实例。
loose_thread_pool_size	分组的数量，默认值：4。线程池中的线程被平均分到多个组中进行管理。
loose_thread_pool_oversubscribe	每个组中允许的活跃线程的数量，默认值：32。活跃线程是指正在执行SQL语句的线程，但是不包括以下两种情形： • SQL语句在等待磁盘IO； • SQL语句在等待事务提交。

查询Thread Pool状态

您可以通过如下命令查询Thread Pool状态：

```
show status like "thread_pool%";
```

示例：

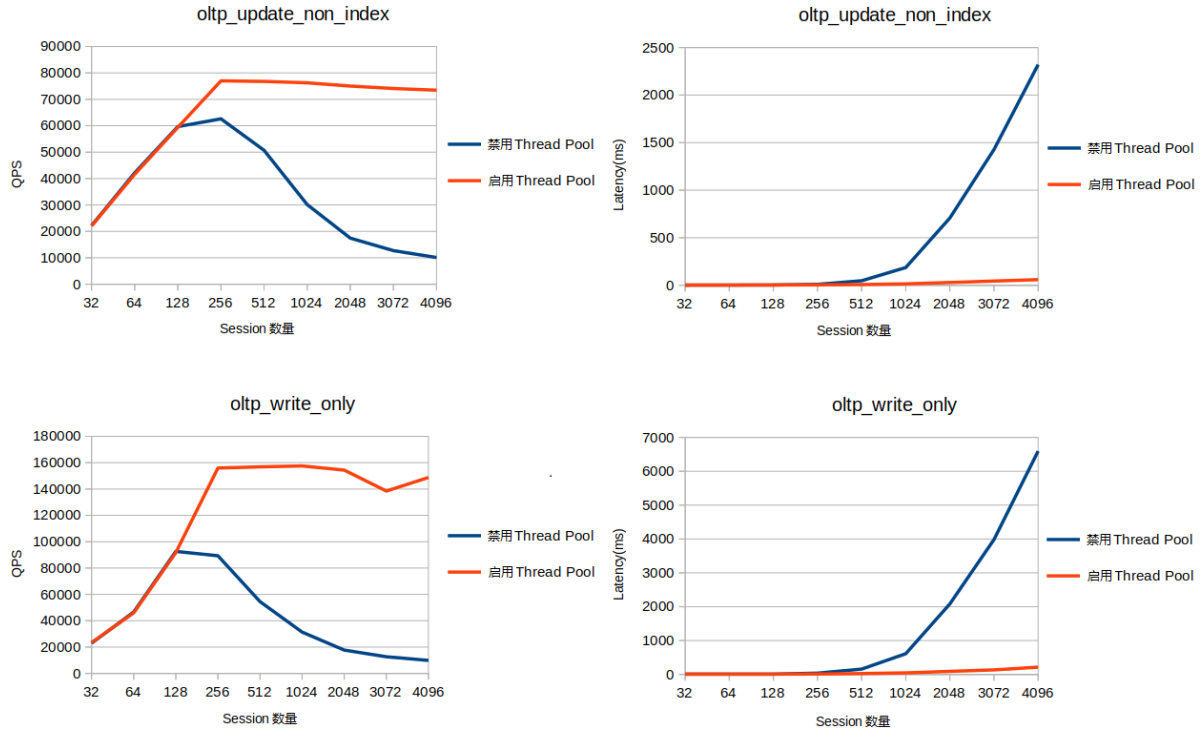
```
mysql> show status like "thread_pool%";
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| thread_pool_active_threads | 1     |
| thread_pool_big_threads    | 0     |
| thread_pool_dml_threads    | 0     |
| thread_pool_idle_threads   | 19    |
| thread_pool_qry_threads    | 0     |
| thread_pool_total_threads  | 20    |
| thread_pool_trx_threads    | 0     |
| thread_pool_wait_threads   | 0     |
+----------------------------+-------+
8 rows in set (0.00 sec)
```

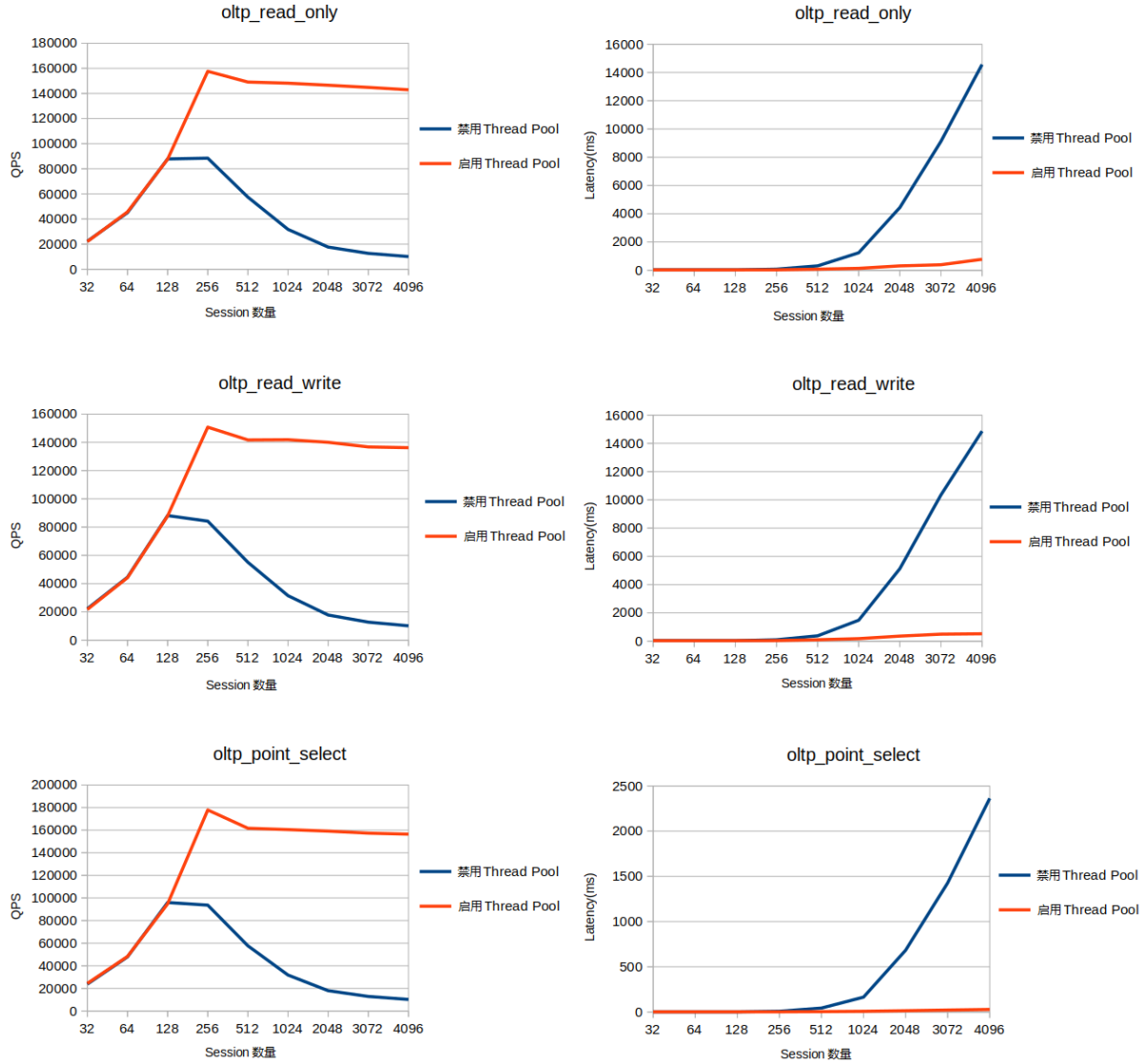
参数说明如下。

参数	说明
thread_pool_active_threads	线程池中的活跃线程数。
thread_pool_big_threads	线程池中正在执行复杂查询的线程数。复杂查询包括有子查询、聚合函数、group by、limit等的查询语句。
thread_pool_dml_threads	线程池中的在执行DML的线程数。
thread_pool_idle_threads	线程池中的空闲线程数。
thread_pool_qry_threads	线程池中正在执行简单查询的线程数。
thread_pool_total_threads	线程池中的总线程数。
thread_pool_trx_threads	线程池中正在执行事务的线程数。
thread_pool_wait_threads	线程池中正在等待磁盘IO、事务提交的线程数。

Sysbench测试

如下是开启线程池和不开启线程池的性能对比。从测试结果可以看出线程池在高并发的情况下有着明显的性能优势。





4.3. Statement Outline

生产环境中，SQL语句的执行计划经常会发生改变，导致数据库不稳定。阿里云利用Optimizer Hint和Index Hint让MySQL稳定执行计划，该方法称为Statement Outline，并提供了工具包（DBMS_OUTLN）便于您快捷使用。

前提条件

RDS实例版本如下：

- MySQL 8.0
- MySQL 5.7

功能设计

Statement Outline支持官方MySQL 8.0、MySQL 5.7的所有hint类型，分为如下两类：

- Optimizer Hint

根据作用域和hint对象，分为Global level hint、Table/Index level hint、Join order hint等。详情请参见MySQL官网。

- Index Hint

  根据Index Hint的类型和范围进行分类。详情请参见MySQL官网

Statement Outline表介绍

AliSQL内置了一个系统表（outline）保存hint，系统启动时会自动创建该表，无需您手动创建。这里提供表的创建语句供您参考：

```
CREATE TABLE `mysql`.`outline` (
  `Id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Schema_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Digest` varchar(64) COLLATE utf8_bin NOT NULL,
  `Digest_text` longtext COLLATE utf8_bin,
  `Type` enum('IGNORE INDEX','USE INDEX','FORCE INDEX','OPTIMIZER') CHARACTER SET utf8 COLL
ATE utf8_general_ci NOT NULL,
  `Scope` enum('','FOR JOIN','FOR ORDER BY','FOR GROUP BY') CHARACTER SET utf8 COLLATE utf8
_general_ci DEFAULT '',
  `State` enum('N','Y') CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL DEFAULT 'Y',
  `Position` bigint(20) NOT NULL,
  `Hint` text COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`Id`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_bin STATS_PERSISTENT=0 COMMENT='Statement outline'
```

参数说明如下。

参数	说明
Id	Outline ID。
Schema_name	数据库名。
Digest	**Digest_text**进行hash计算得到的64字节的hash字符串。
Digest_text	SQL语句的特征。
Type	• Optimizer Hint中，hint类型的取值为OPTIMIZER。 • Index Hint中，hint类型的取值为USE INDEX、FORCE INDEX或IGNORE INDEX。
Scope	仅Index Hint需要提供，分为如下三类： • FOR GROUP BY • FOR ORDER BY • FOR JOIN 空串表示所有类型的Index Hint。
State	本规则是否启用。

参数	说明
Position	• Optimizer Hint中，Position表示Query Block， 因为所有的Optimizer Hint必须作用到 Query Block上，所以，Position从1开始，hint作用在语句的第几个关键字上，Position就是几。 • Index Hint中，Position表示表的位置， 也是从1开始，hint作用在第几个表上，Position就是几。
Hint	• Optimizer Hint中，Hint表示完整的hint字符串，例如 `/*+ MAX_EXECUTION_TIME(1000) */` 。 • Index Hint中，Hint表示索引名字的列表， 例如 `ind_1,ind_2` 。

管理Statement Outline

为了便捷地管理Statement Outline，AliSQL在DBMS_OUTLN中定义了六个本地存储规则。详细说明如下：

- add_optimizer_outline

  增加Optimizer Hint。命令如下：

  ```
  dbms_outln.add_optimizer_outline('<Schema_name>','<Digest>','<query_block>','<hint>','<query>');
  ```

  > **说明** Digest和Query（原始SQL语句）可以任选其一。如果填写Query，DBMS_OUTLN会计算Digest和Digest_text。

  示例：

  ```
  mysql> call dbms_outln.add_optimizer_outline("outline_db", '', 1, '/*+ MAX_EXECUTION_TIME(1000) */',
                                        "select * from t1 where id = 1");
  ```

- add_index_outline

  增加Index Hint。命令如下：

  ```
  dbms_outln.add_index_outline('<Schema_name>','<Digest>',<Position>,'<Type>','<Hint>','<Scope>','<Query>');
  ```

  > **说明** Digest和Query（原始SQL语句）可以任选其一。如果填写Query，DBMS_OUTLN会计算Digest和Digest_text。

  示例：

  ```
  mysql> call dbms_outln.add_index_outline('outline_db', '', 1, 'USE INDEX', 'ind_1', '',
                                    "select * from t1 where t1.col1 =1 and t1.col2 ='xpchild'
  ");
  ```

- preview_outline

  查看匹配Statement Outline的情况，可用于手动验证。命令如下：

```
dbms_outln.preview_outline('<Schema_name>','<Query>');
```

示例：

```
mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 a
nd t1.col2 ='xpchild'");
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| SCHEMA     | DIGEST                                                               | BLOCK_T
YPE | BLOCK_NAME | BLOCK | HINT                |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE
| t1         |     1 | USE INDEX (`ind_1`) |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
1 row in set (0.00 sec)
```

- show_outline

  展示Statement Outline在内存中命中的情况。命令如下：

  ```
  dbms_outln.show_outline();
  ```

  示例：

```
mysql> call dbms_outln.show_outline();
+------+------------+-------------------------------------------------------------------+-
----------+-------+------+-------------------------------------------------------------+------+
----------+-----------------------------------------------------------------------------------
-------+
| ID   | SCHEMA     | DIGEST                                                                |
TYPE      | SCOPE | POS  | HINT                                                        | HIT  |
OVERFLOW | DIGEST_TEXT
|
+------+------------+-------------------------------------------------------------------+-
----------+-------+------+-------------------------------------------------------------+------+
----------+-----------------------------------------------------------------------------------
-------+
|   33 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 |
OPTIMIZER |       |    1 | /*+ SET_VAR(foreign_key_checks=OFF) */                      |    1 |
0 | SELECT * FROM `t1` WHERE `id` = ?                                                     |
|   32 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 |
OPTIMIZER |       |    1 | /*+ MAX_EXECUTION_TIME(1000) */                             |    2 |
0 | SELECT * FROM `t1` WHERE `id` = ?                                                     |
|   34 | outline_db | d4dcef634a4a664518e5fb8a21c6ce9b79fccb44b773e86431eb67840975b649 |
OPTIMIZER |       |    1 | /*+ BNL(t1,t2) */                                           |    1 |
0 | SELECT `t1` . `id` , `t2` . `id` FROM `t1` , `t2`                                     |
|   35 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f |
OPTIMIZER |       |    2 |  /*+ QB_NAME(subq1) */                                      |    2 |
0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )                 |
|   36 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f |
OPTIMIZER |       |    1 | /*+ SEMIJOIN(@subq1 MATERIALIZATION, DUPSWEEDOUT) */  |    2 |
0 | SELECT * FROM `t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )                 |
|   30 | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c |
USE INDEX |       |    1 | ind_1                                                       |    3 |
0 | SELECT * FROM `t1` WHERE `t1` . `col1` = ? AND `t1` . `col2` = ?                      |
|   31 | outline_db | 33c71541754093f78a1f2108795cfb45f8b15ec5d6bff76884f4461fb7f33419 |
USE INDEX |       |    2 | ind_2                                                       |    1 |
0 | SELECT * FROM `t1` , `t2` WHERE `t1` . `col1` = `t2` . `col1` AND `t2` . `col2` = ? |
+------+------------+-------------------------------------------------------------------+-
----------+-------+------+-------------------------------------------------------------+------+
----------+-----------------------------------------------------------------------------------
-------+
7 rows in set (0.00 sec)
```

关于HIT和OVERFLOW的说明如下。

参数	说明
HIT	此Statement Outline命中的次数。
OVERFLOW	此Statement Outline没有找到Query block或相应的表的次数。

- del_outline

  删除内存和表中的某一条Statement Outline。命令如下：

```
dbms_outln.del_outline(<Id>);
```

示例：

```
mysql> call dbms_outln.del_outline(32);
```

> **说明** 如果删除的规则不存在，系统会报相应的警告，您可以使用 `show warnings;` 查看警告内容。

```
mysql> call dbms_outln.del_outline(1000);
Query OK, 0 rows affected, 2 warnings (0.00 sec)
mysql> show warnings;
+---------+------+-----------------------------------------------+
| Level   | Code | Message                                       |
+---------+------+-----------------------------------------------+
| Warning | 7521 | Statement outline 1000 is not found in table |
| Warning | 7521 | Statement outline 1000 is not found in cache |
+---------+------+-----------------------------------------------+
2 rows in set (0.00 sec)
```

- flush_outline

  如果您直接操作了表outline修改Statement Outline，您需要让Statement Outline重新生效。命令如下：

```
dbms_outln.flush_outline();
```

  示例：

```
mysql> update mysql.outline set Position = 1 where Id = 18;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> call dbms_outln.flush_outline();
Query OK, 0 rows affected (0.01 sec)
```

功能测试

验证Statement Outline是否有效果，有如下两种方法：

- 通过preview_outline进行预览。

```
mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 a
nd t1.col2 ='xpchild'");
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| SCHEMA     | DIGEST                                                               | BLOCK_T
YPE | BLOCK_NAME | BLOCK | HINT                |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
| outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE
| t1         |     1 | USE INDEX (`ind_1`) |
+------------+----------------------------------------------------------------------+--------
----+------------+-------+---------------------+
1 row in set (0.01 sec)
```

- 直接使用explain查看。

```
mysql> explain select * from t1 where t1.col1 =1 and t1.col2 ='xpchild';
+----+-------------+-------+------------+------+---------------+-------+---------+-------
+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key   | key_len | ref
| rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+-------+---------+-------
+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ref  | ind_1         | ind_1 | 5       | const
|    1 |   100.00 | Using where |
+----+-------------+-------+------------+------+---------------+-------+---------+-------
+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
mysql> show warnings;
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
| Note  | 1003 | /* select#1 */ select `outline_db`.`t1`.`id` AS `id`,`outline_db`.`t1`.`
col1` AS `col1`,`outline_db`.`t1`.`col2` AS `col2` from `outline_db`.`t1` USE INDEX (`ind
_1`) where ((`outline_db`.`t1`.`col1` = 1) and (`outline_db`.`t1`.`col2` = 'xpchild')) |
+-------+------+-----------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

4.4. Sequence Engine

AliSQL提供了Sequence Engine，简化获取序列值的复杂度。

Sequence Engine介绍

在持久化数据库系统中，无论是单节点中的业务主键，还是分布式系统中的全局唯一值，亦或是多系统中的幂等控制，单调递增的唯一值是常见的需求。不同的数据库系统有不同的实现方法，例如MySQL提供的AUTO_INCREMENT，Oracle、SQL Server提供的SEQUENCE。

在MySQL数据库中，如果业务希望封装唯一值，例如增加日期、用户等信息，使用AUTO_INCREMENT的方法会带来很大不便，在实际的系统设计中，也存在不同的折中方法：

- 序列值由Application或者Proxy来生成，不过弊端很明显，状态带到应用端会增加扩容和缩容的复杂度。
- 序列值由数据库通过模拟的表来生成，但需要中间件来封装和简化获取唯一值的逻辑。

AliSQL提供了Sequence Engine，通过引擎的设计方法，尽可能地兼容其他数据库的使用方法，简化获取序列值复杂度。

Sequence Engine实现了MySQL存储引擎的设计接口，但底层的数据仍然使用现有的存储引擎，例如InnoDB或者MyISAM来保存持久化数据，兼容现有的第三方工具（例如Xtrabackup），所以Sequence Engine仅仅是一个逻辑引擎。

Sequence Engine通过Sequence Handler接口访问Sequence对象，实现NEXTVAL的滚动、缓存的管理等，最后透传给底层的基表数据引擎，实现最终的数据访问。

前提条件

实例版本如下：

- RDS MySQL 8.0（内核小版本为20190816及以上）
- RDS MySQL 5.7（内核小版本为20210430及以上）
- RDS MySQL 5.6（内核小版本为20170901及以上）

> **说明** 实例系列不能是三节点企业版。

使用限制

- Sequence不支持子查询和join查询。
- 可以使用 `SHOW CREATE TABLE` 或者 `SHOW CREATE SEQUENCE` 来访问Sequence结构，但不能使用 `SHOW CREATE SEQUENCE` 访问普通表。
- 不支持建表的时候指定Sequence引擎，Sequence表只能通过创建Sequence的语法来创建。

创建Sequence

创建Sequence语句如下：

```
CREATE SEQUENCE [IF NOT EXISTS] <数据库名>.<Sequence名称>
   [START WITH <constant>]
   [MINVALUE <constant>]
   [MAXVALUE <constant>]
   [INCREMENT BY <constant>]
   [CACHE <constant> | NOCACHE]
   [CYCLE | NOCYCLE]
  ;
```

> **说明** 方括号（[]）中的内容非必填。

参数说明如下。

参数	说明
START	Sequence的起始值。
MINVALUE	Sequence的最小值。
MAXVALUE	Sequence的最大值。 **说明** 如果有参数NOCYCLE，到达最大值后会报如下错误： `ERROR HY000: Sequence 'db.seq' has been run out.`
INCREMENT BY	Sequence的步长。

参数	说明
CACHE/NOCACHE	缓存的大小，为了性能考虑，可以设置较大的缓存，但如果遇到实例重启，缓存内的值会丢失。
CYCLE/NOCYCLE	表示Sequence如果用完了后，是否允许从MINVALUE重新开始。取值： • CYCLE：允许； • NOCYCLE：不允许。

示例：

```
create sequence s
       start with 1
       minvalue 1
       maxvalue 9999999
       increment by 1
       cache 20
       cycle;
```

为了兼容MySQL Dump的备份方式，您也可以使用另外一种创建Sequence的方法，即创建Sequence表并插入一行初始记录。示例如下：

```
CREATE TABLE schema.sequence_name (  `currval` bigint(21) NOT NULL COMMENT 'current value',
  `nextval` bigint(21) NOT NULL COMMENT 'next value',
  `minvalue` bigint(21) NOT NULL COMMENT 'min value',
  `maxvalue` bigint(21) NOT NULL COMMENT 'max value',
  `start` bigint(21) NOT NULL COMMENT 'start value',
  `increment` bigint(21) NOT NULL COMMENT 'increment value',
  `cache` bigint(21) NOT NULL COMMENT 'cache size',
  `cycle` bigint(21) NOT NULL COMMENT 'cycle state',
  `round` bigint(21) NOT NULL COMMENT 'already how many round'
) ENGINE=Sequence DEFAULT CHARSET=latin1;
INSERT INTO schema.sequence_name VALUES(0,0,1,9223372036854775807,1,1,10000,1,0);
COMMIT;
```

Sequence表介绍

由于Sequence是通过真正的引擎表来保存的，所以通过查询创建语句看到仍然是默认的引擎表。示例如下：

```
SHOW CREATE TABLE schema.sequence_name;
CREATE TABLE schema.sequence_name (
  `currval` bigint(21) NOT NULL COMMENT 'current value',
  `nextval` bigint(21) NOT NULL COMMENT 'next value',
  `minvalue` bigint(21) NOT NULL COMMENT 'min value',
  `maxvalue` bigint(21) NOT NULL COMMENT 'max value',
  `start` bigint(21) NOT NULL COMMENT 'start value',
  `increment` bigint(21) NOT NULL COMMENT 'increment value',
  `cache` bigint(21) NOT NULL COMMENT 'cache size',
  `cycle` bigint(21) NOT NULL COMMENT 'cycle state',
  `round` bigint(21) NOT NULL COMMENT 'already how many round'
) ENGINE=Sequence DEFAULT CHARSET=latin1
```

查询语法

Sequence支持的查询语法如下：

- `SELECT nextval(<Sequence名称>),currval(<Sequence名称>) FROM <Sequence名称>;`

  说明 适用于MySQL 8.0、MySQL 5.7。

- `SELECT <Sequence名称>.currval, <Sequence名称>.nextval FROM dual;`

  说明 适用于MySQL 8.0、MySQL 5.7、MySQL 5.6。

示例：

```
mysql> SELECT test.currval, test.nextval from dual;
+--------------+--------------+
| test.currval | test.nextval |
+--------------+--------------+
|           24 |           25 |
+--------------+--------------+
1 row in set (0.03 sec)
```

说明 新创建的Sequence需要先调用一次该Sequence的NEXTVAL才能正常进行查询，否则会报 `Sequence 'xxx' is not yet defined in current session` 错误。

NEXTVAL调用示例：

```
SELECT <Sequence名称>.nextval FROM dual;
```

4.5. Returning

AliSQL提供returning功能，支持DML语句返回Result set，同时提供了工具包（DBMS_TRANS）便于您快捷使用。

背景信息

MySQL的语句执行结果报文通常分为三类：Resultset、OK和ERR。针对DML语句返回的是OK或ERR报文，其中包括影响记录、扫描记录等属性。但在很多业务场景下，执行INSERT、UPDATE、DELETE这样的DML语句后，都会跟随SELECT查询当前记录内容，以进行接下来的业务处理，为了减少一次客户端和服务器的交互，returning功能支持使用DML语句后返回Resultset。

前提条件

实例版本为RDS MySQL 8.0。

语法

```
DBMS_TRANS.returning(<Field_list>,<Statement>);
```

参数说明如下。

参数	说明
Field_list	期望的返回字段，多个字段以英文逗号（,）进行分隔，支持表中原生的字段或星号（*），不支持进行计算或者聚合等操作。
Statement	执行的DML语句，支持INSERT、UPDATE、DELETE。

注意事项

`dbms_trans.returning()` 不是事务性语句，会根据DML语句来继承事务上下文，结束事务需要显式的提交或者回滚。

INSERT Returning

针对INSERT语句，returning返回插入到表中的记录内容。

示例：

```
CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `col1` int(11) NOT NULL DEFAULT '1',
  `col2` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB;
mysql> call dbms_trans.returning("*", "insert into t(id) values(NULL),(NULL)");
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:39:05 |
|  2 |    1 | 2019-09-03 10:39:05 |
+----+------+---------------------+
2 rows in set (0.01 sec)
```

说明

- 如果没有填入Field_list，returning将返回OK或ERR报文。

```
mysql> call dbms_trans.returning("", "insert into t(id) values(NULL),(NULL)");
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
mysql> select * from t;
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:40:55 |
|  2 |    1 | 2019-09-03 10:40:55 |
|  3 |    1 | 2019-09-03 10:41:06 |
|  4 |    1 | 2019-09-03 10:41:06 |
+----+------+---------------------+
4 rows in set (0.00 sec)
```

- INSERT Returning只支持 `insert values` 形式的语法，类似 `create as` 、 `insert select` 形式的则不支持。

```
mysql> call dbms_trans.returning("", "insert into t select * from t");
ERROR 7527 (HY000): Statement didn't support RETURNING clause
```

UPDATE Returning

针对UPDATE语句，returning返回更新后的记录。

示例：

```
mysql> call dbms_trans.returning("id, col1, col2", "update t set col1 = 2 where id >2");
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  3 |    2 | 2019-09-03 10:41:06 |
|  4 |    2 | 2019-09-03 10:41:06 |
+----+------+---------------------+
2 rows in set (0.01 sec)
```

说明 UPDATE Returning不支持多表UPDATE语句。

DELETE Returning

针对DELETE语句，returning返回被删除的记录。

示例：

```
mysql> call dbms_trans.returning("id, col1, col2", "delete from t where id < 3");
+----+------+---------------------+
| id | col1 | col2                |
+----+------+---------------------+
|  1 |    1 | 2019-09-03 10:40:55 |
|  2 |    1 | 2019-09-03 10:40:55 |
+----+------+---------------------+
2 rows in set (0.00 sec)
```

4.6. Lizard事务系统

RDS MySQL 8.0启用全新的Lizard事务系统，本文介绍Lizard的优势和使用方法。

背景信息

官方版本MySQL 8.0持续对日志系统（Redo Log）、锁系统进行优化，大幅提升了系统性能，但InnoDB存储引擎的事务系统，存在读写互相干扰、XA事务不够稳定等问题。

为了更好地提升RDS MySQL数据库的吞吐能力，并支持分布式事务和全局一致性，RDS MySQL 8.0.22（小版本20201231）开始启用全新的事务系统，即Lizard事务系统，Lizard的优势主要体现在如下三个方面：

- 高并发场景提高性能
- 原生闪回查询
- XA事务和全局一致性

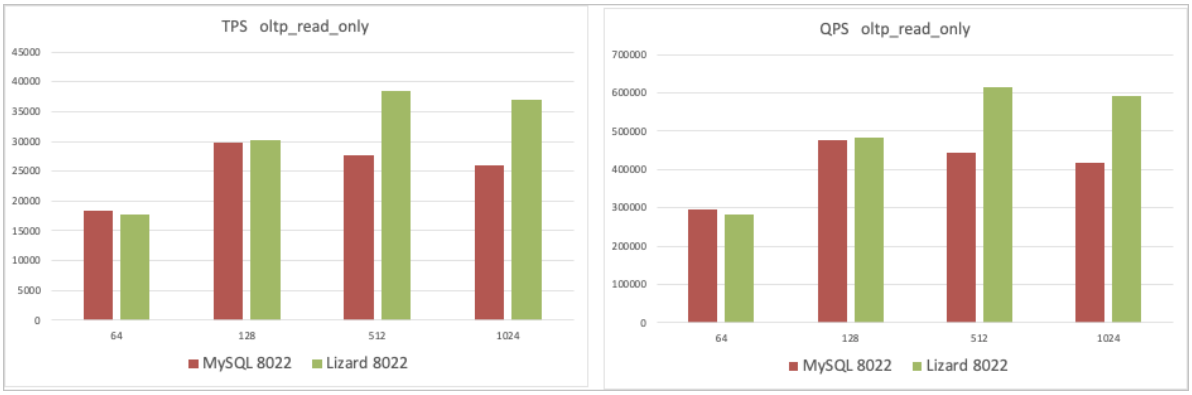
高并发场景提高性能

官方版本MySQL的InnoDB引擎使用的事务系统是一个全局结构，DML流程中事务状态的变更以及查询过程的读视图（Read View）都要访问全局事务系统，造成严重的读写干扰问题，大大限制了系统的吞吐能力。而Lizard事务系统，不再维护在查询过程中用于实现多版本并发控制（MVCC）的Read View，也就是不再访问事务系统，大幅提高MySQL使用多核CPU的能力，有效提升高并发读写混合场景下的事务吞吐能力。

官方版本MySQL和使用Lizard的RDS MySQL进行SysBench性能测试的结果如下：

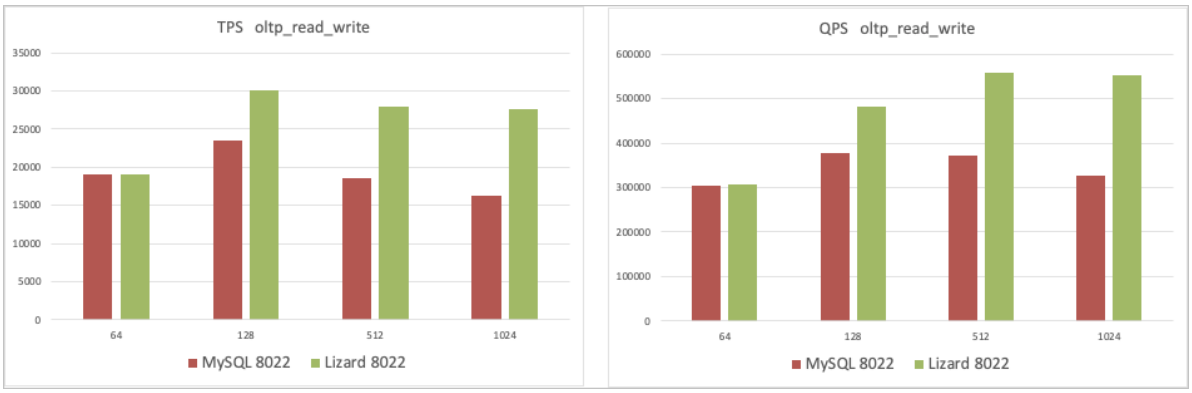
- 测试环境
  - CPU：96核（Intel(R) Xeon(R) Platinum 8163 主频2.5GHz）
  - MySQL配置：
    - innodb_buffer_pool_size：50 GB
    - `sync_binlog = 0`，即每次事务提交时，Binlog会写入缓存，但是不会立刻写入磁盘，会由操作系统决定何时写入磁盘。
    - `innodb_flush_log_at_trx_commit = 2`，即每次事务提交时，Redo日志会写入缓存，但是不会立刻写入磁盘，每秒会执行一次写入磁盘（flush）操作。

      > **说明** 设置 `sync_binlog = 0` 和 `innodb_flush_log_at_trx_commit = 2` 可以降低来自IO子系统的干扰，方便测试高并发场景下Lizard事务系统对多核CPU资源的利用，并且能消除并发争用（Concurrent Contention）。
  - 数据量：30 GB（20张表，每张表500万条记录）
- 测试场景1：SysBench OLTP Read_only



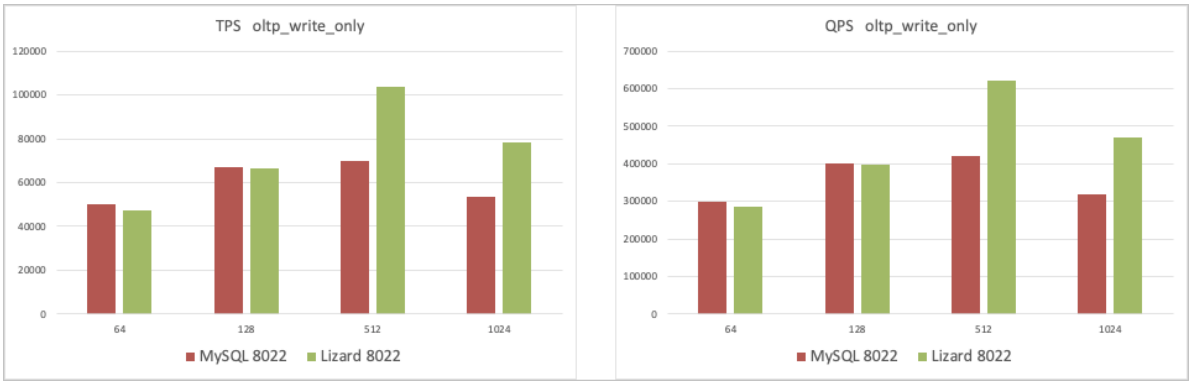
> **说明** 相比官方版本MySQL，RDS MySQL的QPS最多提高42.2%。

- 测试场景2：SysBench OLTP Read_write



> **说明** 相比官方版本MySQL，RDS MySQL的QPS最多提高69.9%。

- 测试场景3：SysBench OLTP Write_only



> **说明** 相比官方版本MySQL，RDS MySQL的QPS最多提高48.1%。

综合所有测试结果，使用Lizard事务系统的RDS MySQL能在高并发情况下，消除事务系统的热点，大幅提升系统的吞吐能力。

原生闪回查询

在RDS MySQL数据库日常运维的过程中，可能会遇到误操作或者非预期的数据修改操作，一旦提交就无法回滚，Lizard事务系统支持原生的闪回查询（Flashback Query），能够指定某个时间点进行一致性查询。

Flashback Query的语法如下：

```
SELECT ... FROM tablename
  AS OF [SCN | TIMESTAMP] expr;
```

示例

- 初始化数据

```
mysql> CREATE TABLE tab (
    -> id int PRIMARY KEY AUTO_INCREMENT,
    -> version int,
    -> gmt_modify timestamp
    -> );
mysql> INSERT INTO tab VALUES (1, 1, now()),(2, 1, now());
mysql> COMMIT;
mysql> SELECT * FROM tab;
+----+---------+---------------------+
| id | version | gmt_modify          |
+----+---------+---------------------+
|  1 |       1 | 2020-12-17 16:40:38 |
|  2 |       1 | 2020-12-17 16:40:39 |
+----+---------+---------------------+
```

- 更新数据

```
mysql> UPDATE tab SET version = version + 1, gmt_modify = now();
mysql> COMMIT;
mysql> SELECT * FROM tab;
+----+---------+---------------------+
| id | version | gmt_modify          |
+----+---------+---------------------+
|  1 |       2 | 2020-12-17 16:40:54 |
|  2 |       2 | 2020-12-17 16:40:54 |
+----+---------+---------------------+
```

- 使用Flashback Query查询

```
mysql> SELECT * FROM tab AS OF TIMESTAMP '2020-12-17 16:40:40';
+----+---------+---------------------+
| id | version | gmt_modify          |
+----+---------+---------------------+
|  1 |       1 | 2020-12-17 16:40:38 |
|  2 |       1 | 2020-12-17 16:40:39 |
+----+---------+---------------------+
mysql> SELECT * FROM tab AS OF TIMESTAMP '2020-12-17 16:40:55';
+----+---------+---------------------+
| id | version | gmt_modify          |
+----+---------+---------------------+
|  1 |       2 | 2020-12-17 16:40:54 |
|  2 |       2 | 2020-12-17 16:40:54 |
+----+---------+---------------------+
```

> **说明** 如果查询的版本时间点太旧，此时undo记录已经被删除（Truncate），就会报错：`ERROR 7546 (HY000): Snapshot too old`。

为了更好地使用Flashback Query，RDS MySQL提供三个参数来管理Flashback Query，说明如下。

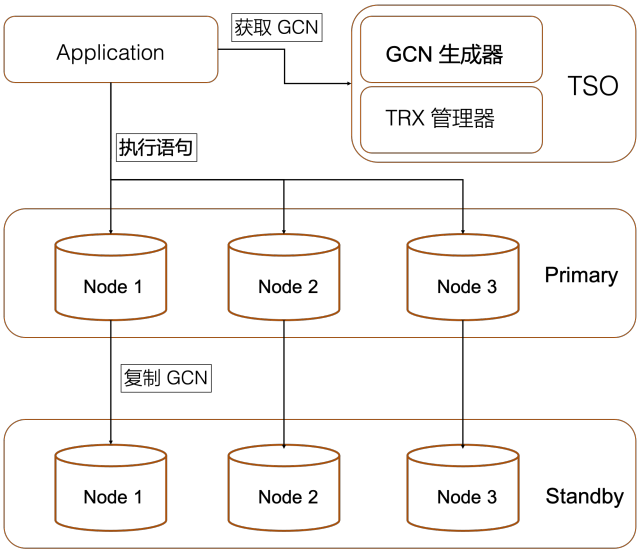
类别	INNODB_UNDO_RETENTION	INNODB_UNDO_SPACE_SUPREMUM_SIZE	INNODB_UNDO_SPACE_RESERVED_SIZE
描述	InnoDB存储引擎保留undo记录的最长时间，单位：秒。 保留的越久，Flashback Query能够支持的回档查询越久，但undo表空间所占用的存储空间也会快速上升。	InnoDB存储引擎保留的undo表空间的最大值，单位：MB。 超过这个值时，会忽略INNODB_UNDO_RETENTION强制清理undo记录。	InnoDB存储引擎预留的undo表空间大小，单位：MB。 在INNODB_UNDO_RETENTION设置有效（非零）的情况下，将使用这部分空间尽可能地多保留undo记录。
命令行格式	--innodb-undo-retention=#	--innodb-undo-space-supremum-size=#	--innodb-undo-space-reserved-size=#
参数范围	全局参数（Global）	全局参数（Global）	全局参数（Global）
是否动态参数	是	是	是
设置变量应用是否提示	否	否	否
数据类型	Integer	Integer	Integer
默认值	0	102400	0
取值范围	0~4294967295	0~4294967295	0~4294967295

XA事务和全局一致性

为了完整地支持XA事务和分布式全局一致性，Lizard事务系统支持外部传入全局事务提交号GCN（Global Commit Number），说明如下：

- 事务提交过程中，支持传入GCN，即COMMT/XA COMMIT by GCN。
- 语句查询过程中，支持传入GCN，即SELECT by GCN。

例如在最简单的分布式结构和场景下，维护一个全局授时服务器（TSO）节点，并购买三个RDS MySQL高可用版实例，即下图的Node1至Node3。



示例命令如下：

- Node 1节点

```
XA BEGIN $xid;
UPDATE account SET balance = balance + 10 WHERE user = '张三';
XA END;
XA PREPARE $xid;
XA COMMIT $xid $GCN;
```

- Node 2节点

```
XA BEGIN $xid
UPDATE account SET balance = balance - 10 WHERE user = '李四';
XA END;
XA PREPARE $xid;
XA COMMIT $xid $GCN;
```

- 查询语句
  - Node 1节点

```
SELECT * FROM account AS OF GCN $GCN where user ='张三';
```

  - Node 2节点

```
SELECT * FROM account AS OF GCN $GCN where user ='李四';
```

> **说明** RDS MySQL数据库支持GCN相关语法，目前在公测中，后续会提供更丰富的使用方法。如需使用，请提交工单进行开通。

5.性能

5.1. Fast Query Cache

针对原生MySQL Query Cache的不足，阿里云进行重新设计和全新实现，推出Fast Query Cache，能够有效提高数据库查询性能。

前提条件

- 实例版本为MySQL 5.7（内核小版本20200331或以上）。
- 实例未开启数据库独享代理。

背景信息

查询缓存（Query Cache）是为了提高查询性能而实现的一种缓存策略，其基本思想是：对于每个符合条件的查询语句，直接对结果集进行缓存，当下次查询命中时，直接从缓存中取出对应的结果集返回，不需要经历SQL的分析、优化、执行等复杂过程，通过节约CPU资源来达到查询加速的目标，是一项非常实用的技术。

MySQL原生Query Cache在设计和实现上存在着较多严重问题：

- 并发处理较差，在多核情况下，可能并发越高性能降低越严重。
- 内存管理较差，内存利用率低并且回收不及时，造成内存浪费。
- 当缓存命中率较低时，性能无提升甚至会出现严重降低。

由于以上问题，MySQL原生Query Cache没有得到广泛应用，在最新版的MySQL 8.0中，取消此功能。阿里云数据库团队对Query Cache进行重新设计和全新实现，解决了以上几个主要问题：

- 优化并发控制

  取消全局锁同步机制，采用无锁机制，重新设计并发场景下的同步问题，能够充分利用多核的处理能力，保证高并发场景下的性能。

- 优化内存管理

  取消内存预分配机制，采用更加灵活的动态内存分配机制，及时回收无效的内存，保证内存的真实利用率。

- 优化缓存机制

  动态检测缓存利用率，实时调整缓存策略，解决命中率偏低或读写混合等场景下的性能降低问题。

相比原生Query Cache，Fast Query Cache可以在不同的业务场景中放心开启，提高查询性能。

使用Fast Query Cache

您可以在RDS控制台设置参数**query_cache_type**和**query_cache_size**使用Fast Query Cache。

参数	说明
query_cache_type	Fast Query Cache功能开关，取值： • 0：默认值，禁用Fast Query Cache。 • 1：使用Fast Query Cache，但可通过SQL_NO_CACHE关键字跳过缓存。 • 2：不启用Fast Query Cache，但可通过SQL_CACHE关键字对特定语句使用缓存。

参数	说明
query_cache_size	Fast Query Cache使用的内存大小，取值范围：0~10485760000，需要为1024的整数倍。单位：Byte。

由于Fast Query Cache功能需要占用额外的内存空间，所以建议使用Fast Query Cache功能时同步修改参数innodb_buffer_pool_size的大小，推荐的修改步骤如下：

1. 修改innodb_buffer_pool_size为原先的90%，分出10%的空间给query_cache_size。例如原先为{DBInstanceClassMemory*7/10}，需要改为{DBInstanceClassMemory*63/100}。具体操作，请参见调整实例Buffer Pool大小。
2. 修改参数query_cache_size。具体操作，请参见设置实例参数。
   - 若能够评估结果集大小，query_cache_size可以设置为 `20% * 结果集大小` 。
   - 若无法准确评估结果集大小，query_cache_size可以设置为 `10% * innodb_buffer_pool_size` 。

   > **说明** 如果变更实例规格，参数query_cache_size的值不会随实例规格变化，请及时修改此参数值。
3. 修改参数query_cache_type为1，开启Fast Query Cache功能。具体操作，请参见设置实例参数。

性能比较

在相同场景下，分别测试QC-OFF（关闭Query Cache）、MySQL-QC（开启MySQL原生Query Cache）和Fast-QC（开启Fast Query Cache）的QPS。

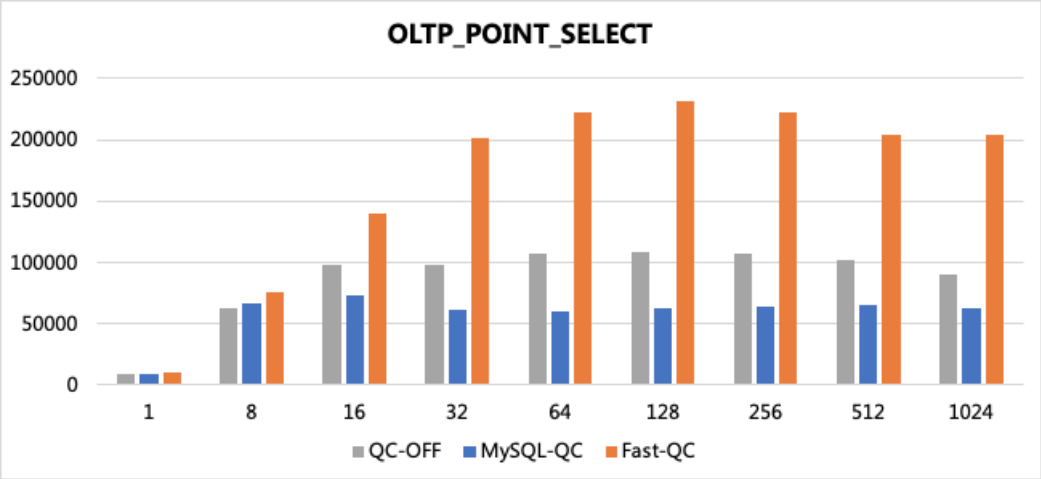
- 测试环境：4核8GB 独享型实例
- 测试工具：Sysbench
- 数据量：250MB（25张表，每张表40000条记录）
- 场景1：全部命中（只读）

  测试场景为Sysbench oltp_point_select，用例中仅包括主键上的点查（point select），将Query Cache设为512MB，内存大于测试数据量，缓存可以全部命中，主要关注不同并发下的性能提升效果。

  全部命中（只读）QPS

并发数	QC-OFF	MySQL-QC（相比QC-OFF提升）	Fast-QC（相比QC-OFF提升）
1	8093	8771（8.38%）	9261（14.43%）
8	62262	65686（5.50%）	75313（20.96%）
16	97083	73027（-24.78%）	139323（43.51%）
32	97337	60567（-37.78%）	200978（106.48%）
64	106283	60216（-43.34%）	221659（108.56%）
128	107781	62844（-41.69%）	231409（114.70%）
256	106694	63832（-40.17%）	222187（108.25%）
512	101733	64866（-36.24%）	203789（100.32%）

并发数	QC-OFF	MySQL-QC（相比QC-OFF提升）	Fast-QC（相比QC-OFF提升）
1024	89548	62291（-30.44%）	203542（127.30%）



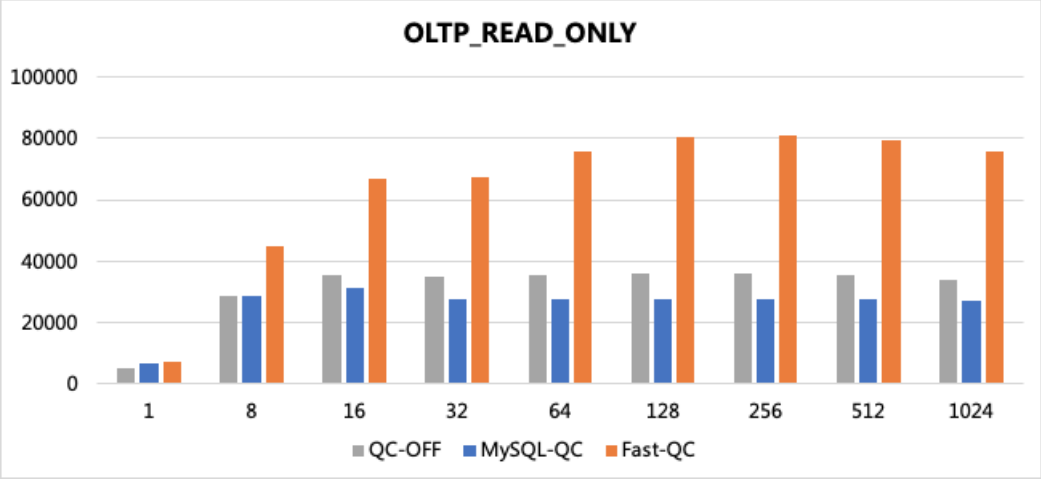
说明 测试结果显示，在较高并发的场景下，MySQL原生Query Cache并发处理性能出现较大幅度的降低，Fast Query Cache在各个并发场景下无性能降低，最高时能够提高一倍的QPS。

- 场景2：高命中率（只读）

  测试场景为Sysbench oltp_read_only，用例中包含返回多条记录的范围查询，将Query Cache设为512MB，内存才相对比较充足，命中率可以达到80%以上，这时主要关注不同并发下的性能提升效果。

  高命中率（只读）QPS

并发数	QC-OFF	MySQL-QC（相比QC-OFF提升）	Fast-QC（相比QC-OFF提升）
1	5099	6467（26.83%）	7022（37.71%）
8	28782	28651（-0.46%）	45017（56.41%）
16	35333	31099（-11.98%）	66770（88.97%）
32	34864	27610（-20.81%）	67623（93.96%）
64	35503	27518（-22.49%）	75981（114.01%）
128	35744	27733（-22.41%）	80396（124.92%）
256	35685	27738（-22.27%）	80925（126.78%）
512	35308	27398（-22.40%）	79323（124.66%）
1024	34044	26861（-22.10%）	75742（122.48%）



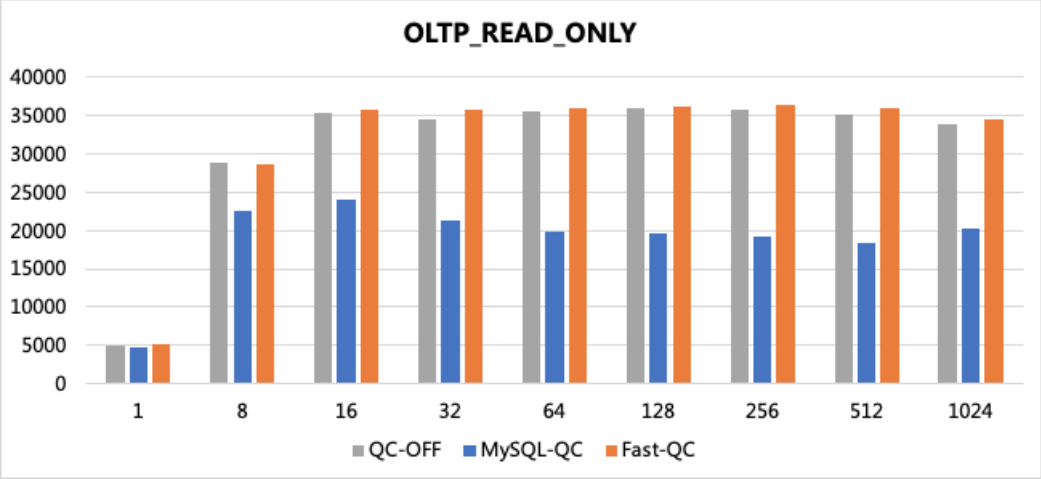
说明 测试结果显示，随着并发数的增加，MySQL原生Query Cache的性能出现明显的降低，Fast Query Cache的性能则会不断提升，最高时能够提高一倍多的QPS。

- 场景3：低命中率（只读）

  测试场景为Sysbench oltp_read_only，用例中包含返回多条记录的范围查询，将Query Cache设为16MB，内存明显严重不足，缓存命中率只有10%左右，内存不足时会涉及缓存项的大量淘汰，影响性能，这时主要关注不同并发下的性能降低程度。

  低命中率（只读）QPS

并发数	QC-OFF	MySQL-QC（相比QC-OFF提升）	Fast-QC（相比QC-OFF提升）
1	5004	4727（-5.54%）	5199（3.90%）
8	28795	22542（-21.72%）	28578（-0.75%）
16	35455	24064（-32.13%）	35682（0.64%）
32	34526	21330（-38.22%）	35871（3.90%）
64	35514	19791（-44.27%）	36051（1.51%）
128	35983	19519（-45.75%）	36253（0.75%）
256	35695	19168（-46.30%）	36337（1.80%）
512	35182	18420（-47.64%）	35972（2.25%）
1024	33915	20168（-40.53%）	34546（1.86%）



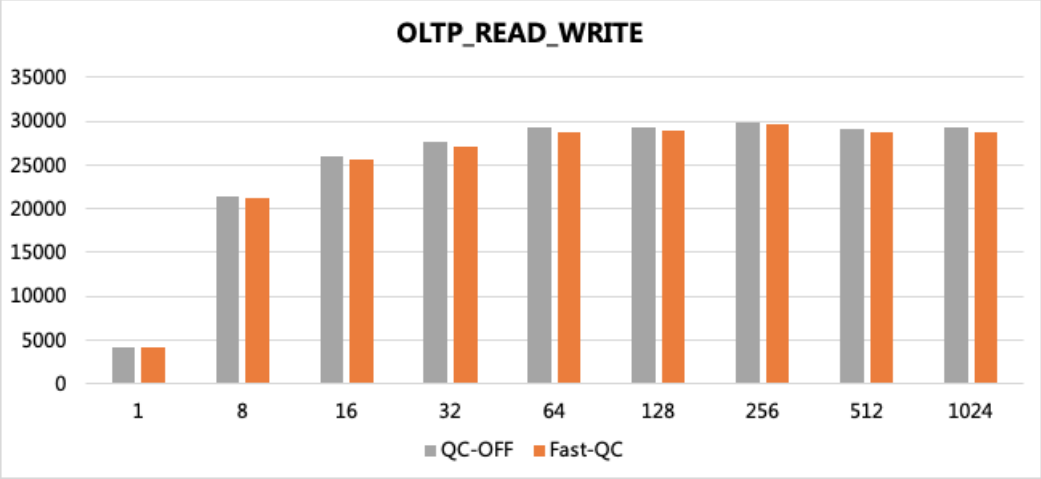
> **说明** 测试结果显示，MySQL原生Query Cache的性能降低明显，最多出现了接近50%的性能损失，Fast Query Cache优化了低命中率场景，几乎不会带来任何额外的性能损失。

- 场景4：读写混合

  测试场景为Sysbench oltp_read_write，每个事务中都有对表的更新操作，可以认为缓存基本处于失效状态，频繁的更新操作涉及缓存的主动淘汰，理论上会比较影响性能，这时主要关注不同并发下的性能衰减程度。

  读写混合QPS

并发数	QC-OFF	Fast-QC（相比QC-OFF提升）
1	4152	4098（-1.30%）
8	21359	21195（-0.77%）
16	26020	25548（-1.81%）
32	27595	26996（-2.17%）
64	29229	28733（-1.70%）
128	29265	28828（-1.49%）
256	29911	29616（-0.99%）
512	29148	28816（-1.14%）
1024	29204	28824（-1.30%）



说明 测试结果显示，Fast Query Cache在读写混合场景下不会出现过多的性能降低，整体性能影响很小。

实践指南

在缓存数据集大小明确的情况下，例如使用SQL_CACHE关键字对指定表开启Query Cache，可以参照前面的测试进行性能评估。接下来将对Fast Query Cache的使用作一些补充说明。

- 适用场景指南
  - Fast Query Cache主要目的是提高读操作性能，建议在读多写少的场景下开启，或者使用SQL_CACHE关键字针对读多写少的表开启。如果写多读少，数据的更新非常频繁，可能会出现很少的性能降低。
  - 开启Fast Query Cache带来的性能提升和缓存命中率直接相关。在全局开启前建议查看InnoDB Buffer Pool的命中率（命中率 = 1 - Innodb_buffer_pool_reads/Innodb_buffer_pool_read_requests），如果命中率低于80%，则不建议开启。您也可以通过TABLE_STATISTICS表查看表级别的读写比，对读写比高的表通过SQL_CACHE关键字显式开启Fast Query Cache。查询TABLE_STATISTICS表请参见Performance Insight。
- 缓存使用方式（query_cache_type）

  query_cache_type参数支持会话级修改，用户可以根据真实业务场景进行灵活设置，请参见以下建议：
  - 对于更新频繁、写多读少等不适合Query Cache的场景，应将query_cache_type全局设置为0。
  - 对于数据量较小、访问模式比较固定、命中率较高的场景，可以将query_cache_type全局设置为1。
  - 对于数据量较大、访问模式不固定、命中率无法保障的场景，可将query_cache_type设置为2，仅对指定的语句，通过SQL_CACHE关键字使用Fast Query Cache。
- 缓存大小（query_cache_size）设置

  query_cache_size和SQL息息相关，如果缓存中有返回多条记录的查询，缓存可能需要是数据量的数倍。如果SQL中不包含范围查询，可以参见以下测试来评估数据量和query_cache_size的关系。
  - 测试环境：4核8GB独享型实例（innodb_buffer_pool_size=6GB）
  - 测试工具：Sysbench
  - 数据量：10GB（100张表，每张表400000条记录）

  测试场景为Sysbench oltp_point_select、64并发、Special分布（20%热点）。测试不同query_cache_size大小对于性能的影响。对应上述的数据量，全量结果集的真实大小为2.5GB。

不同缓存QPS

query_cache_size（MB）	QC-OFF	Fast-QC命中率	Fast-QC（相比QC-OFF提升）
64	98236	22%	99440（1.23%）
128	98236	45%	114155（16.21%）
256	98236	72%	140668（43.19%）
512	98236	82%	151260（53.98%）
1024	98236	84%	153866（56.63%）
2048	98236	87%	159597（62.46%）
4096	98236	92%	169412（72.45%）

Fast Query Cache在不同**query_cache_size**的设置下都不会引起性能退化，对于主键查询操作，在不同缓存命中率下都有性能提升，达到90%以上时，提升效果比较明显；对于范围查询或带 `Order By` 的排序语句，缓存命中率低于90%时，也能节约大量的CPU，带来较大的性能提升。

5.2. Binlog in Redo

Binlog in Redo功能指在事务提交时将Binlog内容同步写入到Redo Log中，减少对磁盘的操作，提高数据库性能。

前提条件

实例版本为MySQL 8.0（内核小版本20200430或以上）。

背景信息

在MySQL关键业务场景中，为了业务数据的安全，事务提交时必须实时保存对应的Binlog和Redo Log，即以下两个参数必须同时设置为1：

```
sync_binlog = 1;
innodb_flush_log_at_trx_commit = 1;
```

由于每个事务提交会对磁盘进行两次I/O操作，虽然Binlog采用了Group Commit的方式合并I/O来提升效率，但两次I/O等待的本质没有改变，影响事务处理的效率，当使用云盘存储时，影响会更明显。I/O合并的效率是由同时提交的并发事务数量决定的，当并发量不够时I/O合并的效果也不理想，表现为少量的写操作事务提交时所需要的时间比较久，响应时间较长。

为了提高事务提交效率，AliSQL精心设计了Binlog in Redo机制（设置参数 `persist_binlog_to_redo = on` 开启），即在事务提交时将Binlog内容同步写入到Redo Log中。当事务提交时，只需要将Redo Log保存到磁盘中，从而减少一次对磁盘的操作，而Binlog文件则采用异步的方式，用单独的线程周期性的保存到磁盘中。在异常重启后，系统会用Redo Log中的Binlog内容来补齐Binlog文件。由于减少了一次I/O操作，性能得到了提升，响应时间变的更短，同时Binlog文件保存次数的减少，极大地降低了文件系统因文件长度实时变化带来的文件同步（fsync）压力，也提升了文件系统的性能。

Binlog in Redo功能不会改变Binlog的格式，基于Binlog的复制及第三方工具也不会受任何影响。

参数介绍

- persist_binlog_to_redo

  Binlog in Redo功能开关。全局系统变量，取值：on或off。修改本参数立刻生效，不需要重启实例。

  > **说明** 您只需要设置 `persist_binlog_to_redo = on` 即可正常使用Binlog in Redo功能，不需要修改其他参数（ `sync_binlog = 1` 自动失效）。

- sync_binlog_interval

  Binlog异步保存的间隔。全局系统变量，当 `persist_binlog_to_redo = on` 时生效。默认值：50，单位：毫秒（ms），通常使用默认值即可。修改本参数立刻生效，不需要重启实例。

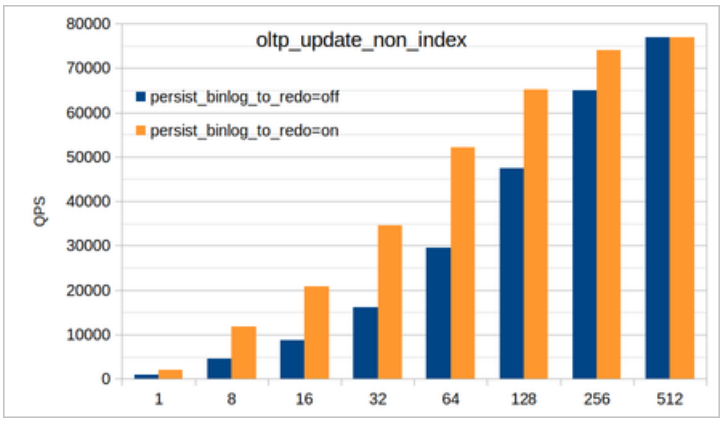
性能压力测试

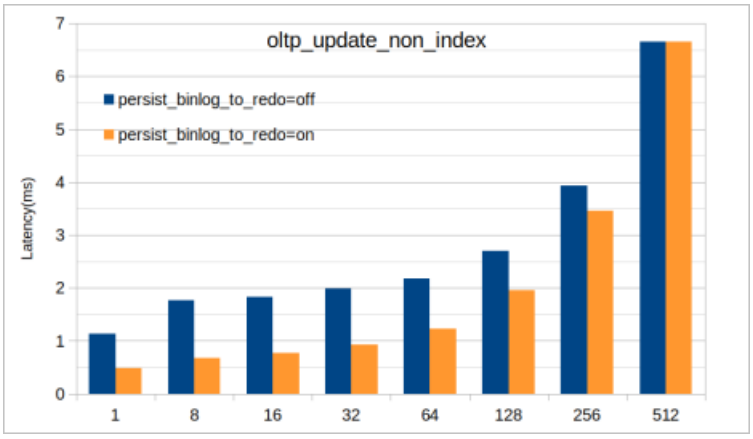
- 测试环境
  - 应用服务器：阿里云ECS实例
  - RDS实例规格： 32核、64 GB内存、ESSD云盘
  - 实例类型：高可用版（数据复制方式为异步复制）
- 测试用例

  使用的Sysbench内置用例如下：
  - oltp_update_non_index
  - oltp_insert
  - oltp_write_only
- 测试结果

- oltp_update_non_index

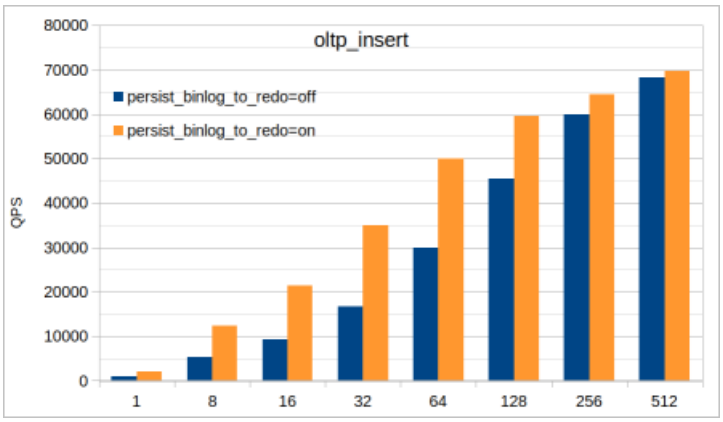
  开启Binlog in Redo后，QPS在低并发场景下提升显著，延迟也较低。

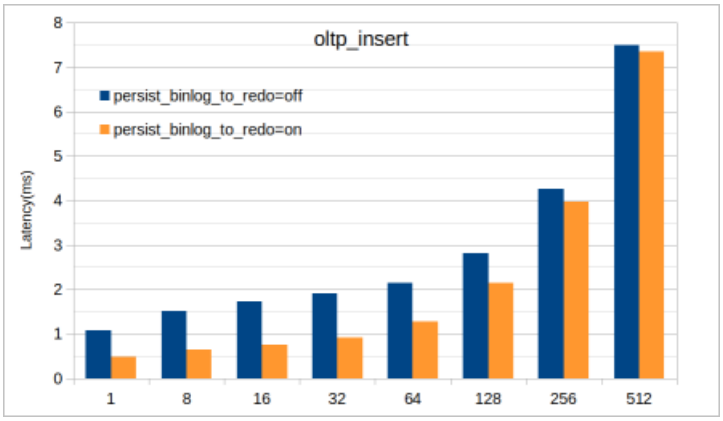




- oltp_insert

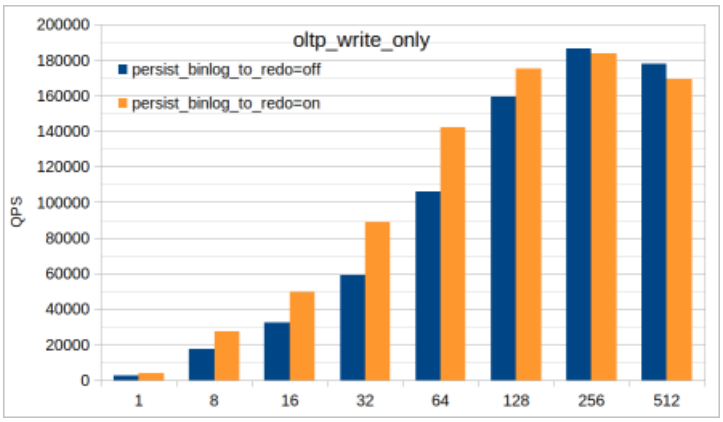
  开启Binlog in Redo后，QPS在低并发场景下提升显著，延迟也较低。

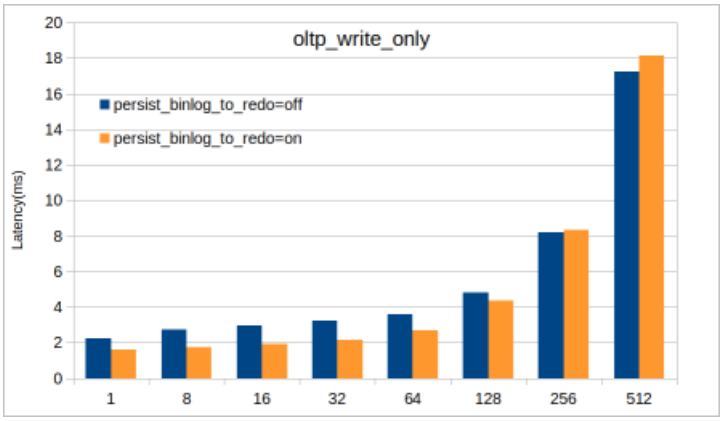




- oltp_write_only

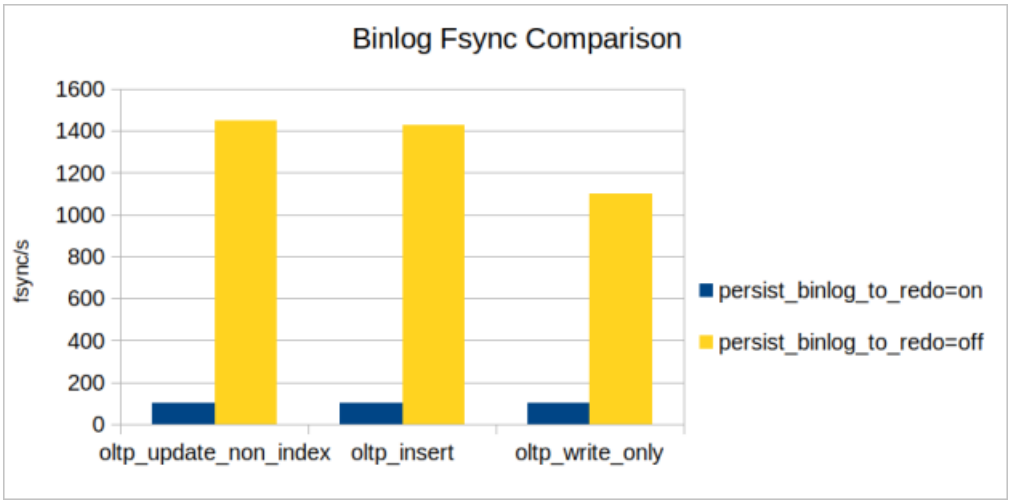
  开启Binlog in Redo后，QPS在低并发场景下有所提升，延迟也较低。





- Binlog Fsync次数对比

  开启Binlog in Redo后，Binlog fsync的次数大幅降低。



测试总结

- oltp_update_non_index和oltp_insert只包含单语句事务，事务提交次数多，而oltp_write_only包含多语句事务（2个UPDATE、1个DELETE、1个INSERT），相比oltp_update_non_index和oltp_insert，事务提交次数较少，所以oltp_update_non_index和oltp_insert的性能提升比oltp_write_only更为明显。
- 在低于256并发时，Binlog in Redo功能可以明显提升性能和降低延迟。对绝大多数的实际使用场景来

说，Binlog in Redo效果显著。

- Binlog in Redo功能开启后会大幅降低Binlog fsync的次数，可以提升文件系统的性能。

5.3. Statement Queue

AliSQL设计了针对语句的排队机制，将语句进行分桶排队，尽量把可能具有相同冲突的语句（例如操作相同行）放在一个桶内排队，减少冲突的开销。

背景信息

MySQL的服务层和引擎层在语句并发执行过程中，有很多串行的点容易导致冲突。例如在DML语句中，事务锁冲突比较常见，InnoDB中事务锁的最细粒度是行级锁，如果语句针对相同行进行并发操作，会导致冲突比较严重，系统吞吐量会随着并发的增加而递减。AliSQL提供Statement Queue机制，能够减少冲突开销、有效提高实例性能。

前提条件

实例版本如下：

- RDS MySQL 8.0基础版或高可用版（内核小版本20191115及以上）
- RDS MySQL 5.7基础版或高可用版（内核小版本20200630及以上）

效果

在单行进行并发UPDATE的场景下测试，AliSQL相比原生MySQL提升了接近4倍的性能。

变量

AliSQL提供了两个变量来定义语句队列的桶数量和大小：

- ccl_queue_bucket_count：表示桶的数量。取值范围：1~64；默认值：4。
- ccl_queue_bucket_size：表示一个桶允许的并发数。取值范围：1~4096；默认值：64。

> **说明** 您可以在RDS控制台修改变量值，详情请参见设置实例参数。

语法

AliSQL支持两种hint语法：

- ccl_queue_value

  根据值进行hash分桶。

  语法：

  ```
  /*+ ccl_queue_value([int | string]) */
  ```

  示例：

  ```
  update /*+ ccl_queue_value(1) */ t set c=c+1 where id = 1;
  update /*+ ccl_queue_value('xpchild') */ t set c=c+1 where name = 'xpchild';
  ```

- ccl_queue_field

  根据where条件中的字段值进行hash分桶。

  语法：

```
/*+ ccl_queue_field(string) */
```

示例：

```
update /*+ ccl_queue_field(id) */ t set c=c+1 where id = 1 and name = 'xpchild';
```

> **说明** ccl_queue_field语法中，where条件只支持原始字段（没有在字段上使用任何函数、计算等）的二元运算，并且二元运算的右侧值必须是数字或者字符串。

接口

AliSQL提供两个接口便于您查询Statement Queue状态：

- dbms_ccl.show_ccl_queue()

  查询当前Statement Queue状态。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+------------------+---------+---------+----------+
|    1 | QUEUE |                64 |       1 |       0 |        0 |
|    2 | QUEUE |                64 |   40744 |      65 |        6 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+------------------+---------+---------+----------+
4 rows in set (0.01 sec)
```

  参数说明如下。

参数	说明
CONCURRENCY_COUNT	最大并发数。
MATCHED	命中规则的总数。
RUNNING	当前并发的数量。
WAITTING	当前等待的数量。

- dbms_ccl.flush_ccl_queue()

  清理内存中的数据。

```
mysql> call dbms_ccl.flush_ccl_queue();
Query OK, 0 rows affected (0.00 sec)
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+-------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+-------------------+---------+---------+----------+
|    1 | QUEUE |                64 |       0 |       0 |        0 |
|    2 | QUEUE |                64 |       0 |       0 |        0 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+-------------------+---------+---------+----------+
4 rows in set (0.00 sec)
```

实践

为避免冗长的应用业务代码的修改，Statement Queue可以配合Statement Outline进行在线业务修改，方便快捷。下文使用SysBench的update_non_index为例进行演示。

- 测试环境
  - 测试表结构

```
CREATE TABLE `sbtest1` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `k` int(10) unsigned NOT NULL DEFAULT '0',
  `c` char(120) NOT NULL DEFAULT '',
  `pad` char(60) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8 MAX_ROWS=1000000;
```

  - 测试语句

```
UPDATE sbtest1 SET c='xpchild' WHERE id=0;
```

  - 测试脚本

```
./sysbench
--mysql-host= {$ip}
--mysql-port= {$port}
--mysql-db=test
--test=./sysbench/share/sysbench/update_non_index.lua
--oltp-tables-count=1
--oltp_table_size=1
--num-threads=128
--mysql-user=u0
```

- 测试过程
  i. 在线增加Statement Outline。

```
mysql> CALL DBMS_OUTLN.add_optimizer_outline('test', '', 1,
                                             ' /*+ ccl_queue_field(id) */ ',
                         "UPDATE sbtest1 SET c='xpchild' WHERE id=0");
Query OK, 0 rows affected (0.01 sec)
```

ii. 查看Statement Outline。

```
mysql> call dbms_outln.show_outline();
+------+--------+-----------------------------------------------------------------+-
----------+-------+------+---------------------------------+------+----------+--------
-------------------------------------+
| ID   | SCHEMA | DIGEST                                                          |
TYPE      | SCOPE | POS  | HINT                            | HIT  | OVERFLOW | DIGEST_
TEXT                                  |
+------+--------+-----------------------------------------------------------------+-
----------+-------+------+---------------------------------+------+----------+--------
-------------------------------------+
|    1 | test   | 7b945614749e541e0600753367884acff5df7e7ee2f5fb0af5ea58897910f023 |
OPTIMIZER |       |    1 |  /*+ ccl_queue_field(id) */  |    0 |        0 | UPDATE `s
btest1` SET `c` = ? WHERE `id` = ? |
+------+--------+-----------------------------------------------------------------+-
----------+-------+------+---------------------------------+------+----------+--------
-------------------------------------+
1 row in set (0.00 sec)
```

iii. 验证Statement Outline生效。

```
mysql> explain UPDATE sbtest1 SET c='xpchild' WHERE id=0;
+----+-------------+---------+------------+-------+---------------+---------+--------
-+-------+------+----------+-------------+
| id | select_type | table   | partitions | type  | possible_keys | key     | key_len
| ref   | rows | filtered | Extra       |
+----+-------------+---------+------------+-------+---------------+---------+--------
-+-------+------+----------+-------------+
|  1 | UPDATE      | sbtest1 | NULL       | range | PRIMARY       | PRIMARY | 4
| const |    1 |   100.00 | Using where |
+----+-------------+---------+------------+-------+---------------+---------+--------
-+-------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
mysql> show warnings;
+-------+------+-----------------------------------------------------------------------
-----------------------------------------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------------------
-----------------------------------------------------+
| Note  | 1003 | update /*+ ccl_queue_field(id) */ `test`.`sbtest1` set `test`.`sbtes
t1`.`c` = 'xpchild' where (`test`.`sbtest1`.`id` = 0) |
+-------+------+-----------------------------------------------------------------------
-----------------------------------------------------+
1 row in set (0.00 sec)
```

iv. 查询Statement Queue状态。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+------------------+---------+---------+----------+
|    1 | QUEUE |                64 |       0 |       0 |        0 |
|    2 | QUEUE |                64 |       0 |       0 |        0 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+------------------+---------+---------+----------+
4 rows in set (0.00 sec)
```

v. 开启测试。

```
sysbench
--mysql-host= {$ip}
--mysql-port= {$port}
--mysql-db=test
--test=./sysbench/share/sysbench/update_non_index.lua
--oltp-tables-count=1
--oltp_table_size=1
--num-threads=128
--mysql-user=u0
```

vi. 验证测试效果。

```
mysql> call dbms_ccl.show_ccl_queue();
+------+-------+------------------+---------+---------+----------+
| ID   | TYPE  | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTING |
+------+-------+------------------+---------+---------+----------+
|    1 | QUEUE |                64 |   10996 |      63 |        4 |
|    2 | QUEUE |                64 |       0 |       0 |        0 |
|    3 | QUEUE |                64 |       0 |       0 |        0 |
|    4 | QUEUE |                64 |       0 |       0 |        0 |
+------+-------+------------------+---------+---------+----------+
4 rows in set (0.03 sec)
mysql> call dbms_outln.show_outline();
+------+--------+-----------+-----------+-------+------+-----------------------------
---+--------+----------+-----------------------------------------------+
| ID   | SCHEMA | DIGEST    | TYPE      | SCOPE | POS  | HINT
| HIT    | OVERFLOW | DIGEST_TEXT                                    |
+------+--------+-----------+-----------+-------+------+-----------------------------
---+--------+----------+-----------------------------------------------+
|    1 | test   | xxxxxxxxx | OPTIMIZER |       |    1 |  /*+ ccl_queue_field(id) */
| 115795 |        0 | UPDATE `sbtest1` SET `c` = ? WHERE `id` = ? |
+------+--------+-----------+-----------+-------+------+-----------------------------
---+--------+----------+-----------------------------------------------+
1 row in set (0.00 sec)
```

> **说明** 查询结果显示Statement Outline命中了115,795次规则，Statement Queue状态显示命中了10,996次排队，当前运行并发63个，排队等待4个。

5.4. Inventory Hint

AliSQL提供Inventory Hint，帮助您快速提交/回滚事务，配合Returning和Statement Queue，能有效提高业务吞吐能力。

背景信息

在秒杀等业务场景中，减少库存是一个常见的需要高并发，同时也需要串行化的任务模型，AliSQL使用排队和事务性Hint来控制并发和快速提交/回滚事务，提高业务吞吐能力。

结合Inventory Hint，RDS的单行热点更新性能可达3.1万TPS（参见单行热点更新测试）。

前提条件

实例版本如下：

- MySQL 8.0
- MySQL 5.7
- MySQL 5.6

语法

新增了三个Hint， 支持SELECT、UPDATE、INSERT、DELETE 语句。

- COMMIT_ON_SUCCESS/ROLLBACK_ON_FAIL

  两个事务Hint为COMMIT_ON_SUCCESS和ROLLBACK_ON_FAIL：

  - COMMIT_ON_SUCCESS：当前语句执行成功就提交事务上下文。
  - ROLLBACK_ON_FAIL：当前语句执行失败就回滚事务上下文。

  语法：

  ```
  /*+ COMMIT_ON_SUCCESS */
  /*+ ROLLBACK_ON_FAIL */
  ```

  示例：

  ```
  UPDATE /*+ COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL */ T
  SET c = c - 1
  WHERE id = 1;
  ```

- TARGET_AFFECT_ROW(NUMBER)

  条件Hint为TARGET_AFFECT_ROW(NUMBER)： 如果当前语句影响行数是指定的就成功，否则语句失败。

  语法：

  ```
  /*+ TARGET_AFFECT_ROW(NUMBER) */
  ```

  示例：

  ```
  UPDATE /*+ TARGET_AFFECT_ROW(1) */ T
  SET c = c - 1
  WHERE id = 1;
  ```

注意事项

- 事务Hint不能运行在autocommit模式下， 例如：

```
mysql> UPDATE /*+ commit_on_success rollback_on_fail target_affect_row(1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
ERROR 7531 (HY000): Inventory transactinal hints didn't allowed in autocommit mode
```

- 事务Hint不能运行在sub statement下，例如：

```
mysql> CREATE TRIGGER tri_1
    -> BEFORE INSERT ON t
    -> FOR EACH ROW
    -> BEGIN
    -> INSERT /*+ commit_on_success */ INTO t1 VALUES (1);
    -> end//
mysql> INSERT INTO t VALUES (2, 1);
ERROR HY000: Inventory transactional hints didn't alllowed in stored procedure
```

- 条件Hint不能运行在SELECT/EXPLAIN statement下， 例如：

```
mysql> EXPLAIN UPDATE /*+ commit_on_success rollback_on_fail target_affect_row(1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
ERROR 7532 (HY000): Inventory conditional hints didn't match with result
```

说明 您可以指定target_affect_row为一个无效的number进行测试，系统会有告警。

```
mysql> EXPLAIN UPDATE /*+ commit_on_success rollback_on_fail target_affect_row(-1) */
t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
+----+-------------+-------+------------+-------+---------------+---------+---------+
-------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len |
ref   | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+
-------+------+----------+-------------+
|  1 | UPDATE      | t     | NULL       | range | PRIMARY       | PRIMARY | 4       |
const |    1 |   100.00 | Using where |
+----+-------------+-------+------------+-------+---------------+---------+---------+
-------+------+----------+-------------+
1 row in set, 2 warnings (0.00 sec)
mysql> show warnings;
+---------+------+-----------------------------------------------------------------------
-----------------------------------------------------------------------+
| Level   | Code | Message
|
+---------+------+-----------------------------------------------------------------------
-----------------------------------------------------------------------+
| Warning | 1064 | Optimizer hint syntax error near '-1) */ t set col1=col1+1 where i
d =1' at line 1                                                        |
| Note    | 1003 | update /*+ COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL */ `test`.`t` set `t
est`.`t`.`col1` = (`test`.`t`.`col1` + 1) where (`test`.`t`.`id` = 1) |
+---------+------+-----------------------------------------------------------------------
-----------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

配合Returning使用

Inventory Hint可以配合Returning使用，实时返回结果集， 例如：

```
mysql> CALL dbms_trans.returning("*", "update /*+ commit_on_success rollback_on_fail target
_affect_row(1) */ t
                                       set col1=col1+1 where id=1");
+----+------+
| id | col1 |
+----+------+
|  1 |   13 |
+----+------+
1 row in set (0.00 sec)
mysql> CALL dbms_trans.returning("*", "insert /*+ commit_on_success rollback_on_fail target
_affect_row(1) */ into
                                       t values(10,10)");
+----+------+
| id | col1 |
+----+------+
| 10 |   10 |
+----+------+
1 row in set (0.01 sec)
```

配合Statement queue使用

Inventory Hint可以配合Statement Queue进行排队，例如：

```
mysql> UPDATE /*+ ccl_queue_field(id) commit_on_success rollback_on_fail target_affect_row(
1) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> UPDATE /*+ ccl_queue_value(1) commit_on_success rollback_on_fail target_affect_row(1
) */ t
    -> SET col1 = col1 + 1
    -> WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

6.稳定

6.1. Faster DDL

优化DDL操作过程中的Buffer Pool管理机制，降低DDL操作带来的性能影响，提升在线DDL操作的并发数。

前提条件

实例版本如下：

- MySQL 8.0（内核小版本为20200630或以上）
- MySQL 5.7（内核小版本为20200630或以上）
- MySQL 5.6（内核小版本为20200630或以上）

背景信息

数据库经常会执行DDL操作，也经常会遇到DDL相关的问题，例如：

- 为什么加索引会造成实例的抖动，影响正常的业务读写？
- 为什么不到1 GB的表执行DDL有时需要十几分钟？
- 为什么使用了临时表的连接退出时会造成实例抖动？

针对这些常见问题，RDS内核团队进行分析后发现MySQL在DDL操作期间的缓存维护逻辑存在性能缺陷，通过深入分析及多次测试，开发Faster DDL功能，优化了Buffer Pool页面管理策略，大幅减少DDL操作导致的锁争用，有效解决或缓解上述问题，让您的实例在正常业务压力下可以安心执行DDL操作。

开启Faster DDL

您可以在控制台修改参数**loose_innodb_rds_faster_ddl**为**ON**，开启Faster DDL。详情请参见设置实例参数。

DDL场景测试

- 测试场景

  选取RDS MySQL 8.0支持的两种In Place Online DDL操作进行验证， 其中CREATE INDEX操作不需要重建表，OPTIMIZE TABLE操作需要重建表。

操作	Instant	In Place	重建表	允许并发DML	只修改元数据
CREATE INDEX	否	是	否	是	否
OPTIMIZE TABLE	否	是	是	是	否

- 测试实例

  MySQL 8.0实例（8核、64 GB），执行DDL操作的表大小为600 MB。

- 测试过程

  使用Sysbench模拟线上业务进行压力测试，在压测期间执行DDL操作，进行反复对比测试。

- 测试结果

操作	平均执行时间（关闭优化）	平均执行时间（开启优化）	性能提升倍数
Create Index	56秒	4.9秒	11.4
Optimize Table	220秒	17秒	12.9

- 测试小结

  在DDL场景下，优化后的AliSQL内核MySQL相比社区版本MySQL，DDL操作执行时间缩短了90%以上。

临时表场景测试

MySQL在很多情况下会使用临时表，例如查询information_schema库里的表、 加速复杂SQL执行时自动创建临时表。在线程退出时系统会集中清理用过的临时表，这也属于一种特殊类型的DDL操作，同样会导致实例的性能抖动。 详情请参见Temp ibt tablespace truncation at disconnection stuck InnoDB under large BP。

- 测试实例

  MySQL 8.0实例（8核、64 GB）。

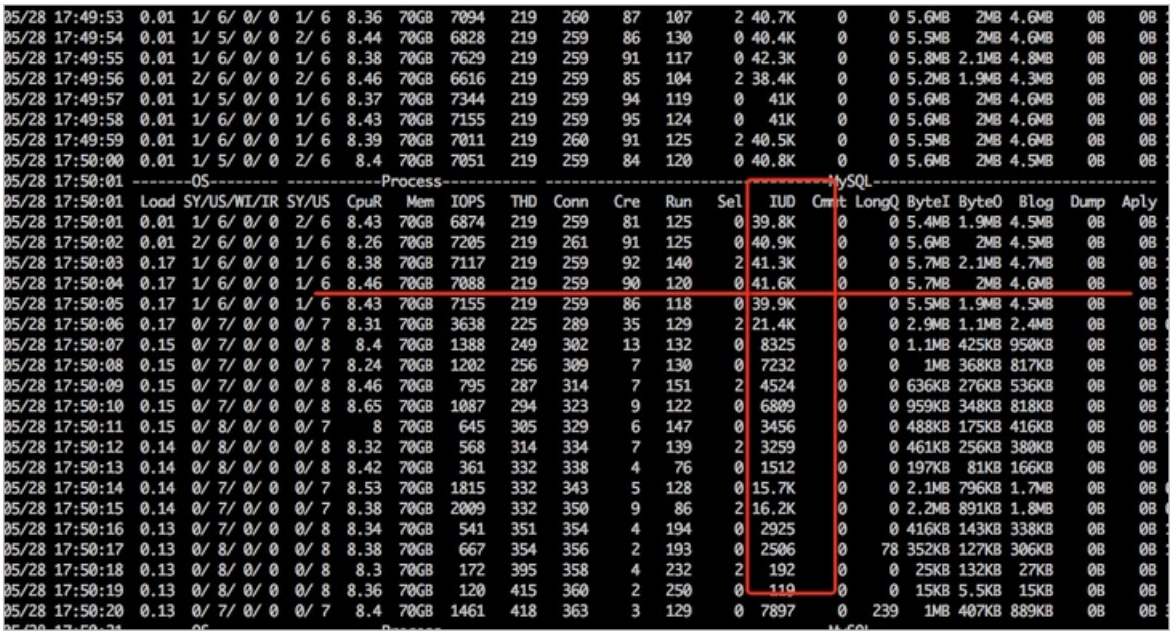
- 测试过程

  使用tpcc-mysql进行压力测试，将Buff Pool基本用满，然后发起单线程短连接的临时表请求。

- 测试结果

对比项	正常情况（无DDL操作）	开启优化	关闭优化
每秒事务数（TPS）	42,000	40,000	<10,000

压测过程中的秒级性能数据如下图所示（红线处为关闭DDL加速功能）：



```
05/28 17:49:53  0.01  1/ 6/ 0/ 0  1/ 6  8.36  70GB  7094   219   260    87   107     2 40.7K     0     0 5.6MB   2MB 4.6MB    0B    0B
05/28 17:49:54  0.01  1/ 5/ 0/ 0  2/ 6  8.44  70GB  6828   219   259    86   130     0 40.4K     0     0 5.5MB   2MB 4.6MB    0B    0B
05/28 17:49:55  0.01  1/ 6/ 0/ 0  1/ 6  8.38  70GB  7629   219   259    91   117     0 42.3K     0     0 5.8MB 2.1MB 4.8MB    0B    0B
05/28 17:49:56  0.01  2/ 6/ 0/ 0  2/ 6  8.46  70GB  6616   219   259    85   104     2 38.4K     0     0 5.2MB 1.9MB 4.3MB    0B    0B
05/28 17:49:57  0.01  1/ 5/ 0/ 0  1/ 6  8.37  70GB  7344   219   259    94   119     0   41K     0     0 5.6MB   2MB 4.6MB    0B    0B
05/28 17:49:58  0.01  1/ 6/ 0/ 0  1/ 6  8.43  70GB  7155   219   259    95   124     0   41K     0     0 5.6MB   2MB 4.6MB    0B    0B
05/28 17:49:59  0.01  1/ 6/ 0/ 0  1/ 6  8.39  70GB  7011   219   260    91   125     2 40.5K     0     0 5.5MB   2MB 4.6MB    0B    0B
05/28 17:50:00  0.01  1/ 5/ 0/ 0  2/ 6   8.4  70GB  7051   219   259    84   120     0 40.8K     0     0 5.6MB   2MB 4.5MB    0B    0B
05/28 17:50:01 -------OS-------- ----------Process---------- ----------------------------------MySQL-----------------------------------
05/28 17:50:01  Load SY/US/WI/IR SY/US  CpuR   Mem  IOPS  THD  Conn   Cre   Run   Sel   IUD  Cmit LongQ ByteI ByteO  Blog  Dump  Aply
05/28 17:50:01  0.01  1/ 6/ 0/ 0  2/ 6  8.43  70GB  6874   219   259    81   125     0 39.8K     0     0 5.4MB 1.9MB 4.5MB    0B    0B
05/28 17:50:02  0.01  2/ 6/ 0/ 0  1/ 6  8.26  70GB  7205   219   261    91   125     0 40.9K     0     0 5.6MB   2MB 4.5MB    0B    0B
05/28 17:50:03  0.17  1/ 6/ 0/ 0  1/ 6  8.38  70GB  7117   219   259    92   140     2 41.3K     0     0 5.7MB 2.1MB 4.7MB    0B    0B
05/28 17:50:04  0.17  1/ 6/ 0/ 0  1/ 6  8.46  70GB  7088   219   259    90   120     0 41.6K     0     0 5.7MB   2MB 4.6MB    0B    0B
05/28 17:50:05  0.17  1/ 6/ 0/ 0  1/ 6  8.43  70GB  7155   219   259    86   118     0 39.9K     0     0 5.5MB 1.9MB 4.5MB    0B    0B
05/28 17:50:06  0.17  0/ 7/ 0/ 0  0/ 7  8.31  70GB  3638   225   289    35   129     2 21.4K     0     0 2.9MB 1.1MB 2.4MB    0B    0B
05/28 17:50:07  0.15  0/ 7/ 0/ 0  0/ 8   8.4  70GB  1388   249   302    13   132     0  8325     0     0 1.1MB 425KB 950KB    0B    0B
05/28 17:50:08  0.15  0/ 7/ 0/ 0  0/ 7  8.24  70GB  1202   256   309     7   130     0  7232     0     0   1MB 368KB 817KB    0B    0B
05/28 17:50:09  0.15  0/ 7/ 0/ 0  0/ 8  8.46  70GB   795   287   314     7   151     2  4524     0     0 636KB 276KB 536KB    0B    0B
05/28 17:50:10  0.15  0/ 7/ 0/ 0  0/ 8  8.65  70GB  1087   294   323     9   122     0  6809     0     0 959KB 348KB 818KB    0B    0B
05/28 17:50:11  0.15  0/ 8/ 0/ 0  0/ 7     8  70GB   645   305   329     6   147     0  3456     0     0 488KB 175KB 416KB    0B    0B
05/28 17:50:12  0.14  0/ 8/ 0/ 0  0/ 8  8.32  70GB   568   314   334     7   139     2  3259     0     0 461KB 256KB 380KB    0B    0B
05/28 17:50:13  0.14  0/ 8/ 0/ 0  0/ 8  8.42  70GB   361   332   338     4    76     0  1512     0     0 197KB  81KB 166KB    0B    0B
05/28 17:50:14  0.14  0/ 7/ 0/ 0  0/ 7  8.53  70GB  1815   332   343     5   128     0 15.7K     0     0 2.1MB 796KB 1.7MB    0B    0B
05/28 17:50:15  0.14  0/ 7/ 0/ 0  0/ 7  8.38  70GB  2009   332   350     9    86     2 16.2K     0     0 2.2MB 891KB 1.8MB    0B    0B
05/28 17:50:16  0.13  0/ 7/ 0/ 0  0/ 8  8.34  70GB   541   351   354     4   194     0  2925     0     0 416KB 143KB 338KB    0B    0B
05/28 17:50:17  0.13  0/ 8/ 0/ 0  0/ 8  8.38  70GB   667   354   356     2   193     0  2506     0    78 352KB 127KB 306KB    0B    0B
05/28 17:50:18  0.13  0/ 8/ 0/ 0  0/ 8   8.3  70GB   172   395   358     4   232     2   192     0     0  25KB 132KB  27KB    0B    0B
05/28 17:50:19  0.13  0/ 8/ 0/ 0  0/ 8  8.36  70GB   120   415   360     2   250     0   119     0     0  15KB 5.5KB  15KB    0B    0B
05/28 17:50:20  0.13  0/ 7/ 0/ 0  0/ 7   8.4  70GB  1461   418   363     3   129     0  7897     0   239   1MB 407KB 889KB    0B    0B
```

- 测试小结

  原生MySQL在每次临时表线程退出时出现剧烈的性能抖动，TPS下降超过70%，开启优化之后性能影响降低至5%。

优化效果

Faster DDL加速功能支持RDS MySQL 5.6、5.7、8.0三个版本，但是不同版本支持加速的DDL类型不同，详情请参见下表。

分类	DDL操作	MySQL 5.6	MySQL 5.7	MySQL 8.0
In Place DDL	详情请参见MySQL 8.0 Online DDL Operations和MySQL 5.7 Online DDL Operations。	否	是	是
表空间管理	开关表空间加密。	否	是	是
	释放或删除表空间。	否	是	是
	丢弃表空间。	是	是	是
删除表	释放或删除表。	是	是	是
Undo操作	释放或删除undo表空间。	否	否	是
刷新表	刷新表及脏页。	是	是	是

Faster DDL解决的缺陷

Faster DDL完美解决了以下MySQL临时缺陷：

- Bug #95582: DDL using bulk load is very slow under long flush_list
- Bug #98869: Temp ibt tablespace truncation at disconnection stuck InnoDB under large BP
- Bug #99021: BUF_REMOVE_ALL_NO_WRITE is not needed for undo tablespace
- Bug #98974: InnoDB temp table could hurt InnoDB perf badly

6.2. Statement Concurrency Control

为了应对突发的数据库请求流量、资源消耗过高的语句访问以及SQL访问模型的变化， 保证MySQL实例持续稳定运行，阿里云提供基于语句规则的并发控制CCL（Concurrency Control），并提供了工具包（DBMS_CCL）便于您快捷使用。

前提条件

实例版本如下：

- MySQL 8.0
- MySQL 5.7

注意事项

- CCL的操作不产生Binlog，所以CCL的操作只影响当前实例。例如主实例进行CCL操作，不会同步到备实例、只读实例或灾备实例。
- CCL提供超时机制以应对DML导致事务锁死锁，等待中的线程也会响应事务超时和线程KILL操作以应对死锁。

功能设计

CCL规则定义了如下三个维度的特征：

- SQL command

  SQL命令类型，例如SELECT、UPDATE、INSERT、DELETE等。

- Object

  SQL命令操作的对象，例如TABLE、VIEW等。

- keywords

  SQL命令的关键字。

创建CCL规则表

AliSQL设计了一个系统表（concurrency_control）保存CCL规则，系统启动时会自动创建该表，无需您手动创建。这里提供表的创建语句供您参考：

```
CREATE TABLE `concurrency_control` (
  `Id` bigint(20) NOT NULL AUTO_INCREMENT,
  `Type` enum('SELECT','UPDATE','INSERT','DELETE') NOT NULL DEFAULT 'SELECT',
  `Schema_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Table_name` varchar(64) COLLATE utf8_bin DEFAULT NULL,
  `Concurrency_count` bigint(20) DEFAULT NULL,
  `Keywords` text COLLATE utf8_bin,
  `State` enum('N','Y') NOT NULL DEFAULT 'Y',
  `Ordered` enum('N','Y') NOT NULL DEFAULT 'N',
  PRIMARY KEY (`Id`)
) /*!50100 TABLESPACE `mysql` */ ENGINE=InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_bin
STATS_PERSISTENT=0 COMMENT='Concurrency control'
```

参数	说明
Id	CCL规则ID。
Type	SQL command，即SQL命令类型。
Schema_name	数据库名。
Table_name	数据库内的表名。
Concurrency_count	并发数。
Keywords	关键字，多个关键字用英文分号（;）分隔。
State	本规则是否启用。
Ordered	Keywords中多个关键字是否按顺序匹配。

管理CCL规则

为了便捷地管理CCL规则，AliSQL在DBMS_CCL中定义了四个本地存储规则。详细说明如下：

- add_ccl_rule

增加规则。命令如下：

```
dbms_ccl.add_ccl_rule('<Type>','<Schema_name>','<Table_name>',<Concurrency_count>,'<Keywords>');
```

示例：

增加规则，SELECT语句的并发数为10。

```
mysql> call dbms_ccl.add_ccl_rule('SELECT', '', '', 10, '');
```

增加规则，SELECT语句中出现关键字key1的并发数为20。

```
mysql> call dbms_ccl.add_ccl_rule('SELECT', '', '', 20, 'key1');
```

增加规则，test.t表的SELECT语句的并发数为20。

```
mysql> call dbms_ccl.add_ccl_rule('SELECT', 'test', 't', 20, '');
```

> **说明** Id越大，规则的优先级越高。

- del_ccl_rule

  删除规则。命令如下：

  ```
  dbms_ccl.del_ccl_rule(<Id>);
  ```

  示例：

  删除规则ID为15的CCL规则。

  ```
  mysql> call dbms_ccl.del_ccl_rule(15);
  ```

  > **说明** 如果删除的规则不存在，系统会报相应的警告，您可以使用 `show warnings;` 查看警告内容。

  ```
  mysql> call dbms_ccl.del_ccl_rule(100);
    Query OK, 0 rows affected, 2 warnings (0.00 sec)
  mysql> show warnings;
  +---------+------+-----------------------------------------------------+
  | Level   | Code | Message                                             |
  +---------+------+-----------------------------------------------------+
  | Warning | 7514 | Concurrency control rule 100 is not found in table |
  | Warning | 7514 | Concurrency control rule 100 is not found in cache |
  +---------+------+-----------------------------------------------------+
  ```

- show_ccl_rule

  查看内存中已启用规则。命令如下：

  ```
  dbms_ccl.show_ccl_rule();
  ```

  示例：

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNING
| WAITTING | KEYWORDS |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
|   17 | SELECT | test   | t     | Y     | N     |                30 |       0 |       0
|        0 |          |
|   16 | SELECT |        |       | Y     | N     |                20 |       0 |       0
|        0 | key1     |
|   18 | SELECT |        |       | Y     | N     |                10 |       0 |       0
|        0 |          |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------
+----------+----------+
```

关于MATCHED、RUNNING和WAITTING的说明如下。

参数	说明
MATCHED	规则匹配成功次数。
RUNNING	此规则下正在并发执行的线程数。
WAITTING	此规则下正在等待执行的线程数。

- flush_ccl_rule

  如果您直接操作了表concurrency_control修改规则，规则不能立即生效，您需要让规则重新生效。命令如下：

  ```
  dbms_ccl.flush_ccl_rule();
  ```

  示例：

  ```
  mysql> update mysql.concurrency_control set CONCURRENCY_COUNT = 15 where Id = 18;
  Query OK, 1 row affected (0.00 sec)
  Rows matched: 1  Changed: 1  Warnings: 0
  mysql> call dbms_ccl.flush_ccl_rule();
  Query OK, 0 rows affected (0.00 sec)
  ```

功能测试

- 测试规则

  设计如下三条规则对应三个维度：

  ```
  call dbms_ccl.add_ccl_rule('SELECT', 'test', 'sbtest1', 3, '');  //SELECT命令操作表sbtest1
  并发数为3
  call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, 'sbtest2');       //SELECT命令关键字sbtest
  2并发数为2
  call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, '');            //SELECT命令并发数为2
  ```

- 测试场景

使用sysbench进行测试，场景如下：

- 64 threads
- 4 tables
- select.lua

- 测试结果

  查看规则并发数情况如下：

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+---------+-------+-------+-------------------+---------+-------
--+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE   | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNIN
G | WAITTING | KEYWORDS |
+------+--------+--------+---------+-------+-------+-------------------+---------+-------
--+----------+----------+
|   20 | SELECT | test   | sbtest1 | Y     | N     |                 3 |     389 |
3 |        9 |          |
|   21 | SELECT |        |         | Y     | N     |                 2 |     375 |
2 |       14 | sbtest2  |
|   22 | SELECT |        |         | Y     | N     |                 2 |     519 |
2 |       34 |          |
+------+--------+--------+---------+-------+-------+-------------------+---------+-------
--+----------+----------+
3 rows in set (0.00 sec)
```

  查看**RUNNING**列，符合预期的并行数量。

6.3. Performance Agent

Performance Agent是AliSQL提供的一种更加便捷的性能数据统计方案。通过MySQL插件的方式，实现MySQL实例内部各项性能数据的采集与统计。

背景信息

Performance Agent在information_schema系统库下新增了一张内存表PERF_STATISTICS，用于统计最近一段时间的性能数据，您可以直接查询该表获取相关指标的性能数据。

前提条件

实例版本如下：

- MySQL 8.0（内核小版本20200229或以上）
- MySQL 5.7（内核小版本20200229或以上）
- MySQL 5.6（内核小版本20200630或以上）

> **说明** 升级内核小版本请参见升级内核小版本。

参数说明

与Performance Agent功能相关的参数说明如下。

参数	说明
performance_agent_enabled	是否开启Performance Agent功能。取值：ON | OFF。默认值为ON。
performance_agent_interval	性能数据采集间隔。单位为秒，默认值为1。
performance_agent_perfstat_volume_size	PERF_STATISTICS表的最大数据条数。默认值为3600。即当采样间隔为1秒时，保存最近1小时的性能数据。

说明 上述三个参数在控制台中不可见，您可通过 `SHOW VARIABLES LIKE '<参数名称>';` 查看参数的状态。

表结构说明

PERF_STATISTICS内存表的结构如下：

```
CREATE TEMPORARY TABLE `PERF_STATISTICS` (
  `TIME` datetime NOT NULL DEFAULT '0000-00-00 00:00:00',
  `PROCS_MEM_USAGE` double NOT NULL DEFAULT '0',
  `PROCS_CPU_RATIO` double NOT NULL DEFAULT '0',
  `PROCS_IOPS` double NOT NULL DEFAULT '0',
  `PROCS_IO_READ_BYTES` bigint(21) NOT NULL DEFAULT '0',
  `PROCS_IO_WRITE_BYTES` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_CONN_ABORT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_CONN_CREATED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_USER_CONN_COUNT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_CONN_RUNNING` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LOCK_IMMEDIATE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LOCK_WAITED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_INSERT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_UPDATE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_DELETE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_SELECT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_COMMIT` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_ROLLBACK` int(11) NOT NULL DEFAULT '0',
  `MYSQL_COM_PREPARE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_LONG_QUERY` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TCACHE_GET` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_TCACHE_MISS` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_TMPFILE_CREATED` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TMP_TABLES` int(11) NOT NULL DEFAULT '0',
  `MYSQL_TMP_DISKTABLES` int(11) NOT NULL DEFAULT '0',
  `MYSQL_SORT_MERGE` int(11) NOT NULL DEFAULT '0',
  `MYSQL_SORT_ROWS` int(11) NOT NULL DEFAULT '0',
  `MYSQL_BYTES_RECEIVED` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_BYTES_SENT` bigint(21) NOT NULL DEFAULT '0',
  `MYSQL_BINLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `MYSQL_IOLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `MYSQL_RELAYLOG_OFFSET` int(11) NOT NULL DEFAULT '0',
  `EXTRA` json NOT NULL DEFAULT 'null'
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

列名	说明
TIME	时间，格式为yyyy-MM-dd HH:mm:ss。
PROCS_MEM_USAGE	物理内存使用量，单位为Byte。
PROCS_CPU_RATIO	CPU使用率。
PROCS_IOPS	系统IO调用次数。
PROCS_IO_READ_BYTES	IO读取数据量，单位为Byte。
PROCS_IO_WRITE_BYTES	IO写入数据量，单位为Byte。
MYSQL_CONN_ABORT	断开连接数。
MYSQL_CONN_CREATED	新建连接数。
MYSQL_USER_CONN_COUNT	当前总的用户连接数。
MYSQL_CONN_RUNNING	当前活跃连接数。
MYSQL_LOCK_IMMEDIATE	当前锁占用数。
MYSQL_LOCK_WAITED	当前锁等待数。
MYSQL_COM_INSERT	插入语句数。
MYSQL_COM_UPDATE	更新语句数。
MYSQL_COM_DELETE	删除语句数。
MYSQL_COM_SELECT	查询语句数。
MYSQL_COM_COMMIT	事务提交数（显式提交）。
MYSQL_COM_ROLLBACK	事务回滚数。
MYSQL_COM_PREPARE	预处理语句数。
MYSQL_LONG_QUERY	慢查询数。
MYSQL_TCACHE_GET	缓存表命中数。
MYSQL_TCACHE_MISS	缓存表未命中数。
MYSQL_TMPFILE_CREATED	临时文件创建数。
MYSQL_TMP_TABLES	临时表创建数。
MYSQL_TMP_DISKTABLES	临时磁盘表创建数。
MYSQL_SORT_MERGE	合并排序次数。

列名	说明
MYSQL_SORT_ROWS	排序行数。
MYSQL_BYTES_RECEIVED	接收数据量，单位为Byte。
MYSQL_BYTES_SENT	发送数据量，单位为Byte。
MYSQL_BINLOG_OFFSET	产生的Binlog文件大小，单位为Byte。
MYSQL_IOLOG_OFFSET	主库发送的Binlog文件大小，单位为Byte。
MYSQL_RELAYLOG_OFFSET	从库应用的Binlog文件大小，单位为Byte。
EXTRA	InnoDB统计信息。EXTRA包含多个字段，为JSON格式。详细字段介绍请参见下方EXTRA字段说明。 **说明** InnoDB统计信息的指标项与 `SHOW STATUS` 命令显示的值相同。

EXTRA字段说明

字段	说明
INNODB_TRX_CNT	事务数。
INNODB_DATA_READ	读取数据量，单位为Byte。
INNODB_IBUF_SIZE	合并记录页数。
INNODB_LOG_WAITS	Log写入等待次数。
INNODB_MAX_PURGE	清除事务数。
INNODB_N_WAITING	锁等待数。
INNODB_ROWS_READ	读取数据行数。
INNODB_LOG_WRITES	日志写次数。
INNODB_IBUF_MERGES	合并次数。
INNODB_DATA_WRITTEN	写入数据量，单位为Byte。
INNODB_DBLWR_WRITES	双写操作写入次数。
INNODB_IBUF_SEGSIZE	当前插入缓冲大小。
INNODB_ROWS_DELETED	删除数据行数。
INNODB_ROWS_UPDATED	更新数据行数。

字段	说明
INNODB_COMMIT_TRXCNT	提交事务数。
INNODB_IBUF_FREELIST	空闲列表长度。
INNODB_MYSQL_TRX_CNT	MySQL事务数。
INNODB_ROWS_INSERTED	插入数据行数。
INNODB_ACTIVE_TRX_CNT	活跃事务数。
INNODB_OS_LOG_WRITTEN	日志文件写入量，单位为Byte。
INNODB_ACTIVE_VIEW_CNT	活跃视图数。
INNODB_RSEG_HISTORY_LEN	TRX_RSEG_HISTORY表长度。
INNODB_AVG_COMMIT_TRXTIME	平均事务提交时间。
INNODB_MAX_COMMIT_TRXTIME	最长事务提交时间。
INNODB_DBLWR_PAGES_WRITTEN	双写操作完成写入次数。

使用方法

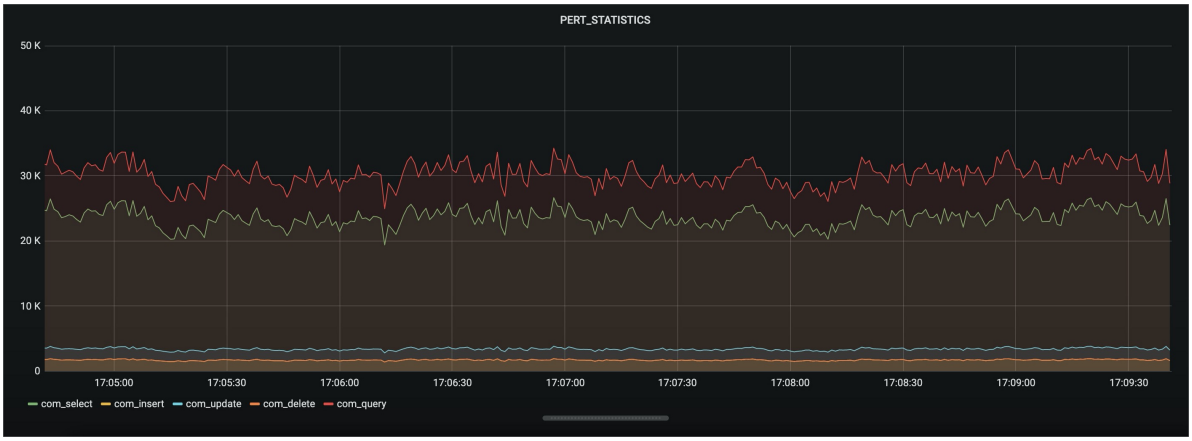
- 直接查询系统表，获取性能数据。 您可以参见如下示例：
  - 查询最近30秒的CPU和内存使用情况，示例如下：

```
MySQL> select TIME, PROCS_MEM_USAGE, PROCS_CPU_RATIO from information_schema.PERF_STATI
STICS order by time DESC limit 30;
+---------------------+-----------------+-----------------+
| TIME                | PROCS_MEM_USAGE | PROCS_CPU_RATIO |
+---------------------+-----------------+-----------------+
| 2020-02-27 11:15:36 |       857812992 |           18.55 |
| 2020-02-27 11:15:35 |       857808896 |           18.54 |
| 2020-02-27 11:15:34 |       857268224 |           19.64 |
| 2020-02-27 11:15:33 |       857268224 |           21.06 |
| 2020-02-27 11:15:32 |       857264128 |           20.39 |
| 2020-02-27 11:15:31 |       857272320 |           20.32 |
| 2020-02-27 11:15:30 |       857272320 |           21.35 |
| 2020-02-27 11:15:29 |       857272320 |            28.8 |
| 2020-02-27 11:15:28 |       857268224 |           29.08 |
| 2020-02-27 11:15:27 |       857268224 |           26.92 |
| 2020-02-27 11:15:26 |       857268224 |           23.84 |
| 2020-02-27 11:15:25 |       857264128 |           13.76 |
| 2020-02-27 11:15:24 |       857264128 |           15.12 |
| 2020-02-27 11:15:23 |       857264128 |           14.76 |
| 2020-02-27 11:15:22 |       857264128 |           15.38 |
| 2020-02-27 11:15:21 |       857260032 |           13.23 |
| 2020-02-27 11:15:20 |       857260032 |           12.75 |
| 2020-02-27 11:15:19 |       857260032 |           12.17 |
| 2020-02-27 11:15:18 |       857255936 |           13.22 |
| 2020-02-27 11:15:17 |       857255936 |           20.51 |
| 2020-02-27 11:15:16 |       857255936 |           28.74 |
| 2020-02-27 11:15:15 |       857251840 |           29.85 |
| 2020-02-27 11:15:14 |       857251840 |           29.31 |
| 2020-02-27 11:15:13 |       856981504 |           28.85 |
| 2020-02-27 11:15:12 |       856981504 |           29.19 |
| 2020-02-27 11:15:11 |       856977408 |           29.12 |
| 2020-02-27 11:15:10 |       856977408 |           29.32 |
| 2020-02-27 11:15:09 |       856977408 |            29.2 |
| 2020-02-27 11:15:08 |       856973312 |           29.36 |
| 2020-02-27 11:15:07 |       856973312 |           28.79 |
+---------------------+-----------------+-----------------+
30 rows in set (0.08 sec)
```

- 查询最近30秒的InnoDB读写的数据行数，示例如下：

```
MySQL> select TIME, EXTRA->'$.INNODB_ROWS_READ', EXTRA->'$.INNODB_ROWS_INSERTED' from i
nformation_schema.PERF_STATISTICS order by time DESC limit 30;
+---------------------+-----------------------------+--------------------------------+
| TIME                | EXTRA->'$.INNODB_ROWS_READ' | EXTRA->'$.INNODB_ROWS_INSERTED' |
+---------------------+-----------------------------+--------------------------------+
| 2020-02-27 11:22:17 | 39209                       | 0                              |
| 2020-02-27 11:22:16 | 36098                       | 0                              |
| 2020-02-27 11:22:15 | 38035                       | 0                              |
| 2020-02-27 11:22:14 | 37384                       | 0                              |
| 2020-02-27 11:22:13 | 38336                       | 0                              |
| 2020-02-27 11:22:12 | 33946                       | 0                              |
| 2020-02-27 11:22:11 | 36301                       | 0                              |
| 2020-02-27 11:22:10 | 36835                       | 0                              |
| 2020-02-27 11:22:09 | 36900                       | 0                              |
| 2020-02-27 11:22:08 | 36402                       | 0                              |
| 2020-02-27 11:22:07 | 39672                       | 0                              |
| 2020-02-27 11:22:06 | 39316                       | 0                              |
| 2020-02-27 11:22:05 | 37830                       | 0                              |
| 2020-02-27 11:22:04 | 36396                       | 0                              |
| 2020-02-27 11:22:03 | 34820                       | 0                              |
| 2020-02-27 11:22:02 | 37350                       | 0                              |
| 2020-02-27 11:22:01 | 39463                       | 0                              |
| 2020-02-27 11:22:00 | 38419                       | 0                              |
| 2020-02-27 11:21:59 | 37673                       | 0                              |
| 2020-02-27 11:21:58 | 35117                       | 0                              |
| 2020-02-27 11:21:57 | 36140                       | 0                              |
| 2020-02-27 11:21:56 | 37592                       | 0                              |
| 2020-02-27 11:21:55 | 39765                       | 0                              |
| 2020-02-27 11:21:54 | 35553                       | 0                              |
| 2020-02-27 11:21:53 | 35882                       | 0                              |
| 2020-02-27 11:21:52 | 37061                       | 0                              |
| 2020-02-27 11:21:51 | 40699                       | 0                              |
| 2020-02-27 11:21:50 | 39608                       | 0                              |
| 2020-02-27 11:21:49 | 39317                       | 0                              |
| 2020-02-27 11:21:48 | 37413                       | 0                              |
+---------------------+-----------------------------+--------------------------------+
30 rows in set (0.08 sec)
```

- 对接性能监控平台，实现实时监控。例如使用Grafana。



6.4. Purge Large File Asynchronously

AliSQL支持通过异步删除大文件的方式保证系统稳定性。

背景信息

使用InnoDB引擎时，直接删除大文件会导致POSIX文件系统出现严重的稳定性问题，因此InnoDB会启动一个后台线程来异步清理数据文件。当删除单个表空间时，会将对应的数据文件先重命名为临时文件，然后清除线程将异步、缓慢地清理文件。

> **说明** AliSQL提供清除文件日志来保证DDL语句的原子性。

使用方法

1. 使用如下命令查看实例全局变量设置：

```
SHOW GLOBAL VARIABLES LIKE '%data_file_purge%';
```

返回结果如下：

```
+----------------------------------------+-------+
| Variable_name                          | Value |
+----------------------------------------+-------+
| innodb_data_file_purge                 | ON    |
| innodb_data_file_purge_all_at_shutdown | OFF   |
| innodb_data_file_purge_dir             |       |
| innodb_data_file_purge_immediate       | OFF   |
| innodb_data_file_purge_interval        | 100   |
| innodb_data_file_purge_max_size        | 128   |
| innodb_print_data_file_purge_process   | OFF   |
+----------------------------------------+-------+
```

参数说明如下：

参数	说明
innodb_data_file_purge	是否启用异步清除策略。

参数	说明
innodb_data_file_purge_all_at_shutdown	正常关机时全部清理。
innodb_data_file_purge_dir	临时文件目录。
innodb_data_file_purge_immediate	取消数据文件的链接但不清理。
innodb_data_file_purge_interval	清理时间间隔。单位：ms。
innodb_data_file_purge_max_size	每次清理单个文件大小的最大值。单位：MB。
innodb_print_data_file_purge_process	是否打印文件清理工作进程。

说明 建议使用如下命令进行设置：

```
set global INNODB_DATA_FILE_PURGE = on;
set global INNODB_DATA_FILE_PURGE_INTERVAL = 100;
set global INNODB_DATA_FILE_PURGE_MAX_SIZE = 128;
```

2. 使用如下命令查看清理进度：

```
select * from information_schema.innodb_purge_files;
```

返回结果如下：

```
+--------+---------------------+--------------------+---------------+------------------
-------+--------------+
| log_id | start_time          | original_path      | original_size | temporary_path
| current_size |
+--------+---------------------+--------------------+---------------+------------------
-------+--------------+
|      0 | 2021-05-14 14:40:01 | ./file_purge/t.ibd |     146800640 | ./#FP_210514 14:4
0:01_9 |     79691776 |
+--------+---------------------+--------------------+---------------+------------------
-------+--------------+
```

参数说明如下：

参数	说明
start_time	清理操作的开始时间。
original_path	表数据文件的原始路径。
original_size	表数据文件的原始大小，单位：byte。

参数	说明
temporary_path	清理中的临时文件路径。
current_size	待清理的剩余临时文件大小，单位：byte。

6.5. Performance Insight

Performance Insight是专注于实例负载监控、关联分析、性能调优的利器，帮助您迅速评估数据库负载，找到性能问题的源头，提升数据库的稳定性。

前提条件

- 实例版本如下：
  - MySQL 8.0
  - MySQL 5.7
- 内核小版本需要为20190915或以上。

> **说明** 您可以在**基本信息**页面的**配置信息**区域查看是否有**升级内核小版本**按钮。如果有按钮，您可以单击按钮查看当前版本；如果没有按钮，表示已经是最新版。详情请参见升级内核小版本。

Performance Insight介绍

Performance Insight由如下两部分组成：

- Object statistics

  Object statistics查询表和索引的统计信息，包括如下两个表：
  - TABLE_STATISTICS：记录读取和修改的行。
  - INDEX_STATISTICS：记录索引的读取行。
- Performance point

  Performance point提供实例的详细性能信息，方便您更快更准确地量化SQL的开销。Performance point包括如下三个维度：
  - CPU：包括执行任务的总时间（Elapsed time）、CPU执行任务的时间（CPU time）等。
  - LOCK：包括服务器MDL锁时间、存储事务锁时间、互斥冲突（仅调试模式）、读写锁冲突等。
  - IO：数据文件读写时间、日志文件写入时间、逻辑读取、物理读取、物理异步读取等。

Object statistics使用方法

1. 确认参数OPT_TABLESTAT和OPT_INDEXSTAT的值为ON。示例如下：

```
mysql> show variables like "opt_%_stat";
  +---------------+-------+
  | Variable_name | Value |
  +---------------+-------+
  | opt_indexstat | ON    |
  | opt_tablestat | ON    |
  +---------------+-------+
```

> **说明** 如果参数找不到或参数值不为ON，请确认您的实例版本是否为MySQL 5.7。

2. 在information_schema数据库查询TABLE_STATISTICS表或INDEX_STATISTICS表，查看表和索引的统计信息。示例如下：

```
mysql> select * from TABLE_STATISTICS limit 10;
  +--------------+--------------+-----------+--------------+------------------------+--
------------+--------------+--------------+
  | TABLE_SCHEMA | TABLE_NAME   | ROWS_READ | ROWS_CHANGED | ROWS_CHANGED_X_INDEXES | R
OWS_INSERTED | ROWS_DELETED | ROWS_UPDATED |
  +--------------+--------------+-----------+--------------+------------------------+--
------------+--------------+--------------+
  | mysql        | db           |         2 |            0 |                      0 |
0 |            0 |            0 |
  | mysql        | engine_cost  |         2 |            0 |                      0 |
0 |            0 |            0 |
  | mysql        | proxies_priv |         1 |            0 |                      0 |
0 |            0 |            0 |
  | mysql        | server_cost  |         6 |            0 |                      0 |
0 |            0 |            0 |
  | mysql        | tables_priv  |         2 |            0 |                      0 |
0 |            0 |            0 |
  | mysql        | user         |         7 |            0 |                      0 |
0 |            0 |            0 |
  | test         | sbtest1      |      1686 |          142 |                    184 |
112 |           12 |           18 |
  | test         | sbtest10     |      1806 |          125 |                    150 |
105 |            5 |           15 |
  | test         | sbtest100    |      1623 |          141 |                    182 |
110 |           10 |           21 |
  | test         | sbtest11     |      1254 |          136 |                    172 |
110 |           10 |           16 |
  +--------------+--------------+-----------+--------------+------------------------+--
------------+--------------+--------------+
  mysql> select * from INDEX_STATISTICS limit 10;
  +--------------+--------------+------------+-----------+
  | TABLE_SCHEMA | TABLE_NAME   | INDEX_NAME | ROWS_READ |
  +--------------+--------------+------------+-----------+
  | mysql        | db           | PRIMARY    |         2 |
  | mysql        | engine_cost  | PRIMARY    |         2 |
  | mysql        | proxies_priv | PRIMARY    |         1 |
  | mysql        | server_cost  | PRIMARY    |         6 |
  | mysql        | tables_priv  | PRIMARY    |         2 |
  | mysql        | user         | PRIMARY    |         7 |
  | test         | sbtest1      | PRIMARY    |      2500 |
  | test         | sbtest10     | PRIMARY    |      3007 |
  | test         | sbtest100    | PRIMARY    |      2642 |
  | test         | sbtest11     | PRIMARY    |      2091 |
  +--------------+--------------+------------+-----------+
```

参数说明如下。

参数	说明
TABLE_SCHEMA	数据库名称。
TABLE_NAME	表名称。
ROWS_READ	读的行数。
ROWS_CHANGED	修改的行数。
ROWS_CHANGED_X_INDEXES	索引修改的行数。
ROWS_INSERTED	插入的行数。
ROWS_DELETED	删除的行数。
ROWS_UPDATED	更新的行数。
INDEX_NAME	索引名称。

Performance point使用方法

1. 确认Performance point的相关参数。正常的示例如下：

```
mysql> show variables like "%performance_point%";
  +----------------------------------------+-------+
  | Variable_name                          | Value |
  +----------------------------------------+-------+
  | performance_point_dbug_enabled         | OFF   |
  | performance_point_enabled              | ON    |
  | performance_point_iostat_interval      | 2     |
  | performance_point_iostat_volume_size   | 10000 |
  | performance_point_lock_rwlock_enabled | ON    |
  +----------------------------------------+-------+
```

> **说明** 如果参数找不到，请确认您的实例版本是否为MySQL 5.7。

2. 在performance_schema数据库查询events_statements_summary_by_digest_supplement表，查看排名前10的SQL语句。示例如下：

```
mysql> select * from events_statements_summary_by_digest_supplement limit 10;
  +--------------------+----------------------------------+----------------------------
--------------+--------------+
  | SCHEMA_NAME        | DIGEST                           | DIGEST_TEXT
| ELAPSED_TIME | ......
  +--------------------+----------------------------------+----------------------------
--------------+--------------+
  | NULL               | 6b787dd1f9c6f6c5033120760a1a82de | SELECT @@`version_comment`
LIMIT ?        |          932 |
  | NULL               | 2fb4341654df6995113d998c52e5abc9 | SHOW SCHEMAS
|         2363 |
  | NULL               | 8a93e76a7846384621567fb4daa1bf95 | SHOW VARIABLES LIKE ?
|        17933 |
  | NULL               | dd148234ac7a20cb5aee7720fb44b7ea | SELECT SCHEMA ( )
|         1006 |
  | information_schema | 2fb4341654df6995113d998c52e5abc9 | SHOW SCHEMAS
|         2156 |
  | information_schema | 74af182f3a2bd265678d3dadb53e08da | SHOW TABLES
|         3161 |
  | information_schema | d3a66515192fcb100aaef6f8b6e45603 | SELECT * FROM `TABLE_STATIS
TICS` LIMIT ?  |         2081 |
  | information_schema | b3726b7c4c4db4b309de2dbc45ff52af | SELECT * FROM `INDEX_STATIS
TICS` LIMIT ?  |         2384 |
  | information_schema | dd148234ac7a20cb5aee7720fb44b7ea | SELECT SCHEMA ( )
|          129 |
  | test               | 2fb4341654df6995113d998c52e5abc9 | SHOW SCHEMAS
|          342 |
  +--------------------+----------------------------------+----------------------------
--------------+--------------+
```

参数说明如下。

参数	说明
SCHEMA_NAME	数据库名称。
DIGEST	Digest_text进行hash计算得到的64字节的hash字符串。
DIGEST_TEXT	SQL语句的特征。
ELAPSED_TIME	实际运行时间。单位：μs。
CPU_TIME	CPU运行时间。单位：μs。
SERVER_LOCK_TIME	服务器锁定时间。单位：μs。
TRANSACTION_LOCK_TIME	存储事务锁定时间。单位：μs。
MUTEX_SPINS	互斥旋转次数。
MUTEX_WAITS	互斥等待次数。
RWLOCK_SPIN_WAITS	读写闩锁的自旋等待数。

参数	说明
RWLOCK_SPIN_ROUNDS	读写闩锁的旋转循环圈数。
RWLOCK_OS_WAITS	读写闩锁的操作系统等待数。
DATA_READS	数据文件读取次数。
DATA_READ_TIME	数据文件读取时间。单位：μs。
DATA_WRITES	数据文件写入次数。
DATA_WRITE_TIME	数据文件写入时间。单位：μs。
REDO_WRITES	日志文件写入次数。
REDO_WRITE_TIME	日志文件写入时间。单位：μs。
LOGICAL_READS	逻辑页读取次数。
PHYSICAL_READS	物理页读取次数。
PHYSICAL_ASYNC_READS	物理异步页读取次数。

3. 在information_schema数据库查询IO_STATISTICS表，查看最近的数据读写情况。示例如下：

```
mysql> select * from IO_STATISTICS limit 10;
  +---------------------+-----------+----------------+
  | TIME                | DATA_READ | DATA_READ_TIME | ......
  +---------------------+-----------+----------------+
  | 2019-08-08 09:56:53 |        73 |            983 |
  | 2019-08-08 09:56:57 |         0 |              0 |
  | 2019-08-08 09:59:17 |         0 |              0 |
  | 2019-08-08 10:00:55 |      4072 |          40628 |
  | 2019-08-08 10:00:59 |         0 |              0 |
  | 2019-08-08 10:01:09 |       562 |           5800 |
  | 2019-08-08 10:01:11 |       606 |           6910 |
  | 2019-08-08 10:01:13 |       609 |           6875 |
  | 2019-08-08 10:01:15 |       625 |           7077 |
  | 2019-08-08 10:01:17 |       616 |           5800 |
  +---------------------+-----------+----------------+
```

参数说明如下。

参数	说明
TIME	日期。
DATA_READ	数据读取次数。
DATA_READ_TIME	数据读取总时间。单位：μs。

参数	说明
DATA_READ_MAX_TIME	数据读取最长时间。单位：μs。
DATA_READ_BYTES	数据读取总大小。单位：byte。
DATA_WRITE	数据写入次数。
DATA_WRITE_TIME	数据写入总时间。单位：μs。
DATA_WRITE_MAX_TIME	数据写入最长时间。单位：μs。
DATA_WRITE_BYTES	数据写入总大小。单位：byte。

7.安全

7.1. Recycle Bin

由于DDL语句无法回滚，开发或运维人员如果误操作（例如DROP TABLE）可能会导致数据丢失。阿里云支持回收站（Recycle Bin）功能，临时将删除的表转移到回收站，还可以设置保留的时间，方便您找回数据，同时提供了工具包（DBMS_RECYCLE）便于您快捷使用。

前提条件

实例版本如下：

- RDS MySQL 8.0
- RDS MySQL 5.7

Recycle Bin参数

Recycle Bin设计了如下五个参数。

参数	说明
loose_recycle_bin	是否打开回收站功能，包括session级别和global级别。您可以在控制台修改参数。默认值：OFF。
loose_recycle_bin_retention	回收站保留时间，单位：秒。默认为604800，即一周。您可以在控制台修改参数。
loose_recycle_scheduler	是否打开回收站的异步清理任务线程。您可以在控制台修改参数。默认值：OFF。
loose_recycle_scheduler_interval	回收站异步清理任务线程的轮询间隔，单位：秒。默认为30。暂不开放。
loose_recycle_scheduler_purge_table_print	是否打印异步清理现场工作的详细日志。暂不开放。

> **说明** 为了防止磁盘空间被占满，建议合理设置保留时间，并打开后台清理任务线程。

Recycle Bin介绍

- 回收/清理机制

- 回收机制

  执行 `TRUNCATE TABLE` 语句时，将原始表移动到专门的recycle bin目录中，并在原位置使用相同的结构创建新表。

  > **说明** 仅RDS MySQL 8.0支持。

  执行 `DROP TABLE/DATABASE` 语句时，只保留相关的表对象，并移动到专门的recycle bin目录中。其它对象的删除策略如下：

  - 如果是与表无关的对象，根据操作语句决定是否保留，不做回收。
  - 如果是表的附属对象，可能会修改表数据的，做删除处理，例如Trigger和Foreign key。但Column statistics不做清理，随表进入回收站。

- 清理机制

  回收站会启动一个后台线程，来异步清理超过**recycle_bin_retention**时间的表对象。在清理回收站表的时候，如果遇到大表，会再启动一个后台线程异步删除大表。

- 权限

  RDS MySQL实例启动时，会初始化一个名为__recycle_bin__的数据库，作为回收站使用的专有数据库。__recycle_bin__是系统级数据库，您无法直接进行修改和删除。

  对于回收站内的表，虽然您无法直接执行 `drop table` 语句，但是可以使用 `call dbms_recycle.purge_table('<TABLE>');` 进行清理。

  > **说明** 账号在原表和回收站表都需要具有DROP权限。

- 回收站表命名规则

  Recycle Bin会从不同的数据库回收到统一的__recycle_bin__数据库中，所以需要保证目标表表名唯一，所以定义了如下命名格式：

  ```
  "__" + <Storage Engine> + <SE private id>
  ```

  参数说明如下。

参数	说明
Storage Engine	存储引擎名称。
SE private id	存储引擎为每一个表生成的唯一值。例如在InnoDB引擎中就是table id。

- 独立回收

  回收的设置只会影响该实例本身，不会影响到binlog复制到的节点（备实例、只读实例和灾备实例）上。例如我们可以在主实例上设置回收，保留7天；在备实例上设置回收，保留14天。

  > **说明** 回收站保留周期不同，将导致实例的空间占用差别比较大。

注意事项

- 如果回收站数据库和待回收的表跨了文件系统，执行 `drop table` 语句将会搬迁表空间文件，耗时较长。

- 如果Tablespace为General，可能会存在多个表共享同一个表空间的情况，当回收其中一张表的时候，不会搬迁相关的表空间文件。

管理Recycle Bin

AliSQL在DBMS_RECYCLE中定义了两个管理接口。详细说明如下：

- show_tables

  展示回收站中所有临时保存的表。命令如下：

  ```
  call dbms_recycle.show_tables();
  ```

  示例：

  ```
  mysql> call dbms_recycle.show_tables();
  +-----------------+---------------+--------------+-------------+---------------------+-
  --------------------+
  | SCHEMA          | TABLE         | ORIGIN_SCHEMA | ORIGIN_TABLE | RECYCLED_TIME       |
  PURGE_TIME          |
  +-----------------+---------------+--------------+-------------+---------------------+-
  --------------------+
  | __recycle_bin__ | __innodb_1063 | product_db    | t1           | 2019-08-08 11:01:46 |
  2019-08-15 11:01:46 |
  | __recycle_bin__ | __innodb_1064 | product_db    | t2           | 2019-08-08 11:01:46 |
  2019-08-15 11:01:46 |
  | __recycle_bin__ | __innodb_1065 | product_db    | parent       | 2019-08-08 11:01:46 |
  2019-08-15 11:01:46 |
  | __recycle_bin__ | __innodb_1066 | product_db    | child        | 2019-08-08 11:01:46 |
  2019-08-15 11:01:46 |
  +-----------------+---------------+--------------+-------------+---------------------+-
  --------------------+
  4 rows in set (0.00 sec)
  ```

参数	说明
SCHEMA	回收站的数据库名。
TABLE	进入回收站后的表名。
ORIGIN_SCHEMA	原数据库名。
ORIGIN_TABLE	原表名。
RECYCLED_TIME	回收时间。
PURGE_TIME	预计从回收站删除的时间。

- purge_table

  手动清理回收站中的表。命令如下：

  ```
  call dbms_recycle.purge_table('<TABLE>');
  ```

> **说明**
> - TABLE为进入回收站后的表名。
> - 账号在原表和回收站表都需要具有DROP权限。

示例：

```
call dbms_recycle.purge_table('__innodb_1063');
```

- restore_table

  恢复回收站内的表。命令如下：

  ```
  call dbms_recycle.restore_table('<RECYCLE_TABLE>','<DEST_DB>','<DEST_TABLE>');
  ```

  参数说明如下。

参数	说明
RECYCLE_TABLE	需要恢复的回收站内的表名。 **说明** 如果仅传入此参数，会恢复到原始表。
DEST_DB	目标数据库名。
DEST_TABLE	目标表名。

> **说明** 由于restore_table命令需要SUPER权限，因此暂不支持手动执行，后续会开放恢复入口。当前如需执行该命令，请提交工单。

示例：

```
mysql> call dbms_recycle.restore_table('__innodb_1063','testDB','testTable');
```

8.最佳实践

8.1. 将PolarDB-X中的InnoDB引擎转换为X-Engine引擎

本文为您介绍如何将PolarDB-X原本的InnoDB引擎转换为X-Engine引擎。

背景信息

目前有较多RDS存量用户想要使用X-Engine，这些用户使用数据库有如下特点：

- 实例版本大多为MySQL 5.6或5.7，MySQL 8.0较少。
- 单实例数据量比较大，达到实例规格所能支持的磁盘空间上限（例如4核8GB的规格最大支持2TB的本地盘）。
- 同时使用PolarDB-X，特别是部分客户使用的是较老版本的PolarDB-X，或者做过一些定制功能（例如SQL透传）。

针对用户的需求，阿里云提供PolarDB-X的转换方案，如果您的业务也需要将PolarDB-X中的InnoDB转换为X-Engine，可以参见本文进行操作。

> **说明** X-Engine的详细介绍请参见X-Engine简介。

切换方案

由于MySQL 8.0使用InnoDB引擎或X-Engine引擎时，在对外接口和使用体验上是相同的，因此在PolarDB-X升级完之后，可以对底层RDS实例进行逐步转换，例如PolarDB-X下面挂载了8个RDS实例，可以先将其中1个RDS实例的引擎转换为X-Engine，运行一段时间，确认没有兼容性和性能问题之后，再转换剩下的7个实例。

转换前验证空间压缩效率

在开始使用X-Engine之前，建议购买一个同规格的X-Engine引擎实例，并通过阿里云DTS服务导入原InnoDB引擎实例的数据，查看空间压缩效率。空间压缩的结果可以为以下两方面提供参考：

- 实例存储空间

  根据空间压缩效率，可以确定转换为X-Engine之后所需购买的实例规格， 例如如果空间占用压缩到原有的30%以下， 则原来购买3TB磁盘空间的实例可以转换为使用1TB磁盘空间的X-Engine实例，或者可以使用相同规格的实例，为未来业务发展留下更多存储空间。

- 数据库分片数

  存储空间降低后，可以考虑减少数据库分片数，例如将原来分布在多个实例上的数据库，合并到同一个实例上，大幅降低成本。

> **说明** X-Engine引擎实例验证完之后可以释放，也可以清空数据以供正式转换时使用。

转换流程

1. 升级PolarDB-X，确保版本号高于v5.4.2-15744202。
   - 查看PolarDB-X版本请参见版本说明。

- 升级PolarDB-X版本请参见升级版本。

> **说明**
> - 如果PolarDB-X版本高于v5.4.2-15744202，请跳过本步骤。
> - 如果业务对老版本特有接口有依赖（例如用于优化性能的SQL透传功能），需要对业务代码做少许修改，以确保兼容性。

2. 选定一个RDS分库实例（InnoDB引擎）作为首个切换的实例，将建表语句导出后修改引擎类型为X-Engine，然后创建所需规格的X-Engine引擎实例，或直接使用验证空间压缩效率时创建的实例，将表结构脚本导入新实例。
   - 创建实例请参见创建RDS MySQL实例。
   - 导出和导入建表语句请参见InnoDB/TokuDB/Myrocks引擎转换为X-Engine引擎方案二。

   > **说明** 通过DTS迁移数据时默认会继承源实例的引擎类型。所以需要先单独导出建表语句的SQL，修改Create语句中的引擎类型为X-Engine引擎. 然后再迁移数据到新建的X-Engine表中。

3. 通过DTS功能将RDS分库实例（InnoDB引擎）的数据同步至X-Engine引擎的RDS实例中，等待数据同步。数据同步请参见RDS MySQL实例间的双向同步。

   > **说明** 您可以使用DTS的双向同步功能保证两个实例的数据一致。

4. 修改PolarDB-X路由规则，将RDS分库实例（InnoDB引擎）上的访问切换到X-Engine引擎实例上。修改PolarDB-X路由规则请提交工单。

   > **说明** 维持首个X-Engine引擎实例在线上运行5天时间，持续监控实例，重点观察请求的处理耗时和异常信息等，同时注意双向同步的同步情况，确保在过程中出现问题时，可以切换回原有的RDS分库实例（InnoDB引擎）。查看引擎监控请参见查看监控信息。

5. 首个X-Engine引擎实例运行没有问题之后，可以切换剩下30%~50%的实例，然后继续观察3~5天。流程请参照之前的2~4步。

   > **说明** 原有RDS分库实例（InnoDB引擎）不要释放或下线，并且每一个RDS分库实例（InnoDB引擎）都需要使用DTS与对应X-Engine实例做双向同步。

6. 切换剩下的所有实例，整个集群切换完成之后继续观察3~5天，之后根据业务运行情况，可以释放所有的DTS同步链路，同时释放原有的RDS分库实例（InnoDB引擎）。

8.2. X-Engine如何支撑钉钉跃居AppStore第一

本文为您介绍RDS的X-Engine引擎如何在成本方面支撑钉钉业务，帮助企业快速实现在线协同办公。

背景信息

钉钉作为中国领先的企业IM工具，在中国有数以亿计用户，从钉钉项目群、钉钉视频通话、钉钉视频会议、钉钉日报等基础功能，再到钉钉平台上演化出来的各种办公室自动化（Office Automation）应用，方便了人与人之间的交流，可以帮助企业快速实现在线协同办公。

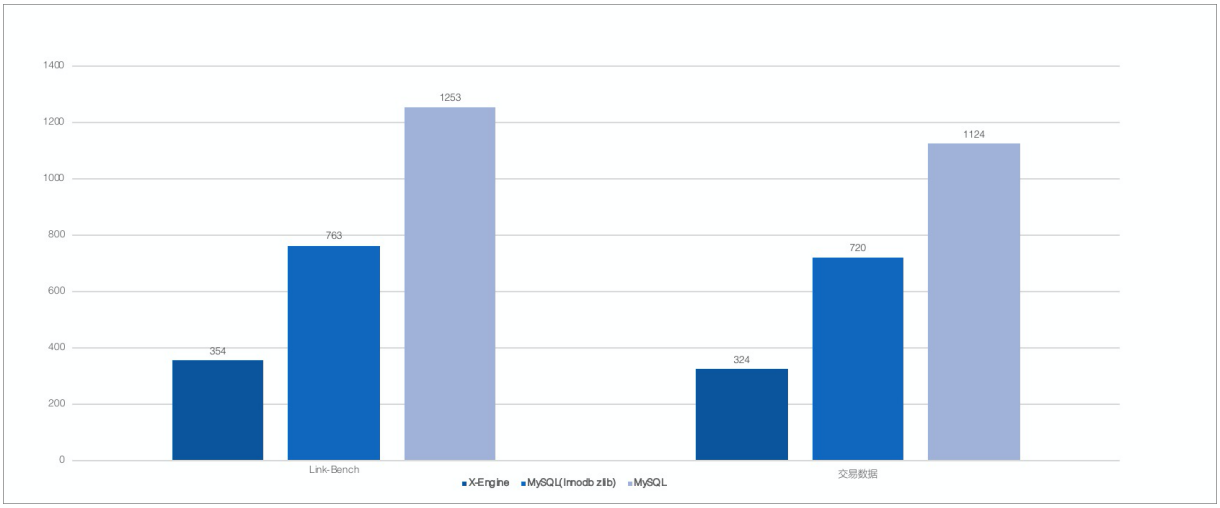
2020年新型冠状病毒肺炎疫情爆发，为了规避集中办公带来的感染风险，大量企业员工选择了在家办公，企业办公协同工具需求瞬间爆发。钉钉迅速冲上了AppStore下载榜单的第一位，导致钉钉访问流量迅速增长，借助于阿里云提供的弹性基础设施，钉钉平稳地渡过了每一次流量洪峰。

如此庞大用户量，钉钉的消息系统除了要保证消息及时正确传递，还要保证已读、未读等特有功能，而且不同于微信这样的用户级IM工具，企业IM需要实现聊天记录永久保存，并且提供多端漫游功能。由于用户量持续爆炸性增长，聊天记录永久保存给钉钉业务带来巨大成本压力，同时还要保证聊天记录的读写性能不会降低。

面对这些挑战，钉钉业务选用了X-Engine作为钉钉消息的最终存储引擎，实现了性能和成本平衡。使用X-Engine有如下优势：

- 存储空间比InnoDB引擎减少了约62%。
- 保留对事务以及二级索引等数据库特性的支持。
- 业务代码不需要修改就可以迁移到X-Engine引擎实例上。
- X-Engine的冷热分离能力能够对最新消息有着最快的处理速度，而对历史消息有着最高压缩比。

在Link-Bench和阿里巴巴内部交易业务两个数据集上测试了X-Engine的存储空间效率。在测试中，对比开启压缩的InnoDB引擎，X-Engine有着2倍空间优势；对比未开启压缩的InnoDB，X-Engine则有着3~5倍空间优势。



X-Engine如何实现低成本

X-Engine可以实现低成本是因为有以下几个特殊技术：

- 紧凑数据页格式

  X-Engine使用Copy-on-write技术，避免原地更新数据页，新数据会写入到新数据页中。由于既有数据不可更新，可以对只读数据页进行紧凑存储并使用前缀编码等方式进行数据压缩，提升页面空间使用效率。而已经失效的历史记录版本则由Compaction操作清理，保证有效记录都紧凑排列。相对于传统存储引擎（例如InnoDB），使用X-Engine可以将存储空间降低至10%~50%。

- 数据压缩及无效记录清理

  编码之后的数据页，可以使用通用压缩算法（zlib、zstd、snappy等）进行压缩，所有处在LSM-tree低层次的数据都会默认压缩。

  数据压缩是以计算资源换存储空间的技术，因此选用一个压缩率小及压缩/解压速度快的压缩算法也非常关键，经过大量对比测试，X-Engine默认选用ZSTD压缩算法，但同时也支持其他算法。

除了使用压缩之外，Compaction操作会对无效记录进行删除，只保留有效记录，Compaction执行越频繁则无效记录占比越低，空间使用效率越高，因此保证合适的Compaction频率也是提升空间使用效率的关键。

为了减少Compaction操作对计算资源的消耗，X-Engine团队研发了FPGA Compaction技术，使用异构计算硬件来加速Compaction过程，实现了在一个FPGA硬件流水线内同时完成Compaction和压缩操作。即使在没有FPGA硬件的主机上，借助合理调度算法，X-Engine也能以较小的性能代价节省存储空间。

- 智能冷热分离

  通常存储系统访问数据都有局部性，大量访问都集中在少部分数据上，这也是缓存系统能有效工作的基本前提。在LSM存储结构中，如果把访问频率高的数据尽可能放在较高层次上，存放在快速存储设备中（例如NVM、DRAM），而把访问频率低的数据放在较低层次中，存放在廉价慢速存储设备中，这就是X-Engine冷热分层概念。

  X-Engine中冷热分离算法主要完成如下几个任务：

  - 在Compaction操作中，挑选出未来最不可能被访问到的数据页和记录，移动到LSM-tree底层。
  - 挑选当前热点数据，在Compaction或者转储过程中回填到内存中（BlockCache/RowCache），避免缓存命中率抖动导致影响性能。
  - AI算法会识别出未来可能被访问到的数据，并提前预读到内存中，减少首次访问缓存未命中率。

  准确识别出数据冷热，可以避免无效压缩或解压带来的计算资源浪费，提升系统吞吐。

更多详细说明请参见X-Engine简介。

相关论文

- X-Engine: An Optimized Storage Engine for Large-scale E-commerce Transaction Processing
- FPGA-Accelerated Compactions for LSM-based Key-Value Store

8.3. 淘宝万亿级交易订单背后的存储引擎

基于X-Engine引擎的PolarDB-X集群支撑了淘宝历史订单数据库业务，解决了使用HBase数据库遗留的问题，降低存储成本的同时，满足了用户随时查询订单的需求。

背景信息

阿里巴巴旗下的淘宝是中国最大在线购物平台，活跃用户数量超过7亿人，商家数量也有数千万。

在如此体量巨大的平台上，每天的实物和虚拟商品交易达到亿级别。每次交易会涉及到会员信息验证、商品库信息查询、订单创建、库存扣减、优惠扣减、订单支付、物流信息更新和确认支付等，每个环节都涉及到数据库记录创建和状态更新，整个流程可能涉及到数百次数据库事务操作，整个数据库集群每天会执行数百亿次事务读写。数据库团队不仅要保证数据库系统性能稳定，还需要考虑每日递增海量数据带来的巨大存储成本压力。

交易订单是整个交易过程最为关键的信息，由于可能涉及到交易纠纷处理，需要随时提供用户查询，必须永久记录在数据库中。淘宝成立至今近17年，与订单相关的数据库记录总量达到了万亿级别，所占用磁盘空间也早已超过PB级。

下文将为您详细介绍淘宝是如何做到既满足用户随时查询订单的低延时需求，又控制存储成本。

架构演进历史

淘宝从2003年成立至今，近17年时间，随着流量不断增加，交易订单数据库架构也经历过数次演进：

- 第一阶段

淘宝起步阶段由于流量较小，使用Oracle数据库存储所有订单信息，订单创建和历史订单查询都在同一数据库进行。

- 第二阶段

  由于历史订单数据量越来越大，单一数据库已经不能同时满足性能和容量需求，于是对交易订单库进行拆分，分为在线库和历史库，将三个月之前的历史订单迁移进历史库，但是由于数据量巨大，不能满足查询需求，因此当时的用户只能查询三个月之内的历史订单信息。

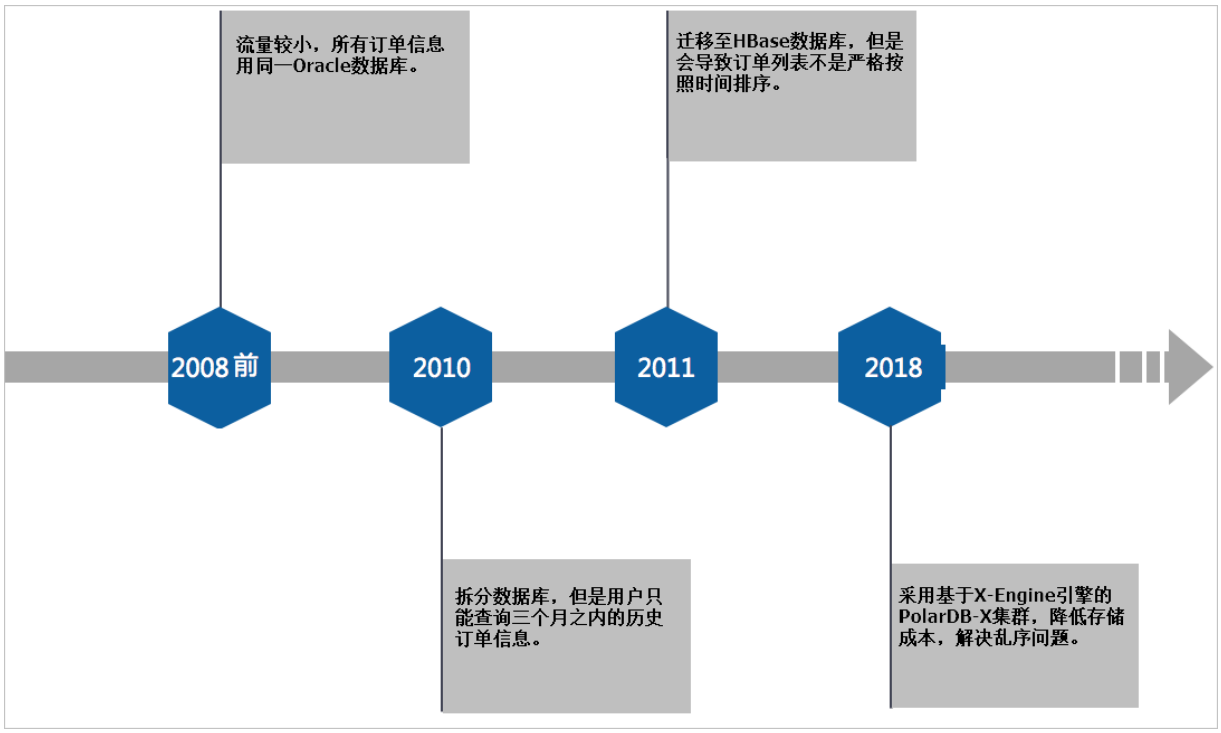
- 第三阶段

  为了解决存储成本和历史订单查询问题，淘宝将历史订单迁移到HBase数据库。

  整体方案是使用主表结合索引表，查询订单详细信息通过主表完成，通过买家或者卖家ID查询订单，则需要借助索引表先得到订单号。这个方案遗留一个问题，订单不一定按照时间顺序迁移到历史订单库，很多类型订单并不迁移到历史订单库，所以会导致订单列表不是严格按照时间排序，用户可能会发现自己的近期订单出现在不正确的位置。

- 第四阶段

  历史订单数据库采用基于X-Engine引擎的PolarDB-X集群，既降低了存储成本，也解决了乱序问题。



业务痛点

回顾淘宝交易订单数据库演进历史，自拆分出历史订单数据库，在后续十年时间里，业务团队和数据库团队一直在应对几个核心挑战：

- 存储成本

  由于每日写入数据量巨大且数据永不删除，必须保证存储成本极低。

- 查询功能

  提供丰富查询功能，例如按时间排序、按订单类型查找等，因此底层数据库需要支持二级索引，且二级索引需要保证一致性和性能。

- 查询延时

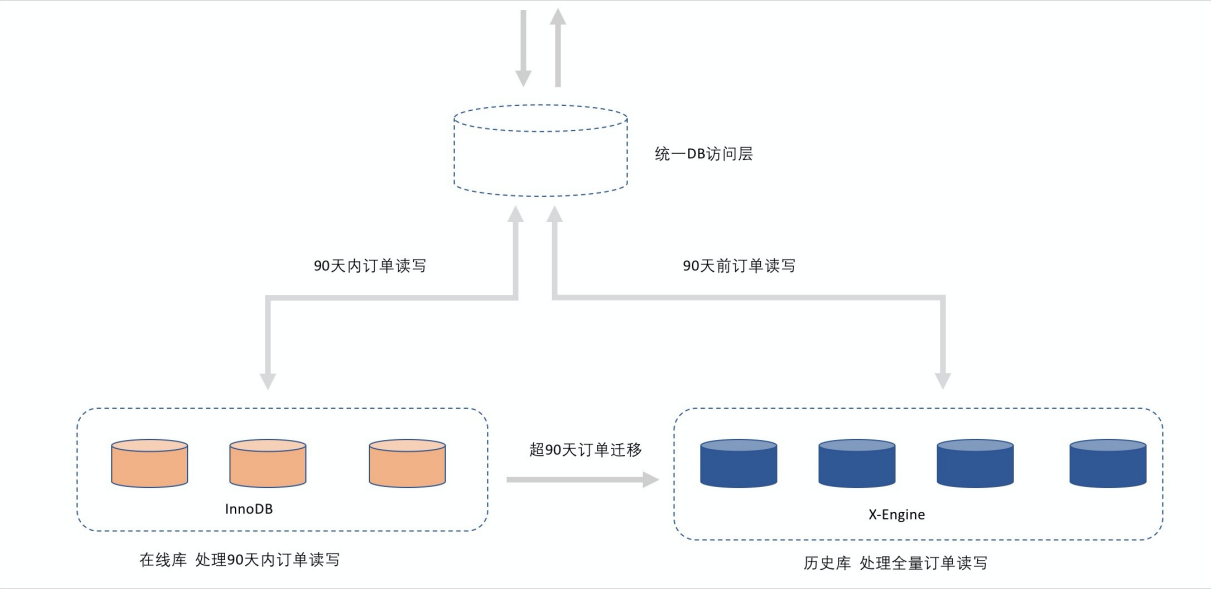
保证较低查询延时，不影响用户体验。虽然90天前的历史订单查询量比90天内少很多，但仍然需要保证延时在一定范围内。

基于X-Engine引擎的历史订单数据库方案

交易订单系统在在线库和历史库分离的架构下迭代了十年时间，很多业务代码对这套分离架构做了兼容，考虑到对业务代码改造以及迁移的风险，我们在初期延续了分离架构，只是将原有HBase集群替换成PolarDB-X集群（X-Engine引擎）。详细方案如下：

- 在线库依然沿用之前MySQL集群（InnoDB引擎），但是只保存最近90天订单，数据量少，可以保证较高的缓存命中率，确保读写延时。
- 通过数据同步将在线库中超过90天的订单迁移到历史库中，并从在线库中删除。
- 历史库存储引擎切换为X-Engine，保存超过90天的所有交易订单数据，超过90天的订单读写，直接操作历史库。

使用新方案后，历史库存储成本相比使用HBase时没有上升；历史库和在线库互相兼容，可以创建完全一样的索引，解决了排序异常问题；历史库的冷热分离保证了读取延时。



总结

淘宝交易订单记录流水型的访问特征是最近写入的记录会被大量访问，随着时间推移，记录访问频次急剧衰减。X-Engine引擎的冷热分离机制能很好地处理这种流水型业务，单一X-Engine数据库集群完全可以解决此类需求。

对于新开业务或者有大量流水型记录存储需求的现有业务，如果业务层面还未做冷热分离，建议您直接使用X-Engine引擎，基于X-Engine引擎的分布式数据库PolarDB-X可以同时解决扩展问题和存储成本问题。

目前X-Engine引擎已经在阿里云上线， 需要的用户可以购买使用。详情请参见创建RDS MySQL实例。

8.4. X-Engine测试最佳实践

本文介绍常用测试工具Sysbench如何对RDS MySQL的X-Engine引擎进行测试，帮助您准确评估X-Engine的性能。

前提条件

- 确认RDS MySQL实例默认存储引擎为X-Engine。

说明 如下所示，XENGINE的Support列值应为DEFAULT。

```
MySQL [(none)]> show storage engines;
+--------------------+---------+----------------------------------------------------------
----------+--------------+------+------------+
| Engine             | Support | Comment
| Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------
----------+--------------+------+------------+
| FEDERATED          | NO      | Federated MySQL storage engine
| NULL         | NULL | NULL       |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it d
isappears) | NO           | NO   | NO         |
| XENGINE            | DEFAULT | X-Engine storage engine
| YES          | YES  | YES        |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary t
ables      | NO           | NO   | NO         |
| InnoDB             | YES     | Supports transactions, row-level locking, and foreig
n keys     | YES          | YES  | YES        |
| PERFORMANCE_SCHEMA | YES     | Performance Schema
| NO           | NO   | NO         |
| Sequence           | YES     | Sequence Storage Engine Helper
| NO           | NO   | NO         |
| MyISAM             | YES     | MyISAM storage engine
| NO           | NO   | NO         |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables
| NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine
| NO           | NO   | NO         |
| ARCHIVE            | YES     | Archive storage engine
| NO           | NO   | NO         |
+--------------------+---------+----------------------------------------------------------
----------+--------------+------+------------+
11 rows in set (0.00 sec)
```

- 确认测试用表存储在X-Engine中。

说明 如下所示，ENGINE=XENGINE。如果建表语句显式注明了ENGINE=INNODB或其他存储引擎，所建的表将不使用X-Engine。

```
MySQL [sbtest]> show create table sbtest1;
+---------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------+
| Table   | Create Table
|
+---------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------+
| sbtest1 | CREATE TABLE `sbtest1` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `k` int(11) NOT NULL DEFAULT '0',
  `c` char(120) COLLATE utf8mb4_general_ci NOT NULL DEFAULT '',
  `pad` char(60) COLLATE utf8mb4_general_ci NOT NULL DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `k_1` (`k`)
) ENGINE=XENGINE AUTO_INCREMENT=2001 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_
ci |
+---------+-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
------------------------------+
1 row in set (0.01 sec)
```

说明 建议您使用1.1.0或更高版本的Sysbench。

使用DTS进行存储空间测试

我们推荐您使用阿里云数据迁移服务DTS将您的真实数据库数据迁移至X-Engine引擎的RDS MySQL实例，然后查看X-Engine的磁盘空间占用情况，这样的测试结果更符合您的实际业务情况。由于X-Engine同时应用了空间友好的存储格式、前缀编码、分层存储、高效压缩算法等多种技术来降低磁盘空间占用，这些技术的实际效果与您数据库中的库表结构、记录长度等因素相关，所以使用您的真实数据进行测试，结果更为准确。

因为DTS不支持在迁移数据的过程中自动切换存储引擎，您需要在使用X-Engine引擎的RDS MySQL实例上手动创建数据库和表，并按前提条件所述确认建表语句指明使用X-Engine存储引擎（ENGINE=XENGINE），然后使用DTS仅迁移数据，不迁移库表结构。

建议您在完成数据导入后，暂勿执行SQL，并通过RDS控制台持续观测实例的CPU和IOPS利用率情况，待CPU和IOPS基本归零后，此时的空间占用数据较为准确。这是因为X-Engine所采用的LSM-tree数据结构依赖后台异步任务来完成数据压缩等降低存储成本的功能，后台任务的执行需要一些时间，会使用一些CPU和IOPS资源。

详细迁移步骤请参见RDS实例间数据迁移。

使用Sysbench进行存储空间测试

为充分测试X-Engine的空间压缩效率，建议将Sysbench测试命令中的table_size参数设为一个较大的数（在MySQL实例磁盘大小允许范围内）。

与使用DTS测试相同，我们建议您在完成数据导入后持续观测实例的CPU和IOPS利用率情况，待CPU和IOPS基本归零后，此时的空间占用数据较为准确。

Sysbench测试命令：

```
sysbench /usr/share/sysbench/oltp_update_index.lua \
  --mysql-host=[RDS实例连接串] \
  --mysql-user=sbtest \
  --mysql-password=sbtest@888 \
  --mysql-db=sbtest \
  --threads=32 \
  --tables=32 \
  --table_size=1000000000 \
  --mysql-storage-engine=XENGINE \
  prepare
```

使用Sysbench进行性能测试

在使用Sysbench进行性能测试时，我们建议您将rand-type设为zipfian，将rand-zipfian-exp设为0.9。

- rand-type为SQL语句填充的随机数的分布。
- zipfian分布是一种常见的带热点倾斜的数据分布，当rand-zipfian-exp为0.9时，使用zipfian分布所生成的随机数更接近真实世界中常见的数据分布，此时测试结果相比使用默认的uniform分布而言更具参考价值。

建议您单次测试进行更长的时间，如3600秒。长时间测试有利于获得更具参考价值的平均性能，降低潜在的干扰因素对性能数据的影响。

建议使用较多的线程数来测试吞吐能力，如512个线程。

如您希望通过对X-Engine进行参数调节来获得更高的性能，请与您的客户经理、售后工程师等阿里云服务人员取得联系，或通过阿里云官网提交工单，我们将为您提供咨询服务。

Sysbench测试命令：

```
sysbench /usr/share/sysbench/oltp_point_select.lua \
  --mysql-host=[RDS实例连接串] \
  --mysql-user=sbtest \
  --mysql-password=sbtest@888 \
  --time=3600 \
  --mysql-db=sbtest \
  --tables=32 \
  --threads=512 \
  --table_size=10000000 \
  --rand-type=zipfian \
  --rand-zipfian-exp=0.9 \
  --report-interval=1 \
  run
```

使用Python脚本批量执行测试

如需要批量进行多个Sysbench测试，推荐使用Python脚本自动执行并记录结果。示例（被测数据库实例连接地址为输入）如下：

```
import subprocess
import time
import sys
def execute_test(test_name, db_conn_string):
  # setup sysbench parameters
  mysql = "--mysql-host=%s" % db_conn_string
  user = "--mysql-user=sbtest"
  password = "--mysql-password=*********"
  time = "--time=3600"
  database = "--mysql-db=sbtest"
  tables = "--tables=32"
  threads = "--threads=512"
  table_size = "--table_size=1000000"
  distribution = "--rand-type=pareto --rand-pareto-h=0.9"
  # formulate the sysbench command
  cmd = 'sysbench ' + test_name + " " + mysql + " " + user+ " " + password + " " + time+ "
" + database+ " " + tables + " " + threads + " " + table_size+ " " + distribution+ " " + "-
-report-interval=1"+ " " +'run'
  # execute
  out = subprocess.check_output(cmd,
    stderr = subprocess.STDOUT, shell=True)
  # output sysbench outputs to a file
  output_file_name = "xengine_result_"+test_name[20:len(test_name)]
  output_file = open(output_file_name, "w")
  output_file.write(out)
  output_file.close()
if __name__ == '__main__':
  # the connection string for the MySQL (X-Engine) instance to be tested
  db_conn_string = sys.argv[1]
  test = [
    "/usr/share/sysbench/oltp_update_index.lua",
    "/usr/share/sysbench/oltp_point_select.lua",
    "/usr/share/sysbench/oltp_read_only.lua",
    "/usr/share/sysbench/oltp_write_only.lua",
    "/usr/share/sysbench/oltp_read_write.lua",
    "/usr/share/sysbench/oltp_insert.lua"
  ]
  for atest in test:
    print("start test:\t%s\t%s" % (atest, time.ctime()))
    execute_test(atest, db_conn_string)
    print("end test:\t%s\t%s" % (atest, time.ctime()))
    # sleep foe some seconds
    # after a period of testing with inserts/updates/deletes, x-engine needs some time to c
omplete
    # its asynchronous background compactions.
    time.sleep(1000)
```

8.5. 使用DMS归档数据到X-Engine

本文介绍如何通过任务编排实现周期性地将MySQL InnoDB表中的历史数据迁移至MySQL X-Engine表，实现低成本的RDS X-Engine存储方案。

前提条件

源库和目标库实例满足以下条件：

- 源库为RDS MySQL InnoDB数据库。
- 目标库为RDS MySQL X-Engine数据库，创建方法请参见创建RDS MySQL实例。
- 管控模式均为安全协同。
- 源库和目标库均已开启跨库查询，开启方法请参见编辑实例。

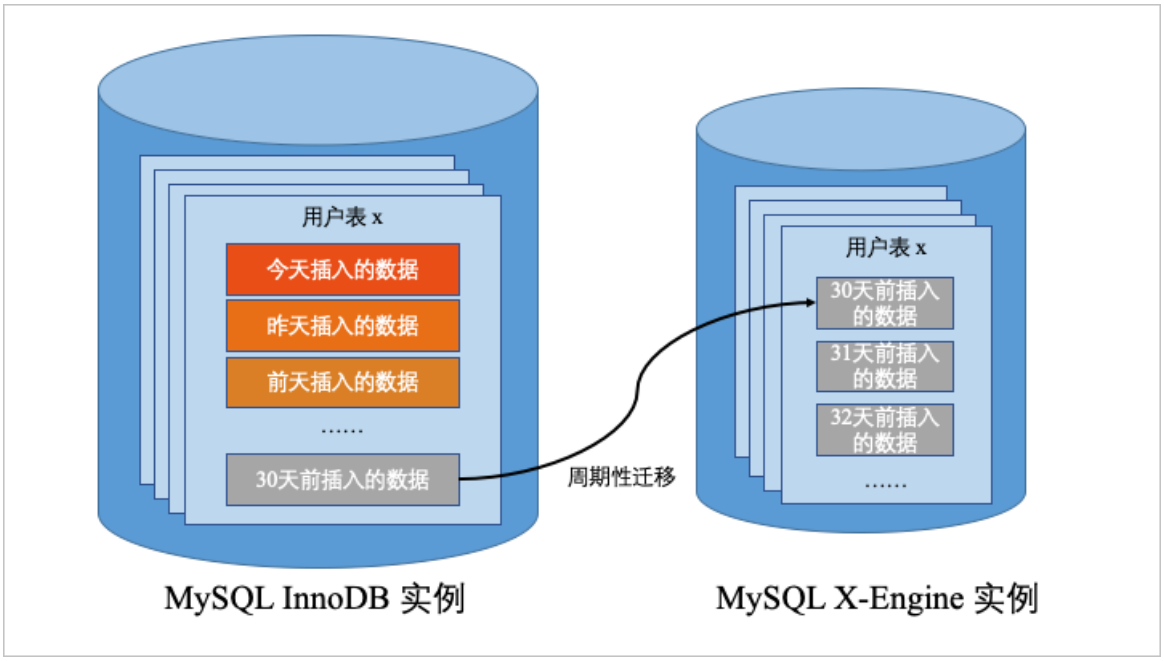
> **说明** 本案例中RDS MySQL InnoDB引擎对应的DBLink名称为dblink_src_rds，RDS MySQL X-Engine引擎对应的DBLink名字为dblink_target_rds。

背景信息

为解决企业随着业务的发展，数据库存储膨胀而导致存储成本急剧上升、数据库性能出现下降等问题，阿里云推出基于X-Engine存储引擎的低成本存储方案。通过将现有MySQL InnoDB表中的历史数据迁移至MySQL X-Engine引擎上，仅保留最近的热数据的方式，可降低企业存储成本，提高数据库安全及可用性。

> **说明** X-Engine是阿里云自研的OLTP数据库存储引擎，已经广泛应用在阿里集团内部诸多业务系统中，包括交易历史库、钉钉历史库等核心应用，大幅缩减了业务成本，同时也作为双十一大促的关键数据库技术，挺过了数百倍平时流量的冲击。X-Engine的性能与InnoDB存储引擎相似，但是存储成本远低于InnoDB，因此拥有极高的性价比。更多信息请参见X-Engine简介。

流程如下图所示：



您可以使用DMS的任务编排功能实现历史数据定期、自动地迁移到X-Engine上，降低人工操作的成本。

本文将介绍在DMS任务编排中，创建可周期性地（每天）将MySQL InnoDB表中的历史数据（一个月前）迁移至MySQL X-Engine表上，然后对InnoDB表进行历史数据清理和表空间优化的任务流。随着任务流的周期运行，MySQL InnoDB表中将只保留最近一个月的数据，而大量历史数据则保存至低成本的X-Engine引擎上。

创建任务流

1. 登录数据管理DMS 5.0。

> **说明** 如果您需要切换到旧版数据管理DMS，单击页面右下角，进入**数据管理DMS**平台。具体操作，请参见数据管理DMS 5.0切换至旧版。

2. 在页面顶部，选择**全部功能 > 数据工厂 > 任务编排**。
3. 在**任务编排**页面的**自由编排任务**区域，单击**新建任务流**。

4. 在**新建任务流**对话框，将**任务流名称**设置为`Rds_innodb_to_X-Engine`，将**描述**设置为`Rds_innodb_to_X-Engine demo`，单击**确认**。即会进入**任务编排**页面。

创建任务节点

本章节将介绍在Rds_innodb_to_X-Engine任务流中创建4个任务节点：

- 创建X-Engine表：创建数据转储的目标表。
- 历史数据迁移：利用DSQL跨库查询的能力将InnoDB表的数据迁移至X-Engine表上。
- 删除InnoDB表的历史数据：删除InnoDB表中已迁移的数据。
- 优化InnoDB表：历史数据删除后，可通过OPTIMIZE命令对InnoDB表空间进行优化，以节省存储空间。

1. 在**任务编排**页面中，创建并配置以下节点：

创建X-Engine表

i. 将左侧任务类型中的**MySQL**拖拽到页面中的空白区域。

ii. 双击画面中的目标任务节点，重命名为：创建X-Engine表，按回车保存节点名。

iii. 单击此节点，在右侧面板中选中**内容设置**页签。

iv. 在**内容设置**页签中，从数据库列表，选择RDS MySQL X-Engine数据库实例。

v. 输入以下建表语句，并单击**保存**。

```
CREATE TABLE IF NOT EXISTS `target_xengine_tbl` (
  `id` BIGINT,
  `price` DECIMAL(10,2),
  `count` INT,
  `trx_time` DATETIME,
  PRIMARY KEY (`id`)
) ENGINE=XENGINE DEFAULT CHARSET=utf8;
```

> **说明** 本案例中将目标表命名为：`target_xengine_tbl`，且该表的表结构与InnoDB业务库里的表结构`src_innodb_tbl`必须一致。

历史数据迁移

i. 将左侧任务类型中的**跨库SQL**拖拽到页面中的空白区域。

ii. 双击画面中的目标任务节点，重命名为：历史数据迁移，按回车保存节点名。

iii. 单击任务流中画布的空白处，在右侧面板中选中**任务流变量**页签，参考下图添加变量名为 `thirty_one_days_ago` 和 `thirty_days_ago` 的变量，并单击**保存**。

在配置完变量后，您可以在SQL任务中，用 `${变量名}` 表示时间点或时长。关于变量的规则与作用详情，请参见任务。

- 新增 `thirty_one_days_ago` 变量，格式为 `yyyy-MM-dd` ，偏移为 `-30日` 。

  表示获取当前日期往前30天的日期（精确至日期），例如当前日期为2021-01-15，该变量的值即为2020-12-15。

- 新增 `thirty_days_ago` 变量，格式为 `yyyy-MM-dd` ，偏移为 `-29天` 。

  表示获取当前日期往前29天的日期（精确至日期），例如当前日期为2021-01-15，该变量的值即为2020-12-16。

iv. 单击此节点，在右侧面板中选中**内容设置**页签。

v. 输入以下语句，并单击**保存**。

```
INSERT INTO `dblink_target_rds`.`target_db`.`target_xengine_tbl`
 (`id`, `price`, `count`, `trx_time`)
SELECT `id`, `price`, `count`, `trx_time`
FROM `dblink_src_rds`.`src_db`.`src_innodb_tbl`
WHERE `trx_time` >= '${thirty_one_days_ago}'
AND `trx_time` < '${thirty_days_ago}';
```

> **说明** 该语句的主要作用是将InnoDB表的部分数据迁移至X-Engine表中。
>
> - 本案例中 `dblink_src_rds` 指源RDS MySQL InnoDB引擎的数据库实例，`dblink_target_rds` 指目标RDS MySQL X-Engine引擎的数据库实例。
> - 迁移数据的范围（WHERE条件）由两个任务流变量来控制：`${thirty_one_days_ago}` 和 `${thirty_days_ago}` 。

删除InnoDB表的历史数据

i. 将左侧任务类型中的**无锁数据变更**拖拽到页面中的空白区域。

ii. 双击画面中的目标任务节点，重命名为：删除InnoDB表的历史数据，按回车保存节点名。

iii. 单击此节点，在右侧面板中选中**内容设置**页签。

iv. 在**内容设置**页签中，从数据库列表，选择RDS MySQL InnoDB数据库实例。

> **说明** 该节点的目的是删除RDS MySQL InnoDB表中对已迁移的数据。

v. 输入以下语句，并单击**保存**。

```
DELETE FROM `src_innodb_tbl`
  WHERE `trx_time` >= '${thirty_one_days_ago}'
  AND `trx_time` < '${thirty_days_ago}';
```

优化InnoDB表

i. 将左侧任务类型中的**无锁数据变更**拖拽到页面中的空白区域。

ii. 双击画面中的目标任务节点，重命名为：优化InnoDB表，按回车保存节点名。

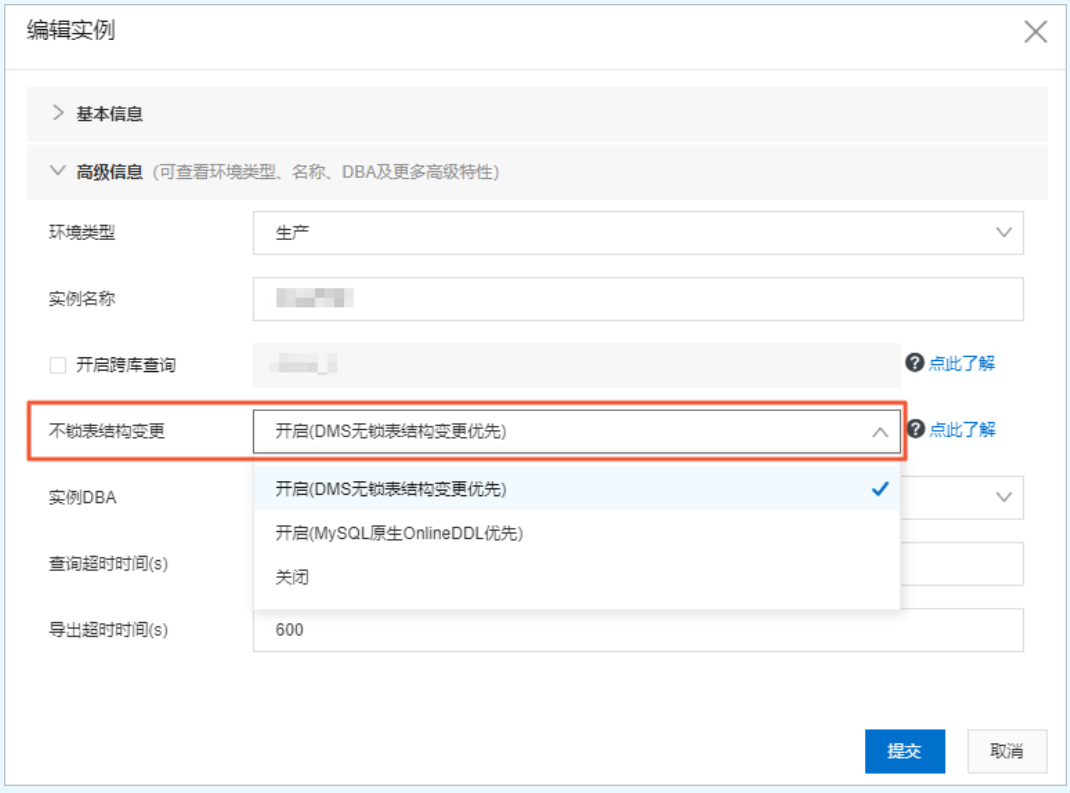
iii. 单击此节点，在右侧面板中选中**内容设置**页签。

iv. 在**内容设置**页签中，从数据库列表，选择RDS MySQL InnoDB数据库实例。

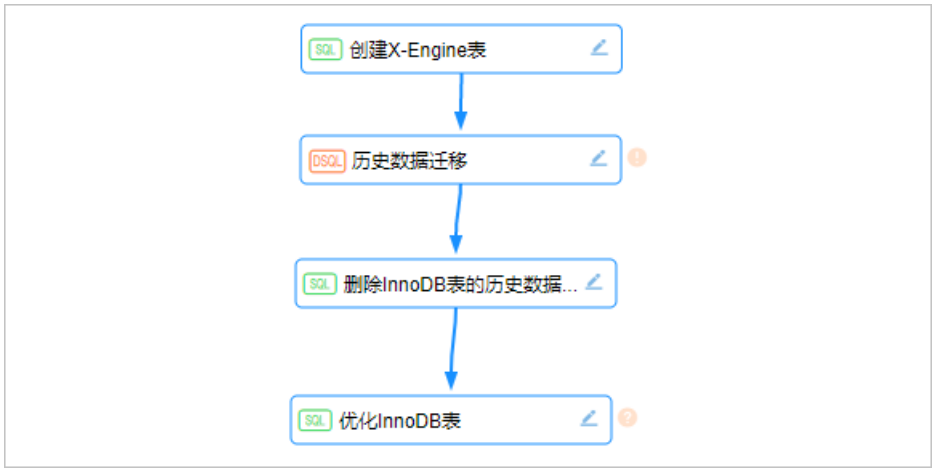
v. 输入以下语句，并单击**保存**。

```
OPTIMIZE TABLE `src_innodb_tbl`;
```

说明 为避免运行OPTIMIZE命令对在线业务的影响，建议您将RDS实例开启OnlineDDL功能（如下图所示），开启方法请参见编辑实例。



2. 在任务流中的画布中，按顺序将4个任务节点进行连接。



开启调度任务

1. 单击任务流中画布的空白处，在右侧面板中选中**调度配置**页签，打开**开启调度**按钮，参考下图完成配置。

> **说明** 为避开业务高峰期，本案例的调度配置为每日凌晨01:00运行，您可以按需配置调度计划。

2. 单击**保存**即可。