ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for RDS
AliSQL Kernel

Document Version: 20201014

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings> Network> Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Features of AliSQL

This topic provides an overview of the features that are provided by AliSQL. It also provides a comparison between MySQL versions with AliSQL and other MySQL versions.

Introduction to AliSQL

AliSQL is an independent MySQL branch that is developed by Alibaba Cloud. AliSQL provides all of the features that are available in the MySQL Community edition. AliSQL also provides some similar features that you can find in the MySQL Enterprise edition. These similar features include enterprise-grade backup and restoration, thread pool, and parallel query. In addition, AliSQL provides Oracle-compatible features, such as Sequence engine. ApsaraDB RDS for MySQL with AliSQL provides all the basic features of MySQL and a wide range of advanced features such as enterprise-grade security, backup and restoration, monitoring, performance optimization, and read-only instance.

MySQL versions supported

Category	Feature	Description	MySQL 8.0	MySQL 5.7	MySQL 5.6
Functionality	Thread Pool	The thread pool feature separates threads from sessions. If a large number of sessions are created on your RDS instance, you can run a small number of threads to complete the tasks in active sessions.	Supported	Supported	Supported
	Statement outline	The statement outline feature allows you to stably run query plans by using optimizer and index hints. You can install the DBMS_OUTLN package to use this feature.	Supported	Supported	Not supported
	Sequence Engine	The Sequence engine simplifies the generation of sequence values on your RDS instance.	Supported	Not supported	Supported
	Returning	This returning feature allows data manipulation language (DML) statements to return result sets. You can install the DBMS_TRANS package to use this feature.	Supported	Not supported	Not supported
	Fast query cache	The fast query cache is a query cache that is developed by Alibaba Cloud based on the native MySQL query cache. The fast query cache uses a new design and a new implementation mechanism to increase the query performance of your RDS instance.	Not supported	Supported	Not supported

Category	Feature	Description	MySQL 8.0	MySQL 5.7	MySQL 5.6
Performance	Binlog in Redo	The Binlog in Redo feature synchronously writes binary logs to the redo log file when a transaction is committed. This reduces operations on disks and improves the performance of your RDS instance.	Supported	Not supported	Not supported
	Statement Queue	The statement queue feature allows statements to queue in the same bucket. These statements may be executed on the same resources. For example, these statements are executed on the same row of a table. This feature reduces overheads from possible conflicts.	Supported	Supported	Not supported
	Inventory Hint	The inventory hint feature can work with the returning and statement queue features to rapidly commit and roll back transactions. This allows you to increase the throughput of your application.	Supported	Supported	Supported
Stability	Faster DDL	The faster DDL feature provides an optimized buffer pool management mechanism. This mechanism reduces the impact of data definition language (DDL) operations on the performance of your RDS instance. This mechanism also increases the number of concurrent DDL operations that are allowed.	Supported	Supported	Supported
	Statement concurrency control	The concurrency control (CCL) feature allows you to control the concurrency of statements based on syntax rules. You can install the DBMS_CCL package to use this feature.	Supported	Supported	Not supported
	Performance Agent	The performance agent feature is provided as a plug-in of MySQL. You can use this feature to calculate and analyze the performance metrics of your RDS instance.	Supported	Supported	Supported
	Purge Large File Asynchronously	The Purge Large File Asynchronously feature allows you to asynchronously delete files from your RDS instance. This ensures the stability of your RDS instance.	Supported	Supported	Supported

Category	Feature	Description	MySQL 8.0	MySQL 5.7	MySQL 5.6
	Performance Insight	The performance insight feature supports load monitoring, association analysis, and performance optimization at the instance level. You can evaluate loads on your RDS instance and resolve performance issues. This allows you to improve the stability of your RDS instance.	Supported	Supported	Not supported
Security	Data Protect	The data protection feature controls the permissions on delete operations. This allows you to protect your data from being accidentally deleted.	Supported	Supported	Supported
	Recycle bin	The recycle bin feature allows you to temporarily store deleted tables. It also allows you to specify a retention period within which you can retrieve the deleted tables. You can install the DBMS_RECYCLE package to use this feature.	Supported	Not supported	Not supported

Features

Category	Feature	MySQL Community edition	MySQL Enterprise edition	MySQL 5.7 and MySQL 8.0 with AliSQL	ApsaraDB RDS for MySQL
Enterprise-grade value-added services	24/7 support	Not supported	Supported	Supported	Supported
	Emergency troubleshooting	Not supported	Supported	Supported	Supported
	Expert support	Not supported	Supported	Supported	Supported
	MySQL Database Server	Supported	Supported	Supported	Supported
	MySQL Document Store	Supported	Supported	Supported for MySQL 8.0	Supported for MySQL 8.0
	MySQL Connectors	Supported	Supported	Supported for versions released to the public	Supported for versions released to the public

Category	Feature	MySQL Community edition	MySQL Enterprise edition	MySQL 5.7 and MySQL 8.0 with AliSQL	ApsaraDB RDS for MySQL
MySQL Features	MySQL Replication	Supported	Supported	Supported	Supported
	MySQL Router	Supported	Supported	MaxScale supported for MySQL 8.0	Single-tenant database proxies supported
	MySQL Partitioning	Supported	Supported	Supported	Supported
	Storage Engine	InnoDB MyISAM NDB	InnoDB MyISAM NDB	InnoDB X-Engine	InnoDB X-Engine
Oracle Compatibility	Sequence Engine	Not supported	Not supported	Supported for MySQL 8.0	Supported for MySQL 8.0
MySQL Enterprise Monitor	Enterprise Dashboard	Not supported	Supported	Under development	Enhanced Monitor
	Query Analyzer	Not supported	Supported	Under development	Performance Insight
	Replication Monitor	Not supported	Supported	Under development	Supported
	Enhanced OS Metrics	Not supported	Not supported	Not supported	Enhanced Monitor
MySQL Enterprise Backup	Hot backup for InnoDB	Not supported	Supported	Supported	Supported
	Full, Incremental, Partial, Optimistic Backups	Not supported	Supported	Supported	Database- and table-level backup supported
	Full, Partial, Selective, Hot Selective restore	Not supported	Supported	Supported	Database- and table-level restoration supported
	Point-In-Time-Recovery	Not supported	Supported	Supported	Supported

Category	Feature	MySQL Community edition	MySQL Enterprise edition	MySQL 5.7 and MySQL 8.0 with AliSQL	ApsaraDB RDS for MySQL
	Cross-Region Backup	Not supported	Not supported	Not supported	Cross-region backup supported
	Recycle bin	Not supported	Not supported	Supported for MySQL 8.0	Supported for MySQL 8.0
	Flashback	Not supported	Not supported	Supported	Supported
MySQL Enterprise Security	Enterprise TDE	Local key replacement supported	Supported	BYOK-based TDE and key rotation supported	BYOK-based TDE and key rotation supported
	Enterprise Disk Data Encryption at Rest	Not supported	Not supported	Not supported	BYOK-based disk encryption supported
	Enterprise Encryption	SSL	Supported	SSL	SSL
	Enterprise Audit	Not supported	Supported	SQL Explorer supported	SQL Explorer supported
	SM4 encryption algorithm	Not supported	Not supported	Supported	Supported
MySQL Enterprise Scalability	Thread Pool	Not supported	Supported	Supported for MySQL 8.0	Supported for MySQL 8.0
	Enterprise Readonly Request Extention	Not supported	Not supported	Supported	Read-only instances supported
MySQL Enterprise Reliability	SQL Outline	Not supported	Not supported	Supported	Supported
	Hot Massive Update	Not supported	Not supported	Supported	Supported
	Hot SQL Limit	Not supported	Not supported	Supported	Supported
	Hot SQL Firewall	Not supported	Not supported	Supported	Supported

Category	Feature	MySQL Community edition	MySQL Enterprise edition	MySQL 5.7 and MySQL 8.0 with AliSQL	ApsaraDB RDS for MySQL
MySQL Enterprise High-Availability	Enterprise Automatic Failover Switch	Not supported	Not supported	Third-party high-availability mechanism required	Supported for the RDS High-availability Edition
	Multi-Source Replication	Supported	Supported	Supported	Highly available read-only instances supported

2.Release notes of minor AliSQL versions

This topic describes the release notes of minor AliSQL versions.

ApsaraDB RDS for MySQL 8.0

20200831

- New features:
  - An option is added to disable parallel scan for the `COUNT (*)` function.
  - Start global transaction identifiers (GTIDs) and end GTIDs are introduced to the mysqlbinlog plug-in.
  - Various log sequence numbers (LSNs) in the redo log are supported.
    - innodb_lsn: the LSN of each record in the redo log.
    - innodb_log_checkpoint_lsn: the LSN of the last checkpoint.
    - innodb_log_write_lsn: the LSN of each record that is written into the redo log.
    - innodb_log_ready_for_write_lsn: the LSN of the last record that is written into the log buffer.
    - innodb_log_flush_lsn: the LSN of each record that is flushed from the redo log to the disk.
    - innodb_log_dirty_pages_added_up_to_lsn: the LSN of each record that logs a page as dirty.
    - innodb_log_oldest_lsn: the LSN of each record that logs an update to a page.
- Performance optimization:
  - The concurrency control (CCL) mechanism is optimized to better determine how transactions can wait and concurrently run.
  - The CCL mechanism is optimized to better prioritize the stored procedures that are to run.
- Bugs fixed:
  - The bug that prevents the recursively called interpreter from checking the memory size is fixed.
  - The bug that prevents you from modifying table definitions when transparent data encryption (TDE) is enabled is fixed.
  - The bug that causes the event scheduler to leak memory is fixed.

20200630

- New features:
  - The faster DDL feature is introduced to provide an optimized buffer pool management mechanism. This mechanism reduces the impact of data definition language (DDL) operations and increases the number of concurrent DDL operations that are allowed. For more information, see Faster DDL.
  - The maximum number of connections that are allowed is increased to 500,000.
- Performance optimization:
  - Thread pools are optimized.

  - The memory allocation mechanism is optimized. You can specify the maximum number of memory resources that are allowed for Performance Schema based on the instance type.
  - SQL log files are no longer detected.
  - TDE is optimized to cache the keys that are provided by Alibaba Cloud Key Management Service (KMS).
  - The status of threads that are managed by the CCL mechanism is modified. For more information, see Statement concurrency control.

- Bugs fixed:
  - The bug that causes the system to consider a semicolon (;) to be a part of the command used to create an outline is fixed.
  - The bug that causes the server to unexpectedly exit in the event of table modifications is fixed.
  - The bug that causes earlier versions to disallow the memory and array keywords supported in later versions is fixed.
  - The bug that causes the system to incorrectly count the number of waits when commands are read from a client is fixed.
  - The bug that causes failures in minor engine version updates is fixed.

20200430

- New features:
  - The Binlog in Redo feature is introduced. This feature writes binary logs into redo log files before the binary logs are written to the disk. This reduces I/O consumption and improves database performance. For more information, see Binlog in Redo.
  - The data protection feature is introduced. This feature supports the customization of security policies that are used to manage the permissions on DROP and TRUNCATE statements. This allows you to avoid data losses that are caused by the unintentional execution of these statements. For more information, see Data Protect.
  - The code for row caching in the X-Engine storage engine is restructured.
  - The XA_RECOVER_ADMIN permission is provided.
- Performance optimization:
  - The code that is used to scan data when operations are performed on a temporary InnoDB table is optimized. This allows the system to scan only dirty pages instead of the entire buffer pool.
  - The global parameter opt_readonly_trans_implicit_commit is renamed as rds_disable_explicit_trans. This ensures compatibility with MySQL 5.6.
  - The SQL Explorer (SQL Audit) feature is optimized, so it does not log upgrades to RDS instances.
  - Memory resources that are consumed by DDL operations on X-Engine tables are reduced.
- Bugs fixed:
  - The bug that causes the sizes of X-Engine tables stored on the disk to be inconsistent with the statistical information in the INFORMATION_SCHEMA schema is fixed.
  - The bug that causes the system to initialize X-Engine logs when the error log file is re-opened is fixed.

20200331

- New feature:

  The `TRUNCATE TABLE` statement is supported. After you execute this statement on a table, this statement moves the table to the recycle bin. Then, this statement creates a table by using the schema of the table that you truncate. For more information, see Recycle bin.

- Performance optimization:
  - The output of Transmission Control Protocol (TCP) errors is disabled by default.
  - The performance of thread pools with the default configuration is improved.
- Bugs fixed:
  - The bug that databases and tables become invalid because the names of partitioned tables are separated with a pound key and a letter p (`#p`) is fixed.
  - The bug that causes the statements managed by the CCL mechanism to be case-sensitive is fixed.
- Changes incorporated: Changes in MySQL 8.0.17 and MySQL 8.0.18 are incorporated. For more information, see Changes in MySQL 8.0.17 and Changes in MySQL 8.0.18.

20200229

- New features:
  - The performance agent feature is introduced. For more information, see Performance Agent. This feature is provided as a MySQL plug-in. It allows you to collect and analyze the performance metrics of an RDS instance.
  - Network round-trip time is introduced to the semi-synchronous mode. This allows you to better understand the performance of an RDS instance.
- Performance optimization:
  - Statement-level CCL is allowed on read-only RDS instances.
  - Outlines are supported for secondary RDS instances.
  - The database proxy feature is enhanced to optimize short-lived connections.
  - The time that is required to execute a PAUSE statement is reduced in various CPU architectures.
  - A memory table is introduced to present the running status of thread pools.
- Bugs fixed:
  - The bug that causes the system to forbid the ppoll function and replace the ppoll function with the poll function in Linux kernels earlier than version 4.9 is fixed.
  - The bug that causes errors when the system invokes the wrap_sm4_encrypt function is fixed.
  - The bug that causes the system to lock global variables when SQL logs are rotated is fixed.
  - The bug that causes errors in restoration inconsistency checks is fixed.
  - The bug that causes incorrect time values in the io_statistics table is fixed.
  - The bug that causes the system to unexpectedly exit when invalid compression algorithms are invoked is fixed.
  - The bug that causes user columns in MySQL 8.0 and MySQL 5.6 to be incompatible is fixed.

20200110

- New feature:

  Three hints are introduced. These hints can be used in SELECT, UPDATE, INSERT, and DELETE statements to commit and roll back transactions at high speeds. This allows you to increase the throughput of your application. For more information, see Inventory Hint.

- Performance optimization:
  - The CCL mechanism is optimized. When an RDS instance is started, CCL queue structures are initialized before CCL rules.
  - The file deletion mechanism is optimized. When you asynchronously delete small files, links to the small files are canceled.
  - The performance of thread pools is optimized. For more information, see Thread Pool.
  - Restoration inconsistency checks are disabled by default.
  - The permissions that are required to configure variables are changed.
    - The user role that is authorized to configure the following variables is changed to standard user:
      - auto_increment_increment
      - auto_increment_offset
      - bulk_insert_buffer_size
      - binlog_rows_query_log_events
    - The user role that is authorized to configure the following variables is changed to superuser or system variable administrator:
      - binlog_format
      - binlog_row_image
      - binlog_direct
      - sql_log_off
      - sql_log_bin

20191225

- New feature:

  The recycle bin feature is introduced. All of the tables that you delete are moved to the recycle bin. You can specify a retention period within which you can retrieve the deleted tables from the recycle bin. For more information, see Recycle bin.

- Performance optimization:
  - The data proxy feature is enhanced to optimize short-lived connections.
  - A dedicated thread is used to serve the maintain user. This allows you to avoid high availability (HA) failures.
  - The locking mechanism is optimized. If an error occurs when binary logs are flushed by using redo logs, ApsaraDB for RDS can explicitly release the lock that is triggered by file synchronization.
  - The deletion of unnecessary TCP error logs is supported.
  - Thread pools are enabled by default.

- Bugs fixed:

- The bug that causes errors in updates to slow query logs is fixed.
- The bug that causes an incorrect lock scope is fixed.
- The bug that causes errors in core dumps when the system invokes the select function for TDE is fixed.

20191115

New feature:

The Statement Queue feature is introduced. This feature allows statements to queue in the same bucket. These statements may be executed on the same resources. For example, these statements are executed on the same row of a table. This reduces overheads from possible conflicts. For more information, see Statement Queue.

20191101

- New features:
  - The SM4 encryption algorithm is supported for TDE. For more information, see Configure TDE for an ApsaraDB RDS for MySQL instance.
  - Data protection for secondary RDS instances is supported. Only the accounts with the SUPER or REPLICATION_SLAVE_ADMIN role have the permissions to insert, delete, and modify data in the slave_master_info, slave_relay_log_info, and slave_worker_info tables.
  - A mechanism is introduced to increase the priorities of auto-increment keys. If a table does not have a primary key or it does not have a unique key without a null value, the auto-increment key without a null value has the highest priority.
  - A mechanism is introduced to prevent the automatic conversion of tables from the MEMORY to MyISAM storage engines. These tables include system tables. These tables also include tables that are invoked by threads in the initializing state.
  - A mechanism is introduced to flush binary log files to the disk before redo log files.
  - A mechanism is introduced to stop the creation of temporary tables on an RDS instance when the RDS instance is locked.
  - The X-Engine storage engine is provided to store transactions based on a log-structured merge (LSM) tree.
- Performance optimization:
  - The thread pool feature is optimized to reduce mutexes. For more information, see Thread Pool.
  - The performance insight feature is optimized to monitor thread pools. For more information, see Performance Insight.
  - Parameter adjustment:
    - `primary_fast_lookup` : a session parameter. Default value: true.
    - `thread_pool_enabled` : a global parameter. Default value: true.

20191015

- New features:
  - The TDE feature is introduced to support real-time I/O encryption and decryption on data files. Data is encrypted before it is written to the disk and decrypted before it is read from the disk to the memory. For more information, see Configure TDE for an ApsaraDB RDS for MySQL instance.

- The returning feature is introduced. This feature allows data manipulation language (DML) statements to return result sets. In addition, the DBMS_TRANS package is provided for you to use this feature. For more information, see Returning.
- The forced conversion from the MyISAM or MEMORY storage engine to the InnoDB storage engine is supported. If the global variable **force_mysiam_to_innodb** or **force_memory_to_innodb** is set to **ON**, a table is converted from the MyISAM or MEMORY storage engine to the InnoDB storage engine when the table is created or modified.
- A mechanism is introduced to forbid standard accounts from performing primary/secondary switchovers. Only privileged accounts have the permissions to perform primary/secondary switchovers.
- A performance proxy plug-in is provided. This plug-in obtains performance data and saves the data as TXT files to your computer. These files are deleted in a circular manner. Only the latest second-level files are retained.
- A configurable timeout period is introduced for mutexes in InnoDB: This timeout period can be changed by setting the global variable **innodb_fatal_semaphore_wait_threshold**. The default value of the global variable is 600.
- Index hint errors can be ignored by setting the global variable **ignore_index_hint_error**. The default value of the global variable is false.
- The SSL encryption feature can be disabled. For more information, see Configure SSL encryption on an ApsaraDB RDS for MySQL instance.
- The output of TCP errors is supported. TCP errors in read, read-wait, and write-wait events are returned with their error codes by using end_connection events. In addition, logs with information about the errors are generated.

- Bugs fixed:
  - The bug that prevents a Linux operating system from merging local asynchronous I/O (AIO) requests before linear Read Ahead is triggered is fixed.
  - The bug that prevents the proper collection of table and index statistics is fixed.
  - The bug that prevents the system direct access to the primary key index of a table with a primary key is fixed.

20190915

Bug fixed:

The bug that causes memory leaks when the Cmd_set_current_connection process runs is fixed.

20190816

- New features:
  - The thread pool feature is introduced to separate threads from sessions. If a large number of sessions exist, the system can run a small number of threads to complete the tasks in active sessions. For more information, see Thread Pool.
  - The CCL mechanism is introduced. This mechanism allows you to specify the maximum number of concurrent requests that are allowed. This enables the system to handle traffic bursts, process statements that consume excessive resources, and adapt to changes of SQL models. This also ensures the continuity and stability of your database service. For more information, see Statement concurrency control.
  - The statement outline feature is introduced to support optimizer hints and index hints. These hints are used to stabilize the execution of query plans on an RDS instance. For more information, see Statement outline.

  - The Sequence engine is introduced to simplify the acquisition of sequence values. For more information, see Sequence Engine.
  - The Purge Large File Asynchronously feature is introduced to asynchronously delete files. Before you delete a tablespace, the system renames the files in the tablespace as temporary files. Then, a background thread is started to asynchronously delete the temporary files. For more information, see Purge Large File Asynchronously.
  - The performance insight feature is introduced to support load monitoring, association analysis, and performance optimization at the instance level. This feature allows you to evaluate the loads of an RDS instance. This feature also allows you to locate performance issues to ensure the stability of your database service. For more information, see Performance Insight.
  - An optimized instance locking mechanism is introduced. You can delete tables from an RDS instance by using DROP or TRUNCATE statements even if the RDS instance is locked.

- Bugs fixed:
  - The bug that causes the system to incorrectly calculate file sizes is fixed.
  - The bug that allows irrelevant processes to reuse released memory resources is fixed.
  - The bug that causes a host to exit unexpectedly when the available cache size on the host is 0 is fixed.
  - The bug that causes conflicts between implicit primary keys and CTS statements is fixed.
  - The bug that causes the system to incorrectly log slow queries is fixed.

20190601

- Performance optimization:
  - Metadata locking on logging tables is reduced.
  - The code for termination options is restructured.
- Bugs fixed:
  - The bug that prevents the SQL Explorer (SQL Audit) feature from logging precompiled statements is fixed.
  - The bug that prevents the system from filtering out error logs in logging tables with invalid names is fixed.

ApsaraDB RDS for MySQL 5.7 on RDS Basic or High-availability Edition

20200831

- New features:
  - Changes incorporated: Changes in MySQL 5.7.30 are incorporated. For more information, visit GitHub.
  - An optimized CCL mechanism is introduced to better determine how transactions can wait and concurrently run.
  - Start GTIDs and end GTIDs are introduced to the mysqlbinlog plug-in.

  - Various LSNs in the redo log are supported.
    - innodb_lsn: the LSN of each record in the redo log.
    - innodb_log_write_lsn: the LSN of each record that is written into the redo log.
    - innodb_log_checkpoint_lsn: the LSN of the last checkpoint.
    - innodb_log_flushed_lsn: the LSN of each record that is flushed from the redo log to the disk.
    - innodb_log_Pages_flushed: the LSN of each record that logs an update to a page.
- Performance optimization:

  The CCL mechanism is optimized to better prioritize the stored procedures that are to run.
- Bug fixed:

  Some major bugs that cause the server to unexpectedly stop when you shut down the server is fixed.

20200630

- New features:
  - Three hints are introduced to the inventory hint feature. These hints are used in SELECT, UPDATE, INSERT, and DELETE statements to commit and roll back transactions at high speeds. This allows you to increase the throughput of your application. For more information, see Inventory Hint.
  - The CCL mechanism is introduced. This mechanism allows you to specify the maximum number of concurrent requests that are allowed. This enables the system to handle traffic bursts, process statements that consume excessive resources, and adapt to changes of SQL models. This also ensures the continuity and stability of your database service. For more information, see Statement concurrency control.
  - The Statement Queue feature is introduced. This feature allows statements to queue in the same bucket. These statements may be executed on the same resources. For example, these statements are executed on the same row of a table. This reduces overheads from possible conflicts. For more information, see Statement Queue.
  - The statement outline feature is introduced to support optimizer hints and index hints. These hints are used to stabilize the execution of query plans on an RDS instance. For more information, see Statement outline.
  - The faster DDL feature is introduced to provide an optimized buffer pool management mechanism. This mechanism reduces the impact of DDL operations and increases the number of concurrent DDL operations that are allowed. For more information, see Faster DDL.
  - The maximum number of connections that are allowed is increased to 500,000.
- Performance optimization:
  - The `call dbms_admin.show_native_procedure();` command is provided to display all of the procedures on an RDS instance.
  - A new function is provided to delete orphan tables.
  - Thread pools are optimized.
  - Query caching is optimized.
  - The memory allocation mechanism is optimized. You can specify the maximum number of memory resources that are allowed for Performance Schema based on the instance type.

- Bug fixed:

  The bug that causes an audit update thread to enter an infinite loop is fixed.

20200430

- New feature:

  The data protection feature is introduced. This feature supports the customization of security policies that are used to manage the permissions on DROP and TRUNCATE statements. This allows you to avoid data losses that are caused by the unintentional execution of these statements. For more information, see Data Protect.

- Performance optimization:

  Read-write locks are no longer supported in the query cache. The default hash function is changed from LF_hash to murmur3 hash.

- Bugs fixed:

  Two bugs that occur after the system hits the query cache during the execution of transactions at the REPEATABLE_READ isolation level are fixed.

20200331

- New features:
  - The fast query cache is introduced. It is developed by Alibaba Cloud based on the native MySQL query cache. It uses a new design and implementation mechanism to improve query performance. For more information, see Fast query cache.
  - Two metadata locks are introduced from Percona Server 5.7: LOCK TABLES FOR BACKUP (LTFB) and LOCK BINLOG FOR BACKUP (LBFB).
- Performance optimization:
  - Thread pools are optimized to ensure compatibility with earlier MySQL versions.
  - The output of TCP errors is disabled by default.
  - The performance of thread pools with the default configuration is improved.
- Bugs fixed:
  - The bug that causes the system to delete temporary files when you delete large files is fixed.
  - The bug that causes dump threads in thread pools to time out is fixed.
  - The bug that causes the system to incorrectly count the value of the IPK field in the procedure context is fixed.
  - The bug that causes rds_change_user to incur pfs thread leakage and release is fixed.
- Changes incorporated: Changes in MySQL 5.7.28 are incorporated. For more information, visit GitHub.

20200229

- New features:
  - The performance agent feature is introduced. For more information, see Performance Agent. This feature is provided as a MySQL plug-in. It allows you to collect and analyze the performance metrics of an RDS instance.
  - Network round-trip time is introduced to the semi-synchronous mode. This allows you to better understand the performance of an RDS instance.

- Performance optimization:
  - The time that is required to execute a PAUSE statement is reduced in various CPU architectures.
  - The database proxy feature is enhanced to optimize short-lived connections.
  - A memory table is introduced to present the running status of thread pools.
- Bugs fixed:
  - The bug that causes DDL redo logs that are not secure is fixed.
  - The bug that causes incorrect time values in the io_statistics table is fixed.
  - The bug that causes the server to unexpectedly exit in the event of table modifications is fixed.
  - The bugs in MySQL test cases are fixed.

20200110

Performance optimization:

- The file deletion mechanism is optimized. When you asynchronously delete small files, links to the small files are canceled.
- The performance of thread pools is optimized. For more information, see Thread Pool.
- The default value of the thread_pool_enabled parameter is changed to OFF.

20191225

- New feature:

  The management of internal accounts is supported. This allows you to manage user permissions and protect your data.
- Performance optimization:
  - The data proxy feature is enhanced to optimize short-lived connections.
  - A dedicated thread is used to serve the maintain user. This allows you to avoid HA failures.
  - The deletion of unnecessary TCP error logs is supported.
  - Thread pools are optimized.
- Bugs fixed:
  - The bug that causes the mysqld process to unexpectedly exit when the read/write splitting function is enabled is fixed.
  - The bug that causes errors in core dumps when the system uses a keyring is fixed.

20191115

Bug fixed:

The bug that causes the system to display variables in SQL logs that are generated from primary/secondary switchovers is fixed.

20191101

- New features:
  - The SM4 encryption algorithm is supported for TDE. For more information, see Configure TDE for an ApsaraDB RDS for MySQL instance.

    - A mechanism is introduced to allow the system to access the primary index of a table with a primary key.
    - A mechanism is introduced to prevent the automatic conversion of tables from the MEMORY to MyISAM storage engines. These tables include system tables. These tables also include tables that are invoked by threads in the initializing state.
- Performance optimization:
    - The thread pool feature is optimized to reduce mutexes. For more information, see Thread Pool.
    - An SQL log caching mechanism is introduced to increase SQL logging performance.
    - The performance insight feature is optimized to monitor thread pools. For more information, see Performance Insight.
    - The thread pool feature is enabled by default. For more information, see Thread Pool.
- Bugs fixed:
    - The bug that prevents the release of locks on user tables when these tables are being managed or maintained is fixed.
    - More TCP errors are added.

20191015

- New features:
    - The rotation of slow query logs is supported. Every CSV slow query log file is assigned a unique name and a new file. This prevents data losses during the collection of slow query logs. You can run the `show variables like '%rotate_log_table%';` command to check whether the rotation of slow query logs is enabled.
    - A performance proxy plug-in is provided. This plug-in obtains performance data and saves the data as TXT files to your computer. These files are deleted in a circular manner. Only the latest second-level files are retained.
    - The forced conversion from the MEMORY to InnoDB storage engines is supported. If the global variable **rds_force_memory_to_innodb** is set to **ON**, a table is converted from the MEMORY to InnoDB storage engines when the table is created or modified.
    - The keyring-rds plug-in is introduced to TDE. This plug-in allows ApsaraDB for RDS to communicate with the administration system or Alibaba Cloud Key Management Service (KMS).
    - The output of TCP errors is supported. TCP errors in read, read-wait, and write-wait events are returned with their error codes by using end_connection events. In addition, logs with information about the errors are generated.
- Bug fixed:

  The bug that causes Error 1290 in DDL operations is fixed.

20190925

Parameter adjustment:

- The default value of the system variable auto_generate_certs is changed from true to false.
- The global read-only variable auto_detect_certs is introduced. Valid values: true and false. Default value: false. This variable is supported only when code is compiled by using OpenSSL on the server. This variable specifies whether the server automatically searches for SSL

certificate and key files in the data directory.

20190915

New feature:

The thread pool feature is introduced to separate threads from sessions. If a large number of sessions exist, the system can run a small number of threads to complete the tasks in active sessions. For more information, see Thread Pool.

20190815

- New features:
  - The Purge Large File Asynchronously feature is introduced to asynchronously delete files. Before you delete a tablespace, the system renames the files in the tablespace as temporary files. Then, a background thread is started to asynchronously delete the temporary files. For more information, see Purge Large File Asynchronously.
  - The performance insight feature is introduced to support load monitoring, association analysis, and performance optimization at the instance level. This feature allows you to evaluate the loads of an RDS instance. This feature also allows you to locate performance issues to ensure the stability of your database service. For more information, see Performance Insight.
  - An optimized instance locking mechanism is introduced. You can delete tables from an RDS instance by using DROP or TRUNCATE statements even if the RDS instance is locked.
- Bugs fixed:
  - The bug that allows you to set the rds_prepare_begin_id option in the `set rds_current_connection` command is fixed.
  - The bug that prevents the system from updating information about locked accounts is fixed.
  - The bug that allows you to use actual as a keyword in table names is fixed.
  - The bug that causes the overflow of timestamps in slow query logs is fixed.

20190510

New feature: Temporary tables can be created in transactions.

20190319

New feature: Thread IDs can be configured in handshake packets.

20190131

- The upgrade to MySQL 5.7.25 is supported.
- JeMalloc that is used for memory management is disabled.
- The bug that causes the system to incorrectly calculate the value of the internal variable net_lenth_size is fixed.

20181226

- New feature: Dynamic modifications to the system variable binlog-row-event-max-size are supported. This allows you to expedite the replication of tables that do not have a primary key.
- Bug fixed: The bug that prevents the proxy instance of an RDS instance from applying for memory resources is fixed.

20181010

- Implicit primary keys are supported.
- The replication of tables that do not have a primary key between primary and secondary RDS instances is accelerated.
- Native AIO is provided to improve I/O performance.

20180431

New features:

- The RDS High-availability Edition is supported.
- The SQL Audit feature is supported. For more information, see SQL audit.
- The protection for RDS instances on which snapshot backups are being created is enhanced.

ApsaraDB RDS for MySQL 5.7 on RDS Enterprise Edition

20191128

- New feature:

  The read/write splitting function is introduced.

- Bugs fixed:
  - The bug that causes the system to incorrectly calculate the value of the Second_Behind_Master metric for a follower is fixed.
  - The bug that causes dead locks during the re-execution of table-level parallel replication transactions is fixed.
  - XA-related bugs are fixed.

20191016

- New features:
  - The upgrade from the RDS High-availability Edition to the RDS Enterprise Edition is supported for RDS instances that use local SSDs.
  - The GTID function that is provided by the MySQL Community edition is supported. This function is disabled by default.
  - All of the Alibaba Cloud-proprietary AliSQL features and functions that are released in the RDS Basic and High-availability Editions before the minor version 20190915 are incorporated.
- Bug fixed:

  The bug that causes the system to disable binary logs for reset secondary RDS instances is fixed.

20190909

- New features:
  - The execution of large transactions is accelerated. This applies when the synchronous mode is used to replicate data between primary and secondary RDS instances that run the RDS Enterprise Edition.
  - Binary logs can be dumped from a leader or a follower.
  - The creation of read-only RDS instances is supported.
  - The InnoDB storage engine is used for system tables by default.
- Bugs fixed:

  - The bug that invalidates the commands that are run by a follower to delete logs is fixed.
  - The bug that causes slave threads to unexpectedly exit when the slave_sql_verify_checksum parameter is set to OFF and the binlog_checksum parameter is set to crc32 is fixed.

20190709

New features:

- The RDS Enterprise Edition is supported.
- The disabling of the semi-sync plug-in is supported.
- Table-level parallel replication and write set-level parallel replication are supported.
- The pk_access module is introduced to expedite queries that are run based on primary keys.
- Thread pools are supported.
- All of the Alibaba Cloud-proprietary AliSQL features and functions that are released for MySQL 5.7 in the RDS Basic and High-availability Editions before the minor version 20190510 are incorporated.

ApsaraDB RDS for MySQL 5.6

20200831

- New features:

  Various LSNs in the redo log are supported.

  - innodb_lsn: the LSN of each record in the redo log.
  - innodb_log_write_lsn: the LSN of each record that is written into the redo log.
  - innodb_log_checkpoint_lsn: the LSN of the last checkpoint.
  - innodb_log_flushed_lsn: the LSN of each record that is flushed from the redo log to the disk.
  - innodb_log_Pages_flushed: the LSN of each record that logs an update to a page.

- Bugs fixed:
  - The bug that causes incorrect SHOW_HA_ROWS enumeration types is fixed.
  - The bug that causes the system to incorrectly count the value of the IPK field in the procedure context is fixed.
  - The bug that causes the server to unexpectedly exit when you query data from the INFORMATION_SCHEMA schema is fixed.
  - The bug that causes an audit update thread to enter an infinite loop is fixed.
  - The bug that prevents secondary RDS instances from reporting data replication latencies is fixed.

20200630

- New features:
  - The performance agent feature is introduced. For more information, see Performance Agent. This feature is provided as a MySQL plug-in. It allows you to collect and analyze the performance metrics of an RDS instance.
  - The maximum number of connections that are allowed is increased to 500,000.

  - The faster DDL feature is introduced to provide an optimized buffer pool management mechanism. This mechanism reduces the impact of DDL operations and increases the number of concurrent DDL operations that are allowed. For more information, see Faster DDL.

- Performance optimization:
  - The global parameter max_execution_time is introduced. If the execution duration of an SQL statement exceeds the value of this parameter, the execution is paused.
  - Thread pools are optimized.
- Bug fixed:

  The bug that causes the system to incorrectly count the number of waits when commands are read from a client is fixed.

20200430

- The data protection feature is introduced. This feature supports the customization of security policies that are used to manage the permissions on DROP and TRUNCATE statements. This allows you to avoid data losses that are caused by the unintentional execution of these statements. For more information, see Data Protect.
- The mdl_info table is provided to store information about metadata locks.
- The bug that causes conflicts when the thread pool and ic_reduce features are both enabled is fixed.

20200331

Performance optimization:

- The performance of thread pools with the default configuration is improved.
- The output of TCP errors is disabled by default.

20200229

- New feature:

  The read/write splitting function is supported for database proxies.
- Performance optimization:
  - Thread pools are optimized.
  - The time that is required to execute a PAUSE statement is reduced in various CPU architectures.
- Bug fixed:

  The bug that causes the system to partially commit XA transactions is fixed.

20200110

- New feature:

  The thread pool feature is introduced to separate threads from sessions. If a large number of sessions exist, the system can run a small number of threads to complete the tasks in active sessions. For more information, see Thread Pool.
- Performance optimization:

  The file deletion mechanism is optimized. When you asynchronously delete small files, links to the small files are canceled.

- Bugs fixed:
  - The bug that causes the system to incorrectly calculate the sleep time of the page cleaner is fixed.
  - The bug that causes the `SELECT @@global.gtid_executed` statement to incur a failover failure is fixed.
  - The bug that causes the IF CLIENT KILLED AFTER ROLLBACK TO SAVEPOINT PREVIOUS STMTS COMMITTED error is fixed.

20191212

Performance optimization:

The deletion of unnecessary TCP error logs is supported.

20191115

Bug fixed:

The bug that causes the overflow of timestamps in slow query logs is fixed.

20191101

Bugs fixed:

- The bug that causes the system to rotate slow query logs when you update common logs is fixed.
- Some display-related bugs are fixed.

20191015

- New features:
  - The rotation of slow query logs is supported. Every CSV slow query log file is assigned a unique name and a new file. This prevents data losses during the collection of slow query logs. You can run the `show variables like '%rotate_log_table%';` command to check whether the rotation of slow query logs is enabled.
  - A new SM4 encryption algorithm is introduced to replace the original SM4 encryption algorithm.
  - The Purge Large File Asynchronously feature is introduced to asynchronously delete files. Before you delete a tablespace, the system renames the files in the tablespace as temporary files. Then, a background thread is started to asynchronously delete the temporary files. For more information, see Purge Large File Asynchronously.
  - The output of TCP errors is supported. TCP errors in read, read-wait, and write-wait events are returned with their error codes by using end_connection events. In addition, logs with information about the errors are generated.
  - An SQL log caching mechanism is introduced to increase SQL logging performance.
- Bugs fixed:
  - The bug that prevents responses to the pstack command when a large number of connections are established is fixed. This is implemented by disabling the pstack command.
  - The bug that causes conflicts between implicit primary keys and `CREATE TABLE AS SELECT` statements is fixed.
  - The bug that prevents the system from deleting the temporary files that are created from binary log files is fixed.

20190815

An optimized instance locking mechanism is introduced. You can delete tables from an RDS instance by using DROP or TRUNCATE statements even if the RDS instance is locked.

20190130

Bugs that compromise database stability are fixed.

20181010

The rocksdb_ddl_commit_in_the_middle parameter is introduced to MyRocks. If this parameter is set to on, some DDL statements call the COMMIT operation when they are executed.

201806** (ApsaraDB RDS for MySQL 5.6.16)

New feature: Microsecond-level time precision is supported for slow query logs.

20180426 (ApsaraDB RDS for MySQL 5.6.16)

- Invisible indexes are supported. For more information, see AliSQL 5.6.32 Release Notes (2017-07-16).
- The bug that causes the system to apply threads on secondary RDS instances is fixed.
- The bug that compromises database performance when updates to partitioned tables are applied on secondary RDS instances is fixed.
- The bug that causes TokuDB to rebuild tables on which ALTER TABLE COMMENT statements are executed is fixed. For more information, see AliSQL 5.6.32 Release Note (2018-05-01).
- The bug that triggers deadlocks when SHOW SLAVE STATUS or SHOW STATUS statements are executed is fixed.

20171205 (ApsaraDB RDS for MySQL 5.6.16)

- The bug that triggers deadlocks when OPTIMIZE TABLE and ONLINE ALTER TABLE statements are executed at the same time is fixed.
- The bug that triggers conflicts between sequences and implicit primary keys is fixed.
- The bug that prevents the proper execution of SHOW CREATE SEQUENCE statements is fixed.
- The bug that causes the system to incorrectly collect statistics on TokuDB tables is fixed.
- The bug that triggers deadlocks when OPTIMIZE statements are executed in parallel on tables is fixed.
- The bug that causes the system to incorrectly record character sets in QUERY_LOG_EVENT is fixed.
- The bug that prevents an RDS instance from stopping due to signal processing issues is fixed. For more information, see AliSQL 5.6.32 Release Notes (2017-10-10).
- The bug that is caused by the execution of RESET MASTER statements is fixed.
- The bug that causes secondary RDS instances to be in a constant waiting state is fixed.
- The bug that prevents the system from updating the status of primary and secondary RDS instances after primary/secondary switchovers is fixed in the RDS Enterprise Edition.
- The bug that causes the database process to unexpectedly exit due to the execution of SHOW CREATE TABLE statements is fixed.

20170927 (ApsaraDB RDS for MySQL 5.6.16)

The bug that causes the system to query tables from TokuDB based on incorrect indexes is fixed.

20170901 (ApsaraDB RDS for MySQL 5.6.16)

- New features:
  - The upgrade of SSL encryption to TLS 1.2 is supported. For more information, see AliSQL 5.6.32 Release Notes (2017-10-10).
  - Sequences are supported.
- Bug fixed: The bug that causes the system to return an incorrect result set for the NOT IN operator is fixed.

20170530 (ApsaraDB RDS for MySQL 5.6.16)

New feature: The privileged account of an RDS instance is granted the permissions to close the connections that are established by all of the standard accounts created on the RDS instance.

20170221 (ApsaraDB RDS for MySQL 5.6.16)

New feature: The read/write splitting function is supported.For more information, see Read/write splitting overview.

ApsaraDB RDS for MySQL 5.5

20181212

The bug that causes the `gettimeofday(2)` function to return an incorrect time value is fixed. The returned time value is used to calculate the timeout period. If the returned time value is incorrect, some operations never time out.

3.X-Engine

3.1. X-Engine overview

X-Engine is an online transaction processing (OLTP) database storage engine that is developed by the Database Products Business Unit of Alibaba Cloud to suit the needs of PolarDB. This storage engine now is widely used in a number of business systems of Alibaba Group to reduce costs. These include the transaction history database and DingTalk chat history database. In addition, X-Engine is a crucial database technology that empowers Alibaba Group to withstand bursts of traffic that may surge by hundreds of times than usual during Double 11, a shopping festival in China.

Background information

X-Engine aims to cope with the challenges faced by the internal businesses of Alibaba Group. Alibaba Group has been deploying MySQL databases on a large scale since 2010. However, the explosive growth of data volume year by year still imposes the following challenges on these databases:

- To process highly concurrent transactions.
- To store large amounts of data.

You can increase the processing and storage capabilities by adding servers on which you can create more databases. However, this is not an efficient approach. Alibaba Cloud has been devoted to leveraging technical means to maximize performance with minimal resources.

The performance of the conventional database architecture has been carefully studied. Michael Stonebreaker, a leader in the database field and a winner of the Turing Award, wrote a paper on this topic: *OLTP Through the Looking Glass, and What We Found There* .In the paper, he pointed out that conventional general-purpose relational databases spend less than 10% of their time efficiently processing data. The remaining 90% of their time is wasted on other work, such as waiting for locked resources to be released, managing buffers, and synchronizing logs.

This is caused by significant changes to the hardware systems that we depend on in recent years. These include multi-core and many-core CPUs, new processor architectures such as the cache-only memory architecture (COMA) and the non-uniform memory access (NUMA), various heterogeneous computing devices such as GPUs and field-programmable gate arrays (FPGAs). However, the database software built on these hardware systems has not changed much. Such software includes the mechanism that fixes page sizes based on B-tree indexing, the mechanism that processes transactions and restores data by using the recovery and isolation exploiting semantics (ARIES) algorithms, and the concurrency control mechanism based on independent lock managers. These software mechanisms are designed based on slow disks and therefore cannot achieve the potential performance of the preceding hardware systems.

Alibaba Cloud has developed X-Engine to suit the needs of the hardware systems used today.

Architecture

With the pluggable storage engine of MySQL, X-Engine can be seamlessly integrated with MySQL and benefit from the tiered storage architecture.

X-Engine is designed to store large amounts of data, increase the capability of processing concurrent transactions, and reduce storage costs. In most scenarios with large amounts of data, the data is not evenly accessed. Frequently accessed data (hot data) actually accounts for a small proportion. X-Engine divides data into multiple levels based on access frequency. In addition, X-Engine determines storage structures and writes the data to appropriate storage devices based on the access characteristics of each level of data.

X-Engine uses the log-structured merge-tree (LSM tree) architecture that is redesigned for tiered storage.

- X-Engine stores hot data and updated data in the memory by leveraging a number of memory database technologies to expedite the execution of transactions. These technologies include lock-free data structures and append-only data structures.
- X-Engine uses a transaction processing pipeline mechanism to run transaction processing stages in parallel, which greatly increases the throughput.
- Less frequently accessed data (cold data) is gradually deleted or merged into persistent storage levels, and stored in the hierarchical system with abundant storage devices, such as NVMs, SSDs, and HDDs.
- A lot of improvements are made to compactions that impose a significant impact on performance.
  - The data storage granularity is refined based on the fact that data update hotspots are concentrated. This ensures that data is reused as much as possible in the compaction process.
  - The hierarchy of the LSM tree is refined to reduce I/O and computing costs and to minimize the storage space usage incurred by compactions.
- More fine-grained access control and caching mechanisms are used to optimize read performance.

> **Note** The architecture and optimization technologies of X-Engine are summarized into a paper titled *X-Engine: An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing* .This paper was presented at the 2019 SIGMOD Conference, the top conference in the database field. This was the first time that a company from mainland China published technological achievements in OLTP database kernels at a top international conference.

Highlights

- FPGA hardware is used to accelerate compactions and further maximize the performance of your database system. This marks the first time that hardware acceleration is applied to the storage engine of an OLTP database. The achievement has been summarized into a paper titled *FPGA-Accelerated Compactions for LSM-based Key Value Store* .This paper has been accepted by the 18th USENIX Conference on File and Storage Technologies (FAST'20).
- The data reuse technology is used to reduce the costs of compactions and reduces performance jitter caused by data deletion from the cache.
- Queued multi-transaction processing and pipeline processing are used to reduce the thread context switching overheads and calculate the task ratio in each stage. This makes the entire pipeline work evenly and increases transaction processing performance by more than 10 times compared with other similar storage engines such as RocksDB.
- Copy-on-write is used to prevent data pages from being updated when they are read. This

allows read-only data pages to be encoded and compressed and reduces the storage space usage by 50% to 90% compared with conventional storage engines, such as InnoDB.

- Bloom filter is used to quickly determine whether the target data exists, succinct range filter (SuRF) is used to determine whether the range data exists, and row cache is used to cache hot data rows to accelerate read operations.

Basic logic of LSM

The essence of LSM is that all write operations append data to the memory. Each time the written data is accumulated to a certain amount, the data is frozen as a level and then flushed to persistent storage. All rows of the written data are sorted based on primary keys, regardless of whether the data is stored in the memory or persistent storage. In the memory, data is stored in a sorted in-memory data structure, such as a skip list or B-tree. In persistent storage, data is stored in a read-only, fully sorted persistent storage structure.

To make a common storage engine support transaction processing in common storage systems, you must introduce a temporal factor, based on which we can build an independent view for each transaction. These views are not affected in the event of concurrent transactions. For example, the storage engine sorts the transactions, assigns them sequence numbers (SNs) that increase monotonically and globally, and logs the SN of each transaction. This allows the storage engine to determine visibility among independent transactions.

If data is continuously written to the LSM storage structure and none of other actions is performed, the LSM storage structure will eventually become the structure shown in the following figure.

This structure is writer-friendly, because the written data simply has to be appended to the latest memory table. To implement crash recovery, you only need to record the data to redo logs. New data does not overwrite old data, and therefore appended records form a natural multi-SN structure.

However, when more persistence levels of data are accumulated and frozen, query performance decreases. The multi-SN records generated for different transaction commits with the same primary key are distributed across different levels, as are those with different keys. In this case, read operations need to search all the levels of data and merge the found data to obtain the final results.

Compactions are introduced to LSM to resolve this problem. Compactions in LSM have two objectives:

- Control the hierarchy of LSM

  In most cases, the data volume increases by a multiple number of times as the LSM level decreases. This is to improve read performance.

  Data access in a storage system is localized, and a large proportion of access traffic is concentrated on a small portion of data. This is the basic prerequisite for effective operations in the cache system. In the LSM storage structure, you can store hot data at a high LSM level on high-speed storage devices, such as NVMs and DRAMs, and cold data at a low level on low-speed storage devices that cost less. This is the basis of hot and cold data separation in X-Engine.

- Merge data

Compactions continuously merge data at adjacent LSM levels and write the merged data to the lower LSM levels. During the compaction process, the system reads the to-be-merged data from two or more adjacent levels and then sort the data based on keys. If multiple records with the same key have different SNs, the system retains only the record with the latest SN (which is greater than the earliest SN of the current transaction that is being executed), discards the records with earlier SNs, and writes the record with the latest SN to a new level. This process is resource-consuming.

In addition to the separation of hot and cold data, compactions also require considerations on other factors such as the data update frequency. Queries for a large number of multi-SN records waste more I/O and CPU resources. Therefore, records that have the same key but different SNs must be preferably merged to reduce the number of SNs per record. Alibaba Cloud designs a proprietary compaction scheduling mechanism for X-Engine.

A highly optimized LSM

In X-Engine, lock-free skip lists are used in the memory tables, which expedites the execution of highly concurrent read and write queries. A proper data structure must be planned at each LSM level to ensure efficient organization of data at persistent levels.

- Data structuring

  In X-Engine, each level is divided into fixed-sized extents. An extent stores the data with a continuous key range at the level. A set of meta indexes are created for the extents at each level. All of these indexes together with all of the active and immutable memory tables form a metadata tree. The metadata tree has a structure similar to the structure of the B-tree, and its root nodes are metadata snapshots. The metadata tree helps quickly locate extents.

  Except for the active memory tables to which data is being written, all structures in X-Engine are read-only and cannot be modified. When a point in time is specified, for example, when the log sequence number (LSN) is 1000, the structure referenced by metadata snapshot 1 in the preceding figure contains the snapshots of all the data logged at the moment associated with the LSN 1000. This is also why this structure is called a snapshot.

  The metadata structure itself does not change after it is generated. All read operations start from this snapshot structure. This is the basis on which X-Engine implements snapshot-level isolation. All operations such as compactions and memory table freezes are implemented using copy-on-write. Specifically, the result of each modification is written into a new extent. Then, a new meta index structure is generated. Finally, a new metadata snapshot is generated.

  For example, each compaction generates a new metadata snapshot, as shown in the following figure.

  In this example, metadata snapshot 2 is slightly different from metadata snapshot 1. Only some leaf nodes and index nodes that have changed are modified.

  > **Note** This data structuring technology is similar to that presented in the paper titled B-trees, Shadowing, and Clones, which will help you understand this process.

- Transaction processing

  With its lightweight write mechanism, LSM has significant advantages in write operations. However, transaction processing is not as simple as writing updated data to a system. A complex process is required to ensure atomicity, consistency, isolation, and durability (ACID). X-Engine divides the entire transaction processing process into two phases:

  i. The read and write phase

     In the read and write phase, X-Engine checks for write-write conflicts and read-write conflicts in the transaction and determines whether the transaction can be executed, rolled back, or locked. If no transaction conflicts are detected, all the modified data is written to the transaction buffer.

  ii. The commit phase

     The commit phase includes the entire process of writing data to WALs, writing data to memory tables, committing the data, and returning results to the user. This process involves both I/O operations (logging and returning results) and CPU operations (copying logs and writing data to memory tables).

  To increase the throughput during transaction processing, the system concurrently processes a large number of transactions. A single I/O operation is costly, and therefore most storage engines tend to commit a number of transactions at a time, which is called "group commit". This allows you to combine I/O operations. However, the transactions that are to be committed at a time still need to wait for a long period of time. For example, when logs are being written to a disk, nothing else is done except waiting for the data to be flushed to disks.

  To further increase the throughput during transaction processing, X-Engine adopts a pipeline technology that divides the commit phase into four independent and more fine-grained stages:

  i. Copying logs to the log buffer
  ii. Flushing logs to disks
  iii. Writing data to memory tables
  iv. Committing the data

  When a transaction commit thread enters the commit phase, it can freely choose any stage of the pipeline to process the data. In this way, threads can concurrently process data at different stages. If the tasks for each stage are properly divided based on sizes, all the stages of the pipeline can be nearly fully loaded. In addition, transaction processing threads instead of background threads are used in the commit phase. Each stage is either executing tasks in a stage or processing requests. This process does not involve any waiting or switching, and therefore the capabilities of each thread are fully utilized.

- Read operations

  In LSM, if multiple records with the same key have different SNs, the records with later SNs are appended to the record with the earliest SN. Records that have the same key but different SNs may be stored at different levels. The system must identify the appropriate SN of each requested record in compliance with the visibility rules that are defined based on the transaction isolation levels. In most cases, the system searches for records with the latest SNs from the highest level to the lowest level.

For single-record queries, the query process ends after the single record is found. If the record is located at a high level, for example, in a memory table, it will be returned quickly. If the record is located at a low level, for example, at a level used for random reading, the system must search downwards level by level. In this case, a bloom filter can be used to skip some levels to expedite the query, but this involves more I/O operations. A row cache is introduced into X-Engine to expedite single-row queries. The row cache stores data above all the persistent data levels. When a single-row query does not hit data in memory tables, it will hit data in the row cache. The row cache needs to store each record with the latest SN at all persistence levels. However, the records in the row cache may change. For example, every time after a read-only memory table is flushed to a persistence level, the records in the row cache must be updated accordingly. This operation is subtle and requires careful design.

For range scans, it is impossible to determine the level where the data associated with a specific key range is stored. In this case, the final result can be returned only after all levels are scanned for the data and the data is merged. X-Engine adopts a series of methods to address this problem. For example, SuRF presented at the best paper at SIGMOD 2018 provides a range scan filter to reduce the number of levels to be scanned. The asynchronous I/O and prefetching mechanism are also provided to address this problem.

The core to read operations is the cache design. A row cache handles single-row queries. A block cache handles requests missed by the row cache or range scan requests. However, in LSM, a compaction incurs updates to a large number of data blocks at a time. This causes a large amount of data in the block cache to expire instantly and results in a sharp performance jitter. The following optimizations are made to address this problem:

- Reduces the granularity of compaction.
- Reduces the amount of data modified during each compaction.
- Updates the existing cached data only when the data is modified during each compaction.

- Compaction

  Compactions are important. The system needs to read data associated with overlapped key ranges from adjacent levels, merge the data, and write the merged data to a new level. This is the cost of simple write operations. The storage architecture of X-Engine is redesigned to optimize compactions.

  As mentioned previously, X-Engine divides each level of data into fixed-sized extents. An extent is equivalent to a small but complete Sorted String Table (SSTable), which stores the data with a continuous key range at the level. A key range is further divided into smaller continuous segments that are called data blocks. Data blocks are equivalent to pages in conventional databases, except that data blocks are read-only and their lengths are not fixed.

  A comparison between metadata snapshots 1 and 2 unveils the intent of the extent design. Only a small portion of overlapped data and the meta index node need to be modified each time. The structures of metadata snapshots 1 and 2 actually share a large number of data structures. This is called data reuse, and the extent size is a crucial factor that determines the data reuse rate. As a completely reusable physical structure, the extent size is minimized to reduce the amount of overlapped data. However, the extent size must be proper. If the extent size is abnormally small, a large number of indexes will be required, which increases management costs.

In X-Engine, the data reuse rate is high in compactions. Assume that you want to merge the extents that contain overlapped key ranges at Level 1 and Level 2. In this case, the merge algorithm scans the data row by row. Any physical structure, including data blocks and extents, that does not overlap with the data at other levels can be reused. The difference between the reuse of extents and the reuse of data blocks is that the meta indexes of extents can be modified while data blocks only support data copying. Data blocks are not recommended although they significantly reduce CPU utilization.

The following figure shows a typical data reuse process in a compaction.

The data reuse process is completed by using row-by-row iteration. However, this fine-grained data reuse causes data fragmentation.

Data reuse benefits the compaction itself, reduces I/O and CPU consumption during the compaction, and improves the overall performance of the system. For example, in the compaction process, data does not need to be completely rewritten, which greatly reduces the storage space occupied by written data. In addition, most of the data remains unchanged, and therefore the cached data remains valid after data updates. This reduces read performance jitters caused by the expiration of the cached data during the compaction.

In fact, optimizations to compactions are only part of what X-Engine does. X-Engine also optimizes the compaction scheduling policies and defines the method of selecting extents, the granularity of compactions, and the execution priorities of the specified compactions. These all affect the performance of the system. Although no perfect policies exist, X-Engine has accumulated valuable experience and defined a number of rules to define proper compaction scheduling policies.

Scenarios

For more information, see Best practices of X-Engine.

Get started with X-Engine

For more information, see Usage notes.

Follow-up development

As a storage engine for MySQL, X-Engine must be continuously improved in terms of its compatibility with MySQL systems. Based on the most urgent needs, some features such as foreign keys will be gradually enhanced, and more data structures and index types will be supported.

The core value of X-Engine lies in cost-effectiveness. Continuously improving performance at lower costs is a long-term fundamental goal. Alibaba Cloud continues its exploration for new approaches that make X-Engine more efficient on operations, such as compaction scheduling, cache management and optimization, data compression, and transaction processing.

X-Engine will not be limited to a storage engine for standalone databases. It will serve as the core of the Alibaba Cloud proprietary distributed PolarDB to provide enterprise-grade database services.

3.2. Usage notes

This topic describes the X-Engine storage engine supported by ApsaraDB RDS for MySQL. This engine can process transactions and reduce disk usage.

Introduction

X-Engine is an online transaction processing (OLTP) database storage engine developed by the Database Products Business Unit of Alibaba Cloud to suit the needs of ApsaraDB PolarDB. This storage engine is widely used in many business systems of Alibaba Group to reduce costs. These systems include the transaction history database and DingTalk chat history database. In addition, X-Engine is a crucial database technology that empowers Alibaba Group to withstand bursts of traffic that may surge to hundreds of times greater than usual during the Double 11 shopping festival in China.

X-Engine is optimized for large-scale e-commerce transaction processing. The paper *X-Engine: An Optimized Storage Engine for Large-scale E-Commerce Transaction Processing* written by the X-Engine R&D team describes the pioneering work X-Engine has achieved in the database engine field. This paper was accepted by the Industrial Track of SIGMOD 2019.

Unlike the InnoDB storage engine, X-Engine adopts the log-structured merge-tree (LSM tree) architecture for tiered storage. LSM tree has the following significant advantages:

- The small size of hotspot datasets that require indexes improves write performance.
- The bottom-layer persistent data pages are read-only. In addition, they are stored in a compact format and are compressed by default to reduce storage costs.

In addition to the advantages of LSM tree, X-Engine brings the following innovations in engineering implementation:

- Continuously optimized write performance: Continuous optimization allows X-Engine to deliver write performance that is over 10 times higher than the write performance of RocksDB that runs in the LSM tree architecture.
- Data reuse at the storage layer: Data reuse optimizes the performance of compaction operations and reduces the impact of compaction operations on system resources in the traditional LSM tree architecture. This allows you to keep system performance stable.
- Hybrid storage: You can deploy various storage media, such as SSDs and HDDs. These storage media provide different I/O capabilities on the same RDS instance. The hybrid storage architecture works with the tiered storage architecture of X-Engine to intelligently store hot and cold data separately. This allows you to reduce overall costs without compromising performance.
- Multi-level caching, refilling, and prefetching: These allow X-Engine to use the fine-grained access mechanism and cache technology to make up for the read performance shortcomings of the engines that adopt the LSM tree architecture.

The preceding optimizations make X-Engine an alternative to the traditional InnoDB storage engine. In addition to supporting transactions, X-Engine can also reduce occupied storage space by up to 90% and thus lower storage costs. X-Engine is especially suitable for businesses that have a large data volume and require high read/write performance.

> Note For more information about the use scenarios of X-Engine, see Best practices of X-Engine.

Prerequisites

Your RDS instance runs MySQL 8.0 on High-availability Edition or Basic Edition.

Purchase an RDS instance that uses X-Engine

If you want to use X-Engine for your RDS instance, select MySQL 8.0 for Database Engine on the **Basic Configuration** page and select **X-Engine (Low Cost)** for Storage Engine on the **Instance Configuration** page when you purchase an RDS instance. For more information about other parameters, see Create an ApsaraDB RDS for MySQL instance.

Note

- If you want to use X-Engine for an RDS instance that runs MySQL 5.5, 5.6, or 5.7, you must migrate the data of the RDS instance to a new RDS instance that runs MySQL 8.0. For more information, see Migrate data between RDS instances.
- If you want to convert the storage engine of an RDS instance to X-Engine, see Convert the storage engine from InnoDB, TokuDB, or MyRocks to X-Engine.

Create X-Engine tables

If you select X-Engine when you create an instance, the table created within the instance uses X-Engine by default. Execute the following statement to view the default engine used by an instance:

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

If the default engine is X-Engine, you do not need to specify the storage engine in the table creation statement.

After you have created a table, data is stored in X-Engine.

Note You can create tables that use the InnoDB engine in an instance that uses X-Engine. If you use Data Transmission Service (DTS) to migrate an InnoDB table to an X-Engine instance, the destination table also uses InnoDB. For more information, see Solution 2 in Convert the storage engine from InnoDB, TokuDB, or MyRocks to X-Engine.

Limits

- Limits on resource allocations if X-Engine and InnoDB are used together

  When you use X-Engine for an instance, 95% of the memory is used as the write cache and block cache to speed up reading and writing. The InnoDB buffer pool does not occupy much memory. Do not use tables that use InnoDB to store a large volume of data within an instance that uses X-Engine. Otherwise, the X-Engine performance may deteriorate due to a low cache hit ratio. We recommend that you use only a single engine type for tables within an ApsaraDB for RDS instance that runs MySQL 8.0.

- Limits on engine features

  X-Engine has some limits on features. Some features are in development. Other features that are not listed in the following table are the same as those of InnoDB.

Category	Feature	X-Engine	Remarks
SQL features	Foreign key	Not supported.	None
	Temporary table	Not supported.	None
	Partition table	Not supported. X-Engine does not support the creation, addition, deletion, modification, or query of partitions.	None
	Generated Column	Not supported.	None
	Handler API	Not supported.	None
Column properties	Maximum column size (LONGBLOB/LONGTEXT/JSON)	32 MB	None
	GIS data type	Not supported. X-Engine does not support the following GIS data types: GEOMETRY, POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, and GEMOMETRYCOLLECTION.	None
Indexes	Hash index	Not supported.	None
	Spatial index	Not supported. X-Engine does not support the creation and use of full-text indexes.	None
Transactions	Transaction isolation level	Two isolation levels are provided: ○ Read Committed (RC) ○ Repeatable Read (RR)	None
	Maximum transaction size	32 MB	Support for larger transactions is under development.
	Savepoint	Not supported.	None

Category	Feature	X-Engine	Remarks
	XA transaction	Not supported.	Support for XA transactions is under development.
Locks	Lock granularity	○ Table-level locks supported. ○ Row-level locks supported. ○ Gap locks not supported.	None
	Skip Locked Lock Nowait	Not supported.	None
Character sets	Character sets supported by non-indexed columns	Supported.	None
	Character sets supported by indexed columns	○ Latin1 (latin1_bin) ○ GBK (gbk_chinese_ci and gbk_bin) ○ UTF-8 (utf8_general_ci and utf8_bin) ○ UTF-8MB4 (utf8mb4_0900_ai_ci, utf8mb4_general_ci, and utf8mb4_bin)	None
Primary/secondary replication	Binary log formats	stmt/row/mixed **Note** The default binary log format is the row-based format. The statement-based and row-based log formats may lead to data security issues in specific concurrency scenarios.	None

- Limits on large transactions

  X-Engine does not support large transactions. If a transaction modifies a large number of rows, X-Engine uses the commit in middle feature. For example, if you use a transaction to modify more than 10,000 rows, X-Engine commits this transaction and starts a new transaction. However, the commit in middle feature cannot strictly follow atomicity, consistency, isolation, durability (ACID). Exercise caution when you use the commit in middle feature. Examples:

- Start a transaction to insert more than 10,000 rows. During the insertion, a portion of the committed data can be queried by other requests.
- Start a transaction to modify more than 10,000 rows. If a portion of the data is committed in the middle of the transaction, you cannot roll the transaction back.

```
drop table t1;
create table t1(c1 int primary key , c2 int)ENGINE=xengine;
begin;
call insert_data(12000); // 12,000 rows is inserted, and a commit in middle operation is triggered.
As a result, the first 10,000 rows of data are committed.
rollback;// Only the last 2,000 rows can be rolled back.
select count(*) from t1; // The committed 10,000 rows of data can be queried.
+----------+
| count(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.00 sec)
```

- Start a transaction to delete or modify more than 10,000 rows. Some rows are omitted.

```
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;// The deletion triggers a commit in middle operation, and the two r
ows of data inserted by the current transaction are not deleted.
commit;
select * from t1;
+-------+-------+
| c1    | c2    |
+-------+-------+
| 10001 | 10001 |
| 10002 | 10002 |
+-------+-------+
2 rows in set (0.00 sec)
```

Parameters

Note When you create an RDS instance, you can select X-Engine as the default storage engine. You can also adjust the parameter template based on the parameters described in the following table to suit your business requirements. For more information, see Create an ApsaraDB RDS for MySQL instance.

Category	Parameter	Description	Remarks
Performance	xengine_arena_block_size	The unit used when a memory table requests new memory from the operating system and the external memory management system of jemalloc.	Read-only after startup.
	xengine_batch_group_max_group_size	The maximum number of groups of a transaction pipeline.	Read-only after startup.
	xengine_batch_group_max_leader_wait_time_us	The maximum wait time of a transaction pipeline.	Read-only after startup.
	xengine_batch_group_slot_array_size	The maximum batch size of a transaction pipeline.	Read-only after startup.
Memory	xengine_block_cache_size	The size of the read block cache.	This parameter cannot be modified.
	xengine_row_cache_size	The size of the row cache.	This parameter cannot be modified.
	xengine_write_buffer_size	The maximum size of a single memory table.	This parameter cannot be modified.
	xengine_block_size	The size of the data block on a disk.	Read-only after initialization. Read-only after startup.
	xengine_db_write_buffer_size	The maximum size of the active memory tables in all subtables.	This parameter cannot be modified.

Category	Parameter	Description	Remarks
	xengine_db_total_write_buffer_size	The maximum size of the active memory tables and immutable memory tables in all subtables.	This parameter cannot be modified.
	xengine_scan_add_blocks_limit	The number of blocks that can be added to the block cache during each range-based scan request.	This parameter cannot be modified.
compaction	xengine_flush_delete_percent_trigger	If the number of records in a memory table exceeds the value of this parameter, the xengine_flush_delete_record_trigger parameter takes effect on the memory table.	None
Lock	xengine_max_row_locks	The maximum number of rows that can be locked in a single SQL request.	This parameter cannot be modified.
	xengine_lock_wait_timeout	The timeout period of lock wait.	This parameter cannot be modified.

Running status metrics

The following table shows the running status metrics of X-Engine. You can view the metrics on the Monitoring page.

Metric	Description
xengine_rows_deleted	The number of rows deleted.
xengine_rows_inserted	The number of rows written.
xengine_rows_read	The number of rows read.
xengine_rows_updated	The number of rows updated.
xengine_system_rows_deleted	The number of deletion operations on an X-Engine system table.
xengine_system_rows_inserted	The number of insert operations on an X-Engine system table.
xengine_system_rows_read	The number of read operations on an X-Engine system table.

Metric	Description
xengine_system_rows_updated	The number of updates on an X-Engine system table.
xengine_block_cache_add	The number of add operations on the block cache.
xengine_block_cache_data_hit	The number of hits in read data blocks.
xengine_block_cache_data_miss	The number of misses in read data blocks.
xengine_block_cache_filter_hit	The number of hits in filter blocks.
xengine_block_cache_filter_miss	The number of misses in filter blocks.
xengine_block_cache_hit	The number of hits in the block cache. The value of this metric is calculated by using the following formula: The value of this metric= The value of the data_hit metric + The value of index_hit metric.
xengine_block_cache_index_hit	The number of hits in index blocks.
xengine_block_cache_index_miss	The number of misses in index blocks.
xengine_block_cache_miss	The number of misses in the block cache. The value of this metric is calculated by using the following formula: The value of this metric= The value of the data_hit metric + The value of the index_hit metric.
xengine_block_cachecompressed_miss	The number of misses in the compressed block cache.
xengine_bytes_read	The number of bytes on the read physical disk.
xengine_bytes_written	The number of bytes that are added to the block cache.
xengine_memtable_hit	The number of hits in memory tables.
xengine_memtable_miss	The number of misses in memory tables.
xengine_number_block_not_compressed	The number of uncompressed blocks.
xengine_number_keys_read	The number of times that keys are read.
xengine_number_keys_updated	The number of times that keys are updated.
xengine_number_keys_written	The number of times that keys are written.
xengine_number_superversion_acquires	The number of times that Superversion is applied for references.

Metric	Description
xengine_number_superversion_cleanups	The number of times that Superversion is cleared. If Superversion is not referenced, it is cleared.
xengine_number_superversion_releases	The number of times that the referenced Superversion is released. If Superversion is not referenced, it is cleared.
xengine_snapshot_conflict_errors	The number of times that an error is reported due to snapshot version conflicts at the RR isolation level.
xengine_wal_bytes	The size of redo logs that are flushed into the disk. Unit: bytes.
xengine_wal_group_syncs	The number of times that GroupCommit is executed by redo logs.
xengine_wal_synced	The number of times that redo logs are synchronized.
xengine_write_other	The number of times that a follower commits transactions in a transaction pipeline.
xengine_write_self	The number of times that a leader commits transactions in a transaction pipeline.
xengine_write_wal	The number of times that redo logs are written.

3.3. Convert the storage engine from InnoDB, TokuDB, or MyRocks to X-Engine

ApsaraDB RDS for MySQL 8.0 supports X-Engine. X-Engine provides better data compression capabilities and reduces disk space costs. This topic describes how to convert the storage engine from InnoDB, TokuDB, or MyRocks to X-Engine.

Context

X-Engine is an online transaction processing (OLTP) database storage engine that is developed by Alibaba Cloud to suit the needs of PolarDB. This storage engine now is widely used in a number of business systems of Alibaba Group to reduce costs. These include the transaction history database and DingTalk chat history database. In addition, X-Engine is a crucial database technology that empowers Alibaba Group to withstand bursts of traffic that may surge by hundreds of times than usual during Double 11, a shopping festival in China.

For more information, see Best practices of X-Engine.

Note When you create an RDS instance designed to run MySQL 8.0, we recommend that you specify X-Engine as the default storage engine. You can also specify X-Engine for the RDS instance after instance creation by setting the engine parameter to xengine.

Precautions

- If the table you want to convert uses InnoDB, make sure that the remaining disk space of your RDS instance is twice the data volume of the table before the conversion. After the conversion to X-Engine, the disk space occupied by the data in the table decreases to 10% to 50% of the disk space occupied before the conversion.
- If you use solution 1 described in the following sections to perform the conversion, you must reconfigure parameters and restart your RDS instance. Stop your database services before you perform the conversion.
- If you use solution 2 described in the following sections to migrate all data from your RDS instance to a new RDS instance, you must update the endpoints on your application. We recommend that you perform this migration during off-peak hours.
- Before you convert the storage engine of your online business, make sure that X-Engine is compatible with SQL.
- Change the value of the **default_storage_engine** to **xengine** after the conversion. This ensures that all of the tables created later use X-Engine.

Solution recommendations

- If your RDS instance runs MySQL 8.0 (with a kernel version of 20200229 or later), we recommend that you use Solution 1. This way, you do not need to configure various tools.

  Note If the kernel version of your RDS instance does not meet requirements, you can update the kernel version on the **Basic Information** page. In the Configuration Information section, check whether the Upgrade Version button exists. If the button exists, click it to view and update the kernel version. If the button does not exist, you are already using the latest kernel version. For more information, see Upgrade the minor engine version of an ApsaraDB RDS for MySQL instance.

- If your RDS instance runs MySQL 5.6 or 5.7, we recommend that you use Solution 2.

Solution 1

This solution allows you to enable X-Engine by using a parameter template and then use DDL statements to convert the storage engine to X-Engine. This solution is easy and fast, but requires you to restart your RDS instance. In addition, data manipulation language (DML) operations may be blocked, and migrating large-sized tables is time-consuming.

1. Log on to the ApsaraDB for RDS console.
2. In the top navigation bar, select the region where the target RDS instance resides.
3. Find the target instance and click its ID.

4. In the left-side navigation pane, click **Parameters**.
5. In the upper-left corner of the **Editable Parameters** tab, click Apply Template. In the dialog box that appears, select **MySQL_8.0_X-Engine_High-availability_Default Parameter Template** from the Apply Template drop-down list and click **OK**.

   > **Note** This operation restarts the instance. After the restart, 95% of the memory resources are allocated to X-Engine. Do not use X-Engine and InnoDB at the same time.

6. Use DMS to log on to an ApsaraDB for RDS instance.
7. In the top navigation bar, choose **SQL Operations** > **SQL Window**.
8. Run the following command:

   ```
   alter table <The name of the target database>. <The name of the target database> engine xengi
   ne;
   ```

   Example:

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

Solution 2

This solution allows you to synchronize the data of a table in real time from your original RDS instance to a new RDS instance by using Alibaba Cloud Data Transmission Service (DTS). After the data synchronization is complete, you can switch your business to the new RDS instance.

> **Note** The new RDS instance inherits the storage engine of your original RDS instance by default. You must export the SQL statements for table creations, and change the storage engine to X-Engine in the SQL statements. Then, migrate the data to the new X-Engine table.

1. Perform the following steps to export scripts of all schemas of the original RDS instance.
   i. Log on to the original RDS instance by using Alibaba Cloud Data Management (DMS).
   ii. In the top navigation bar, choose **Data Operation** > **Export**.
   iii. Choose **New** > **Export Database**.
   iv. Configure parameters as prompted to export scripts of all schemas and click **OK**. In the dialog box that appears, click **Yes**.

   > **Note** The dialog box is displayed because **Extended Options** is selected in the **Advanced Options** dialog box. You can ignore it.

2. Decompress the schema scripts and change InnoDB or TokuDB to X-Engine.

3. Purchase a new RDS instance that runs MySQL 8.0 and has the same specifications as the original RDS instance. Make sure that you select the X-Engine parameter template.

   > **Note** When you create an RDS instance, you can use the default X-Engine parameter template to specify X-Engine as the default storage engine. For more information, see Create an ApsaraDB RDS for MySQL instance.

4. Perform the following steps to import the schema scripts into the new RDS instance.

   i. Log on to the new RDS instance by using DMS.

   ii. In the top navigation bar, choose **Data Operation** > **Import**.

   iii. On the page that appears, click **New Task**.

   iv. In the dialog box that appears, configure parameters as prompted and click **Start**.

   > **Note** After the scripts are imported, you can run the `show create table <The name of the target table>;` command to verify that the table uses the X-Engine storage engine.

5. Synchronize data from the original RDS instance to the new RDS instance. For more information, see Configure two-way data synchronization between ApsaraDB RDS for MySQL instances.

   > **Warning** Do not select **Initial schema synchronization** during synchronization initialization.

Result

After the synchronization is complete, you can check whether the data synchronization is successful. You can then test whether X-Engine is compatible with SQL. If X-Engine is compatible with SQL, you can convert the storage engine of your online business to X-Engine.

3.4. Benefits of X-Engine

This topic describes the benefits of X-Engine. X-Engine provides 50% lower storage costs, but can still ensure the same performance as InnoDB.

Background information

X-Engine is a storage engine that is developed by Alibaba Cloud to reduce the disk usage and overall database cost of your ApsaraDB RDS for MySQL instance. X-Engine stores data in a tiered storage architecture and uses the Zstandard algorithm to compress data at a high compression ratio. You can understand the advantages of X-Engine over InnoDB and TokuDB by comparing their storage overheads and performance.

Note The major technological innovations of X-Engine have been released at three top academic conferences: ACM SIGMOD 2019 and VLDB2020 in the database field and the USENIX Conference on File and Storage Technologies (FAST) 2020 in the storage field.

Test environment

The RDS instance used for testing is created with the rds.mysql.s3.large type and a storage capacity of 2 TB. This instance type supports four CPU cores and 8 GB of memory.

50% lower storage costs than InnoDB

The preceding figure shows a comparison of disk usage between X-Engine and InnoDB.

Both X-Engine and InnoDB use their default configurations and the default table schema provided by SysBench. Each table contains 10 million data records, and the total number of tables gradually increases from 32 to 736. As the data volume grows, the disk usage of X-Engine increases at a lower speed and drops by up to 42% compared with InnoDB. In addition, X-Engine requires less disk space to store large-sized individual data records. For example, after an image database is migrated to X-Engine, it occupies only 14% of the disk space required in InnoDB.

InnoDB does not compress data in most of its business scenarios. If data compression is enabled, the disk usage of InnoDB decreases by about 33%, but the query performance also sharply decreases. For example, the performance for updates based on primary keys decreases by about 90%. This interrupts your business. X-Engine can compress data to reduce storage costs while maintaining stable performance.

Sysbench test command:

```
#InnoDBprepare Database
sysbench/usr/share/sysbench/oltp_update_index.lua\
    --mysql-host=[RDS instance connection string]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --mysql-db=sbtest\
    --threads=32\
    --tables=[32-736]\
    --table_size=10000000\
    --mysql-storage_engine=INNODB\
    prepare

#X-Engineprepare Database
sysbench/usr/share/sysbench/oltp_update_index.lua\
    --mysql-host=[RDS instance connection string]\
    --mysql-user=sbtest\
    --mysql-password=sbtest\
    --mysql-db=sbtest\
    --threads=32\
    --tables=[32-736]\
    --table_size=10000000\
    --mysql-storage_engine=XENGINE\
    prepare
```

Lower storage overheads than TokuDB

TokuDB no longer offers low cost storage options. Also, Percona no longer offers support, maintenance, and updates for TokuDB. However, X-Engine offers lower storage overheads than TokuDB. We recommend that you change the storage engine of your ApsaraDB RDS for MySQL instance from TokuDB to X-Engine.

TokuDB uses a fractal tree structure. This structure consists of more leaf nodes than the B+-tree structure that is used by InnoDB. These leaf nodes are populated with data records and stored as data blocks. Therefore, TokuDB supports a higher data compression ratio than InnoDB. However, the fractal tree structure does not support tiered storage. The tiered storage architecture of X-Engine not only consists of blocks that are populated with data records but also offers optimized storage. This allows X-Engine to support lower storage overheads than TokuDB.

The preceding figure shows a comparison of disk usage between X-Engine and TokuDB.

A total of 32 tables are created on the RDS instance used for testing. Each table contains 100 million data records. These data records occupy 411 GB of disk space in TokuDB and 400 GB of disk space in X-Engine. This proves the improvements and advantages of X-Engine over TokuDB in terms of storage costs.

Sysbench test command:

```
#TokuDBprepare Database
sysbench/usr/share/sysbench/oltp_update_index.lua\
   --mysql-host=[RDS instance connection string]\
   --mysql-user=sbtest\
   --mysql-password=sbtest\
   --mysql-db=sbtest\
   --threads=32\
   --tables=[32-736]\
   --table_size=1000000000\
   --mysql-storage_engine=TokuDB\
   prepare

#X-Engineprepare Database
sysbench/usr/share/sysbench/oltp_update_index.lua\
   --mysql-host=[RDS instance connection string]\
   --mysql-user=sbtest\
   --mysql-password=sbtest\
   --mysql-db=sbtest\
   --threads=32\
   --tables=[32-736]\
   --table_size=1000000000\
   --mysql-storage_engine=XENGINE\
   prepare
```

Tiered storage and tiered access to increase QPS

X-Engine reduces the disk usage for cold data to lower the overall storage cost while maintaining a stable rate of queries per second (QPS) for hot data.

- The tiered storage architecture of X-Engine allows you to store hot and cold data at different tiers and compress cold data by default.
- X-Engine applies technologies such as prefix encoding to every data record. This allows you to reduce storage overheads.
- In most of the actual business scenarios, data is skewed because the volume of hot data is lower than that of cold data. Despite the skewed data, the tiered access architecture of X-Engine allows you to increase the QPS.

The preceding figure shows the performance of X-Engine for processing point queries to skewed data.

This test uses the popular method of Zipf distribution to control the degree of data skew. If the degree of data skew (namely, the Zipf factor) is high, more point queries hit hot data in the cache instead of cold data on disks. This decreases the access latency and increases the QPS. In addition, the compression of cold data only has a small impact on the QPS.

The tiered storage and tiered access architectures of X-Engine eliminate the impact of cold data on most of the SQL statements that are executed to query hot data. These architectures also increase the QPS by 2.7 times than when all data is evenly accessed.

Sysbench test command:

```
sysbench/usr/share/sysbench/oltp_point_select.lua\
   --mysql-host=[RDS instance connection string]\
   --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
```

Equally matched performance of X-Engine and InnoDB for cold data queries

X-Engine and InnoDB offer an equally matched QPS and transactions per second (TPS) for processing queries to a large volume of cold data, especially archived and historical data.

The preceding figure shows a comparison of cold data query performance between InnoDB and X-Engine.

In most of the online transactional processing (OLTP) scenarios, X-Engine and InnoDB offer equally matched performance for processing frequent updates (oltp_update_index and oltp_write_only) and point queries (oltp_point_select).

However, when X-Engine queries data for a specific time range or checks the uniqueness of a single data record, its tiered storage architecture requires a scan or access to a number of storage tiers. As a result, X-Engine processes range queries (oltp_read_only) and inserts (OLTP_insert) at a slightly lower speed than InnoDB.

X-Engine processes data reads and writes (oltp_read_write) at the same speed as InnoDB.

Sysbench test command:

```
#oltp_read_only
sysbench/usr/share/sysbench/oltp_read_only.lua\
  --mysql-host=[RDS instance connection string]\
  --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
```

Summary

X-Engine is a storage engine that is tailored to the cost-effectiveness requirements of ApsaraDB RDS for MySQL. It offers performance that is comparable to InnoDB but at lower storage costs. X-Engine has been used in a number of core businesses of Alibaba Group. These include the DingTalk chat history database, Taobao image database, and Taobao transaction history database.

Get started with X-Engine

If you are new to ApsaraDB RDS for MySQL, select X-Engine as the default storage engine when you create an instance. For more information, see Create an ApsaraDB RDS for MySQL instance.

4.Feature

4.1. Thread Pool

ApsaraDB for RDS provides the Thread Pool feature to maximize performance. This feature separates threads from sessions. It allows sessions to share threads and complete more tasks with less threads.

Benefits

By default, each session creates an exclusive thread in MySQL. If a large number of sessions are active, they will compete for resources. In addition, your database system needs to process a heavy workload of thread scheduling, and a large amount of data in the cache becomes invalid. This decreases your database performance.

The thread pool of ApsaraDB for RDS grants priorities to SQL statements based on the statement types. The thread pool also provides a concurrency control mechanism to limit the number of connections. This ensures high database performance in the event of a large number of highly concurrent connections. The benefits of the thread pool are as follows:

- When a large number of threads are running concurrently, the thread pool automatically limits the number of concurrent threads to a proper range. Within this range, your database system processes a moderate workload of thread scheduling, and most of the data in the cache remains valid.
- When a large number of transactions are executed concurrently, the thread pool automatically grants different priorities to SQL statements and transactions. Based on the priorities, the thread pool limits the number of concurrent statements and transactions separately. This mitigates resource competition.
- The thread pool grants high priorities to SQL statements that are used to manage data and ensures that these statements are preferentially executed. This delivers stable execution of operations such as connection establishment, management, and monitoring even if your database system is heavily loaded.
- The thread pool grants low priorities to complicated SQL statements that are used to query data and limits the maximum number of concurrent statements. This prevents a large number of complicated SQL statements from exhausting resources and making the database service unavailable.

Prerequisites

Your RDS instance is running MySQL 5.6, 5.7, or 8.0.

Use the thread pool

The following table describes the parameters of the thread pool. You can configure these parameters in the ApsaraDB for RDS console. For more information, see Reconfigure the parameters of an ApsaraDB RDS for MySQL instance.

Parameter	Description

Parameter	Description
thread_pool_enabled	Specifies whether to enable the Thread Pool feature. Valid values: • ON • OFF Default value: ON. Note • You can enable or disable the Thread Pool feature only by using this parameter. The **thread_handling** parameter has phased out. • Enabling or disabling the Thread Pool feature does not require an instance restart.
thread_pool_size	The number of groups in the thread pool. Default value: 4. Threads in the thread pool are evenly divided into groups and managed by group.
thread_pool_oversubscribe	The number of active threads allowed per group. Default value: 32. A thread is active if it is executing an SQL statement. However, if the SQL statement is in one of the following states, the thread is inactive: • The SQL statement is waiting for disk I/O. • The SQL statement is waiting for the involved transaction to be committed.

Query the status of the thread pool

Run the following command to query the status of the thread pool:

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

Example:

```
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)
```

The following table describes the parameters that describe the status of the thread pool.

Parameter	Description
thread_pool_active_threads	The number of active threads in the thread pool.
thread_pool_big_threads	The number of threads that are executing complicated SQL statements in the thread pool. Complicated SQL statements contain subqueries, aggregate functions, and clauses such as GROUP BY and LIMIT.
thread_pool_dml_threads	The number of threads that are executing data manipulation language (DML) statements in the thread pool.
thread_pool_idle_threads	The number of idle threads in the thread pool.
thread_pool_qry_threads	The number of threads that are executing simple SQL statements in the thread pool.
thread_pool_total_threads	The total number of threads in the thread pool.
thread_pool_trx_threads	The number of threads that are executing transactions in the thread pool.
thread_pool_wait_threads	The number of threads that are waiting for disk I/O and those that are waiting for transactions to be committed in the thread pool.

Use SysBench to test the thread pool

The following figures show comparisons of performance between business scenarios with the thread pool enabled and disabled. Based on the test results, the thread pool significantly increases your database performance in the event of a large number of highly concurrent sessions.

4.2. Statement outline

Databases may be unstable because the execution plan of SQL statements is constantly changing. Alibaba Cloud provides the statement outline feature to make stable execution plans by using optimizer and index hints. The DBMS_OUTLN package can be installed to use the statement outline feature.

Prerequisites

The RDS instance version is one of the following:

- MySQL 8.0
- MySQL 5.7

Feature design

The statement outline feature supports the following types of hints provided by MySQL 8.0.

- Optimizer hint

  Optimizer hints are classified by scope and object, and are divided into various types, such as global level hint, table level hint, index level hint, and JOIN_ORDER hint. For more information, see Optimizer Hints.

- Index hint

  Index hints are classified by scope and type. For more information, see Index Hints.

Introduction to the outline table

AliSQL uses a system table named outline to store hints. The instance system automatically creates the table when the system is started. You can refer to the following statements that create the outline table.

```
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 COLLATE 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'
```

The following table describes the parameters.

Parameter	Description
Id	The ID of the outline table.
Schema_name	The name of the database.
Digest	The 64-byte hash string calculated from **Digest_text** during the hash calculation.
Digest_text	The digest of the SQL statement.
Type	• The hint type of optimizer hints is OPTIMIZER. • The hint type of index hints is USE INDEX, FORCE INDEX, or IGNORE INDEX.
Scope	This parameter is only specified for index hints. Valid values: • FOR GROUP BY • FOR ORDER BY • FOR JOIN An empty string indicates index hints of all types.
State	Specifies whether to enable the statement outline.

Parameter	Description
Position	• For optimizer hints, the Position parameter is the position of the keyword in query blocks because all optimizer hints are applied to query blocks. The value of Position indicates the order of the keyword that is applied by hints. The valid values of Position starts from 1. • For index hints, the Position parameter is the position of the table. The value of Position indicates the order of the table that is applied by hints. The valid value starts from 1.
Hint	• For optimizer hints, Hint indicates an integrated hint string, such as `/*+ MAX_EXECUTION_TIME(1000) */` . • For index hints, Hint indicates a list of index names, such as `ind_1,ind_2` .

Manage the statement outline

AliSQL provides six management interfaces in the DBMS_OUTLN package. They are described as follows:

- add_optimizer_outline

  Adds optimizer hints. The statement is as follows:

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

  Note You can enter either of the Digest or Query SQL statements. If you enter the query statement, DBMS_OUTLN calculates the values of Digest and Digest_text.

  Example:

  ```
  CALL DBMS_OUTLN.add_optimizer_outline("outline_db", '', 1, '/*+ MAX_EXECUTION_TIME(1000) */',
                                "select * from t1 where id = 1");
  ```

- add_index_outline

  Adds index hints. The statement is as follows:

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

  Note You can enter either of the Digest or Query SQL statements. If you enter the query statement, DBMS_OUTLN calculates the values of Digest and Digest_text.

  Example:

  ```
  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

  Queries the status of the SQL statement matching the statement outline, which can be used for manual verification. The statement is as follows:

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

  Example:

  ```
  mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 and t1.col2 ='x
  pchild'");
  +------------+------------------------------------------------------------------+------------+-----------+-------+-
  --------------------+
  | SCHEMA     | DIGEST                                                           | BLOCK_TYPE | BLOCK_NAME | BLOCK | HINT
  |
  +------------+------------------------------------------------------------------+------------+-----------+-------+-
  --------------------+
  | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE      | t
  1          |     1 | USE INDEX (`ind_1`) |
  +------------+------------------------------------------------------------------+------------+-----------+-------+-
  --------------------+
  1 row in set (0.00 sec)
  ```

- show_outline

  Displays the in-memory hit rate of the statement outline. The statement is as follows:

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

  Example:

```
mysql> call dbms_outln.show_outline();
+------+------------+------------------------------------------------------------------+-----------+-------+------+
-------------------------------------------------------+------+----------+------------------------------------------
------------------------------------------+
| ID   | SCHEMA     | DIGEST                                                           | TYPE      | SCOPE | POS  | HINT
| HIT  | OVERFLOW | DIGEST_TEXT                                                                       |
+------+------------+------------------------------------------------------------------+-----------+-------+------+
-------------------------------------------------------+------+----------+------------------------------------------
------------------------------------------+
|   33 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 | OPTIM
IZER |       |    1 | /*+ SET_VAR(foreign_key_checks=OFF) */                |    1 |        0 | SELECT * FROM `t1` WH
ERE `id` = ?                                     |
|   32 | outline_db | 36bebc61fce7e32b93926aec3fdd790dad5d895107e2d8d3848d1c60b74bcde6 | OPTIM
IZER |       |    1 | /*+ MAX_EXECUTION_TIME(1000) */                       |    2 |        0 | SELECT * FROM `t1` WHER
E `id` = ?                                      |
|   34 | outline_db | d4dcef634a4a664518e5fb8a21c6ce9b79fccb44b773e86431eb67840975b649 | OPTIMI
ZER |       |    1 | /*+ BNL(t1,t2) */                                     |    1 |        0 | SELECT `t1` . `id` , `t2` . `id` FROM `t1
` , `t2`                               |
|   35 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f | OPTIMIZ
ER |       |    2 |  /*+ QB_NAME(subq1) */                                |    2 |        0 | SELECT * FROM `t1` WHERE `t1` . `
col1` IN ( SELECT `col1` FROM `t2` )              |
|   36 | outline_db | 5a726a609b6fbfb76bb8f9d2a24af913a2b9d07f015f2ee1f6f2d12dfad72e6f | OPTIMIZ
ER |       |    1 | /*+ SEMIJOIN(@subq1 MATERIALIZATION, DUPSWEEDOUT) */  |    2 |        0 | SELECT * FROM
`t1` WHERE `t1` . `col1` IN ( SELECT `col1` FROM `t2` )              |
|   30 | outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | USE IN
DEX |       |    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)
```

The following table describes the HIT and OVERFLOW parameters.

Parameter	Description
HIT	The number of times that the statement outline finds the destination query block or table.

Parameter	Description
OVERFLOW	The number of times that the statement outline does not find the destination query block or table.

- del_outline

  Deletes a statement outline from the memory and the table. The statement is as follows:

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

  Example:

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

  > **Note** If the statement outline that you want to delete does not exist, the system displays a corresponding error. You can execute the `SHOW WARNINGS;` statement to view the error message.
  >
  > ```
  > 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

  If you modify the statement outline in the outline table, you need to execute the following statement so that the statement outline takes effect again. The statement is as follows:

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

  Example:

  ```
  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)
  ```

Feature test

There are two methods to verify whether the statement outline takes effect.

- Use the preview_outline interface.

```
mysql> call dbms_outln.preview_outline('outline_db', "select * from t1 where t1.col1 =1 and t1.col2 ='x
pchild'");
+------------+------------------------------------------------------------------+------------+------------+-------+-
--------------------+
| SCHEMA     | DIGEST                                                           | BLOCK_TYPE | BLOCK_NAME | BLOCK | HINT
|
+------------+------------------------------------------------------------------+------------+------------+-------+-
--------------------+
| outline_db | b4369611be7ab2d27c85897632576a04bc08f50b928a1d735b62d0a140628c4c | TABLE      | t
1          |     1 | USE INDEX (`ind_1`) |
+------------+------------------------------------------------------------------+------------+------------+-------+-
--------------------+
1 row in set (0.01 sec)
```

- Execute the EXPLAIN statement.

```
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`,`outl
ine_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)
```

4.3. Sequence Engine

AliSQL provides the Sequence Engine feature. This feature allows you to use a Sequence engine on your ApsaraDB for RDS instance to simplify the generation of sequence values.

Introduction

Unique, monotonically increasing sequence values are commonly required for the primary keys in a single-node persistent database system, for the globally unique identifiers (GUIDs) in a distributed persistent database system, and for the idempotence among multiple persistent database systems. Each database system may have its unique way to ensure unique sequence values. For example, MySQL provides the AUTO_INCREMENT attribute, and Oracle and SQL Server provide the SEQUENCE attribute.

However, in MySQL databases, it is inconvenient to encapsulate unique sequence values such as dates or user names by using the AUTO_INCREMENT attribute. To make the generation of unique sequence values easier, the following alternative solutions are provided:

- Use an application or a proxy to generate sequence values. However, in this case, the states of the sequence values are sent to the application. This makes scaling more complicated.
- Use a simulated table to generate sequence values. However, middleware must be provided to encapsulate and simplify the logic that is used to obtain the generated sequence values.

A Sequence engine is compatible with various database engines and simplifies the generation of sequence values.

A Sequence engine is compatible with various storage engines used with MySQL. However, the underlying persistent data is still stored by using existing storage engines such as InnoDB and MyISAM. This ensures compatibility with third-party tools such as XtraBackup. Therefore, a Sequence engine is merely a logical engine.

A Sequence engine uses Sequence Handler to access sequence objects. This allows you to increase the value of a sequence by using the NEXTVAL operator and manage the cached data. The data is passed to the underlying base table engine to support data access.

Prerequisites

Your RDS instance is running one of the following database engine versions:

- MySQL 5.6
- MySQL 8.0

Limits

- You cannot perform subqueries or JOIN queries in a Sequence engine.
- You can use the `SHOW CREATE TABLE` or `SHOW CREATE SEQUENCE` statement to access a sequence. However, you cannot use the `SHOW CREATE SEQUENCE` statement to access a regular table.
- You cannot specify a Sequence engine during table creation. If you want to specify a Sequence engine for a table, you must execute the statement described in the "Create a sequence" section.

Create a sequence

To create a sequence, execute the following statement:

```
CREATE SEQUENCE [IF NOT EXISTS] schema.sequence_name
  [START WITH <constant>]
  [MINVALUE <constant>]
  [MAXVALUE <constant>]
  [INCREMENT BY <constant>]
  [CACHE <constant> | NOCACHE]
  [CYCLE | NOCYCLE]
 ;
```

The following table describes the parameters that you need to configure.

Parameter	Description
START	The start value of the sequence.
MINVALUE	The minimum value of the sequence.
MAXVALUE	The maximum value of the sequence. **Note** If the sequence is specified as NOCYCLE, the following error is reported when the maximum value is reached: `ERROR HY000: Sequence 'db.seq' has been run out.`
INCREMENT BY	The increment at which the value of the sequence increases.
CACHE/NOCACHE	The size of the cache. You can set a larger cache size for better performance. However, if your RDS instance is restarted, sequence values stored in the cache will be lost.
CYCLE/NOCYCLE	Specifies whether the value of the sequence restarts from the specified MINVALUE after reaching the maximum value. Valid values: • CYCLE: The value of the sequence will restart from the specified MINVALUE after reaching the maximum value. • NOCYCLE: The value of the sequence will not restart from the specified MINVALUE after reaching the maximum value.

Example:

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

If you want to use the mysqldump program to back up your RDS instance, you can create a sequence table and insert an initial row into the sequence table. Example:

```
CREATE SEQUENCE 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=InnoDB DEFAULT CHARSET=latin1

INSERT INTO schema.sequence_name VALUES(0,0,1,9223372036854775807,1,1,10000,1,0);
COMMIT;
```

Introduction to sequence tables

Sequences are stored in the tables that are created based on the default storage engine. If you query the sequences that you created, the system returns the tables that are created based on the default storage engine. Example:

```
SHOW CREATE [TABLE|SEQUENCE] schema.sequence_name;

CREATE SEQUENCE 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=InnoDB DEFAULT CHARSET=latin1
```

Statements supported

A Sequence engine supports the following statements:

```
 SELECT [nextval | currval | *] FROM seq;
 SELECT nextval(seq),currval(seq);
 SELECT seq.currval, seq.nextval from dual;
```

4.4. Returning

This topic describes the Returning feature of AliSQL. This feature enables data manipulation language (DML) statements to return result sets and provides the DBMS_TRANS package for you to track the execution of DML statements.

Context

The execution results of MySQL statements are divided into three types: result sets, OK packets, and ERR packets. An OK or ERR packet contains attributes such as the number of affected and the number of scanned records. However, the execution of a DML statement (INSERT, UPDATE, or DELETE) is often followed by the execution of the SELECT statement to query current records. In such cases, the Returning feature enables the server to respond to the client only once by combining the execution results of the two statements into a result set.

Prerequisites

Your RDS instance is running MySQL 8.0.

Syntax

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

The following table describes the parameters that you need to configure.

Parameter	Description
Field_list	The fields to return. If you enter more than one field, separate them with commas (,). Native fields and wildcards (*) in the specified table are supported. However, operations such as calculation and aggregation are not supported.
Statement	The DML statement to execute. Only the INSERT, UPDATE, and DELETE statements are supported.

Precautions

`dbms_trans.returning()` is not a transactional statement. It inherits the context of the specified transaction based on the DML statement that you want to execute. To terminate the transaction, you must explicitly commit it or roll it back.

INSERT Returning

The server returns the records that were inserted into the specified table by using the INSERT statement.

Example:

```
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)
```

Note

- If you do not specify the Field_list parameter, the server returns an OK or ERR packet.

```
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)
```

- The Returning feature only supports statements that are similar to `INSERT VALUES`. It does not support statements such as `CREATE AS` and `INSERT SELECT`.

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

UPDATE Returning

The server returns the records that were updated in the specified table by the using UPDATE statement.

Example:

```
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)
```

Note The Returning feature does not allow the UPDATE statement to be executed on more than one table.

DELETE Returning

The server returns the records that were deleted from the specified table by using the DELETE statement.

Example:

```
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)
```

5.Performance

5.1. Fast query cache

The fast query cache is a query cache that is developed by the Database Products Business Unit of Alibaba Cloud based on the native MySQL query cache. It uses a new design and implementation mechanism to increase the query performance of your ApsaraDB for RDS instance.

Prerequisites

Your RDS instance runs MySQL 5.7 (with a kernel version of 20200331 or later).

Background information

A query cache is a cache policy that allows you to expedite queries by saving CPU resources. It stores the text of each qualified statement with the result set that was returned. If an identical statement is received at a later time, the database server directly retrieves the result set from the query cache. This eliminates the need to analyze, optimize, and execute the statement again.

However, the native MySQL query cache has the following drawbacks in terms of design and implementation:

- It cannot process a large number of concurrent queries at fast speeds. This speed is further reduced if multiple CPU cores are configured.
- It cannot utilize memory resources properly or reclaim memory resources in a timely manner. This results in a waste of memory resources and low memory usage.
- If the cache hit ratio is low, query performance does not increase and may even decrease.

Therefore, the native MySQL query cache is not widely used. This technology is no longer provided in the latest version of MySQL 8.0. In contrast, the fast query cache has the following benefits:

- Optimized concurrency control

  The global locking mechanism that is used in the native MySQL query cache to synchronize the running of threads is removed. The fast query cache uses a new synchronization mechanism. This mechanism allows you to fully utilize the capability of multiple CPU cores and process a large number of concurrent queries at fast speeds.

- Optimized memory management

  The memory preallocation mechanism used in the native MySQL query cache is removed. The fast query cache uses a dynamic memory allocation mechanism that is more flexible. This mechanism allows you to reclaim invalid memory resources in a timely manner and increase the utilization of memory resources.

- Optimized caching

  The fast query cache detects cache usage dynamically and adjusts the cache policy in real time. This allows you to ensure stable query performance if the cache hit ratio is low or your RDS instance is used to process both read and write requests.

Unlike the native MySQL query cache, the fast query cache can be used in a wide range of business scenarios to increase query performance.

Enable the fast query cache

For better query performance, you can reduce the memory space for the InnoDB buffer pool. However, we recommend that you increase the memory space for the fast query cache. The fast query cache is in the test invitation phase. If you are interested, submit a ticket.

Compare the performance of the native MySQL query cache and the fast query cache

For the following example, compare the queries per second (QPS) with different query cache configurations under the same conditions in various test cases. These query cache configurations are QC-OFF (no query cache is enabled), MySQL-QC (the native MySQL query cache is enabled), and RDS-QC (the fast query cache is enabled).

- Test environment: a dedicated RDS instance with 4 CPU cores and 8 GB of memory
- Test tool: SysBench
- Test data volume: 250 MB (25 tables in total, 40,000 records per table)
- Test case 1: Test the QPS for read-only queries with a cache hit ratio higher than 99%.

  The oltp_point_select script is used. Execute only primary key-based POINT SELECT statements. In addition, set the query cache size to 512 MB. This size is greater than the test data volume. It allows the cache hit ratio to reach more than 99%. In this test case, focus on how much the QPS increases based on the number of concurrent queries.

QPS for read-only queries with a cache hit ratio higher than 99%

Number of concurrent queries	QC-OFF	MySQL-QC (QPS increase compared with QC-OFF)	RDS-QC (QPS increase compared with QC-OFF)
1	7,947	8,771 (10.38%)	9,196 (15.72%)
8	61,685	65,686 (6.49%)	74,603 (20.94%)
16	102,800	73,027 (-28.96%)	141,856 (37.99%)
32	102,222	60,567 (-40.75%)	199,209 (94.88%)
64	110,230	60,216 (-45.37%)	218,456 (98.18%)
128	111,274	62,844 (-43.52%)	223,885 (101.20%)
256	109,978	63,832 (-41.96%)	218,692 (98.85%)
512	107,379	64,866 (-39.59%)	211,062 (96.56%)
1,024	102,610	62,291 (-39.29%)	198,787 (93.73%)

> **Note** Based on the test result, as the number of concurrent queries increases, the QPS of the native MySQL query cache sharply decreases. However, the QPS of the fast query cache remains stable and can even increase by up to 100%.

- Test case 2: Test the QPS for read-only queries with a cache hit ratio higher than 80%.

  The oltp_read_only script is used. Run queries including range queries that each return more than one record. In addition, set the query cache size to 512 MB. This size ensures sufficient memory capacity and allows the cache hit ratio to reach more than 80%. In this test case, focus on how much the QPS increases based on the number of concurrent queries.

QPS for read-only queries with a cache hit ratio higher than 80%

Number of concurrent queries	QC-OFF	MySQL-QC (QPS increase compared with QC-OFF)	RDS-QC (QPS increase compared with QC-OFF)
1	4,944	6,467 (30.82%)	6,860 (38.77%)
8	28,195	28,651 (1.62%)	42,720 (51.52%)
16	35,292	31,099 (-11.88%)	63,227 (79.15%)
32	34,067	27,610 (-18.95%)	63,133 (85.32%)
64	35,706	27,518 (-22.93%)	70,556 (97.61%)
128	36,303	27,733 (-23.61%)	74,146 (104.24%)
256	36,386	27,738 (-23.77%)	75,550 (107.64%)
512	36,083	27,398 (-24.07%)	74,317 (105.96%)
1,024	35,077	26,861 (-23.42%)	71,205 (103.00%)

> **Note** Based on the test result, as the number of concurrent queries increases, the QPS of the native MySQL query cache significantly decreases. However, the QPS of the fast query cache can increase by up to 100%.

- Test case 3: Test the QPS for read-only queries with a cache hit ratio of about 10%

  The oltp_read_only script is used. Run queries including range queries that each return more than one record. In addition, set the query cache size to 16 MB. This size results in insufficient memory capacity and deletion of cached data and allows the cache hit ratio to drop to about 10%. In this test case, focus on how much the QPS decreases based on the number of concurrent queries.

QPS for read-only queries with a cache hit ratio of about 10%

Number of concurrent queries	QC-OFF	MySQL-QC (QPS increase compared with QC-OFF)	RDS-QC (QPS increase compared with QC-OFF)
1	4,944	4,727 (-4.38%)	4,917 (-0.54%)
8	28,195	22,542 (-20.05%)	27,897 (-1.06%)
16	35,292	24,064 (-31.81%)	34,625 (-1.89%)
32	34,067	21,330 (-37.39%)	33,592 (-1.39%)
64	35,706	19,791 (-44.57%)	35,571 (-0.38%)
128	36,303	19,519 (-46.23%)	36,055 (-0.68%)
256	36,386	19,168 (-47.32%)	36,243 (-0.39%)
512	36,083	18,420 (-48.95%)	35,679 (-1.12%)
1,024	35,077	20,168 (-42.50%)	34,595 (-1.37%)

> **Note** Based on the test result, as the number of concurrent queries increases, the QPS of the native MySQL query cache significantly decreases by up to 50%. However, the fast query cache allows you to ensure that the QPS decrease is within 2%.

- Test case 4: Test the QPS for read/write queries.

  The oltp_read_write script is used. Run transactions that each perform updates on tables. These frequent updates result in deletion of data from the query cache, and consequently the query cache is considered invalid. In this test case, focus on how much the QPS decreases based on the number of concurrent queries.

QPS for read/write queries

Number of concurrent queries	QC-OFF	RDS-QC (QPS increase compared with QC-OFF)
1	4,188	4,199 (0.27%)
8	21,587	21,263 (-1.50%)
16	26,224	25,956 (-1.02%)
32	27,887	27,741 (-0.52%)
64	29,852	29,426 (-1.43%)

Number of concurrent queries	QC-OFF	RDS-QC (QPS increase compared with QC-OFF)
128	30,024	29,724 (-1.00%)
256	29,835	29,615 (-0.74%)
512	29,525	29,683 (0.54%)
1,024	29,905	29,512 (-1.31%)

Note Based on the test result, as the number of concurrent queries increases, the fast query cache allows you to ensure that the QPS decrease is within 2%.

Practice guidelines

Assume that you can determine the size of the data sets that need to be cached. For example, you enable a query cache on a specified table by using the SQL_CACHE option. In this situation, you can evaluate the QPS by using the preceding test cases. A few more tips on using the fast query cache are as follows:

- Enable the fast query cache in various scenarios
  - The fast query cache aims to increase the QPS for read queries. We recommend that you enable the fast query cache if your RDS instance processes a large number of read queries but a small number of write queries. Otherwise, we recommend that you enable the fast query cache only for tables that are frequently read but to which data is infrequently written. You can enable the fast query cache for a specified table by using the SQL_CACHE option. If your RDS instance processes a small number of read queries but a large number of write queries, the data of your RDS instance is frequently updated. In this situation, if you enable the fast query cache, the QPS may decrease by up to 2%.
  - The QPS increase produced by the fast query cache varies based on the cache hit ratio. Before you enable the fast query cache for your RDS instance, we recommend that you obtain the hit ratio of the InnoDB buffer pool. If the hit ratio is lower than 80%, we recommend that you do not enable the fast query cache. The hit ratio of the InnoDB buffer pool is calculated by using the following formula: Hit ratio = (1 - The value of the Innodb_buffer_pool_reads parameter/The value of the Innodb_buffer_pool_read_requests parameter) x 100%. You can also obtain the read/write ratio of each table from the TABLE_STATISTICS table. If a table has a high read/write ratio, enable the fast query cache for the table by using the SQL_CACHE option. For more information about how to query the TABLE_STATISTICS table, see Performance Insight.
- Manage the fast query cache by using the query_cache_type parameter

  The fast query cache is managed by using the same method as the native MySQL query cache. You can manage the fast query cache by using the **query_cache_type** parameter.

Parameter	Value	Description
query_cache_type	OFF	Disables the fast query cache. This is the default value.
	ON	Enables the fast query cache. However, you can use the SQL_NO_CACHE option to specify that your RDS instance does not retrieve result sets from the fast query cache.
	DEMAND	Disables the fast query cache. However, you can enable the fast query cache later by using the SQL_CACHE option.

You can set the **query_cache_type** parameter for a specific session based on your business scenario.

- If your RDS instance processes a small number of read queries but a large number of write queries, the data of your RDS instance is frequently updated. In this situation, set the **query_cache_type** parameter to OFF for your RDS instance.
- If your RDS instance features a small volume of data, fixed query types, and high hit ratio, set the **query_cache_type** parameter to ON for your RDS instance.
- If your RDS instance features a large data volume, changing query types, and unstable hit ratio, set the **query_cache_type** parameter to DEMAND for your RDS instance. You can enable the fast query cache only for a specific statement by using the SQL_CACHE option.

- Specify a proper query cache size by using the query_cache_size parameter

  The query_cache_size parameter determines SQL statement execution efficiency. If you want to cache the results of queries that each return more than one record, you may need to specify a query cache size that is many times larger than the data volume. If you do not run range queries, you can evaluate the relationship between the data volume and the query_cache_size parameter as follows:

  - Test environment: a dedicated RDS instance with 4 CPU cores and 8 GB of memory (The size of the InnoDB buffer pool is set to 6 GB by using the innodb_buffer_pool_size parameter.)
  - Test tool: SysBench
  - Test data volume: 10 GB (100 tables in total, 400,000 records per table)

  Test case: Specify different cache sizes by using the query_cache_size parameter and execute the oltp_point_select script for each cache size. Make sure that 64 threads run concurrently to query data. In this case, hot data accounts for 20%. This way, you can obtain the impacts of different cache sizes on the QPS. The actual result set is 2.5 GB in size.

QPS with different cache sizes

query_cache_size (MB)	QC-OFF	RDS-QC hit ratio	RDS-QC (QPS increase compared with QC-OFF)
128	104,473	45%	110,483 (5.75%)
256	104,473	72%	128,297 (22.80%)
512	104,473	82%	137,153 (31.28%)
1024	104,473	84%	133,624 (27.90%)

query_cache_size (MB)	QC-OFF	RDS-QC hit ratio	RDS-QC (QPS increase compared with QC-OFF)
2048	104,473	87%	125,766 (20.38%)

When you set the query_cache_size parameter, note the following information:

- If you can determine the result set size, set the query_cache_size parameter to a value that is `20% of the result set size` .
- If you cannot determine the result set size, set the query_cache_size parameter to a value that is `20% of the value of the innodb_buffer_pool_size parameter` .
- The memory capacity of your RDS instance is limited. We recommend that you properly adjust the values of the query_cache_size and innodb_buffer_pool_size parameters at the same time. This allows you to refrain from exhausting the memory capacity that is allowed.

5.2. Binlog in Redo

The Binlog in Redo function synchronously writes binary logs to the redo log file when a transaction is committed. This reduces operations on disks and improves database performance.

Prerequisites

Your RDS instance runs MySQL 8.0 (with a kernel version of 20200430 or later).

Context

To ensure data security in crucial MySQL business scenarios, the system stores both binary and redo logs when a transaction is committed. Both the following parameters must be set to 1:

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

Each time a transaction is committed, the system performs two I/O operations. One is to write the binary logs to disks, and the other is to write the redo logs to disks. Although Group Commit is enabled for binary logs, the system must still wait for the two I/O operations to complete. This affects the efficiency of transaction processing, especially when standard or enhanced SSDs are used. The performance of I/O merging is based on the number of concurrent transactions that are committed at the same time. When the number of concurrent transactions is small, the performance is low. For example, when a small number of write transactions are committed, the system response is slow.

To increase the efficiency of committing transactions, AliSQL provides the Binlog in Redo function. You can enable the function by setting the `persist_binlog_to_redo` parameter to on. When a transaction is committed, the system synchronously writes binary logs to the redo log file and stores only the redo log file to disks. This reduces I/O consumption. The binary log files are then asynchronously stored to disks by using a separate thread at regular intervals. If a restart operation is triggered upon an exception, the system uses the binary logs in the redo log file to complement the binary log files. In this way, the database performance improves and the system response is faster. Also, the number of times that the binary log files are stored is reduced. This significantly relieves the pressure on the file system while increasing performance. This pressure can be resulted from the calls of the fsync functions that are triggered by file updates in real time. The fsync function synchronizes files to disks.

Binlog in Redo does not change the format of binary logs. Replication and third-party tools that are based on binary logs are not affected.

Parameters

- persist_binlog_to_redo

  The switch that is used to enable or disable the Binlog in Redo function. This parameter is a global system variable. Valid values: on and off. The parameter change immediately takes effect. You do not need to restart your RDS instance.

  > **Note** If you want to enable Binlog in Redo, you only need to set the `persist_binlog_to_redo` parameter to on. You do not need to modify the settings of other parameters. The setting `sync_binlog = 1` automatically becomes invalid.

- sync_binlog_interval

  The interval at which binary logs are asynchronously stored. This parameter is a global system variable. It takes effect only when the `persist_binlog_to_redo` parameter is set to on. Default value: 50. Unit: milliseconds (ms). In normal cases, the default value is recommended. The parameter change immediately takes effect. You do not need to restart your RDS instance.

Stress testing

- Test environment
  - Application server: an Alibaba Cloud ECS instance
  - RDS instance type: 32 CPU cores, 64 GB of memory, and enhanced SSDs
  - RDS edition: High-availability Edition with asynchronous data replication
- Test cases

  Sysbench provides the following test cases:
  - oltp_update_non_index
  - oltp_insert
  - oltp_write_only
- Test results

- oltp_update_non_index

  After Binlog in Redo is enabled, the queries per second (QPS) significantly increases and the latency is low when the number of concurrent queries is small.

- oltp_insert

  After Binlog in Redo is enabled, the QPS significantly increases and the latency is low when the number of concurrent queries is small.

- oltp_write_only

  After Binlog in Redo is enabled, the QPS slightly increases and the latency is low when the number of concurrent queries is small.

- Number of times that the fsync function is called for binary logs

  After Binlog in Redo is enabled, the number of times is significantly reduced.

Test conclusion

- oltp_update_non_index and oltp_insert test single-statement transactions, and the transactions are committed on a frequent basis. oltp_write_only tests multi-statement transactions, and the transactions are committed on a less frequent basis. This type of transaction contains two UPDATE statements, one DELETE statement, and one INSERT statement. Performance improvement in oltp_update_non_index and oltp_insert is more notable than that in oltp_write_only.
- When the number of concurrent transactions is less than 256, Binlog in Redo significantly improves database performance and reduces latency. In most scenarios, Binlog in Redo provides significant benefits.
- Binlog in Redo significantly reduces the number of times that the fsync function is called for binary logs. This improves the performance of the file system.

5.3. Statement Queue

The Statement Queue feature of AliSQL allows statements to queue in the same bucket. These statements may be executed on the same resources. For example, these statements are executed on the same row of a table. This reduces overheads from possible conflicts.

Context

During the execution of concurrent statements, the MySQL server and engine are likely to conflict with each other in a number of serial operations. Take transactional lock conflicts triggered by data manipulation language (DML) statements as an example. The InnoDB storage engine supports resource locking accurate to rows. If you execute a number of DML statements concurrently on a row, serious conflicts may occur. The overall throughput of your database system decreases in proportion with the number of concurrent statements. The Statement Queue feature reduces overheads from these conflicts and increases the performance of your database system.

Prerequisites

The RDS instance version is one of the following:

- MySQL 8.0
- MySQL 5.7

Benefits

AliSQL executes UPDATE statements concurrently on a single row four times faster than native MySQL.

Variables

AliSQL provides two variables that are used to define the bucket quantity and size of a statement queue:

- ccl_queue_bucket_count: the number of buckets allowed in the statement queue. Valid values: 1 to 64. Default value: 4.
- ccl_queue_bucket_size: the number of concurrent statements allowed per bucket. Valid values: 1 to 4096. Default value: 64.

> **Note** You can reconfigure the variables in the ApsaraDB for RDS console. For more information, see Reconfigure the parameters of an ApsaraDB RDS for MySQL instance.

Syntaxes

AliSQL supports two hint syntaxes:

- ccl_queue_value

  AliSQL uses a hash algorithm to determine the bucket into which each statement is placed based on the value of a specified field.

  Syntax:

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

  Example:

  ```
  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

AliSQL uses a hash algorithm to determine the bucket into which each statement is placed based on the value of the field that is specified in the WHERE clause.

Syntax:

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

Example:

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

> **Note** In the ccl_queue_field hint, the WHERE clause only supports binary operators on raw fields. These raw fields have not been altered by using functions or computation operations. In addition, the right operand of such a binary operator must be a number or string.

Functions

AliSQL provides two functions that are used to query the status of a statement queue:

- dbms_ccl.show_ccl_queue()

  This function is used to query the status of the current statement queue.

```
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)
```

  The following table describes the parameters in this function.

Parameter	Description
CONCURRENCY_COUNT	The maximum number of concurrent queries allowed.
MATCHED	The total number of rules matched.
RUNNING	The number of statements that are being executed concurrently.
WAITTING	The number of statements that are waiting in queue.

- dbms_ccl.flush_ccl_queue()

This function is used to delete data about the statement queue from the memory and query the status of the statement queue.

```
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)
```

Practices

Statement Queue can work with Statement outline to support online updates of your application code. In the following example, SysBench is used to execute the update_non_index.lua script:

- Test environment
  - Schema

    ```
    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;
    ```

  - Statement

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

- Script

  ```
  ./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
  ```

- Procedure

  i. Create a statement outline in online mode.

  ```
  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. View the statement outline that you created.

  ```
  mysql> call dbms_outln.show_outline();
  +------+--------+------------------------------------------------------------------+-----------+-------+------
  +-------------------------------+------+----------+--------------------------------------------+
  | ID   | SCHEMA | DIGEST                                                           | TYPE      | SCOPE | POS  | HINT
  | HIT  | OVERFLOW | DIGEST_TEXT                                |
  +------+--------+------------------------------------------------------------------+-----------+-------+------
  +-------------------------------+------+----------+--------------------------------------------+
  |    1 | test   | 7b945614749e541e0600753367884acff5df7e7ee2f5fb0af5ea58897910f023 | OPTIMIZE
  R |       |    1 |  /*+ ccl_queue_field("id") */  |    0 |        0 | UPDATE `sbtest1` SET `c` = ? WHERE `id` = ?
  |
  +------+--------+------------------------------------------------------------------+-----------+-------+------
  +-------------------------------+------+----------+--------------------------------------------+
  1 row in set (0.00 sec)
  ```

  iii. Verify that the statement outline has taken effect.

```
mysql> explain UPDATE sbtest1 SET c='xpchild' WHERE id=0;
+----+-------------+---------+------------+-------+---------------+---------+---------+-------+------+------
----+-------------+
| id | select_type | table   | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtere
d | 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`.`sbtest1`.`c` = 'xpc
hild' where (`test`.`sbtest1`.`id` = 0) |
+-------+------+----------------------------------------------------------------------------------------------
----------------------------+
1 row in set (0.00 sec)
```

iv. Query the status of the statement queue used.

```
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. Start the test.

```
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. View the test result.

```
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 | DIGES
T_TEXT                                 |
+------+--------+-----------+-----------+-------+------+-------------------------------+--------+----------
---------------------------------------------+
|    1 | test   | xxxxxxxxx | OPTIMIZER |       |    1 |  /*+ ccl_queue_field("id") */ | 115795 |        0 | UPDA
TE `sbtest1` SET `c` = ? WHERE `id` = ? |
+------+--------+-----------+-----------+-------+------+-------------------------------+--------+----------
---------------------------------------------+
1 row in set (0.00 sec)
```

> **Note** Based on the query results, the statement outline is hit 115,795 times, the statement queue is hit 10,996 times, a total of 63 statements are being executed concurrently, and four statements are waiting in queue.

5.4. Inventory Hint

This topic describes the Inventory Hint feature provided by AliSQL. This feature can work with the Returning and Statement Queue features to commit and roll back transactions rapidly.

Background information

In business scenarios such as seckilling, inventory reduction is a common task model that requires high concurrency and serialization. In this model, AliSQL uses queues and transactional hints to control concurrency and commit or roll back transactions. This increases the throughput of your business.

Prerequisites

The RDS instance version is one of the following:

- MySQL 8.0
- MySQL 5.7
- MySQL 5.6

Syntax

The following three hints are introduced to specify tables in SELECT, UPDATE, INSERT, and DELETE statements.

- COMMIT_ON_SUCCESS and ROLLBACK_ON_FAIL

  These are two transactional hints.

  - COMMIT_ON_SUCCESS: specifies to commit the transaction if the execution of the statement to which this hint is applied succeeds.
  - ROLLBACK_ON_FAIL: specifies to roll the transaction back if the execution of the statement to which this hint is applied fails.

  Syntax:

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

  Example:

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

- TARGET_AFFECT_ROW(NUMBER)

  This is a conditional hint. After you apply it to a statement, the execution of the statement succeeds only when the number of affected rows is the same as the number specified in this hint.

  Syntax:

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

  Example:

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

Precautions

- The transactional hints do not support the autocommit mode. If you use a transactional hint in a statement with the autocommit mode, an error is reported. Example:

```
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
```

- Transactional hints cannot be used in substatements. If you use a transactional hint in a substatement, an error is reported. Example:

```
 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
```

- The conditional hint cannot be used in a SELECT or EXPLAIN statement. If you use the conditional hint in a SELECT or EXPLAIN statement, an error is reported. Example:

```
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
```

Note You can specify an invalid number in the TARGET_AFFECT_ROW hint and check whether the system reports errors:

```
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 id =1' at line 1
|
| Note    | 1003 | update /*+ COMMIT_ON_SUCCESS ROLLBACK_ON_FAIL */ `test`.`t` set `test`.`t`.`c
ol1` = (`test`.`t`.`col1` + 1) where (`test`.`t`.`id` = 1) |
+---------+------+----------------------------------------------------------------------------------------------
------------------------------------------+
2 rows in set (0.00 sec)
```

Work with Returning

You can use Inventory Hint with Returning for the system to return real-time result sets. Example:

```
mysql> CALL dbms_trans.returning("*", "update /*+ commit_on_success rollback_on_fail target_affect_r
ow(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_ro
w(1) */ into
                              t values(10,10)");
+----+------+
| id | col1 |
+----+------+
| 10 |   10 |
+----+------+
1 row in set (0.01 sec)
```

Work with Statement Queue

You can use Inventory Hint with Statement Queue for the system to queue statements. Example:

```
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.Stability

6.1. Faster DDL

This topic describes the faster DDL feature that provides an optimized buffer pool management mechanism. This mechanism allows you to reduce the impact of data definition language (DDL) operations and increase the number of concurrent DDL operations that are allowed.

Prerequisites

Your RDS instance runs one of the following MySQL versions:

- MySQL 8.0 (with a minor engine version of 20200630 or later.)
- MySQL 5.7 (with a minor engine version of 20200630 or later.)
- MySQL 5.6 (with a minor engine version of 20200630 or later.)

Context

DDL operations are common for RDS instances. When you use your RDS instance, you may encounter issues related to DDL operations. For example, you may encounter the following issues:

- When you add indexes, why does a performance jitter occur and interrupt the read and write operations on your RDS instance?
- Why does it require more than 10 minutes to perform a DDL operation on a table whose size is less than 1 GB?
- When a connection that generates temporary tables is closed, why does a performance jitter occur?

The database engine team of ApsaraDB for RDS has performed in-depth analyses and intensive tests to locate these issues. Based on the analysis and test results, the team has identified defects in the cache maintenance logic that is used to manage DDL operations. To fix these issues, the team has developed the faster DDL feature. The optimized buffer pool management mechanism provided by this feature reduces competition for locks that are triggered by DDL operations. When your RDS instance processes a normal number of workloads, this allows you to ensure the performance of your RDS instance during DDL operations.

Enable faster DDL

You can enable the faster DDL feature by setting the **loose_innodb_rds_faster_ddl** parameter to **ON** in the ApsaraDB for RDS console. For more information, see Reconfigure the parameters of an ApsaraDB RDS for MySQL instance.

Test with DDL operations

- Test scenario

  Use the in-place algorithm to perform online DDL operations by executing the following MySQL 8.0-supported statements: CREATE INDEX and OPTIMIZE TABLE. The CREATE INDEX statement creates an index on a table without the need to rebuild the table. The OPTIMIZE TABLE statement creates an index on a table with the need to rebuild the table.

Operation	Instant	In-place	Rebuilds table	Permits concurrent DML operations	Only modifies metadata
CREATE INDEX	No	Yes	No	Yes	No
OPTIMIZE TABLE	No	Yes	Yes	Yes	No

- Test instance

  The RDS instance that is used for the test runs MySQL 8.0. It provides 8 CPU cores and 64 GB of memory. The size of the table on which you perform DDL operations is 600 MB.

- Test procedure

  Use SysBench to perform a stress test. In this test, perform online DDL operations and compare the operation results.

- Test result

Operation	Average execution duration (with faster DDL disabled)	Average execution duration (with faster DDL enabled)	Performance increase times
CREATE INDEX	56 seconds	4.9 seconds	11.4
OPTIMIZE TABLE	220 seconds	17 seconds	12.9

- Test summary

  The faster DDL feature enables ApsaraDB RDS for MySQL with AliSQL to reduce the execution duration of a DDL operation by more than 90% compared with the MySQL Community Edition.

Test with temporary tables

Temporary tables are common in MySQL. For example, the system creates temporary tables that are used to query tables from the information_schema database or to expedite the execution of complex SQL statements. When a thread exits, all of the related temporary tables are deleted. This is known as a specific type of DDL operation that causes a performance jitter on your RDS instance. For more information, see Temp ibt tablespace truncation at disconnection stuck InnoDB under large BP.

- Test instance

  The RDS instance that is used for the test runs MySQL 8.0. It provides 8 CPU cores and 64 GB of memory.

- Test procedure

  Use tpcc-mysql to perform a stress test. In this test, run queries to make sure that the buffer pool reaches near full capacity. Then, initiate single-threaded requests over short-lived connections to generate temporary tables.

- Test result

Comparison item	DDL operations not included	Faster DDL enabled	Faster DDL disabled
Transactions per second (TPS)	42,000	40,000	< 10,000

The following figure shows the second-level performance data that is obtained from the stress test. The red highlighted parts indicate the TPSs that are supported by the RDS instance when the faster DDL feature is disabled.

- Test summary

  Every time when a thread that generates temporary tables exits, the native MySQL causes a severe performance jitter. The jitter decreases the TPS by more than 70%. After the faster DDL feature is enabled, the TPS decrease is reduced to 5%.

Optimization effect

The faster DDL feature supports MySQL 5.6, 5.7, and 8.0. However, the supported DDL operations can vary based on the selected MySQL version.

Category	DDL operation	MySQL 5.6	MySQL 5.7	MySQL 8.0
In-place DDL	For more information, see MySQL 8.0 Online DDL Operations and MySQL 5.7 Online DDL Operations.	No	Yes	Yes
Tablespace management	Enables or disables tablespace encryption.	No	Yes	Yes
	Releases or deletes a tablespace.	No	Yes	Yes
	Discards a tablespace.	Yes	Yes	Yes
Table deletion	Releases or deletes a table.	Yes	Yes	Yes
Undo operation	Releases or deletes an undo tablespace.	No	No	Yes
Table refresh	Refreshes a table and its dirty pages.	Yes	Yes	Yes

Defects fixed by faster DDL

The faster DDL feature fixes the following defects:

- 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

Alibaba Cloud provides the concurrency control (CCL) feature to ensure the stability of ApsaraDB RDS MySQL instances in case of unexpected request traffic, resource-consuming statements, and SQL access model changes. The DBMS_CCL package can be installed to use the CCL feature.

Prerequisites

The RDS instance version is one of the following:

- MySQL 8.0
- MySQL 5.7

Precautions

- CCL operations only affect the current instance because no binlogs are generated. For example, CCL operations performed on the primary instance are not synchronized to the secondary instance, read-only instance, or disaster recovery instance.
- CCL includes a timeout mechanism which resolves transaction deadlocks caused by DML statements. The waiting threads also respond to the transaction timeout and kill threads to prevent deadlocks.

Feature design

The CCL provides features based on the following dimensions:

- SQL command

  The types of SQL statements, such as SELECT, UPDATE, INSERT, and DELETE.

- Object

  The objects managed by SQL statements, such as tables and views.

- keywords

  The keywords of SQL statements.

Create a CCL table

AliSQL uses a system table named concurrency_control to store CCL rules. The instance system automatically creates the table when the system is started. You can refer to the following statements that create the concurrency_control table.

```
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'
```

Parameter	Description
Id	The ID of the CCL rule.
Type	The type of the SQL statement.
Schema_name	The name of the database.
Table_name	The name of the table in the database.
Concurrency_count	The number of concurrent threads.
Keywords	The keyword. Multiple keywords are separated by semicolons (;).
State	Specifies whether to enable the CCL rule.
Ordered	Specifies whether to match multiple keywords in sequence.

Manage CCL rules

AliSQL provides four management interfaces in the DBMS_CCL package. They are described as follows:

- add_ccl_rule

  Use the following statement to create a rule.

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

  Example:

  The number of concurrent threads of the SELECT statement is 10.

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

The number of concurrent threads of the SELECT statement is 20, and the keyword of the statement is key1.

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

The number of concurrent threads of the SELECT statement in the test.t table is 20.

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

> **Note** The rule with a larger Id has higher priority.

- del_ccl_rule

  Use the following statement to delete a rule.

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

  Example:

  Delete the CCL rule whose ID is 15.

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

  > **Note** If the CCL rule that you want to delete does not exist, the system displays an error. You can execute the `SHOW WARNINGS;` statement to view the error message.
  >
  > ```
  > 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

  Use the following statement to view the enabled rules in the memory.

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

  Example:

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+-------+-------+-------+-------------------+---------+---------+----------+----------+
| ID   | TYPE   | SCHEMA | TABLE | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNING | WAITTI
NG | 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 |          |
+------+--------+--------+-------+-------+-------+-------------------+---------+---------+----------+----------+
```

The following table describes the MATCHED, RUNNING, and WAITTING parameters.

Parameter	Description
MATCHED	The number of times the rule is matched.
RUNNING	The number of concurrent threads under the rule.
WAITTING	The number of threads to be run under the rule.

- flush_ccl_rule

  If you modify the rules in the concurrency_control table, you must execute the following statement to validate the rules again.

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

  Example:

  ```
  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)
  ```

Feature test

- Test rules

  Execute the following statements to create the rules for three dimensions.

```
call dbms_ccl.add_ccl_rule('SELECT', 'test', 'sbtest1', 3, '');  // The SELECT statement manages the sbt
est1 table and the number of concurrent threads is 3.
call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, 'sbtest2');      // The keyword of the SELECT statement is s
btest2 and the number of concurrent threads is 2.
 call dbms_ccl.add_ccl_rule('SELECT', '', '', 2, '');            // The number of concurrent threads of the SELE
CT statement is 2.
```

- Test scenarios

  Use sysbench to test in the following scenarios:

  - 64 threads
  - 4 tables
  - select.lua

- Test results

  Execute the following statement to view the number of concurrent threads under the rules.

```
mysql> call dbms_ccl.show_ccl_rule();
+------+--------+--------+---------+-------+-------+-------------------+---------+---------+----------+----------
+
| ID   | TYPE   | SCHEMA | TABLE   | STATE | ORDER | CONCURRENCY_COUNT | MATCHED | RUNNING | WAIT
TING | 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)
```

The numbers displayed in the **RUNNING** column are the same as the numbers specified when you create the rules.

6.3. Performance Agent

This topic describes the Performance Agent feature provided by AliSQL as a plug-in to collect statistics of performance data on ApsaraDB RDS for MySQL instances.

Background information

Performance Agent adds a memory table named PERF_STATISTICS to the information_schema system database. This table stores the performance data generated over a recent period of time. You can query performance data from this table.

Prerequisites

The RDS instance runs one of the following database engine versions:

- MySQL 8.0 (The kernel version of the RDS instance is 20200229 or later)
- MySQL 5.7 (The kernel version of the RDS instance is 20200229 or later)

Note For information about updating the kernel version, see Upgrade the minor engine version of an ApsaraDB RDS for MySQL instance.

Parameters

The following table describes the parameters you must configure for Performance Agent. For more information, see Reconfigure parameters for an RDS MySQL instance.

Parameter	Description
performance_agent_enabled	Specifies whether to enable the Performance Agent feature. Valid values: ON | OFF. Default value: ON.
performance_agent_interval	The interval at which you want to collect performance data. Unit: seconds. Default value: 1.
performance_agent_perfstat_volume_size	The maximum number of data records that are allowed in the PERF_STATISTICS memory table. Default value: 3600. If you set the performance_agent_interval parameter to 1, the system retains the performance data generated within the last hour.

Schema

The schema of the PERF_STATISTICS memory table is as follows:

```
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;
```

Column	Description
TIME	The time when the data record was generated. The time is in the yyyy-MM-dd HH:mm:ss format.

Column	Description
PROCS_MEM_USAGE	The amount of physical memory occupied by the data record. Unit: bytes.
PROCS_CPU_RATIO	The CPU utilization of the data record.
PROCS_IOPS	The number of I/O operations that the system invoked.
PROCS_IO_READ_BYTES	The amount of data that was read by I/O operations. Unit: bytes.
PROCS_IO_WRITE_BYTES	The amount of data that was written by I/O operations. Unit: bytes.
MYSQL_CONN_ABORT	The number of disconnected connections.
MYSQL_CONN_CREATED	The number of new connections.
MYSQL_USER_CONN_COUNT	The total number of connections.
MYSQL_CONN_RUNNING	The number of active connections.
MYSQL_LOCK_IMMEDIATE	The number of locks held by the data record.
MYSQL_LOCK_WAITED	The number of locks for which the data record waited.
MYSQL_COM_INSERT	The number of statements executed to insert data.
MYSQL_COM_UPDATE	The number of statements executed to update data.
MYSQL_COM_DELETE	The number of statements executed to delete data.
MYSQL_COM_SELECT	The number of statements executed to query data.
MYSQL_COM_COMMIT	The number of transactions explicitly committed.
MYSQL_COM_ROLLBACK	The number of transactions rolled back.
MYSQL_COM_PREPARE	The number of statements that were pre-processed.
MYSQL_LONG_QUERY	The number of slow queries.
MYSQL_TCACHE_GET	The number of cache hits.
MYSQL_TCACHE_MISS	The number of cache misses.
MYSQL_TMPFILE_CREATED	The number of temporary files created.
MYSQL_TMP_TABLES	The number of temporary tables created.
MYSQL_TMP_DISKTABLES	The number of temporary disk tables created.
MYSQL_SORT_MERGE	The number of times that data was merged and sorted.

Column	Description
MYSQL_SORT_ROWS	The number of rows sorted.
MYSQL_BYTES_RECEIVED	The amount of data received. Unit: bytes.
MYSQL_BYTES_SENT	The amount of data sent. Unit: bytes.
MYSQL_BINLOG_OFFSET	The size of the binary log file generated. Unit: bytes.
MYSQL_IOLOG_OFFSET	The size of the binary log file sent from the primary instance. Unit: bytes.
MYSQL_RELAYLOG_OFFSET	The size of the binary log file sent from the secondary instance. Unit: bytes.
EXTRA	The statistics information about InnoDB. The EXTRA parameter consists of multiple fields in the JSON format. For more information, see Fields in the EXTRA parameter. **Note** The values of the metrics in the InnoDB statistics information are the same as the values obtained by executing the `SHOW STATUS` statement.

Fields in the EXTRA parameter

Field	Description
INNODB_TRX_CNT	The number of transactions.
INNODB_DATA_READ	The amount of data read. Unit: bytes.
INNODB_IBUF_SIZE	The number of pages merged.
INNODB_LOG_WAITS	The number of times that InnoDB waited to write logs.
INNODB_MAX_PURGE	The number of transactions deleted.
INNODB_N_WAITING	The number of locks for which InnoDB waited.
INNODB_ROWS_READ	The number of rows read.
INNODB_LOG_WRITES	The number of times that logs were written by InnoDB.
INNODB_IBUF_MERGES	The number of times that data was merged by InnoDB.
INNODB_DATA_WRITTEN	The amount of data written. Unit: bytes.
INNODB_DBLWR_WRITES	The number of double write operations.
INNODB_IBUF_SEGSIZE	The size of data inserted into the buffer.

Field	Description
INNODB_ROWS_DELETED	The number of rows deleted.
INNODB_ROWS_UPDATED	The number of rows updated.
INNODB_COMMIT_TRXCNT	The number of transactions committed.
INNODB_IBUF_FREELIST	The length of the idle list.
INNODB_MYSQL_TRX_CNT	The number of MySQL transactions.
INNODB_ROWS_INSERTED	The number of rows inserted.
INNODB_ACTIVE_TRX_CNT	The number of active transactions.
INNODB_OS_LOG_WRITTEN	The amount of log data written. Unit: bytes.
INNODB_ACTIVE_VIEW_CNT	The number of active views.
INNODB_RSEG_HISTORY_LEN	The length of the TRX_RSEG_HISTORY table.
INNODB_AVG_COMMIT_TRXTIME	The average time taken to commit a transaction.
INNODB_MAX_COMMIT_TRXTIME	The maximum time taken to commit a transaction.
INNODB_DBLWR_PAGES_WRITTEN	The number of writes completed by double write operations.

Procedure

- Query the system table to obtain performance data.
  - Query the CPU utilization and memory usage within the last 30 seconds. Example:

```
MySQL> select TIME, PROCS_MEM_USAGE, PROCS_CPU_RATIO from information_schema.PERF_STATISTICS 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)
```

- Query the rows that are read and written by InnoDB within the last 30 seconds. Example:

```
MySQL> select TIME, EXTRA->'$.INNODB_ROWS_READ', EXTRA->'$.INNODB_ROWS_INSERTED' from inf
ormation_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)
```

- Connect to a performance monitoring platform to monitor your database performance in real time. For example, connect to Grafana.

6.4. Purge Large File Asynchronously

This topic describes how to use the Purge Large File Asynchronously function to delete files from an ApsaraDB for RDS instance running AliSQL. This function is designed to ensure database stability by deleting large files asynchronously.

Context

If your ApsaraDB for RDS instance runs the InnoDB storage engine, directly deleting large files from the instance compromises the stability of your POSIX file system. As a result, InnoDB starts a background thread to delete large files asynchronously. InnoDB renames data files housing tablespaces to identify them as temporary files before starting to delete the tablespaces asynchronously.

> ⓘ **Note** AliSQL ensures the atomicity of Data Definition Language (DDL) statements by deleting log files.

Procedure

1. View the global variable settings of your RDS instance, as shown in the following example:

```
mysql> SHOW GLOBAL VARIABLES LIKE '%data_file_purge%';
 +---------------------------------------+-------+
 | 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   |
 +---------------------------------------+-------+
```

The following table describes these variables.

Variable	Description
innodb_data_file_purge	Specifies whether to enable the Purge Large File Asynchronously function.
innodb_data_file_purge_all_at_shutdown	Specifies whether to delete all files when the host server of your RDS instance is shut down.

Variable	Description
innodb_data_file_purge_dir	The directory for stored temporary files.
innodb_data_file_purge_immediate	Specifies whether to revoke data file links, but not to delete them.
innodb_data_file_purge_interval	The intervals at which files are deleted. Unit: ms.
innodb_data_file_purge_max_size	The maximum size of a single file that can be deleted. Unit: MB.
innodb_print_data_file_purge_process	Specifies whether to display the file deletion process.

> **Note** We recommend that you set the following variables to the values provided in the example:
>
> ```
> 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. Run the following command to view the file deletion progress:

```
select * from information_schema.build_current_task
```

6.5. Performance Insight

This topic describes how to use the Performance Insight function for load monitoring, association analysis, and optimizing performance. This function helps you quickly evaluate the loads of your ApsaraDB for RDS instance and locate performance problems to ensure database stability.

Prerequisites

- Your RDS instance runs one of the following database engine versions:
  - MySQL 8.0
  - MySQL 5.7
- The kernel version of your RDS instance is 20190915 or later.

> **Note** Log on to the ApsaraDB for RDS console, find the target RDS instance, and navigate to the **Basic Information** page. Then in the **Configuration Information** section, check whether the **Upgrade Kernel Version** button is available. If the button is available, click it to view the kernel version of your RDS instance. If the button is not available, you are already using the latest kernel version. For more information, see Upgrade the minor engine version of an ApsaraDB RDS for MySQL instance.

Overview

The Performance Insight function consists of the following two parts:

- Object Statistics

  Object Statistics queries statistics from indexes and the following two tables:

  - TABLE_STATISTICS: records rows with read and modified data.
  - INDEX_STATISTICS: records rows with data read from indexes.

- Performance Point

  Performance Point collects performance details of your RDS instance. Using these details, you can quantify the overheads of SQL statements faster and more accurately. Performance Point measures database performance using the following three dimensions:

  - CPU: includes but is not limited to the total time spent executing an SQL statement and the time spent by CPU executing an SQL statement.
  - Lock: includes the time occupied by locks such as metadata locks on the server, storage transaction locks, mutual exclusions (mutexes) (in debugging mode only), and readers-writer locks.
  - I/O: includes the time taken to perform operations such as reading and writing data files, writing log files, reading binary logs, reading redo logs, and asynchronously reading redo logs.

Use Object Statistics

1. Check that the values of the OPT_TABLESTAT and OPT_INDEXSTAT parameters are ON. Example:

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

   > **Note** If these parameters cannot be found or their values are not ON, check that your RDS instance is running MySQL 5.7.

2. Query the TABLE_STATISTICS or INDEX_STATISTICS table in the information_schema database to obtain table or index statistics. Examples:

```
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 |
 +--------------+--------------+------------+-----------+
```

The following table describes parameters of the responses to the sample query requests.

Parameter	Description
TABLE_SCHEMA	The name of a database.

Parameter	Description
TABLE_NAME	The name of a table.
ROWS_READ	The number of rows read from a table.
ROWS_CHANGED	The number of rows modified in a table.
ROWS_CHANGED_X_INDEXES	The number of rows modified by using indexes in a table.
ROWS_INSERTED	The number of rows inserted into a table.
ROWS_DELETED	The number of rows deleted from a table.
ROWS_UPDATED	The number of rows updated in a table.
INDEX_NAME	The name of an index.

Use Performance Point

1. View the global variable settings of your RDS instance, as shown in the following example:

```
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    |
 +--------------------------------------+-------+
```

> **Note** If these variables cannot be found, check that your RDS instance is running MySQL 5.7.

2. Query the events_statements_summary_by_digest_supplement table in the performance_schema database to obtain the top 10 SQL statements in various dimensions. Example:

```
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 ?                     |          17
933 |
 | NULL               | dd148234ac7a20cb5aee7720fb44b7ea | SELECT SCHEMA ( )                         |         1006
|
 | information_schema | 2fb4341654df6995113d998c52e5abc9 | SHOW SCHEMAS                              |
2156 |
 | information_schema | 74af182f3a2bd265678d3dadb53e08da | SHOW TABLES                               |
3161 |
 | information_schema | d3a66515192fcb100aaef6f8b6e45603 | SELECT * FROM `TABLE_STATISTICS`
LIMIT ? |        2081 |
 | information_schema | b3726b7c4c4db4b309de2dbc45ff52af | SELECT * FROM `INDEX_STATISTICS`
LIMIT ? |        2384 |
 | information_schema | dd148234ac7a20cb5aee7720fb44b7ea | SELECT SCHEMA ( )                         |
129 |
 | test               | 2fb4341654df6995113d998c52e5abc9 | SHOW SCHEMAS                              |          342 |
 +--------------------+----------------------------------+-------------------------------------------+------------
--+
```

The following table describes parameters of the response to the sample query request.

Parameter	Description
SCHEMA_NAME	The name of a database.
DIGEST	The 64-byte hash string obtained from the **DIGEST_TEXT** parameter.
DIGEST_TEXT	The digest of an SQL statement.
ELAPSED_TIME	The total time spent executing an SQL statement. Unit: μs.
CPU_TIME	The time spent by CPU executing an SQL statement. Unit: μs.
SERVER_LOCK_TIME	The time occupied by metadata locks on the server during the execution of an SQL statement. Unit: μs.

Parameter	Description
TRANSACTION_LOCK_TIME	The time occupied by storage transaction locks during the execution of an SQL statement. Unit: μs.
MUTEX_SPINS	The number of mutex spins triggered during the execution of an SQL statement.
MUTEX_WAITS	The number of spin waits triggered by mutexes during the execution of an SQL statement.
RWLOCK_SPIN_WAITS	The number of spin waits triggered by readers-write locks during the execution of an SQL statement.
RWLOCK_SPIN_ROUNDS	The number of rounds in which the background thread looped in the spin-wait cycles triggered by readers-write locks during the execution of an SQL statement.
RWLOCK_OS_WAITS	The number of operating system waits triggered by readers-write locks during the execution of an SQL statement.
DATA_READS	The number of times the system read data from data files during the execution of an SQL statement.
DATA_READ_TIME	The time spent reading data from data files during the execution of an SQL statement. Unit: μs.
DATA_WRITES	The number of times the system wrote data into data files during the execution of an SQL statement.
DATA_WRITE_TIME	The time spent writing data into data files during the execution of an SQL statement. Unit: μs.
REDO_WRITES	The number of times the system wrote data into log files during the execution of an SQL statement.
REDO_WRITE_TIME	The time spent writing data into log files during the execution of an SQL statement. Unit: μs.
LOGICAL_READS	The number of times the system read logical pages during the execution of an SQL statement.
PHYSICAL_READS	The number of times the system read physical pages during the execution of an SQL statement.
PHYSICAL_ASYNC_READS	The number of times system read physical asynchronous pages during the execution of an SQL statement.

3. Query the IO_STATISTICS table in the information_schema database to obtain information about recent data read and write operations: Example:

```
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 |
 +---------------------+-----------+----------------+
```

The following table describes parameters of the response to the query request.

Parameter	Description
TIME	The point in time at which data read and write operations were performed.
DATA_READ	The number of times the system read data.
DATA_READ_TIME	The total time spent reading data. Unit: μs.
DATA_READ_MAX_TIME	The maximum time spent reading data. Unit: μs.
DATA_READ_BYTES	The total amount of data read. Unit: bytes.
DATA_WRITE	The number of times the system wrote data.
DATA_WRITE_TIME	The total time spent writing data. Unit: μs.
DATA_WRITE_MAX_TIME	The maximum time spent writing data. Unit: μs.
DATA_WRITE_BYTES	The total amount of data written. Unit: bytes.

7.Security

7.1. Data Protect

This topic describes the data protection feature provided by ApsaraDB RDS for MySQL. This feature controls permissions to perform high-risk operations.

Prerequisites

The instance runs one of the following MySQL versions:

- MySQL 8.0 (The kernel version is 20200430 or later.)
- MySQL 5.7 (The kernel version is 20200430 or later.)
- MySQL 5.6 (The kernel version is 20200430 or later.)

Context

Data protection takes effect on the following database operation commands:

- High-risk data operation commands
  - `Drop Table`
  - `Truncate Table`
  - `Alter Table Drop Paritition`
  - `Alter Table Truncate Partition`
  - `Alter Table Exchange Paritition`
  - `Drop Tablespace`
- Extended commands
  - `DROP View`
  - `ALTER View`
  - `Drop Function`
  - `Drop Procedure`
  - `Drop Trigger`
  - `Purge Binary Logs`

> ? **Note** Data protection is applied to the extended commands to ensure the running of application code.

Parameters

The data protection feature involves the following four parameters:

- rds_data_protect_level

  Specifies the level of data protection. Valid values:

  - NONE: disables data protection.
  - DDL: blocks DROP and TRUNCATE operations on databases and tables.
  - ALL: blocks all DROP and TRUNCATE operations, including the operations on views, stored procedures, functions, and triggers.

> ⓘ **Note** We recommend that you configure a data protection level in the non-maintenance or non-publishing phase and disable data protection in the maintenance or publishing phase.

- rds_data_protect_ignore

  Specifies a list of databases that do not need to be protected. For example, this parameter can be used in scenarios where development and production databases are created on the same RDS instance. You can specify that the development databases are not protected.

- rds_data_protect_admin

  Specifies which users can delete data when the rds_data_protect_control parameter is set to USER.

- rds_data_protect_control

  Specifies a data protection policy. The following protection policies are supported:

  - USER: Only users specified by the rds_data_protect_admin parameter or users who have the SUPER_ACL permissions can delete data. This value applies to most business scenarios on the cloud.
  - SUPER: Only users who have the SUPER_ACL permissions can delete data. You can use SUPER_ACL to implement precise data protection for common on-premises applications.
  - MAINTAIN: Only users with the SUPER_ACL and MAINTAIN permissions can delete data. The MAINTAIN permissions allow users to initiate connections from Alibaba Cloud. This value applies to scenarios where you want to delete data on Alibaba Cloud.
  - LOCAL: Only users with the SUPER_ACL and MAINTAIN permissions can delete data by logging on to the instance over local connections. This value applies to core applications. If you configure this value, you cannot delete data by logging on to the instance over remote connections. You must log on to the physical server.

Enable data protection

Data protection is in the invitational preview. You can submit a ticket to enable this feature.

7.2. Recycle bin

Data definition language (DDL) statements cannot be rolled back. If a table is unintentionally deleted by using a DROP TABLE statement, the table data may be lost. Alibaba Cloud provides the recycle bin feature that allows you to temporarily store deleted tables. You can specify a retention period within which you can retrieve the deleted tables. In addition, Alibaba Cloud provides the DBMS_RECYCLE package that is used to manage the deleted tables in the recycle bin.

Parameters

The following table describes the parameters that you must configure for the recycle bin feature.

Parameter	Description
loose_recycle_bin	Specifies whether to enable the recycle bin feature. You can enable this feature for your RDS instance or a specific session. You can reconfigure this parameter in the ApsaraDB for RDS console.
loose_recycle_bin_retention	The period for which you want to retain tables in the recycle bin. Unit: seconds. Default value: 604800. The default value indicates seven days. You can reconfigure this parameter in the ApsaraDB for RDS console.
recycle_scheduler	Specifies whether to enable the thread that is used to asynchronously delete tables from the recycle bin. This parameter is temporarily unavailable.
recycle_scheduler_interval	The polling interval that is followed by the thread to asynchronously delete tables from the recycle bin. Unit: seconds. Default value: 30. This parameter is temporarily unavailable.
recycle_scheduler_purge_table_print	Specifies whether to log the operations that are performed by the thread to asynchronously delete tables from the recycle bin. This parameter is temporarily unavailable.

Introduction

- Recycling and deletion
  - Recycling

    When you execute a `TRUNCATE TABLE` statement to delete a table, the system moves the deleted table to the recycle bin. Then, the system creates an empty table that has the same structure as the deleted table. The empty table resides in the same location as the deleted table.

    When you execute a `DROP TABLE or DROP DATABASE` statement to delete a table or a database, the system moves only the deleted tables to the recycle bin. The system deletes the other objects based on the following policies:

    - If no relationships exist between an object and the deleted tables, the system determines whether to retain the object based on the executed statement.
    - If an object is based on the deleted tables and may cause modifications to the data in these tables, the system deletes the object. These objects include triggers and foreign keys. The system does not delete column statistics. These statistics are stored to the recycle bin with the deleted tables.
  - Deletion

    The recycle bin starts a background thread to asynchronously delete tables from the recycle bin. These tables are stored in the recycle bin longer than the period that is specified by the **recycle_bin_retention** parameter. If a table in the recycle bin is large, the system starts another background thread to asynchronously delete the large table.
- Permission control

  When you start your RDS instance that runs MySQL, a database named __recycle_bin__ is initialized to store the data that is moved to the recycle bin. The __recycle_bin__ database is a system database. You cannot modify or delete the database.

You cannot delete tables from the recycle bin by executing `DROP TABLE` statements. However, you can use the `call dbms_recycle.purge_table('<TABLE>');` method to delete tables from the recycle bin.

> Note The account that you use must have the permissions to delete tables from your RDS instance and the recycle bin by executing DROP TABLE statements.

- Table naming in the recycle bin

  Tables in the __recycle_bin__ database originate from different databases and may have the same name. To ensure that each table has a unique name in the recycle bin, Alibaba Cloud implements the following naming conventions:

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

  The following table describes the parameters in the naming conventions.

Parameter	Description
Storage Engine	The name of the storage engine that is used by the table.
SE private id	The unique value that is generated by the storage engine to identify the table. For example, the unique value that is used to identify an InnoDB table is the ID of the table.

- Independent recycling

  The recycle bin configuration that you specify on an RDS instance is applied only to that instance. Therefore, the recycle bin configuration that you specify on your primary RDS instance will not be applied to its secondary, read-only, or disaster recovery RDS instances to which binary logs are replicated. For example, you can specify a 7-day retention period on your primary RDS instance and a 14-day retention period on the secondary RDS instances separately.

  > Note The storage usage of an RDS instance varies based on the retention period that you specify on that instance.

Precautions

- After you execute a `DROP TABLE` statement to delete a table, the system may migrate the related data file from the tablespace that stores the table. This applies if the __recycle_bin__ database and the table reside in different file systems. In addition, this process is time-consuming.
- A general tablespace may store more than one table. If you execute a DROP TABLE statement to delete a table from a general tablespace, the system does not migrate the related data file from the general tablespace.

Prerequisites

Your RDS instance runs MySQL 8.0.

Manage the recycle bin

AliSQL provides the following two management methods in the DBMS_RECYCLE package:

- show_tables

  Displays all of the tables that are temporarily stored in the recycle bin. The following code snippet is an example of the show_tables method:

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

  Example:

  ```
  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 11:01:4
  6 |
  | __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)
  ```

Parameter	Description
SCHEMA	The name of the database that stores the table after the table is moved to the recycle bin.
TABLE	The name of the table after the table is moved to the recycle bin.
ORIGIN_SCHEMA	The name of the database that stores the table before the table is moved to the recycle bin.
ORIGIN_TABLE	The name of the table before the table is moved to the recycle bin.
RECYCLED_TIME	The time when the table was moved to the recycle bin.
PURGE_TIME	The time when the table is expected to be deleted from the recycle bin.

- purge_table

  Manually deletes a table from the recycle bin. The following code snippet is an example of the purge_table method:

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

Note

- The TABLE variable specifies the new name of the table after the table is moved to the recycle bin.
- The account that you use must have the permissions to delete tables from your RDS instance and the recycle bin by executing DROP TABLE statements.

Example:

```
mysql> call dbms_recycle.purge_table('__innodb_1063');
Query OK, 0 rows affected (0.01 sec)
```

8.Best practices

8.1. Convert the storage engine of DRDS from InnoDB to X-Engine

This topic describes how to convert the storage engine of DRDS from InnoDB to X-Engine.

Context

Most users of existing ApsaraDB for RDS instances want to use X-Engine. These users have the following characteristics:

- Most RDS instances run MySQL 5.6 or MySQL 5.7. Few RDS instances run MySQL 8.0.
- A single instance has a large volume of data, which reaches the upper limit of disk space supported by the instance type. For example, an RDS instance with 4 CPU cores and 8 GB of memory supports up to 2 TB of local SSDs.
- The users also use DRDS. In addition, some users use an old version of DRDS and have customized functions, such as SQL passthrough.

To address the user requirements, Alibaba Cloud allows you to convert the storage engine of DRDS from InnoDB to X-Engine by following the procedure in this topic.

> **Note** For more information about X-Engine, see X-Engine overview.

Conversion plan

RDS for MySQL 8.0 instances provide consistent API operations and user experience regardless of whether they use InnoDB or X-Engine. In this situation, after a DRDS upgrade, we recommend that you convert the storage engines for the RDS instances from InnoDB to X-Engine one by one. For example, if a DRDS instance has eight RDS instances, first convert the storage engine for one of the eight RDS instances. Monitor the instance running for a period of time. If you confirm that no compatibility or performance issues occur, convert the storage engines for the remaining seven RDS instances.

Compression ratio verification before conversion

Before conversion, we recommend that you purchase an RDS instance that is powered by X-Engine with the same specifications as the existing RDS instance that is powered by the InnoDB storage engine. Then, you can use Alibaba Cloud Data Transmission Service (DTS) to import data from the existing RDS instance to the purchased instance. This way, you can check the compression efficiency. The compression efficiency allows you to determine the following items:

- Instance storage capacity

  Based on the compression efficiency, you can determine the instance specifications that you need to purchase after you convert a storage engine from InnoDB to X-Engine. For example, if the space required after compression is below 30% of the original space, you can purchase an RDS instance with 1 TB of disk space after you convert the storage engine of an RDS instance that originally requires 3 TB of disk space. Alternatively, you can purchase an RDS instance with the same specifications to reserve storage space for future business development.

- Number of database shards

  After storage space is reduced, you can reduce the number of database shards. For example, you can merge databases that are distributed across instances to one instance. This reduces costs.

> ⓘ **Note** You can release the RDS instance that is powered by X-Engine after you complete the verification, or you can clear the instance for official conversion later.

Conversion procedure

1. Upgrade DRDS to a version later than V5.4.2-15744202.

   > ⓘ **Note**
   > - If the DRDS version is later than v5.4.2-15744202, skip this step.
   > - To ensure compatibility, you must adjust the service code. This applies if your business is based on specific API operations that are provided in an earlier version of DRDS, for example, the SQL passthrough function for performance optimization.

2. Select an RDS instance with the InnoDB storage engine from DRDS as the first instance for conversion. Export table creation statements and change the engine type to X-Engine. Then, create an RDS instance with the target specifications and X-Engine. Alternatively, use the RDS instance that you created when you verify the compression efficiency and import the table structure scripts into this instance.
   - For more information about how to create an RDS instance, see Create an ApsaraDB RDS for MySQL instance.
   - For more information about how to import and export table creation statements, see Convert the storage engine from InnoDB, TokuDB, or MyRocks to X-Engine.

   > ⓘ **Note** If you use DTS to migrate data, the engine type of the source instance is inherited by default. You must separately export the table creation statements and change the engine type to X-Engine before you can migrate data to the destination instance that is powered by X-Engine.

3. Use DTS to synchronize data from the RDS instance with the InnoDB storage engine to the RDS instance with X-Engine. For more information about data synchronization, see Configure two-way data synchronization between ApsaraDB RDS for MySQL instances.

   > ⓘ **Note** You can use the two-way synchronization function of DTS to ensure data consistency between the two RDS instances.

4. Modify DRDS routing rules and redirect the access requests to the RDS instance with the InnoDB storage engine to the RDS instance with X-Engine. If you want to modify the DRDS routing rules, submit a ticket.

> ⓘ **Note** Run the first RDS instance with X-Engine for five days. Monitor the instance running and consider the request processing time, exception information, and two-way synchronization progress. If an exception occurs, you must make sure that services can be switched back to the original RDS instance with the InnoDB storage engine. For more information, see View the resource and engine metrics of an ApsaraDB RDS for MySQL instance.

5. After you confirm that the first RDS instance with X-Engine is running as normal, proceed with the conversion for 30% to 50% of the remaining RDS instances and then monitor the instance running for three to five days. In this case, repeat the preceding steps 2 to 4.

   > ⓘ **Note** Do not release or bring the original RDS instances with InnoDB offline because these instances are required to perform DTS two-way synchronization with new RDS instances with X-Engine.

6. Perform the conversion for all the remaining RDS instances. After you complete the conversion for all the instances of DRDS, monitor the instance running for three to five days. If the new instances run as normal, release all DTS synchronization links and the original RDS instances with InnoDB.

8.2. DingTalk secures App Store top rank with X-Engine

This topic describes how X-Engine of ApsaraDB for RDS helps reduce the costs of DingTalk and implement online collaborative offices.

Context

DingTalk is a leading enterprise-grade instant messaging (IM) tool in China. It serves hundreds of millions of users across China. Its basic functions include video conferences and daily reports. DingTalk Open Platform also provides various office automation (OA) applications to facilitate communication between co-workers.

In 2020, COVID-19 is a serious problem. To avoid the risk of infection caused by work in centralized offices, a large number of employees have opted to work from home. The demand for collaborative office tools suddenly increases. In this situation, DingTalk is quickly elevated to the top of the App Store download list. This results in a sharp increase in DingTalk access. DingTalk is based on the elastic infrastructure provided by Alibaba Cloud. This ensures that all the traffic peaks are smoothly handled.

To serve a large number of users, DingTalk must ensure the timely and correct delivery of messages, and provide specific functions, such as read and unread messages. Unlike user-level IM tools such as WeChat, enterprise-grade IM tools must include the permanent storage of chat records and provide the multi-terminal roaming function. This function allows users to receive messages from multiple terminals. As the number of users sharply increases, DingTalk faces challenges in the costs incurred by the permanent storage of chat records while ensuring the performance of read and write operations on the chat records.

To address these challenges, DingTalk uses X-Engine as the storage engine for messages. This achieves a balance between the costs and performance. X-Engine has the following advantages:

- The storage space required by X-Engine is about 62% less than that required by the InnoDB storage engine.
- Specific database functions such as transactions and secondary indexes are supported.
- Service code can be migrated to RDS instances that are powered by X-Engine without changes.
- X-Engine separates hot and cold data to accelerate the processing of current messages. It also implements the most efficient compression algorithm for historical messages.

X-Engine storage efficiency is tested on two datasets: Link-Bench and Alibaba internal transaction business. In the test, X-Engine requires 2-fold less storage space than the InnoDB storage engine with compression enabled, and 3- to 5-fold less storage space than the InnoDB engine with compression disabled.

Low costs achieved by X-Engine

X-Engine adopts the following technologies to ensure low costs:

- Compact pages

  X-Engine uses the Copy-on-write technology to write new data to new pages without updating the original pages. The new pages are read-only and cannot be directly updated. These pages are stored in a compact manner, and the data is compressed by using specific algorithms, such as prefix encoding. This improves the storage efficiency. You can use the compaction operation to clear invalid records. This ensures a compact arrangement of valid records. X-Engine requires only 10% to 50% of the storage space compared with conventional storage engines, such as InnoDB.

- Data compression and cleaning of invalid records

  Pages after encoding can be compressed by using general compression algorithms, such as zlib, zstd, and snappy. Data at a low level of a log-structured merge-tree (LSM tree) is compressed by default.

  Data compression sacrifices computing resources for storage space. We recommend that you select compression algorithms that provide a low compression ratio and a high speed of compression and decompression. After a large number of comparative tests, X-Engine selects zstd as the default compression algorithm with additional support for other compression algorithms.

  In addition, the compaction operation is introduced to delete invalid records. This way, only valid records are retained. The more frequently the compaction operation is performed, the lower the proportion of invalid records, and the higher the storage efficiency. Therefore, you must perform the compaction operation at a suitable frequency.

  The X-Engine team also develops the field-programmable gate array (FPGA) compaction technology to reduce the computing resource consumption of the compaction operation. This technology uses heterogeneous computing hardware to accelerate the compaction process. It streamlines compaction and compression operations by using FPGA hardware. On a host without FPGA hardware, X-Engine can use a suitable scheduling algorithm to save storage space at a lower performance cost.

- Intelligent separation between hot and cold data

In normal access to a storage system, most access requests direct to a small portion of data. This is why the cache works. In an LSM tree structure, frequently accessed data is stored at a high level to a fast storage device, such as NVM and DRAM. Infrequently accessed data is stored at a low level to a slow storage device. This is the hot and cold data separation in X-Engine.

The separation algorithm completes the following tasks:

- In the compaction operation, the pages and records that are least likely to be accessed are selected and moved to the bottom of the LSM tree.
- Current hot data is selected and backfilled to memory (BlockCache and RowCache) in the compaction or dump process. This prevents compromised performance from jitters in cache hit rates.
- The AI algorithm recognizes data that may be accessed in the future and pre-reads it into memory. This increases the hit rates for accessing cache at the first time.

Hot data and cold data are accurately identified to avoid computing resource waste due to invalid compression or decompression. This improves system throughput.

For more information, see X-Engine overview.

Related papers

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

8.3. Storage engine that processes trillions of Taobao orders

Taobao historical orders are supported by a PolarDB-X cluster based on X-Engine. This fixes the known issues caused by the use of HBase databases, reduces storage costs, and allows users to query orders at all times.

Context

Taobao is the largest online shopping platform in China. It serves more than 700 million active users and tens of millions of sellers.

The large platform provides support for about 100 million transactions on physical and virtual commodities every day. Each transaction process involves different phases, such as member information verification, commodity library inquiry, order creation, inventory reduction, discounts, order payment, logistics information update, and payment confirmation. Each phase involves database record creation and status update. The entire process requires hundreds of database transactions, and the entire database cluster performs tens of billions of transaction read and write operations every day. The database team faces the challenge of huge storage costs incurred by the increasing volume of data every day while ensuring the stable performance of the database system.

Orders are the most critical information in the entire transaction process and must be permanently stored in databases. If a transaction dispute arises, the order records must be provided for users to query. Over the last nearly 17 years since Taobao was founded in 2003, the total number of database records related to orders has reached the trillion level, and the disk space occupied by these records has exceeded the PB level.

The following sections describe how Taobao ensures low latency every time that users query orders without increasing storage costs.

Architecture evolution

The architecture of transaction order databases has evolved through four phases as the traffic increases.

- Phase 1

  In this initial phase, the traffic was low, and Taobao used an Oracle database to store all order information. Order creation and historical order queries were performed on the same database.

- Phase 2

  As the volume of historical order data increased, the single database can no longer meet the performance and capacity requirements at the same time. Therefore, the database was split into an online database and a historical database. Historical orders that were generated three months ago were migrated to the historical database. However, the historical database contained too much data to allow queries. In this phase, users can only query historical orders for the last three months.

- Phase 3

  To fix the issues related to storage costs and historical order queries, Taobao migrated historical orders to an HBase database.

  HBase provides both primary and indexing tables. Users can query the primary tables for order details and the indexing tables for order IDs based on the IDs of buyers or sellers. In this situation, orders may not be migrated to the historical order database in chronological order, and many types of orders are not migrated to this database. As a result, the order list is not sorted by time, and users cannot search for orders by using the listed sequence of orders.

- Phase 4

  The historical order database is built in a PolarDB-X cluster based on X-Engine. This reduces storage costs and fixes the out-of-time-order issue.

Business pain points

The architecture evolution shows that the business team and the database team have suffered from the following pain points over the last 10 years since the historical order database was introduced:

- Storage costs

  A large volume of data is written every day and the data is never deleted. Low-cost storage is required.

- Query performance

  Diversified query functions are required to meet specific requirements, such as query by time and query by order type. Databases must support secondary indexes that can ensure data consistency and performance.

- Query latency

  The query latency must be low to ensure user experience. For example, queries on historical orders of 90 days ago are much fewer than those in the last 90 days, but these queries still require low latency.

Historical order database solution based on X-Engine

The transaction order system has been iterated for 10 years in terms of the architecture, where online and historical databases are separated. Most service code is compatible with this architecture, which is also inherited in this solution. This architecture mitigates risks caused by the reconstruction and migration of service code. Initially, the HBase cluster is replaced with the PolarDB-X cluster that is based on X-Engine.

- The online database is still deployed in a MySQL cluster that is based on the InnoDB storage engine, and stores only orders for the last 90 days. The data volume is small, which ensures a high cache hit rate and reduces read/write latency.
- Orders that were generated 90 days ago are migrated from the online database to the historical database through data synchronization and are deleted from the online database.
- The storage engine of the historical database is converted to X-Engine. This database stores all orders that were generated 90 days ago. If you want to perform read or write operations on these orders, access the historical database.

After this new solution is used, the storage costs are the same as those incurred by the use of the HBase database. The historical database is compatible with the online database, and identical indexes can be created on the two databases. This fixes the out-of-time-order issue. In the historical database, hot data is separated from cold data to reduce read latency.

Summary

The transaction order records on Taobao are stored in the streamline mode. Recently written records are frequently accessed at first, and the access frequency sharply decreases over time. X-Engine separates hot and cold data and is suitable for this type of access. A single database cluster based on X-Engine is sufficient for these access scenarios.

Assume that a new or existing business needs to store a large number of streamline records. If hot data and cold data are not separated on the business layer, we recommend that you use the distributed PolarDB-X cluster based on X-Engine to ensure scalability without increasing storage costs.

Alibaba Cloud has launched X-Engine. You can purchase it if required. For more information, see Create an ApsaraDB RDS for MySQL instance.

8.4. Best practices for X-Engine testing

This topic describes how to use SysBench to test the performance of X-Engine used with ApsaraDB RDS for MySQL. This helps you evaluate the performance of X-Engine.

Prerequisites

- The default storage engine of your RDS instance is X-Engine.

Note If X-Engine is used, the value of the XENGINE parameter must be DEFAULT in the Support column.

```
MySQL [(none)]> show storage engines;
+--------------------+---------+----------------------------------------------------------------+--------------
+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoi
nts |
+--------------------+---------+----------------------------------------------------------------+--------------
+------+------------+
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL
| NULL       |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO
| NO   | NO         |
| XENGINE            | DEFAULT | X-Engine storage engine                                        | YES          | YES  | YE
S          |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO
| NO   | NO         |
| InnoDB             | YES     | Supports transactions, row-level locking, and foreign keys     | YES
| YES  | YES        |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO
| NO         |
| Sequence           | YES     | Sequence Storage Engine Helper                                 | NO           | NO   | N
O          |
| 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)
```

- The table used for testing is stored in X-Engine.

Note In this example, the table used for testing is created with the ENGINE parameter set to XENGINE. If you set the ENGINE parameter to INNODB or another storage engine, the table used for testing is stored in the specified storage engine rather than 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_gener
al_ci |
+---------+--------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------
--------------------------+
1 row in set (0.01 sec)
```

Note We recommend that you use SysBench 1.1.0 or later.

Use DTS to test storage space usage

We recommend that you use Alibaba Cloud Data Migration Service (DTS) to migrate your actual database data to your RDS instance and then check the disk usage of X-Engine. In this case, the test results are more close to your actual business situation. X-Engine adopts technologies such as space-friendly storage format, prefix encoding, tiered storage, and efficient compression algorithms to reduce disk usage. The actual effect of these technologies varies based on schemas and record length in your databases. Therefore, using your actual database data allows you to obtain more accurate test results.

DTS does not support an automatic conversion of the storage engine during data migration. You must manually create databases and tables on your RDS instance that runs X-Engine, set the ENGINE parameter to XENGINE in the table creation statements as described in the "Prerequisites" section, and migrate data by using DTS. Do not migrate the schemas.

We recommend that you do not execute SQL statements immediately after the data import is complete. You can monitor the CPU utilization and input/output operations per second (IOPS) usage of your RDS instance in the ApsaraDB for RDS console. After the CPU utilization and IOPS approach zero, you can execute SQL statements to test the performance of X-Engine. In this case, the disk usage is calculated more accurately. This is because the log-structured merge-tree (LSM tree) architecture used by X-Engine depends on background asynchronous tasks to implement functions such as data compression. These functions reduce storage costs. The background asynchronous tasks take some time and consume CPU and IOPS resources.

For more information, see Migrate data between ApsaraDB for RDS instances.

Use SysBench to test the storage space usage

To fully test the compression efficiency of X-Engine, we recommend that you set the table_size parameter in the following command to a large value within the disk size range allowed for your RDS instance.

We recommend that you monitor the CPU utilization and IOPS usage of your RDS instance in the ApsaraDB for RDS console. After the CPU utilization and IOPS approach zero, the space usage is calculated more accurately.

Run the following command to test the storage space usage:

```
sysbench /usr/share/sysbench/oltp_update_index.lua \
 -- mysql-host=[The endpoint of your RDS instance] \
 --mysql-user=sbtest \
 --mysql-password=sbtest@888 \
 --mysql-db=sbtest \
 --threads=32 \
 --tables=32 \
 --table_size=1000000000 \
 --mysql-storage-engine=XENGINE \
 prepare
```

Use SysBench to test performance

If you use SysBench for performance testing, we recommend that you set the rand-type parameter to zipfian and the rand-zipfian-exp parameter to 0.9.

- rand-type: specifies the type of the distribution that is used to generate random numbers in SQL statements.
- Zipfian distribution: a common data distribution with hot spots. When the rand-zipfian-exp parameter is set to 0.9, the random numbers generated by using Zipfian distribution are closer to those generated in the real world. The test results are more valuable in comparison to those generated by using the default uniform distribution.

We recommend that you conduct a single test for a long period of time, such as 3,600 seconds. The test results of average performance obtained from a long-time test is more valuable and less affected by potential interference factors.

We recommend that you use a large number of threads, for example, 512 threads, to test the throughput.

To improve the performance of X-Engine by configuring parameters, contact your Alibaba Cloud account manager or after-sales engineers. You can also submit a ticket to receive consulting services.

Run the following command to test the performance:

```
sysbench /usr/share/sysbench/oltp_point_select.lua \
  -- mysql-host=[The endpoint of your RDS instance] \
  --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
```

Use a Python script to perform multiple tests at a time

If you want to perform multiple tests at a time by using SysBench, we recommend that you use a Python script that can automatically perform the tests and record the test results. When you execute the script, you are prompted to enter the endpoint of your RDS instance. Example:

```
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 complete
    # its asynchronous background compactions.
    time.sleep(1000)
```

8.5. Use DMS to archive data to X-Engine

This topic describes how to use the task orchestration feature of Data Management Service (DMS) to regularly migrate historical data from an ApsaraDB RDS for MySQL database that uses the InnoDB engine to an ApsaraDB RDS for MySQL database that uses X-Engine. This is a cost-effective storage solution.

Prerequisites

Database instances are registered in DMS and meet the following conditions:

- The source database is an ApsaraDB RDS for MySQL database that uses the InnoDB engine.
- The destination database is an ApsaraDB RDS for MySQL database that uses X-Engine. For information about how to create an ApsaraDB RDS for MySQL instance, see Create an ApsaraDB RDS for MySQL instance.
- Both the source and destination instances are in Secure Collaboration mode.
- Cross-database query is enabled for both the source and destination instances. For information about how to enable this feature, see Modify an instance.

> **Note** In this example, the database link of the source instance is dblink_src_rds, and that of the destination instance is dblink_target_rds.

Context

As business develops and data accumulates in a database, various problems may arise, such as surging storage costs and reduced performance of the database. Therefore, Alibaba Cloud provides a cost-effective storage solution based on X-Engine, a storage engine that is developed by Alibaba Cloud. Assume that your business is supported by an ApsaraDB RDS for MySQL database that uses the InnoDB engine. You can migrate historical data in this database to an ApsaraDB RDS for MySQL database that uses X-Engine, where data can be stored at lower costs. In this way, only hot data is stored in the business database. This improves the security and availability of your business and reduces storage costs.

> **Note** X-Engine is an online transaction processing (OLTP) storage engine that is developed by Alibaba Cloud. This storage engine is widely used in many business systems of Alibaba Group, including the transaction history database and DingTalk chat history database. This significantly reduces business costs. In addition, X-Engine is a crucial database technology that empowers Alibaba Group to cope with bursts of traffic that may surge to hundreds of times greater than usual during Double 11. X-Engine provides similar performance as the InnoDB engine, but costs far less than the InnoDB engine. Therefore, X-Engine is a highly cost-effective storage engine. For more information, see X-Engine overview.

The process is shown in the figure below:

You can use the task orchestration feature of DMS to automatically and regularly migrate historical data to a database that uses X-Engine. This helps reduce labor costs.

In this example, a task flow that consists of four tasks is created by using the task orchestration feature in DMS. The first task in the task flow creates a destination table in the ApsaraDB RDS for MySQL database that uses X-Engine. The second task backs up historical data in a source table, namely, data that has been stored in the source table for more than a month, to the destination table on a daily basis. The third task deletes the historical data in the source table. Then, the last task executes an OPTIMIZE TABLE statement to optimize the source table. This task flow is run at a specified time point every day. The source database only stores data that is generated over the last month. Large amounts of historical data are migrated to the destination database that uses X-Engine, which is more cost-effective.

Create a task flow

1. Log on to the DMS console.
2. In the top navigation bar, choose **Data Factory** > **Task Orchestration**. The Home tab of the Task Orchestration page appears.
3. In the **Free orchestration tasks** section, click **New task flow**.
4. In the **New Task Flow** dialog box, enter relevant information in the **Task Flow Name** and **Description** fields and click **OK**. In this example, set the task flow name to `Rds_innodb_to_X-Engine` and enter `Rds_innodb_to_X-Engine demo` in the Description field. The **Task Orchestration** page appears.

Create tasks

Create the following tasks in the task flow:

- Create a destination table: This task creates a destination table in the destination database. This table will be used to store historical data of the source table.
- Back up historical data: This task uses cross-database query to back up historical data from a source table in the source database to the destination table.
- Delete the historical data in the source table: After the historical data in the source table is backed up to the destination table, this task deletes the historical data in the source table.
- Optimize the source table: After the historical data in the source table is deleted, this task executes an OPTIMIZE TABLE statement to optimize the source table and save storage space.

1. On the **Task Orchestration** page, create and configure the following tasks:Create a destination table
   i. In the navigation tree, find the **MySQL** task node and drag the task node to the canvas.
   ii. Double-click this task node on the canvas, modify its name, and then press Enter. In this example, rename the task node as Create a destination table.
   iii. Click this task node. The **Content** tab appears on the right.
   iv. On the **Content** tab, select the destination database from the drop-down list. In this example, select the ApsaraDB RDS for MySQL database that uses X-Engine.

v. Enter an SQL statement for creating a destination table in the destination database and click **Save**. In this example, enter the following SQL statement:

```
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;
```

> **Note** In this example, the name of the destination table is `target_xengine_tbl`. The schema of this table must be the same as that of the source table, which is named `src_innodb_tbl`.

Back up historical data

i. In the navigation tree, find the **Cross Database SQL** task node and drag the task node to the canvas.

ii. Double-click this task node on the canvas, modify its name, and then press Enter. In this example, rename the task node as Back up historical data.

iii. Click this task node. The **Content** tab appears on the right.

iv. On the Content tab, enter an SQL statement for backing up historical data from the source table to the destination table and click **Save**. In this example, enter the following SQL statement:

```
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}';
```

> **Note** This SQL statement is used to back up historical data from the source table in the source database to the destination table in the destination database.
>
> - In this example, the database link `dblink_src_rds` refers to the source ApsaraDB RDS for MySQL instance that uses the InnoDB engine. The database link `dblink_target_rds` refers to the destination ApsaraDB RDS for MySQL instance that uses X-Engine.
> - The following two variables are used in the WHERE clause to specify the range of historical data to be backed up: `${thirty_one_days_ago}` and `${thirty_days_ago}`.

v. Click the blank area on the canvas. The Scheduling tab appears on the right. Click the **Variables** tab, configure one or more variables as needed, and then click **Save**. In this example, configure two variables: `thirty_one_days_ago` and `thirty_days_ago`, as shown in the following figure.

Delete the historical data in the source table

i. In the navigation tree, find the **MySQL** task node and drag the task node to the canvas.

ii. Double-click this task node on the canvas, modify its name, and then press Enter. In this example, rename the task node as Delete the historical data in the source table.

iii. Click this task node. The **Content** tab appears on the right.

iv. On the **Content** tab, select the source database from the drop-down list. In this example, select the ApsaraDB RDS for MySQL database that uses the InnoDB engine.

> **Note** This task is designed to delete the historical data, which has been backed up to the destination table, in the source table.

v. Enter an SQL statement for deleting the historical data in the source table and click **Save**. In this example, enter the following SQL statement:

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

Optimize the source table

i. In the navigation tree, find the **MySQL** task node and drag the task node to the canvas.

ii. Double-click this task node on the canvas, modify its name, and then press Enter. In this example, rename the task node as Optimize the source table.

iii. Click this task node. The **Content** tab appears on the right.

iv. On the **Content** tab, select the source database from the drop-down list. In this example, select the ApsaraDB RDS for MySQL database that uses the InnoDB engine.

v. Enter an SQL statement for optimizing the source table and click **Save**. In this example, enter the following SQL statement:

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

> **Note** To prevent the OPTIMIZE TABLE statement from affecting online business, we recommend that you allow changing schemas without locking tables for the source ApsaraDB RDS for MySQL instance, as shown in the following figure. For more information, see Modify an instance.

2. On the canvas, connect the four task nodes to form a task flow. Draw lines between the task nodes in the order that the task nodes are created.

Configure scheduling properties

1. Click the blank area on the canvas. The **Scheduling** tab appears on the right. Turn on the switch at the top of the **Scheduling** tab and complete the configurations. In this example, configure the task flow to be run at 01:00 every day, as shown in the following figure.

> **Note** In this example, to avoid business peak hours, the task flow is configured to be run at 01:00 every day. You can customize the scheduling properties based on your actual needs.

2. Click **Save**.