Data Warehouse Service
8.2.0

Technical White Paper

Issue	02
Date	2023-01-04

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Contents

1 GaussDB(DWS)

1.1 Product Positioning

GaussDB for Data Warehouse Service (GaussDB(DWS)) is an online data processing database based on the public cloud infrastructure and platform, and provides scalable, fully managed, and out-of-the-box analytic database services. It is a native cloud service based on the Huawei converged data warehouse GaussDB, and is fully compatible with ANSI SQL:99 and SQL:2003 standards, as well as the PostgreSQL and Oracle ecosystems. GaussDB(DWS) provides competitive solutions for PB-level big data analytics in various industries.

DWS comes in four types: standard data warehouse, cloud data warehouse, stream data warehouse, and hybrid data warehouse, helping you build the top data warehouse in terms of enterprise-level kernel, real-time analysis, collaborative computing, convergent analysis, and cloud native.

Oriented to data analysis scenarios, the standard data warehouse provides enterprise-level data warehouse services with high performance, high scalability, high reliability, high security, and easy O&M. It is capable of data analysis at a scale of 2048 nodes and 20 petabytes of data. A company can migrate its large-scale data warehouse to the cloud and without performance deterioration.

The stream data warehouse is built on top of the standard data warehouse. It integrates stream and time sequence engines to provide real-time data ingestion and high-concurrency real-time analysis capabilities. It can be used for IoT real-time analysis.

The cloud data warehouse uses decoupled storage and compute. It is a flexible data warehouse that supports cold and hot data analysis, elastic scaling of storage and computing resources, on-demand resource provisioning, and pay-per-use pricing. It can be used for small- and medium-sized data warehouses with fewer than 50 nodes, which are suitable for the analytics services that integrate databases, warehouses, data marts, and data lakes.

The hybrid data warehouse provides high-concurrency, high-performance, and low-latency transaction processing capabilities based on large-scale data query and analysis capabilities. It is suitable for hybrid transaction/analytical processing (HTAP). A database can be used for both production and analysis. Currently, HA features are not supported. It can be deployed only in standalone mode, which is cost-effective and can be used for lightweight services.

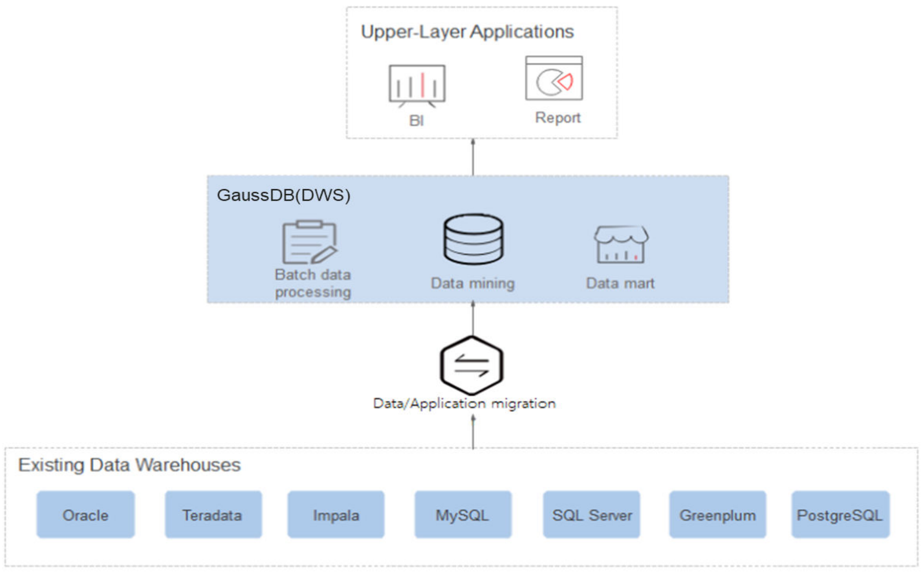
GaussDB(DWS) can be used in a wide range of fields, such as finance, Internet of Vehicles (IoV), government and enterprise, e-commerce, energy, and carrier. It has been listed in the Gartner Magic Quadrant for Data Management Solutions for Analytics for two consecutive years, thanks to its large-scale scalability, enterprise-grade reliability, and higher cost-effectiveness over conventional data warehouses.

1.2 Application Scenario

Data Warehouse Migration

The data warehouse is an important data analytics system for enterprises. However, in this digital age, enterprises' own data warehouses fail to process the ever-growing service volumes due to their poor scalability and high costs. GaussDB(DWS) provides a good answer to these pain points. It is an enterprise-grade cloud data warehouse that boasts high performance, low cost, and smooth scale-out, helping enterprise data warehouses handle the growing data volumes at ease.

Figure 1-1 Data warehouse migration



Advantages

- Seamless migration

  GaussDB(DWS) provides various migration tools to ensure the smooth migration of popular data analytics systems, such as Teradata, Oracle, MySQL, SQL Server, PostgreSQL, Greenplum, and Impala.

- Compatibility with conventional data warehouses

  GaussDB(DWS) supports the SQL 2003 standard and stored procedures. It is compatible with some Oracle syntax and data structures, and can be seamlessly connected to common BI tools, saving service migration efforts.

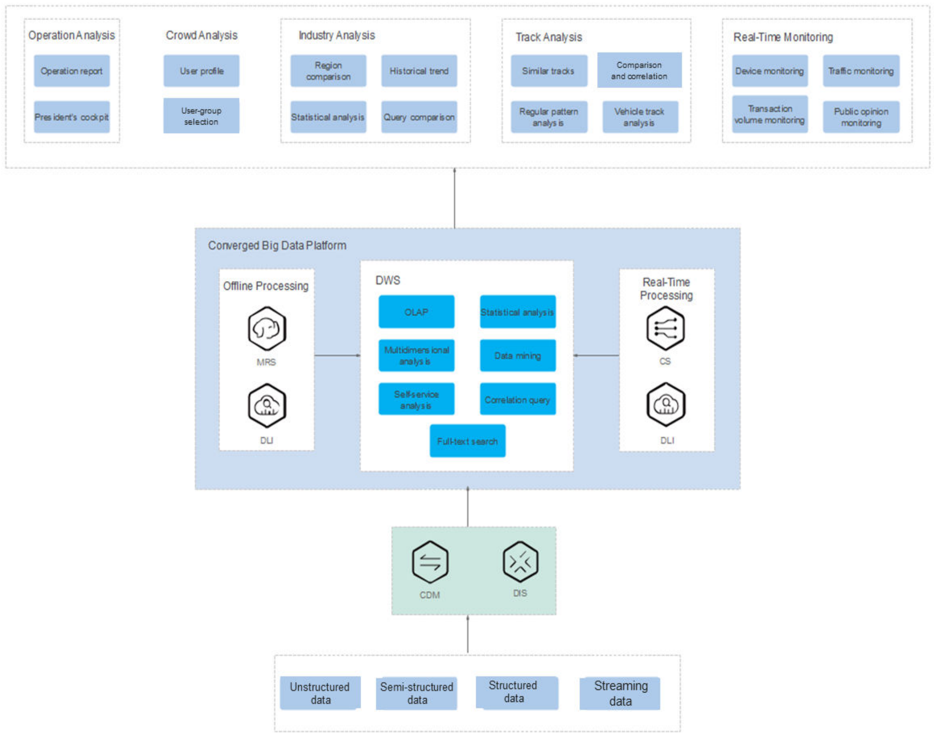
- Secure and reliable

  GaussDB(DWS) supports data encryption and connects to Database Security Service (DBSS) to ensure data security on the cloud. In addition, GaussDB(DWS) supports automatic full and incremental backup of data, improving data reliability.

Convergent Analysis of Big Data

Data has become the most important asset. Enterprises must be able to integrate their data resources and build big data platforms to mine the full value of their data. How to quickly mine values from massive data becomes a key factor for customers to implement predictive analysis.

Figure 1-2 Convergent analysis of big data



Advantages

- Unified analysis entrance

  The GaussDB(DWS) SQL serves as the unified entrance of upper-layer applications, so that application developers can access all data using the SQL.

- Real-time interactive analysis

  Analysis personnel can obtain immediately-actionable information from the big data platform in real time using broad analysis requests.

- Auto scaling

  Adding nodes allows you to easily expand into PB-range capacity while enhancing query and analysis performance of the system.

Enhanced ETL + Real-Time BI Analysis

The data warehouse is the pillar of the BI system for collecting, storing, and analyzing massive volumes of data. It powers business decision analysis for the IoT, finance, education, mobile Internet, and Online to Offline (O2O) industries.

Advantages

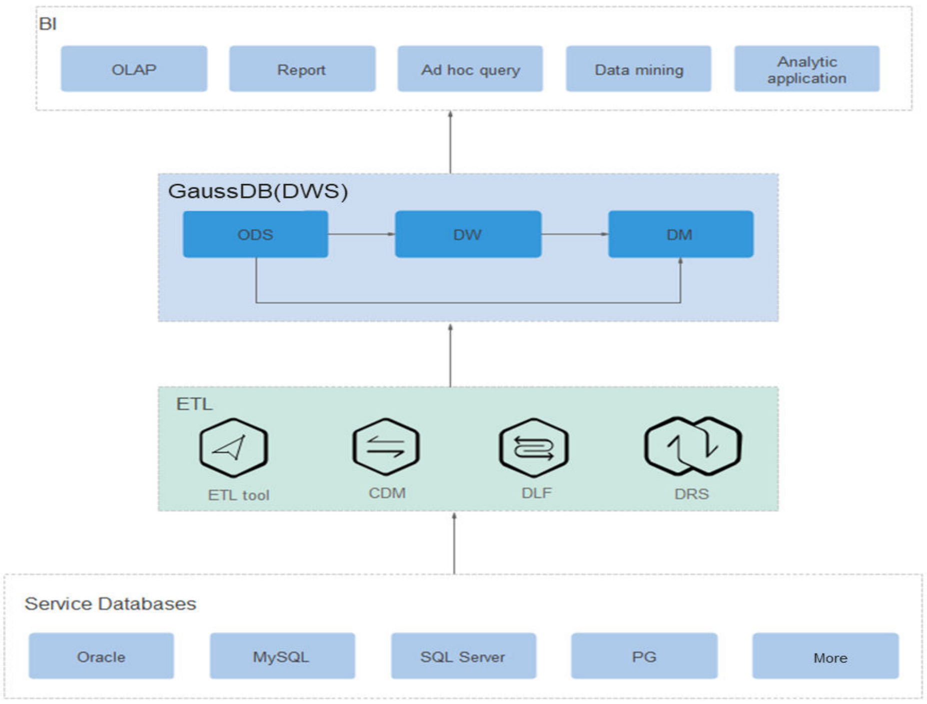
- Data migration

  Ability to import data in batches in real time from multiple data sources.
- High Performance

  Cost-effective PB-level data storage and response to correlation analysis of trillions of data records within seconds.
- Real-time processing

  Real-time consolidation of service data to produce actionable insights in operational decision-making.

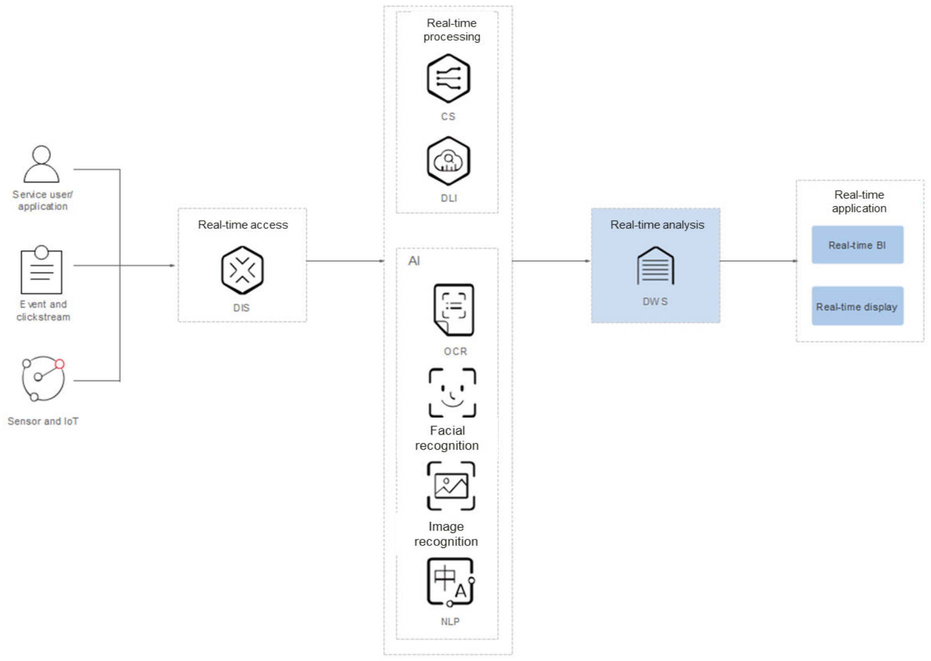
Figure 1-3 Enhanced ETL + real-time BI analysis



Real-Time Data Analysis

In the mobile Internet and IoT domains, huge volumes of data must be processed and analyzed in real time to extract the full value from data. GaussDB(DWS) quickly imports and queries data and supports real-time data analysis using times series, stream, and AI engines.

Figure 1-4 Real-time data analysis



Advantages

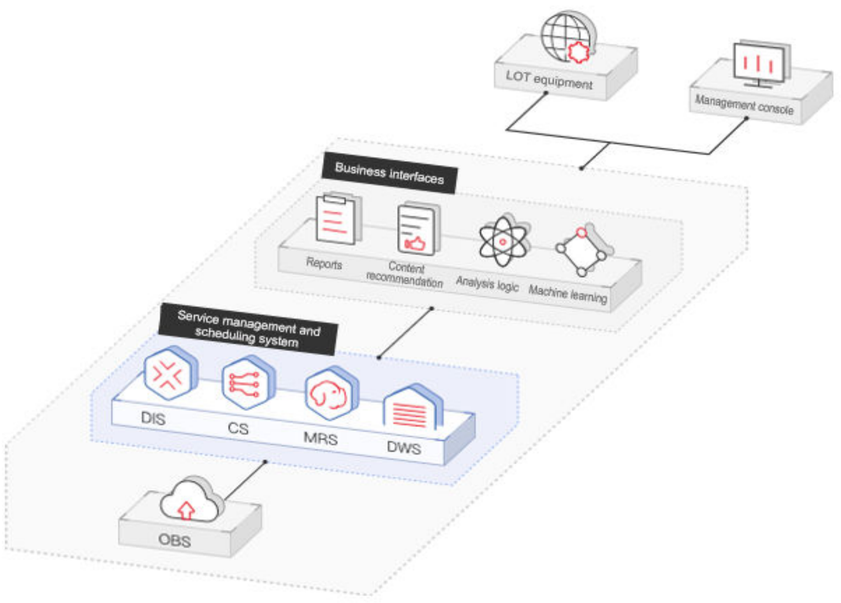
- Real-Time import of streaming data

  Data from IoT and Internet applications can be written into GaussDB(DWS) in real time after being processed by the stream computing and AI services.
- Real-Time Monitoring and Prediction

  Device monitoring, control, optimization, supply, self-diagnosis, and self-healing based on data analysis and prediction.
- Convergent AI analysis

  You can conduct association analysis on results of AI-based image and text data analysis and other service data on GaussDB(DWS).
- **IoT**

Figure 1-5 IoT



GaussDB(DWS) helps you analyze massive amounts of data from Internet of Things (IoT) in real time and perform optimization based on the results. It is widely used in industrial IoT, O2O service system, and IoV solutions.

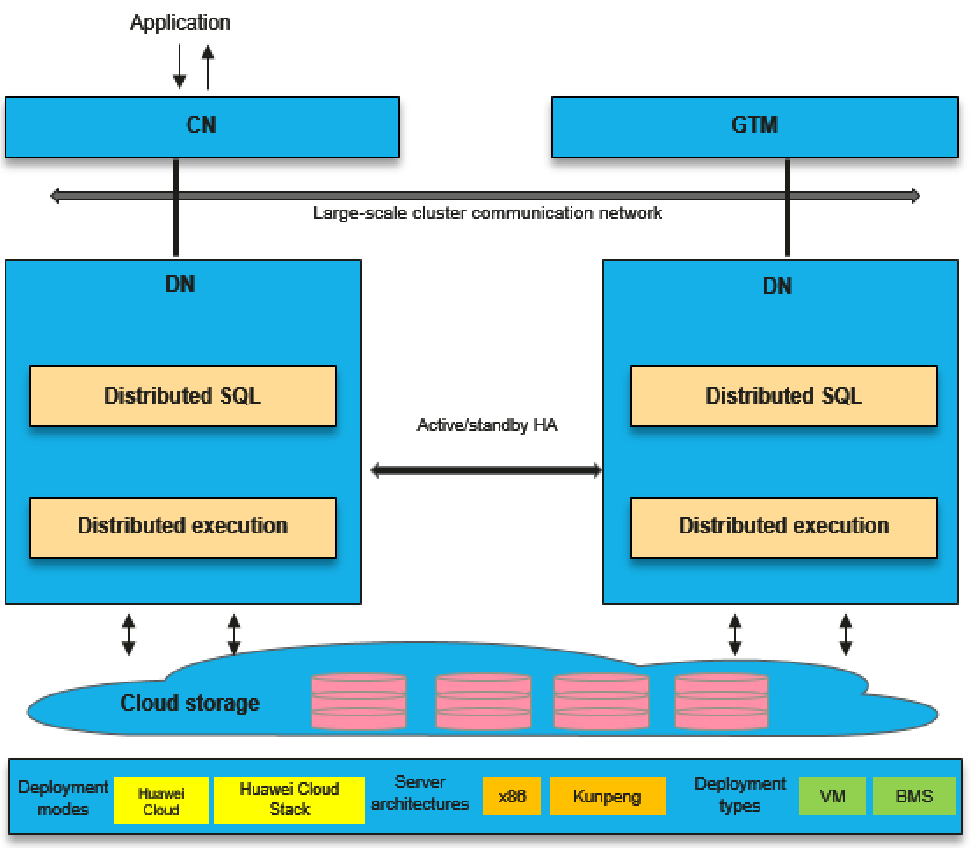
Advantages of GaussDB(DWS) are as follows:

- Real-time archiving of stream data: importing stream data from IoT devices and the gateway to GaussDB(DWS) using HUAWEI CLOUD DIS
- Device monitoring and prediction: device monitoring, control, optimization, supply, self-diagnosis, and self-healing based on data analysis and prediction
- Information recommendation: information recommended to users based on the data collected by their networked devices.

1.3 GaussDB(DWS) Architecture and Advantages

GaussDB(DWS) clusters are distributed parallel database clusters based on the shared-nothing architecture.

Figure 1-6 GaussDB(DWS) architecture



This architecture renders GaussDB(DWS) the following key advantages:

GaussDB(DWS) can be used in multiple cloud forms, including HUAWEI CLOUD, HUAWEI CLOUD Stack, HCSO, and Intelligent EdgeSite (IES), satisfying differentiated deployment and O&M demands.

GaussDB(DWS) uses the Huawei-developed GaussDB database kernel and is compatible with PostgreSQL. The GaussDB database is transformed from a single OLTP database to an enterprise-grade, MPP-based, and distributed OLAP database oriented to massive data analysis.

DWS comes in four types: standard data warehouse, cloud data warehouse, stream data warehouse, and hybrid data warehouse, They provide processing capability of large volume of data across multiple industries and a unified management platform with the following advantages and features:

Ease of use

- Visualized one-stop management

  GaussDB(DWS) allows you to easily complete the entire process from project concept to production deployment. With the GaussDB(DWS) management console, you do not need to install data warehouse software or deploy data warehouse servers. GaussDB(DWS) offers you a high-performance and high-availability enterprise-grade data warehouse cluster in a couple of minutes.

With just a few clicks, you can easily connect applications to the data warehouse, back up data, restore data, and monitor data warehouse resources and performance.

- Heterogeneous database migration tools

  GaussDB(DWS) provides various migration tools to migrate SQL scripts of Oracle and Teradata to GaussDB(DWS).

High performance

- Cloud-based distributed architecture

  GaussDB(DWS) adopts the MPP architecture so that service data is separately stored on numerous nodes. Data analytics tasks are quickly executed in parallel on the nodes where data is stored.

- Response to query of trillions of data records within seconds

  GaussDB(DWS) improves data query performance by executing multi-thread operators in parallel, running commands in registers in parallel with the vectorized computing engine, and using the Low Level Virtual Machine (LLVM) compiler to reduce redundant judgment conditions.

  GaussDB(DWS) provides you with a better data compression ratio (column-store), higher index performance (column-store), and better point update and query (row-store) performance.

- Fast data loading

  GaussDB(DWS) provides you with GDS, a high-speed parallel bulk data loading tool.

High scalability

- On-demand scale-out: With the shared-nothing open architecture, nodes can be added at any time to enhance the data storage, query, and analysis capabilities of the system. Up to 2048 nodes can be deployed.
- Enhanced linear performance after scale-out: The capacity and performance increase linearly with the cluster scale. The linear rate is 0.8.
- Service continuity: During scale-out, data can be added, deleted, modified, and queried, and DDL operations (**DROP**/**TRUNCATE**/**ALTER TABLE**) can be performed. Online table-level scale-out ensures service continuity.

Robust reliability

- ACID

  Support for the atomicity, consistency, isolation, and durability (ACID) feature, which ensures strong data consistency for distributed transactions.

- Comprehensive HA design

  All software processes of GaussDB(DWS) are in active/standby mode. Logical components, such as the CNs and DNs of each cluster, also work in active/standby mode. This ensures data reliability and consistency as well as service continuity when any single point of failure (SPOF) occurs.

- Security

  GaussDB(DWS) supports transparent data encryption and can connect to DBSS to better protect user privacy and data security with network isolation and security group rule setting options. In addition, GaussDB(DWS) supports automatic full and incremental backup of data for higher reliability.

Convergent analysis

- Convergence of multiple modes: You can perform direct calculation and convergent analysis of stream, time series, GIS, full-text, and AI data on GaussDB(DWS).
- Convergence of multiple sources: You can use standard SQL statements to query data on the Hadoop Distributed File System (HDFS) and object storage service (OBS) without data migration.
- Cluster acceleration: The shared cluster Express is provided based on OBS data access for more efficient convergent computing and analysis capabilities.

Strong security

- Transparent encryption: Database data files are encrypted to prevent malicious attackers from bypassing the database permission control mechanism at the OS layer or stealing disks to access user data.
- Data masking: Built-in masking functions for digits, characters, and time types are provided. In addition, masking rules can be customized to effectively protect sensitive data while efficiently accessing big data.

2 Platforms and Technical Specifications Supported by GaussDB(DWS)

2.1 Technical Specifications

Software Specifications

Table 2-1 lists the technical specifications of GaussDB(DWS).

Table 2-1 Technical specifications

Technical Specifications	Maximum Value
Data capacity	20 PB
Number of cluster nodes	2048
Size of a single table	1 PB
Size of data in each row	1 GB
Size of a single column in each record	1 GB
Number of records in each table	2^{55}
Number of columns in each table	1,600
Number of indexes in each table	Unlimited
Number of columns in the index of each table	32
Number of constraints in each table	Unlimited

Technical Specifications	Maximum Value
Number of concurrent connections	80 for analytical long transactions, 500 for online short queries, and 5000 for short transactions
Number of partitions in a partitioned table	32,768
Size of each partition in a partitioned table	1 PB
Number of records in each partition in a partitioned table	2^{55}

Product Performance

Table 2-2 lists the performance specifications of GaussDB(DWS).

Table 2-2 GaussDB performance

Performance	Data import: 80 MB/s for row-store tables and 150 MB/s for column-store tables per node	Server configurations: CPU: 2 x 12-core Intel Xeon E5-2690 CPUs Memory: 256 GB Disk: 20 x 600 GB SAS disks Network: 10GE
	Data export: 100 MB/s per node	
	Full table scan: 300 million records per second	
	Point query: precise query of trillions of records in seconds	
	GROUP BY: 20 million records per second per node	
	JOIN: 5 million records per second per node	
	ORDER BY: 3 million records per second per node	

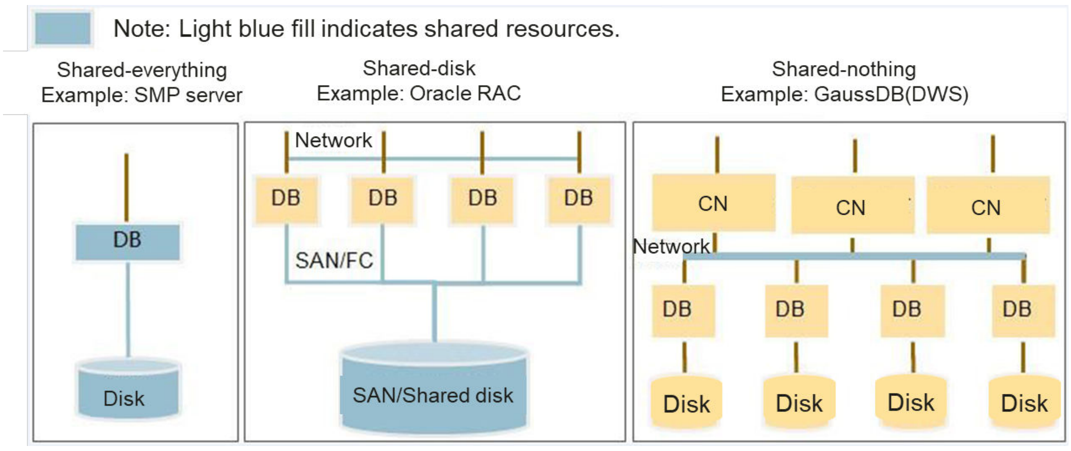
3 GaussDB(DWS) Core Technologies

3.1 Shared-Nothing Architecture

GaussDB(DWS) is an MPP system employing the shared-nothing architecture. It consists of multiple independent logical nodes that do not share system resources with each other, such as CPUs, memory, and storage. In such an architecture, service data is stored on multiple physical nodes. Data analysis tasks are executed in parallel on the nodes where data is stored. The massively parallel data processing significantly improves response speed.

Figure 4-1 compares the shared-nothing architecture and other architecture.

Figure 3-1 Architecture comparison



The shared-nothing architecture has the following advantages:

- High scalability
  - Provides on-demand scaling for BI and data analysis to process high-concurrency massive data.
  - Provides an automatic parallel processing mechanism.
- Implements automatic internal parallel processing without manual partitioning or optimization.

  - Allows you to load and access data as you would in a common database.
  - Distributes data on all parallel nodes.
  - Every node processes only partial data.
- Optimal I/O processing
  - All nodes process data in parallel.
  - Nodes share nothing with each other and have no I/O conflicts.
- Increased storage space and improved query and loading performance with added nodes

3.2 Data Distribution in a Distributed System

Background

DWS uses a two-layer data layout mechanism achieve high-performance query and import of PB-level data. At the first layer, users can specify a data distribution policy (hash distribution or replication distribution) when creating a table. When data is written to the system, the system determines the node where the data is stored based on the corresponding distribution policy. At the second layer, the node partitions its stored data according to partitioning rules.

3.2.1 Distributed Data Storage

GaussDB(DWS) horizontally partitions tables, distributing tuples in a table to multiple nodes. This allows you to filter out unnecessary data when querying to quickly locate data and significantly improve database performance.

Horizontal partitioning distributes data in a table to multiple nodes based on a specific data distribution policy.GaussDB(DWS) supports the data distribution policies described in **Table 3-1**. When executing the CREATE TABLE statement, you can configure the **DISTRIBUTE BY** parameter to enable data distribution on a specific table.

Table 3-1 Distribution policies

Policy	Description	Scenario	Advantage & Disadvantage
Replicatio n	Full data in a table is stored on each DN in the cluster.	Small tables and dimension tables	● The advantage of replication is that each DN has full data of the table. During the join operation, data does not need to be redistributed, reducing network overheads and reducing plan segments (each plan segment starts a corresponding thread). ● The disadvantage of replication is that each DN retains the complete data of the table, resulting in data redundancy. Generally, replication is only used for small dimension tables.
Hash	Table data is distributed on all DNs in the cluster.	Fact tables containing a large amount of data	● The I/O resources of each node can be used during data read/write, greatly improving the read/write speed of a table. ● Generally, a large table (containing over 1 million records) is defined as a hash table.

Policy	Description	Scenario	Advantage & Disadvantage
Polling (Round-robin) **Supported by 8.1.2 and later versions**	Each row in the table is sent to each DN in turn. Data can be evenly distributed on each DN.	Fact tables that contain a large amount of data and cannot find a proper distribution key in hash mode	● Round-robin can avoid data skew, improving the space utilization of the cluster. ● Round-robin does not support local DN optimization like a hash table does, and the query performance of Round-robin is usually lower than that of a hash table. ● If a proper distribution key can be found for a large table, use the hash distribution mode with better performance. Otherwise, define the table as a round-robin table.

3.2.2 Data Partitioning

Description

Most database products partition data. In the DWS distributed system, data partitioning is a process of horizontally partitioning data on a node based on a specified policy. A table is divided into multiple partitions based on a specified range, and data in different partitions does not overlap.

GaussDB(DWS) supports range partitioning and list partitioning. In range partitioning, records are divided and inserted into multiple partitions of a table. Each partition stores data of a specific range (ranges in different partitions do not overlap). If you configure the **PARTITION** parameter when running the **CREATE TABLE** statement, data in the table will be partitioned.

Benefits

Table 3-2 uses an xDR scenario to describe the benefits provided after data is partitioned based on time slices.

Table 3-2 Partitioning benefits

Scenarios	Benefits
The rows frequently accessed in a table are located in one or a few partitions.	Significantly reduces search space and improves access performance.
Most partition records need to be queried or updated.	Significantly improves performance because only one partition rather than the whole table needs to be scanned.
Records that need to be loaded or deleted in batches are located in one or a few partitions.	Improves processing performance because related partitions can be directly read or deleted. Reduces de-fragmentation workloads because records can be deleted in batches.

Data partitioning provides the following benefits:

- **Improves manageability:** Tables and indexes are divided into smaller and more manageable units, In this way, data management can be performed by partitions. Database administrators will perform maintenance in the designated area of the table.
- **Improves deletion performance:** You can delete an entire partition rather than delete data row by row.

  The syntax for deleting a partitioned table and a common table is the same: DROP TABLE.
- **Improves query performance:** You can restrict the volume of data to be checked or manipulated to make queries quicker.

  With partition pruning, also known as partition elimination, the CN filters out unexpected partitions and scans only the remaining partitions. Partition pruning greatly improves query performance.

  Intelligent partition connection: Partitioning can also improve the performance of multi-table joins by using a technique known as partition-wise joins. Partition-wise joins can be applied when two tables are joint and at least one of these tables is partitioned using a join key. Partition-wise joins break a large join into smaller joins of "identical" data sets. "Identical" here is defined as covering the same set of partitioning key values on both sides of the join, ensuring that only a join of these 'identical' data sets will produce a result and that other data sets do not have to be considered.

3.2.3 Parallel Data Import

Principles

Importing data in parallel on multiple nodes fully uses the computing and I/O capabilities of the nodes to maximize speed. The parallel data import function of GaussDB(DWS) implements high-speed and parallel import of external data in a specified format (CSV or TEXT).

Parallel data import is more efficient than the traditional data import method in which the INSERT statement is used to insert data. The procedure for importing data in parallel is as follows:

- The CN only plans and delivers data import tasks, and the DNs execute these tasks. This reduces CN resource usage, enabling the CN to process external requests.
- The computing capability and network bandwidth of all the DNs are fully utilized, improving data import performance.

The following uses the Hash distribution policy as an example to describe the GaussDB(DWS) data import process. Figure 3-2 shows the parallel data import process.

Figure 3-2 Parallel data import

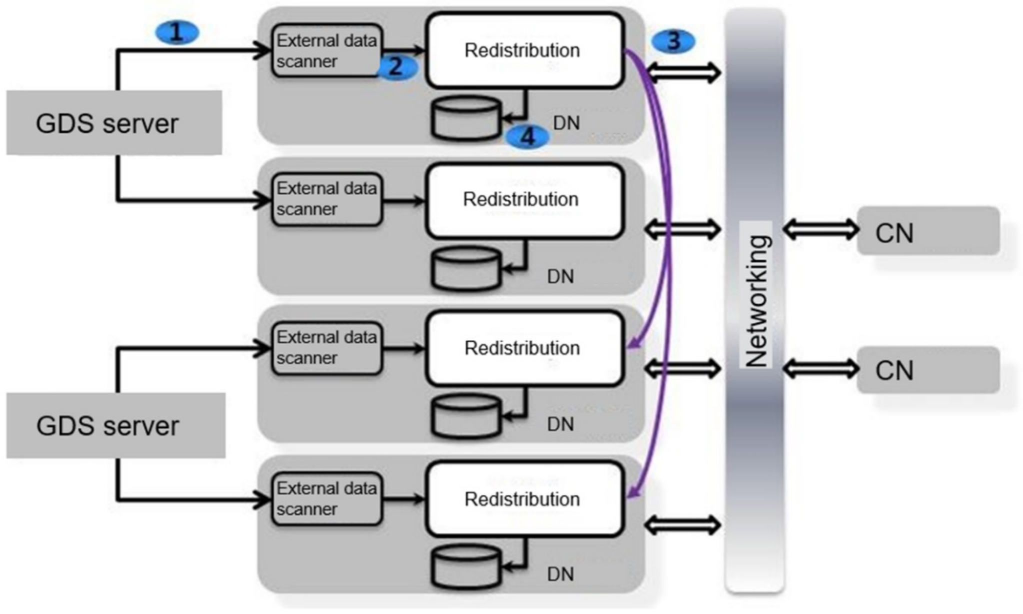


Table 3-3 Procedure description

Process	Description
Creating a table that complies with the Hash distribution policy	When running the **CREATE TABLE** statement, a service application presets the Hash distribution policy (specifies an attribute of a table as a distribution field).
Setting the partitioning policy	When executing the CREATE TABLE statement, a service application presets a partitioning rule (specifies an attribute of a table as a partitioning field). All Hash data in each DN is partitioned based on the preset partitioning rule.
1	During data import, GDS splits a specified data file into data blocks with a fixed size.

Process	Description
2	DNs download these data blocks from GDS in parallel.
3 4	Each DN processes data blocks in parallel and parses out a data tuple from the data blocks. The physical location of each tuple is determined based on the Hash value obtained based on the distribution column. ● If data is distributed on remote nodes based on the Hash values, you need to redistribute it to target DNs. ● If data is distributed on local nodes based on the Hash values, store it on local DNs.
Writing data into partitions	After data is sent to the node where Hash is used, it is written into the partition data file based on the partitioning logic. While data is written into a partitioned table in GaussDB(DWS), you can exchange partitions to improve the writing performance.
Gauss Data ServiceGeneral Data Service (GDS): Multiple GDSs can be deployed on a data server to improve the import performance.	

3.3 Fully Parallel Query

Description

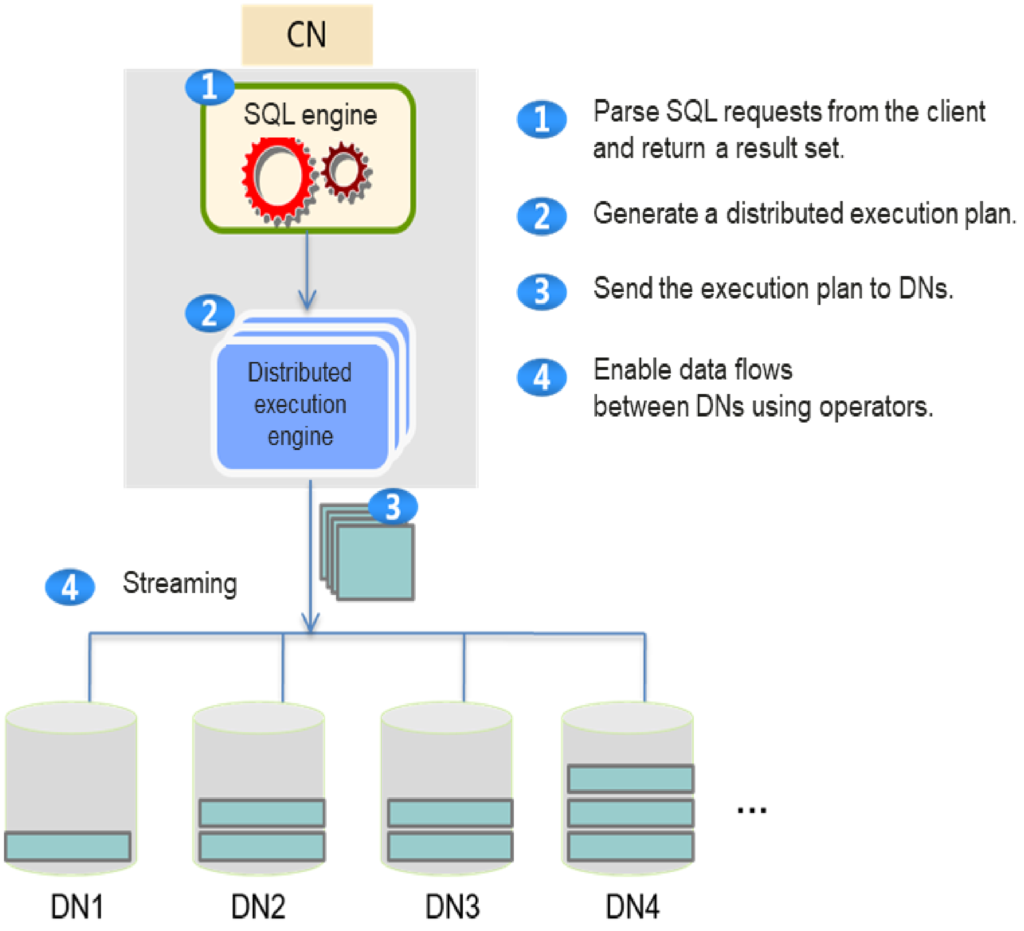
Fully parallel distributed query processing is the core technology of GaussDB(DWS). It minimizes data flow between nodes during query, thereby making query more efficient.

To efficiently analyze data, GaussDB(DWS) employs a set of high-performance distributed executors, which input the execution plan generated by the SQL engine, process tuples based on the execution plan, and return the result to the client.

Technical Principles

Figure 3-3 demonstrates the fully parallel distributed query technology of GaussDB(DWS).

Figure 3-3 Fully parallel distributed query processing technology


CN
1
SQL engine
2
Distributed execution engine
3
4
Streaming
DN1
DN2
DN3
DN4
...
1 Parse SQL requests from the client and return a result set.
2 Generate a distributed execution plan.
3 Send the execution plan to DNs.
4 Enable data flows between DNs using operators.

- The distributed executor running on the CN provides schedules and distributes executions.
- A new execution operator is introduced to support data flow between DNs. The new operator is called the data flow operator. Data flow can be classified into Gather, Broadcast, and Redistribution flows, based on the relationship between input and output. **Gather** combines multiple query fragments of data into one. **Broadcast** forwards the data of one query fragment to multiple query fragments. **Redistribution** reorganizes the data of multiple query fragments and then redistributes the reorganized data to multiple query fragments.
- Data transmission between DNs depends on the data stream topology constructed using the data distribution and the cost model during query and analysis. By the data stream topology, network connections are set up between DNs to drive data stream on the data flowing topology.

3.4 Vectorized Executor and Hybrid Row-Column Storage Engine

Background

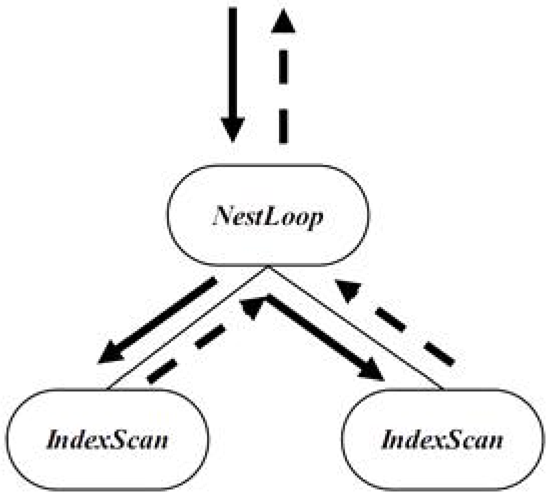
In a wide table containing a huge amount of data, a query usually only involves certain columns. In this case, the query performance of the row-store engine is poor. For example, a single table containing the data of a meteorological agency has 200 to 800 columns. Among these columns, only 10 are frequently accessed. This is where vectorized executor technology steps in, by improving performance and saving storage space.

Vectorized Execution

Figure 3-4 shows a standard vectorized executor. Control flow travels in the downlink direction (shown as solid lines in the following figure) and data flow in the uplink direction (shown as dotted lines in the following figure). The upper-layer node invokes the lower-layer node to ask for data and the lower-layer node only returns one tuple to the upper-layer node at a time.

Instead of returning only one tuple at a time in the traditional executor, the vectorized executor returns a batch of tuples at a time, which significantly improves executor performance with the aid of the column storage feature.

Figure 3-4 Vectorized executor



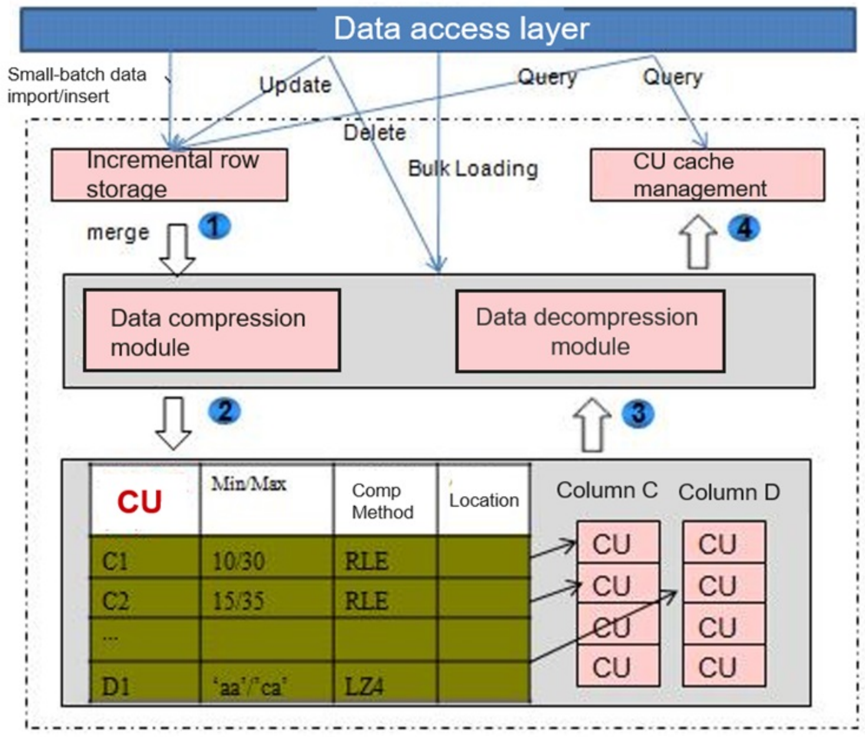
Hybrid Row-Column Storage Executor

GaussDB(DWS) supports both the row and column storage models. You can choose a row- or column-store table as needed.

Generally, if a table contains many columns (called a wide table) and its query involves only a few columns, column storage is recommended. If a table contains only a few columns and a query includes most of the fields, row storage is recommended.

The hybrid row-column storage engine achieves higher data compression ratio (column storage), index performance (column storage), and point update and point query (row storage) performance, as shown in **Figure 3-5**.

Figure 3-5 Hybrid row-column storage engine



Data compression is supported for column storage. You can compress old, inactive data to free up space, reducing procurement and O&M costs.

In GaussDB(DWS), data can be compressed using the Delta Value Encoding, Dictionary, RLE, LZ4, and ZLIB algorithms. The system automatically selects a compression algorithm based on data characteristics. The average compression ratio is 7:1. Compressed data can be directly accessed and is transparent to services, greatly reducing the preparation time before accessing historical data.

The restrictions of the column storage engine are as follows:

- Only the CREATE TABLE, DROP TABLE, and TRUNCATE TABLE operations can be performed for DDL.

  Partition management using DDL statements (such as ADD PARTITION, DROP PARTITION, MERGE PARTITION, and EXCHANGE) can be used.

  The **CREATE TABLE LIKE** statement is supported.

  The **ALTER TABLE** statement is partially supported.

  Other DDL statements are not supported.
- Among DML syntax, UPDATE, COPY, BULKLOAD, and DELETE are supported.
- A trigger and primary foreign key are not supported.
- Psort indexes, B-tree indexes, and GIN indexes are supported. For details about the constraints, see **CREATE INDEX** .

3.5 Resource Monitoring and Management

Background

To use resources appropriately, GaussDB(DWS) provides resource monitoring and management methods to allocate compute and storage resources that affect job running. This prevents performance deterioration or even system running problems caused by improper resource usage. Resource monitoring and management include resource monitoring, load management, and disk space management.

3.5.1 Load Management

Description

Load management balances system compute resources through service concurrency control to prevent resource contention between services, achieving harmonious coexistence of jobs and optimal resource utilization. In addition, the cgroup technology is introduced to manage CPU quotas.

Technical Principles

Load management is classified into static load management and dynamic load management, which are controlled by the **enable_dynamic_workload** parameter. In addition, GaussDB(DWS) provides priority control to control the priorities of tenant jobs.

- Static Load Management

  Each CN controls the memory and concurrency separately. The actual concurrency of a cluster is the sum of the concurrency of all CNs. The actual memory used by the cluster is the sum of the memory occupied by the jobs running on each CN. Because the memory of each CN is controlled separately, the actual memory used by a tenant on the DN may exceed the memory quota of the tenant.
- Dynamic Load Management

  The CCN is added to control the concurrency and memory of complex jobs. Each CN requests queuing information from the CCN. The CCN accumulates the estimated memory used by tenants to obtain the used memory of each tenant. When the available memory of a tenant is insufficient, the CCN triggers queuing. After the job running is complete, the CCN attempts to wake up the jobs in the queue. The sum of used memory of all tenants is the used memory of the cluster. In addition, the CCN periodically collects the memory usage information on DNs to update the available memory of the cluster. If the available memory is insufficient, jobs are queued. The job running must meet the memory limits of both the tenant and cluster.
- Priority Control

  GaussDB(DWS) implements load identification and intra-queue priority control based on query_band. It provides more flexible load identification methods and identifies load queues based on job types, application names, and script names. Users can flexibly configure query_band identification queues based on service scenarios. In addition, priority control of job delivery in the queue is implemented. In the future, priority control of resources in the queue will be gradually implemented.

Benefits

Load management implements resource isolation and priority control between tenants. It ensures that jobs of tenants with high priorities run preferentially without affecting resources of tenants with low priorities, achieving effective and optimal resource utilization.

For more information, see **Resource Management**.

3.5.2 Space Control

Description

User space management includes permanent tablespace management, temporary tablespace management, and intermediate calculation result set flushing space management. When creating group users and service users, the database administrator can specify the permanent tablespace, temporary tablespace, and intermediate calculation result set flushing space limit. When a user's operation involves the increase or decrease of the managed space, the system checks whether the space exceeds the specified limit. If the space exceeds the limit, the user is not allowed to perform the operation and the available disk space is limited to the specified space limit.

In addition, the product supports schema-based permanent space control. When the space of objects in a schema are added or deleted, the operations that exceed the limit are controlled.

Technical Principles

When a service user executes a service, the size of the data space operated by the service user is recorded in the space quota of the service user. For a permanent tablespace, the quota is evenly allocated to each DN. DNs collect space statistics in real time when data files, temporary table files, and temporary files are added or expanded. When a service user performs a service operation, DNs check whether the space limit is reached and determine whether to perform the operation. If the space limit is exceeded, DNs roll back the operation. When a transaction or session ends, the space is released and the corresponding value is updated.

DNs collect statistics on disk changes involved in user operations in real time, save the statistics to the memory, wait for the CN to collect the statistics, and receive the information delivered by the CN to update the temporary space used by the entire system.

Benefits

By limiting the maximum storage space used by a user, the system prevents user services from squeezing or occupying too many storage resources to ensure that normal service data can be written, ensuring system stability and data availability.

For more information, see **Space Control**

3.6 Distributed Transactions

Background

In a distributed share-nothing architecture, table data is distributed on different nodes. One or more statements on the client may modify data on multiple nodes at the same time. In this case, a distributed transaction is generated. Note the following points using distributed transactions:

1) Atomicity of transactions on each node: Distributed transactions are either all successful or all failed on all nodes.

2) Transaction consistency: The data returned on each node is the same. Data consistency cannot be ensured when a node is faulty.

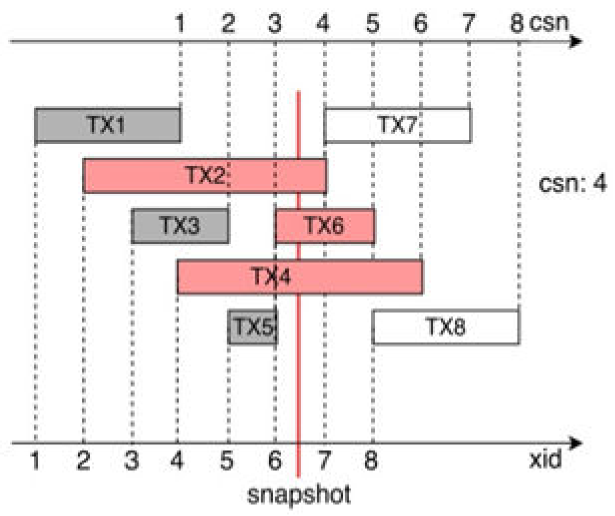
The atomicity of distributed transactions must be ensured. The consistency of distributed transactions depends on the CAP theory. The common standards are CP systems that support strong consistency 2pc and 3pc protocols or AP systems that support eventual consistency TCC and message tables.

Technical Principles

GaussDB(DWS) supports strongly consistent distributed transactions and provides high-availability with consistency and partition tolerance (CP).

CSN

CSN update and mapping between CSNs and XIDs



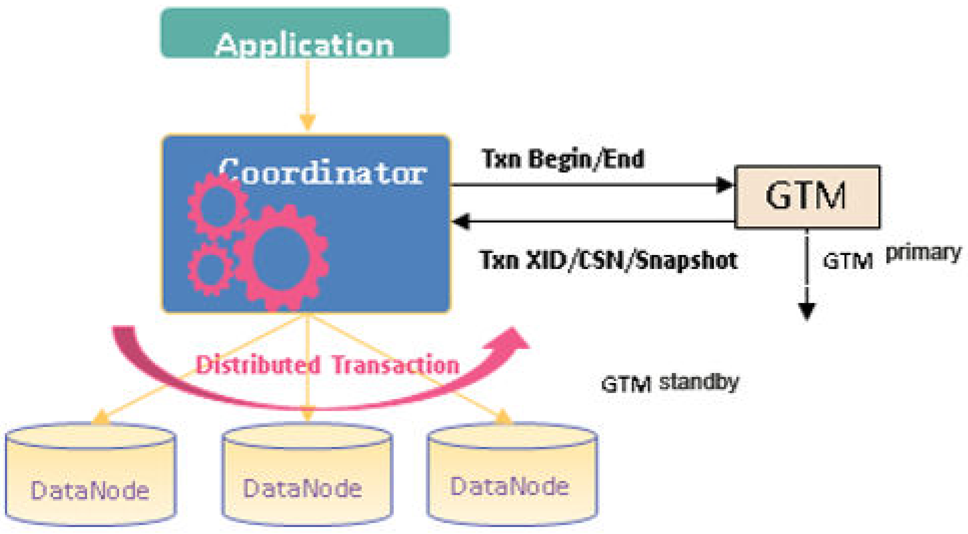
CSN – Commit Sequence Number: transaction submission number

The CSN is an 8-byte unsigned integer that increases monotonically and is maintained by the GTM.

2) When a transaction ends, the CSN value is updated from the GTM.

3) After the CSN mechanism is used, the CSN can be obtained from the GTM.

GTM



GTM is a component in the GaussDB(DWS) distributed framework. It has the following functions:

1. Manage and allocate transaction IDs (increases but does not decrease).

2. Manage and maintain CSN numbers (increase but not decrease)

When executing a modification operation, the CN obtains the transaction ID from the GTM.

At the beginning of the statement, the CN obtains a CSN from the GTM for query.

When a transaction starts or ends, the CN communicates with the GTM to register and destroy transaction information.

Troubleshooting

The gs_clean tool is used to automatically clear residual distributed transactions caused by node faults.

gs_clean queries the residual two-phase transactions on each node, checks whether the transactions are submitted or rolled back on other nodes based on the residual transaction IDs, and clears the residual two-phase transactions based on the final result.

Benefits

GaussDB(DWS) supports strongly consistent distributed transactions. You can use the GaussDB(DWS) database in the same way as using a standalone database.

After the CSN-based transaction mechanism is used, the concurrency performance is greatly improved.

3.7 Online Scale-Out

Background

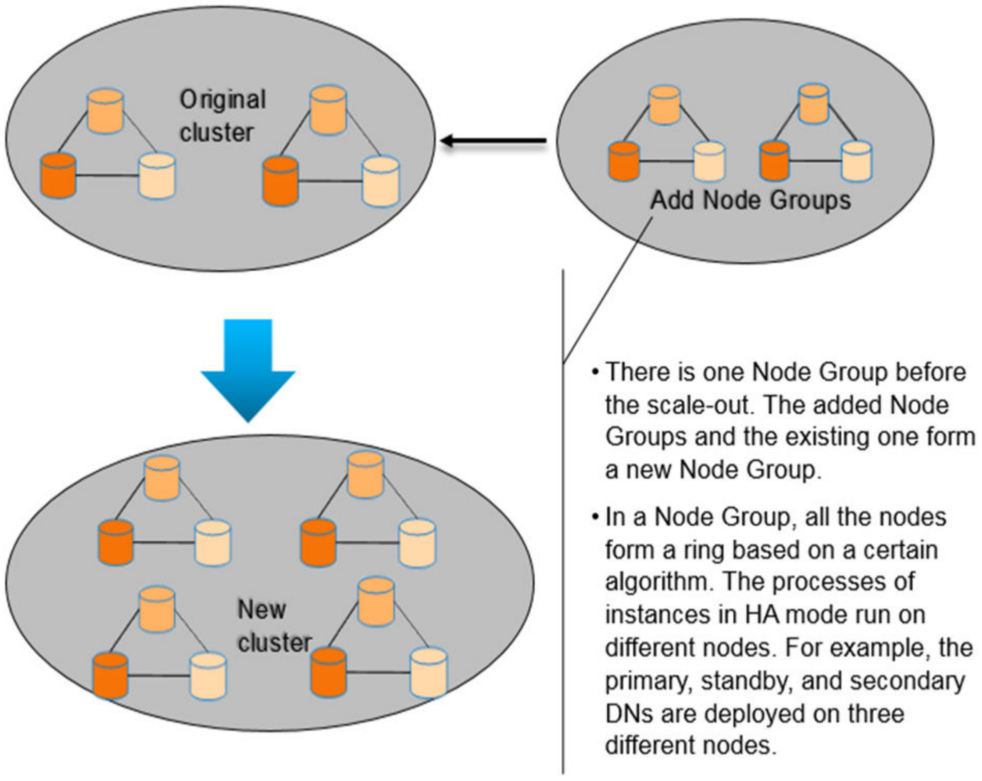
As your business expands, your current system may be unable to provide sufficient disk capacity and satisfactory performance. The distributed GaussDB(DWS) cluster allows you to fully utilize existing machines to scale the existing capacity out, meeting your business needs.

Technical Principles

The node group technology of GaussDB(DWS) supports the scale-out of multiple tables in parallel with a speed up to 400 GB per hour on each new node.

Figure **Figure 3-6** shows the GaussDB(DWS) scale-out process.

Figure 3-6 Scale-out


Original cluster
Add Node Groups
New cluster
• There is one Node Group before the scale-out. The added Node Groups and the existing one form a new Node Group.
• In a Node Group, all the nodes form a ring based on a certain algorithm. The processes of instances in HA mode run on different nodes. For example, the primary, standby, and secondary DNs are deployed on three different nodes.

Benefits

Online scale-out of GaussDB(DWS) has the following advantages:

- Scale-out does not interrupt services.

  Data import and query in GaussDB(DWS) are not interrupted during scale-out.

- Consistent hashing and multi-table parallel scale-out technologies improve query performance.

  Hash technology is used to minimize the amount of data migrating during redistribution.

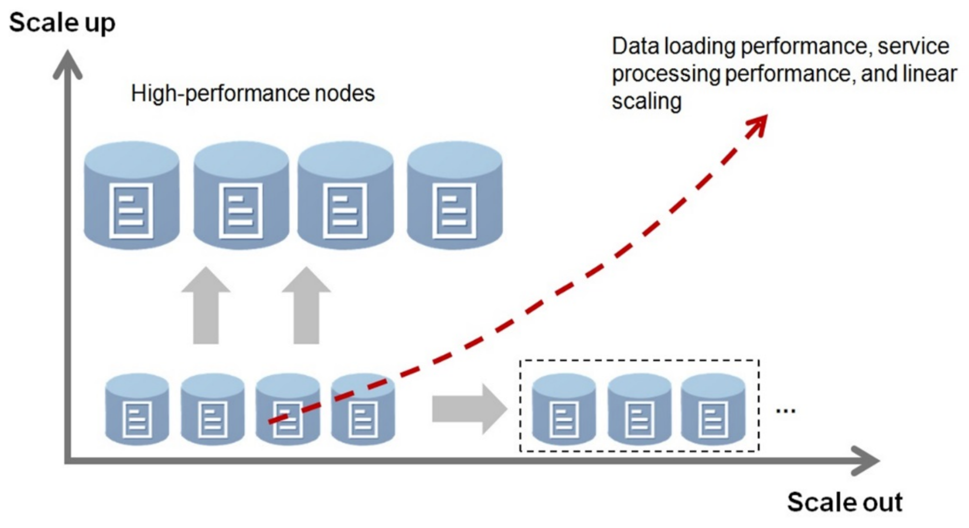
  Tables can be redistributed in parallel or in user-specified order.

  Users can query the scale-out progress in real time.

- As the number of nodes increases, the cluster performance improves linearly.

  As shown in **Figure 3-7**, in the full parallel distributed architecture, the data loading performance, service processing performance, and capacity of a GaussDB(DWS) cluster linearly improves with the number of nodes.

Figure 3-7 High-performance scalability


Scale up
High-performance nodes
Data loading performance, service processing performance, and linear scaling
...
Scale out

3.8 SQL on Anywhere

Background

Various types of engine components provide many interfaces for big data processing. However, for traditional database users, the SQL language is still the most familiar and convenient interface. Their operations will be efficient using SQL statements on a client to perform operations on big data components.

GaussDB(DWS) supports SQL on Anywhere, allowing users in a GaussDB(DWS) database to perform operations on the following systems to build a unified big data computing platform: Hadoop, Oracle, Spark, and other GaussDB(DWS) databases.

3.8.1 SQL on Hadoop

In DWS, you can directly read the structured data stored in HDFS, use the SQL query interfaces it provides, and perform complex analysis and query about the Hadoop native data using the vectorized executor.

Technical Architecture

DWS maps structured HDFS data to its foreign table. You can use the well-developed SQL compiler, vectorized executor, and SQL interfaces provided by the database to analyze and query native Hadoop data in HDFS.

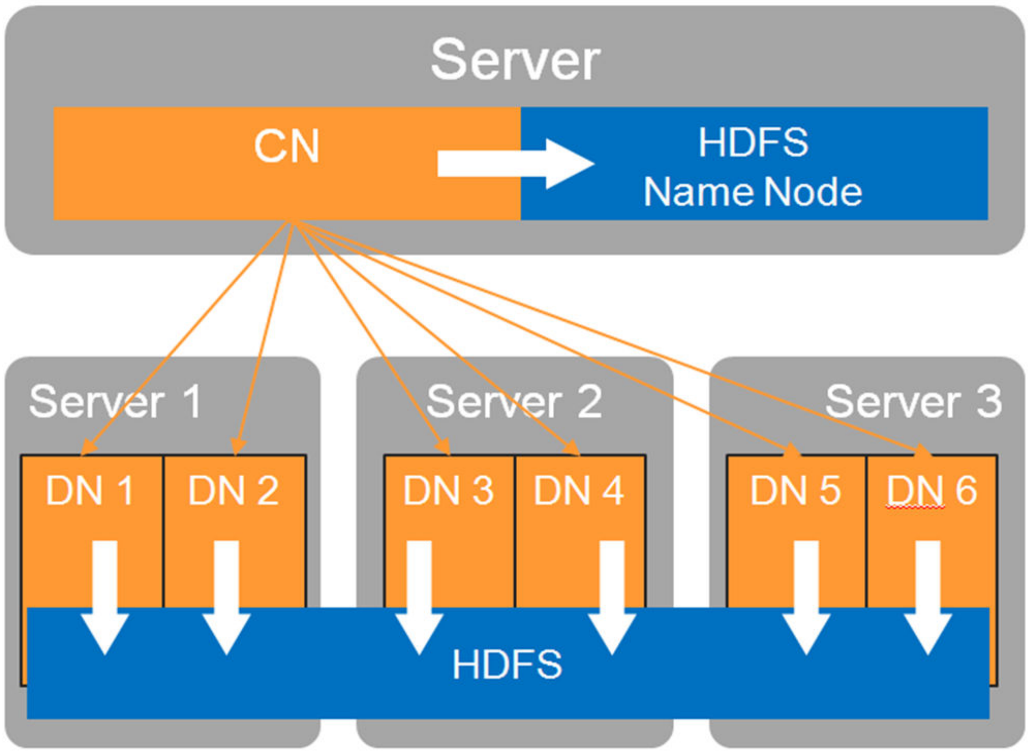
DNs and HDFS data nodes can be deployed in the same physical cluster so that the CN can directly read data from the local HDFS data node, reducing the network overhead caused by remote HDFS read. DNs are also allowed to remotely access other HDFS data nodes.

The SQL compiler of DWS introduces a new component, Scheduler. If access to HDFS is required during SQL compiling, the Scheduler accesses the metadata of the HDFS NameNode to obtain the storage path of external files in HDFS. When the SQL engine of the CN is compiling query statements, it will invoke the Scheduler to obtain and specify the directory that contains the HDFS file to be accessed by each DN.

The allocation principles of the HDFS external data files (between DNs) are as follows:

- A DN gives priority to data stored on its own physical server node when reading data.
- Then, workload balancing between DNs is considered.

Figure 3-8 Logical architecture



Technical Highlights

- **Standard SQL query syntax**

The SQL query syntax inherited from DWS supports structured data analysis and query in HDFS, supports system functions such as connection, aggregation, and character date, supports subqueries and joint access to HDFS structured data and DWSS local data, and supports window functions.

- **HDFS data cost estimation model**

  In addition to the cost-based SQL optimizer of DWS, the product uses the cost evaluation model for structured data access in the HDFS to formulate the best execution plan of HDFS data.

- **Intelligent scanning**

  DWS can directly push predicates to the native Hadoop data stored on HDFS, filter predicates in the compressed data, and perform late materialization for the Hadoop Optimized Row Columnar (ORC) storage, reducing the amount of data to be read in the HDFS.

- **Low Level Virtual Machine (LLVM) optimization**

  Predicates in native Hadoop data are scanned and optimized by LLVM. Then, intermediate representations (IRs) for their conditions are generated and further converted into machine codes. In this way, the performance of filtering and querying predicates is improved.

- **Informational constraints**

  If the columns in a table are unique, you can specify information constraints for certain columns when you create this table in the database, facilitating queries during execution.

- **Vectorized executor**

  DWS interconnects with its mature vectorized executor and improves analysis and query performance for structured ORC column data in the HDFS.

- **Partitioned tables**

  The product is adapted to the partitioned-table data defined by the Hive syntax in the HDFS. It automatically prunes partitioned tables using the DWS SQL optimizer, improving analysis and query performance.

- **Highly efficient distributed reads of HDFS data**

  The DWS SQL compiler uses the Scheduler to balance loads among DWS DNs that access HDFS data and uses the HDFS feature of short-circuit local read to improve data read performance.

  For more information, see **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)**.

3.8.2 GDS-based Cross-Cluster Interconnection

Background

In medium- and large-sized enterprises, datawarehouses are layered. Multiple GaussDB(DWS) clusters are deployed and data is synchronized across clusters. Data synchronization between clusters should support a large amount of data and parallel execution. How to provide efficient cross-cluster data synchronization has become one of the key issues in the data field.

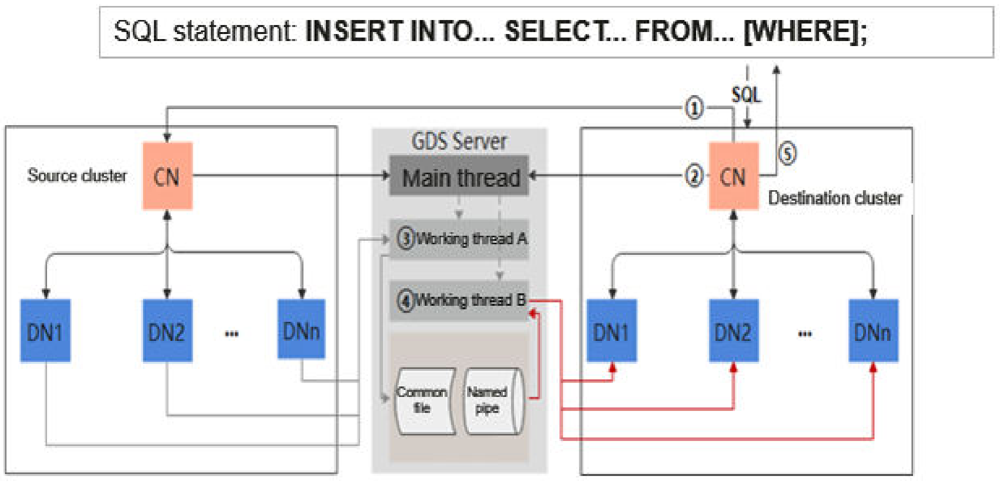
Function

Full data migration between GaussDB(DWS) clusters

Partial data migration based on filter conditions between GaussDB(DWS) clusters

Technical Principles

An SQL statement that triggers synchronization is converted into a pair of GDS import and export jobs through query rewriting. The jobs are executed in the source and destination clusters, forming an efficient and real-time data transfer channel for data migration and synchronization. You can execute synchronization on the destination cluster to pull data from the source cluster or, reversely, execute synchronization on the source cluster to push data to the destination cluster.



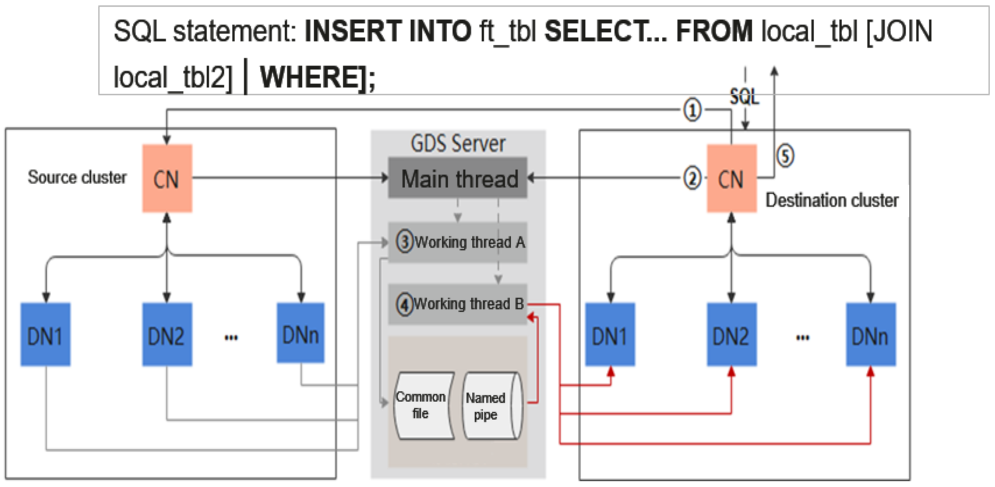
1. Remotely connect to the source cluster, create a GDS write-only foreign table, and initiate an export job.

2. Create a GDS read-only foreign table and initiate an import job.

3. Worker thread A receives data from the source cluster and writes the data to a local file.

4. Worker thread B reads local file data and sends the data to the destination cluster.

5. The destination cluster obtains the final result based on the job results at both ends and returns the final result to the user.



1. Remotely connect to the target cluster, create a GDS read-only foreign table, and initiate an import job.

2. Create a GDS write-only foreign table and initiate the export job.

3. Worker thread A receives data from the source cluster and writes the data to the named pipe.

4. Worker thread B reads the named pipe data and sends it to the destination cluster.

5. The source cluster integrates its export job result and the import job result of the destination cluster, then returns the final result to the user.

Benefits

One SQL statement is used to start the migration service. With GDS, the computing power of the nodes in the clusters at both ends is fully utilized to provide convenient and efficient logical data synchronization or migration between GaussDB(DWS) clusters without occupying disk space, improving system resource utilization.

For more information, see **GDS-based Cross-Cluster Interconnection**.

3.9 Cluster Management and HA

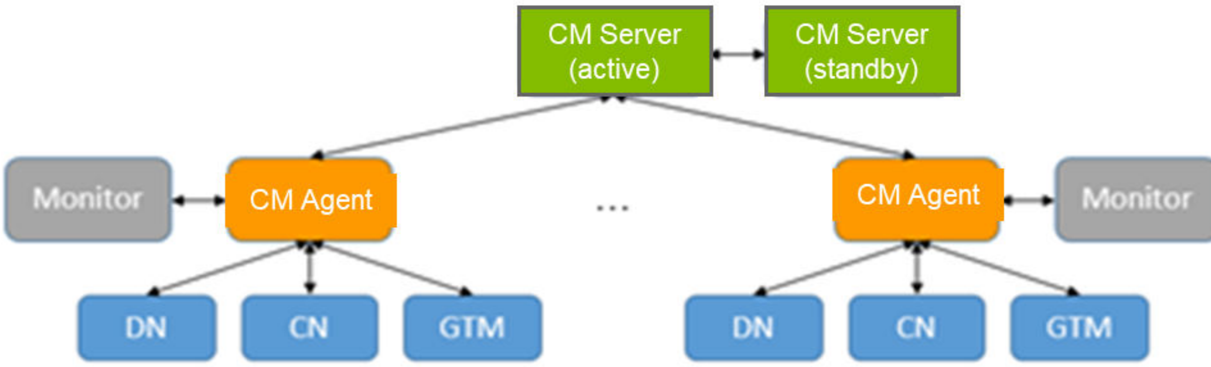
Description

GaussDB(DWS) provides the cluster manager (CM) module to manage and monitor the running status of each functional unit and physical resource in the distributed system, ensuring stable running of the entire system. CMs are classified into primary and standby CMs. In normal cases, only the primary CM provides the GaussDB(DWS) cluster management service. If the primary CM is faulty, the standby CM will be promoted to the primary to provide cluster management.

Technical Principles

The cluster management module consists of the CMServer, CMAgent, and Monitor components, and provides tools for querying cluster status, starting and stopping a cluster, performing primary/standby switchover, and rebuilding instances. CMServer is deployed only on the primary and standby CMs. As the brain of the entire GaussDB(DWS) cluster, CMServer processes various status information reported by CMAgent and determines whether to change the status. Deployed on all nodes, CMAgent functions as an instance agent process, reports the status of CNs, DNs, GTMs, and other instances to CMServer, and receives and executes commands delivered by CMServer. Monitor is deployed on all nodes as a scheduled task that restarts CMAgent when it is stopped.

Figure 3-9 Architecture of cluster management modules

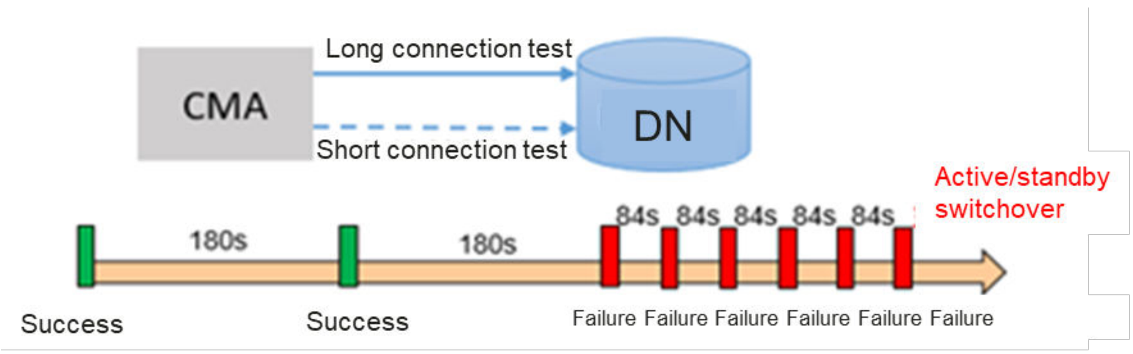


Hang Detection

The cluster management module uses the short connection mechanism to check whether the instance process is in an abnormal state such as network fault, disk I/O suspension, or process/thread suspension. If necessary, the cluster management module triggers the DN/GTM primary/standby switchover or CN removal process.

Take DNs as an example. By default, CMAgent creates instance connections every 180 seconds. If the connection fails, CMAgent retries every 84 seconds. If the connection fails for five consecutive times, the primary/standby DN switchover process is triggered. The complete hang detection period is about 600 seconds.

Figure 3-10 Instance hang detection



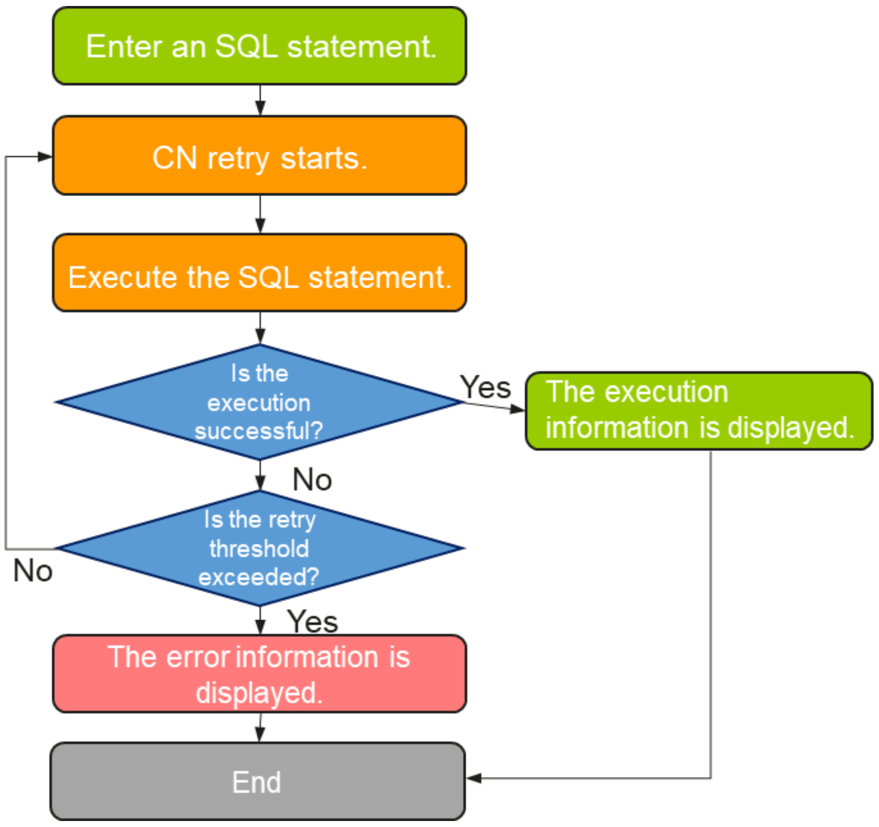
CN Retry

GaussDB(DWS) provides the CN Retry function to automatically retry SQL statements when an exception occurs, improving service continuity.

The CN triggers the retry mechanism when an error is reported during statement execution. For retryable errors, roll back the executed operation and execute the statement again. If the operation still fails, the error information is reported to the client.

Users are unaware of the retry process of SQL statements.

Figure 3-11 CN retry workflow


Enter an SQL statement.
CN retry starts.
Execute the SQL statement.
Is the execution successful?
Yes
The execution information is displayed.
No
Is the retry threshold exceeded?
No
Yes
The error information is displayed.
End

3.10 SQL Self-Diagnosis

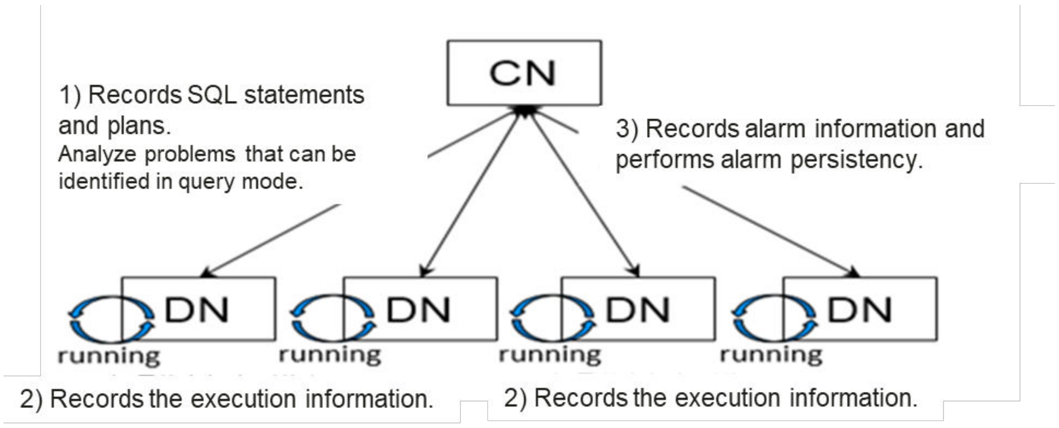
Background

A large number of queries are involved in services. Query problems at the planning and execution phases may appear, for example, whether the estimation is accurate, whether data skew occurs, and whether statistics are not collected and how to collect statistics. SQL self-diagnosis provides users with a more efficient and easy-to-use method for locating performance problems. It helps users simplify the SQL optimization process of batch processing jobs. After entering SQL statements, users can easily obtain SQL problems and optimization suggestions in batches, instead of executing complex optimization such as extracting and rewriting SQL statements in the job set one by one, reproducing the SQL statements that have performance problems, checking Explain Performance to locate the problem, rewriting the SQL statements, and adjusting parameters.

Function Description

Before executing a job, configure GUC parameters. For details, see "resource_track_level" and "resource_track_cost" in *HUAWEI CLOUD Stack 8.0.2 GaussDB(DWS) Developer Guide*. After running a user job, you can view the related system view to obtain the possible performance problems of the query job. These system views provide possible causes of performance problems. Based on the performance alarms, you can optimize the jobs with performance problems by referring to "SQL Self-Diagnosis" in *HUAWEI CLOUD Stack 8.0.2 GaussDB(DWS) Developer Guide*.

Technical Principles


CN
1) Records SQL statements and plans.
Analyze problems that can be identified in query mode.
3) Records alarm information and performs alarm persistency.
DN
DN
DN
DN
running
running
running
running
2) Records the execution information.
2) Records the execution information.

[1] The CN queries and compiles SQL statements to generate a plan tree. The diagnosis analyzer diagnoses the plan tree to identify problems in query mode (including alarms that statistics are not collected and that SQL statements cannot be delivered).

[2] The DN executes the SQL statement and writes the statistics to the shared memory of the DN. If the current SQL is running in operator mode (all supported alarm scenarios can be diagnosed), collect operator execution statistics during SQL running.

[3] The DN executes the SQL statement and returns the execution result to the CN. If the operator mode is used, the DN returns the collected runtime status to the CN. The diagnosis analyzer analyzes the status and records the analysis result.

[4] Statistics in the memory are cleared every 3 minutes. To record all historical information, enable the GUC parameter **enable_resource_record**. The statistics in the memory is persisted to a specific system catalog every 3 minutes.

Benefits

Scenario Description	Benefits
Some column statistics are not collected.	Use ANALYZE to collect statistics and generate a better execution plan.

Scenario Description	Benefits
SQL statements are not pushed down.	Report the cause of the pushdown failure. Optimize SQL statements to push them down.
In a hash join, the larger table is used as the inner table.	A large volume of data is written to disks, which greatly affects the system performance. This problem can be avoided by optimizing the SQL statements.
Nestloop is used in a large-table equivalent join.	Use of Nestloop has a great impact on performance when the data volume is large. This problem can be avoided by optimizing the SQL statements.
A large table is broadcasted.	A large amount of data is transmitted on the network, which greatly affects the system performance. This problem can be avoided by optimizing the SQL statements.
Data skew	Some nodes become the system bottleneck.
Improper index	The index is improper. As a result, a large number of rows need to be scanned. In this case, modify indexes.
Inaccurate estimation	The estimated number of rows deviates greatly from the actual number. As a result, the selected plan is not optimal.

3.11 Transparent Data Encryption

Description

Transparent Data Encryption (TDE) encrypts GaussDB(DWS) data files. Generally, threat mitigation measures are taken to protect data security. For example, design a secure system, encrypt confidential assets, or build a firewall around database servers. However, in a scenario where the physical media (for example, disks) are stolen by attackers or internal personnel, the malicious party can just restore or attach the database and browse the data. One solution is to encrypt the sensitive data in the database and protect the keys that are used to encrypt the data. This prevents anyone without the keys from using the data, but this kind of protection must be planned in advance. The GaussDB(DWS) provides a complete solution.

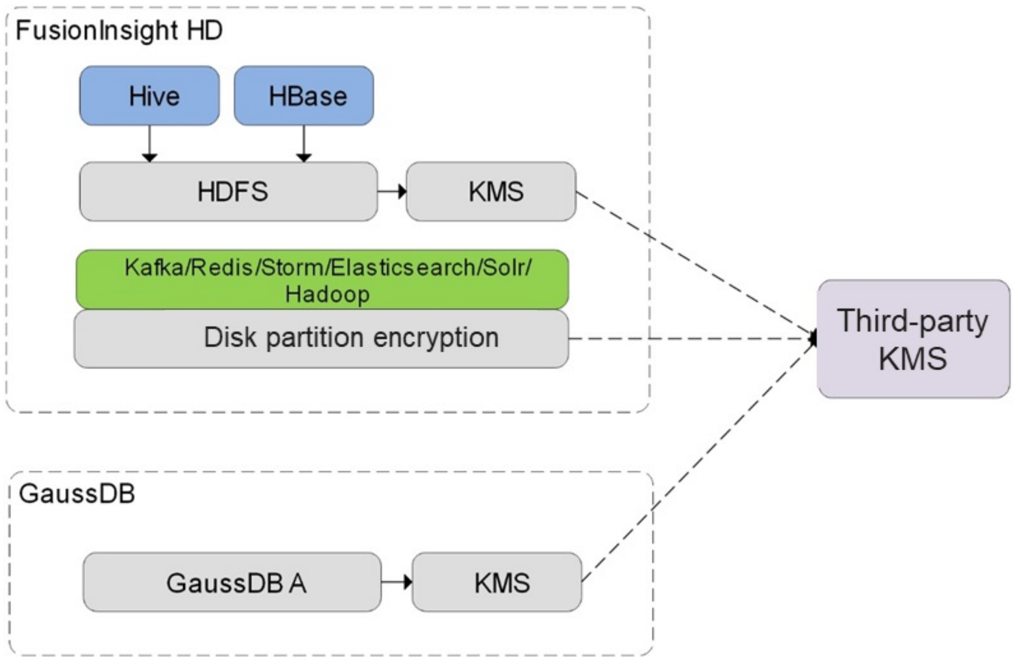
TDE performs real-time I/O encryption and decryption of the data. Users are unaware of the encryption. The encryption uses a database encryption key (DEK), which is not stored in the cluster. The DEK is a symmetric key secured by using the cluster encryption key (CEK) stored in the KMS server. Database servers store only DEK ciphertext. During database startup, each CN/DN uses identity authentication (for example, Kerberos) to connect to the KMS server, decrypts the DEK ciphertext to obtain the key plaintext, and caches it in the memory. Once the host is powered off or the cluster is shut down, keys are deleted. Therefore, do not lose key files in the cluster because they are irrecoverable.

Use Cases

In a traditional database cluster, user data is stored in plaintext in column-store or row-store files. Cluster maintenance personnel or malicious attackers can bypass the database permission control mechanism in the OS or steal disks to access user data. GaussDB(DWS) adopts and enhances the Hadoop KMS. The connection to third-party KMS helps GaussDB(DWS) achieve transparent data encryption for data security.

GaussDB(DWS) obtains keys from the third-party KMS through the Hadoop KMS.

Figure 3-12 Data storage encryption


FusionInsight HD
Hive
HBase
HDFS
KMS
Kafka/Redis/Storm/Elasticsearch/Solr/ Hadoop
Disk partition encryption
Third-party KMS
GaussDB
GaussDB A
KMS

GaussDB(DWS) Transparent Encryption

In GaussDB(DWS) database-level transparent encryption, each GaussDB(DWS) cluster has a CEK, and each database is configured with a DEK. DEKs are encrypted using the CEK and their ciphertext is stored in GaussDB(DWS) clusters. Keys are applied for, encrypted, and decrypted through the KMS service. The encryption algorithm is configured using configuration items. Currently, the AES and SM4 algorithms are supported. The SM4 algorithm supports hardware acceleration in chips of Hi 1620 or later.

Currently, database-level transparent encryption is supported. You need to configure encryption when creating a cluster.

For details about how to configure transparent encryption, see **Encrypting GaussDB(DWS) Databases**.

3.12 Data Masking

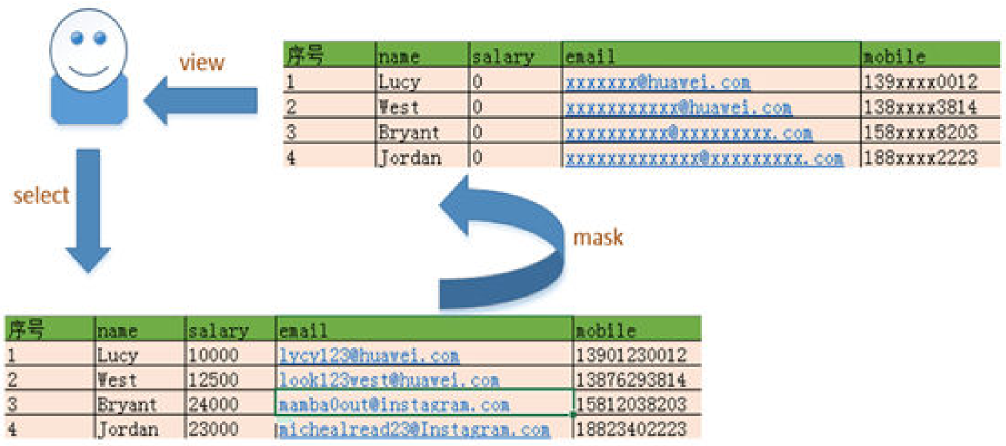
Background

In the big data era, the huge value of data also brings difficulties in privacy protection. Data masking ensures efficient big data sharing and protects sensitive information.

Description

GaussDB(DWS) allows users to create data masking policies by column. Users can create policies for sensitive data in services. Sensitive data is identified by users based on their service scenarios. After a data masking policy is configured, only the administrator and table object owner can access raw data. In addition, the data can be used for actual calculation and is masked only when the database service returns the final result.

Figure 3-13 Data masking effect



序号	name	salary	email	mobile
1	Lucy	0	xxxxxxx@huawei.com	139xxxx0012
2	West	0	xxxxxxxxxxx@huawei.com	138xxxx3814
3	Bryant	0	xxxxxxxxxx@xxxxxxxxx.com	158xxxx8203
4	Jordan	0	xxxxxxxxxxxxx@xxxxxxxxx.com	188xxxx2223

序号	name	salary	email	mobile
1	Lucy	10000	lycy123@huawei.com	13901230012
2	West	12500	look123west@huawei.com	13876293814
3	Bryant	24000	mamba0out@instagram.com	15812038203
4	Jordan	23000	michealread23@Instagram.com	18823402223

Technical Principles

Figure 3-14 Technical principles

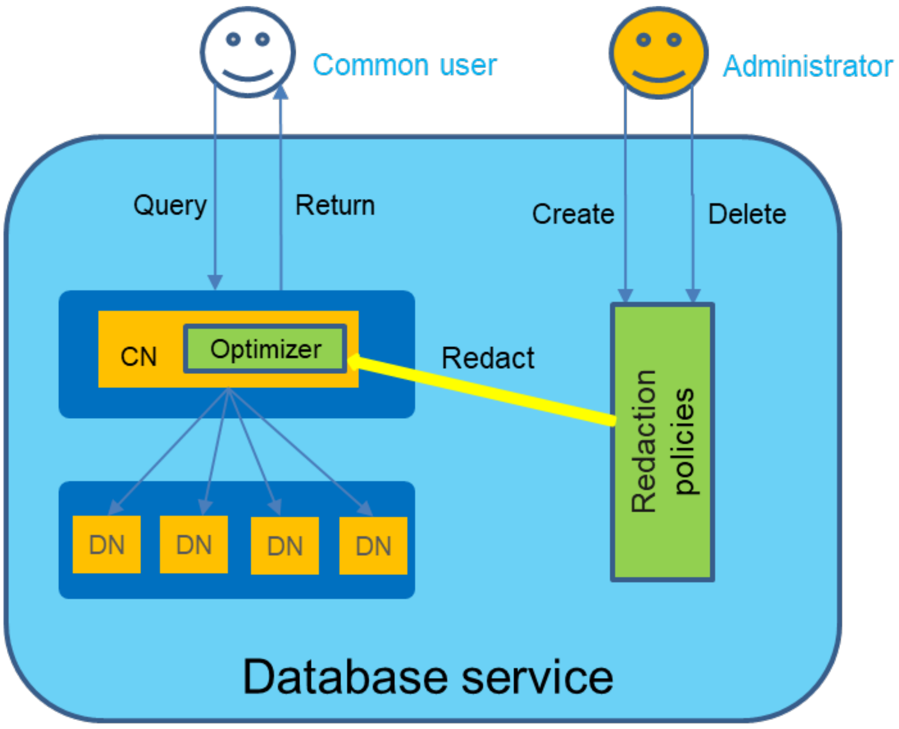


Table 3-4 Interface functions

External Interface	Function
add_policy	Creating a masking policy
alter_policy	Modifying a data masking policy
drop_policy	Deleting a data masking policy
enable_policy	Enabling a data masking policy
disable_policy	Disabling a data masking policy

After a data masking policy is created for a column in a table, all queries involving the column are affected by the policy. Only the administrator and table owner can query and return the original values of the column.

Benefits

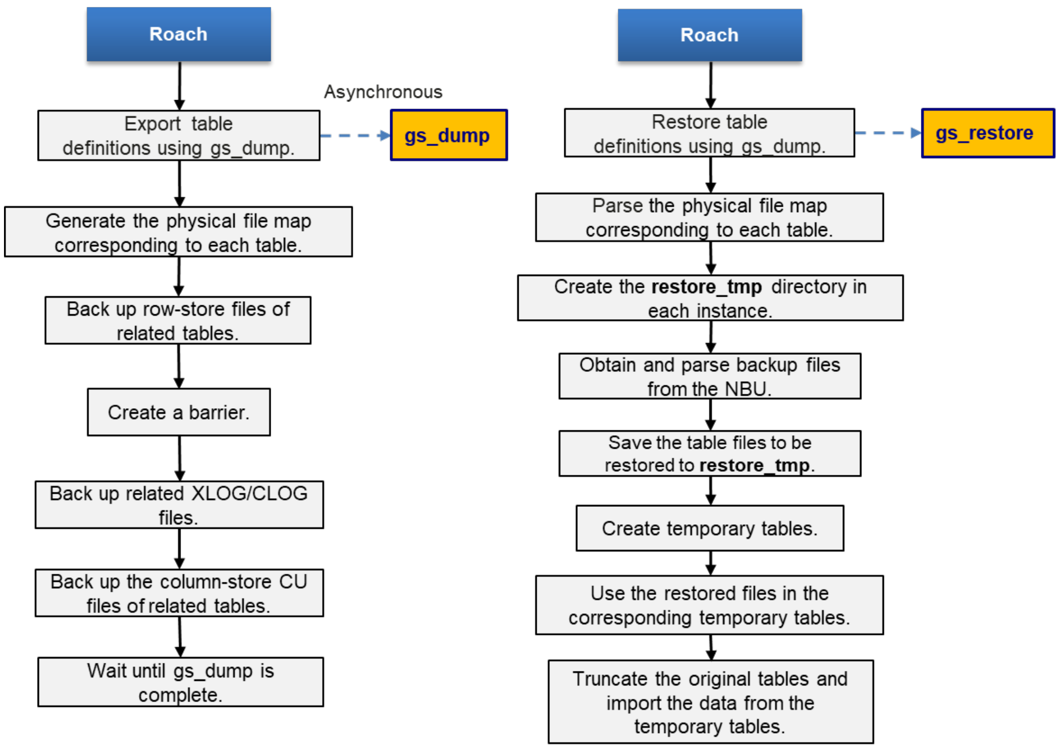
Industries involving sensitive information have great requirements for data masking, such as finance, government, and healthcare. Data masking can be used to prevent sensitive information leakage in application development, testing, and training scenarios.

For details, see **Data Redaction**

3.13 Data Backup and Disaster Recovery

3.13.1 Backup and Restoration

The following figure shows the backup and restoration process of physical fine-grained backup and restoration:


Roach
Asynchronous
Export table definitions using gs_dump.
gs_dump
Generate the physical file map corresponding to each table.
Back up row-store files of related tables.
Create a barrier.
Back up related XLOG/CLOG files.
Back up the column-store CU files of related tables.
Wait until gs_dump is complete.
Roach
Restore table definitions using gs_dump.
gs_restore
Parse the physical file map corresponding to each table.
Create the restore_tmp directory in each instance.
Obtain and parse backup files from the NBU.
Save the table files to be restored to restore_tmp.
Create temporary tables.
Use the restored files in the corresponding temporary tables.
Truncate the original tables and import the data from the temporary tables.

Snapshot

A snapshot is a complete backup that records point-in-time configuration data and service data of a GaussDB(DWS) cluster. A snapshot can be used to restore a cluster. Snapshots are stored on OBS.

NOTE

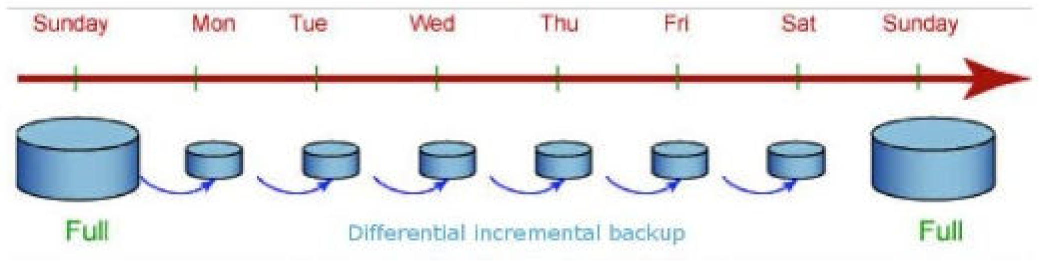
- GaussDB(DWS) provides some free-of-charge storage space for storing snapshot data. However, when you use more storage space than the free-of-charge storage space, the excess space is billed on a pay-per-use basis.
- The free-of-charge space is the same as the size of the total storage space of a cluster (storage space of a single node x number of nodes).
- The snapshot management function depends on OBS.

A snapshot contains the data in databases running in a cluster and the cluster information, including the node quantity, node flavors, and administrator names. To restore a cluster from a snapshot, GaussDB(DWS) uses the cluster information to create a new cluster and then restores all databases from the snapshot. The new cluster created from the snapshot has the same configurations (including the number and flavor of nodes) as those of the original cluster. When restoring a cluster from a snapshot, the parameter values are consistent with those in the snapshot unless you specify them.

There are two types of snapshots: automated and manual.

Backup and Restoration Policies of Automated Snapshots

Automated snapshots adopt differential incremental backups. The automated snapshot created for the first time is a full backup (base version), and then the system creates full backups at a specified interval. Incremental backups are generated between two full backups. The incremental backup records change based on the previous backup. During snapshot restoration, GaussDB(DWS) uses all backups between the latest full backup and the current incremental backup to restore the cluster. Therefore, no data loss occurs. To ensure that every incremental snapshot can be used for data restoration, when its retention period exceeds the upper limit, GaussDB(DWS) does not delete the snapshot immediately. Instead, GaussDB(DWS) retains it for future cluster restoration using other incremental snapshots. GaussDB(DWS) deletes the previous full automated snapshots and related incremental snapshots only after a new full snapshot is created. If you disable the automated snapshot function for an existing cluster, all its automated snapshots will be deleted. However, manual snapshots will not be deleted.

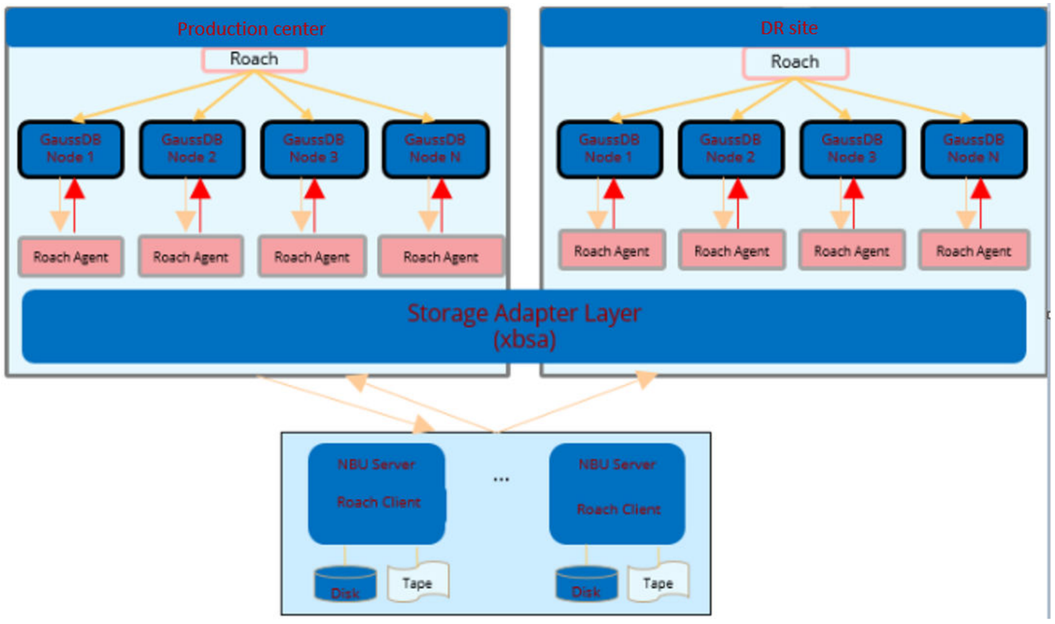


Ecosystem Interconnection

Storage media for database backup include NetBackup, EISOO, A8000, OBS, and disk storage media. Local disk storage competes with database data for storage space on disks. Therefore, remote storage media, such as NetBackup, EISOO and A8000, are used to manage backup data. In addition, these storage services perform technical operations such as data deduplication, and provides information about backup space usage. OBS is a data storage service provided by Huawei. It is easy to use. During backup, the backup process directly sends data to OBS for storage. You do not need to set up an intermediate client on the local backup server. The mainstream backup software uses this architecture, as shown in the following figure. Each node has a backup client. When a backup task starts, the backup task is delivered to each client. Each client creates an **eefproc** process, which invokes the backup command of the database. After the backup data is generated, it is stored in the pipe. The **eefproc** process reads the data and sends it to the backup server.

For the backup and restoration ecosystem interconnection, GaussDB(DWS) uses the standard XBSA interface for backup, and backup vendors implement the XBSA protocol and use the non-intrusive Roach client provided by GaussDB(DWS) to connect to the GaussDB(DWS) backup storage service.

Figure 3-15 Backup and restoration ecosystem interconnection architecture



For details, see **Snapshots**.

3.13.2 Cluster DR

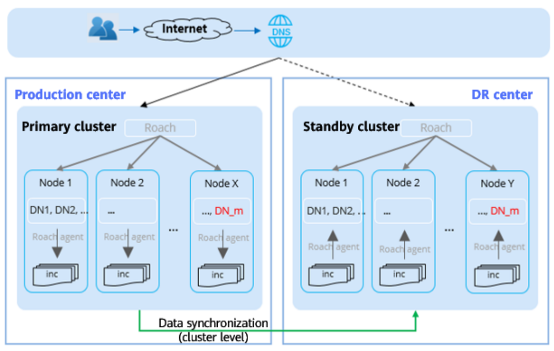
Background

GaussDB(DWS) clusters support disaster recovery. Data of the primary cluster is periodically synchronized to a standby cluster in another region or AZ. After considering several typical DR methods in the industry, such as dual-write on the application side and multiple remote copies, as well as the architecture characteristics of GaussDB(DWS) and the replicability of the DR solution, we have chosen the dual-cluster DR.

Technical Principles

Dual-cluster DR architecture:

- Data synchronization: Node-to-node data synchronization is used between two clusters with the same number of DNs based on the MPP distribution characteristics. The network between the two clusters must be connected mutual trust must be configured.
- Periodic synchronization: Data is synchronized at a configurable interval.
- Data content: Cluster data including row-store data, column-store data, library data, and configuration information is synchronized.



Architecture:
- Cross-DC (cross-AZ/cross-region/offline), dual-cluster (primary/standby), and asynchronous replication through Roach
- DR data: cluster-level

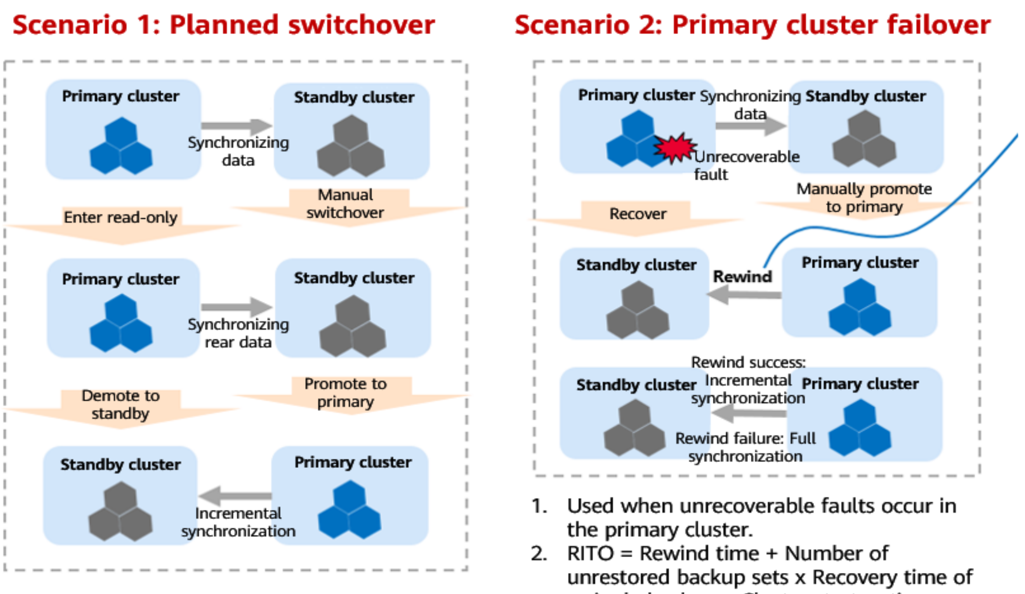
Application scenarios:
Intra-city and remote disaster recovery of enterprise-level data warehouses

RTO/RPO:
RTO/RPO: 30 minutes to hours, service-relevant

Constraints
- The primary and standby clusters must have the same topology and DNs.
- The network between the primary and standby clusters must be connected, and mutual trust can be configured.

Features:
- The two clusters are relatively independent. During non-recovery periods, the primary cluster can be read and written, and the standby cluster can be read.
- The DR architecture does not depend on underlying storage, services are transparent, and PB-scale data synchronization is supported.
- DR of row-store and column-store data is supported..

Cluster switchover:


Scenario 1: Planned switchover
Scenario 2: Primary cluster failover
Primary cluster
Standby cluster
Synchronizing data
Enter read-only
Manual switchover
Synchronizing rear data
Demote to standby
Promote to primary
Incremental synchronization
Unrecoverable fault
Recover
Manually promote to primary
Rewind
Rewind success: Incremental synchronization
Rewind failure: Full synchronization

1. Used for DR drills.
2. RITO = Number of stacked backup sets x Recovery time of a single backup set + Cluster startup time;
3. RPO= 0

1. Used when unrecoverable faults occur in the primary cluster.
2. RITO = Rewind time + Number of unrestored backup sets x Recovery time of a single backup + Cluster startup time
3. RPO = Incremental data of the primary cluster that is not synchronized

There are two types of DR switchover. One is planned switchover, which is used for DR drills. The other is failover, which is performed upon cluster faults.

For a planned switchover, if the RPO is 0, the primary cluster synchronizes data to the standby cluster and becomes the standby cluster. During a failover, the standby cluster immediately becomes the primary cluster. In this case, the RPO is not 0.

Benefits

The dual-cluster HA DR is loosely coupled and is selected by vendors such as Oracle and DB2. The advantages are as follows: (a) The two clusters are independent of each other, and the standby cluster does not affect the primary cluster. (b) The dual-cluster switchover can easily support the software upgrade and application upgrade of the major versions. It helps with the data security protection of large-scale MPP clusters.

For more information, see **DR Overview**.

4 GaussDB(DWS) Tools

4.1 Client Tools

4.1.1 gsql

Background

gsql is a database connection tool provided by GaussDB(DWS) and used by executing commands. You can use gsql to connect to a server and perform O&M on the server. In addition to the basic functions of operating on a database, gsql provides advanced features.

Function Description

- **Connecting to the database**: By default, you can connect to only the local server. To connect to a remote database, configure the server first.
- **Running SQL commands**: You can enter and run SQL commands interactively or run the SQL commands specified in a file.
- **Running meta-commands**: Meta-commands help administrators view database object information, query cache information, format SQL output, and connect to a new database.

4.1.2 Data Studio

Description

Data Studio is a GUI tool that connects to a database to help you execute and debug SQL statements and stored procedures. Data Studio supports basic features of GaussDB(DWS) and provides a GUI for database developers. This improves the efficiency of constructing application programs and simplifies database development.

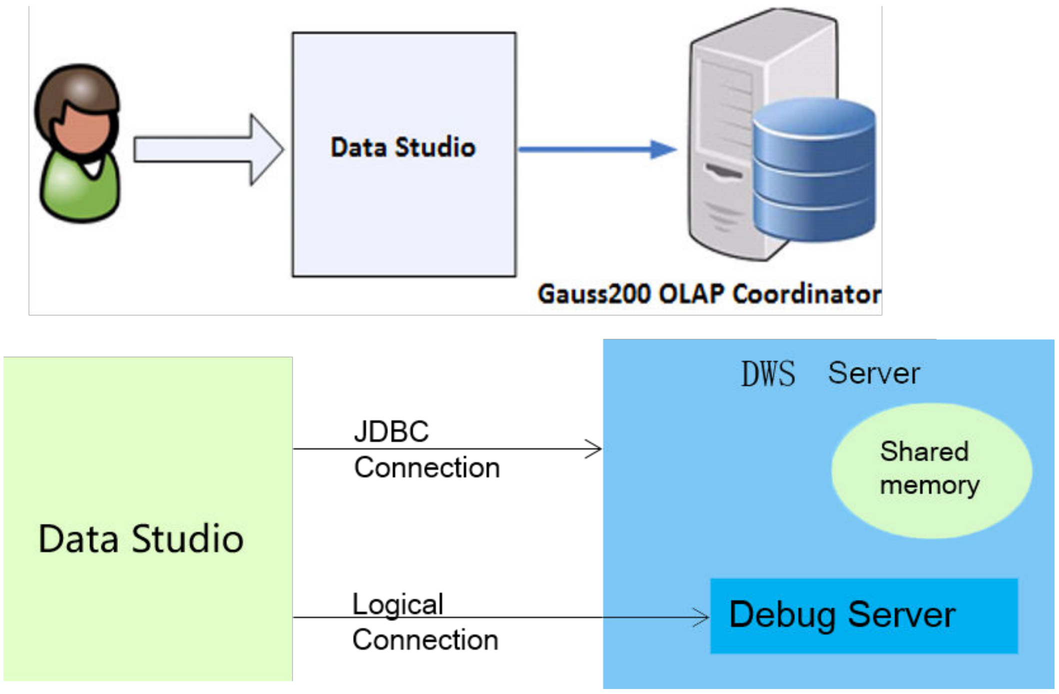
Data Studio provides the following functions for database developers:

- Browsing database objects

- Creating and managing database objects, such as databases, users, tables, and indexes
- Editing and running PL/SQL statements
- Importing and exporting table data
- Debugging SQL statements and stored procedures

Technical Principles

Figure 4-1 DataStudio component interaction



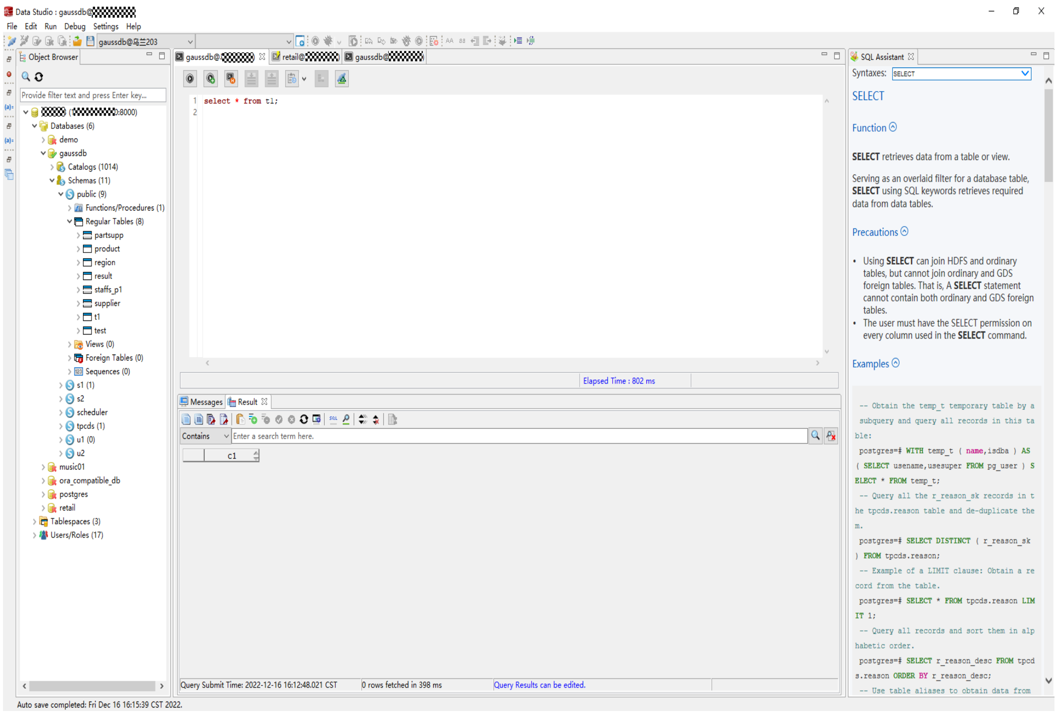
As shown in the preceding figure, Data Studio uses the C/S structure and communicates with the GaussDB(DWS) database through the JDBC driver.

During debugging, two connections are used:

- JDBC connection, which is used for query.
- Logical connection, which is used for other debugging operations, such as breakpoint and variable operations.

The database service and debugging service communicate with each other through the shared memory.

Figure 4-2 Data Studio GUI



```
select * from t1;
```

Elapsed Time : 802 ms

Query Submit Time: 2022-12-16 16:12:48.021 CST | 0 rows fetched in 398 ms | Query Results can be edited.

Auto save completed: Fri Dec 16 16:15:39 CST 2022.

SELECT

Function

SELECT retrieves data from a table or view.

Serving as an overlaid filter for a database table, **SELECT** using SQL keywords retrieves required data from data tables.

Precautions

- Using **SELECT** can join HDFS and ordinary tables, but cannot join ordinary and GDS foreign tables. That is, A **SELECT** statement cannot contain both ordinary and GDS foreign tables.
- The user must have the SELECT permission on every column used in the **SELECT** command.

Examples

```
 -- Obtain the temp_t temporary table by a
 subquery and query all records in this ta
ble:
 postgres=# WITH temp_t ( name,isdba ) AS
( SELECT usename,usesuper FROM pg_user ) S
ELECT * FROM temp_t;
 -- Query all the r_reason_sk records in t
he tpcds.reason table and de-duplicate the
m.
 postgres=# SELECT DISTINCT ( r_reason_sk
) FROM tpcds.reason;
 -- Example of a LIMIT clause: Obtain a re
cord from the table.
 postgres=# SELECT * FROM tpcds.reason LIM
IT 1;
 -- Query all records and sort them in alp
habetic order.
 postgres=# SELECT r_reason_desc FROM tpcd
s.reason ORDER BY r_reason_desc;
 -- Use table aliases to obtain data from
```

No.	Function	Description
1	Managing database objects	Manages database objects such as databases, schemas, tables, columns, constraints, indexes, views, tablespaces, and user roles.
2	SQL editor	Creates, edits, runs, and debugs PL/SQL stored procedures, formats query statements, automatically recommends SQL statements, and fills in templates.
3	Stored procedure debugging	Creates debugging links, uses breakpoints to control the execution of PL/SQL stored procedures, and displays debugging information such as call stacks and variables.
4	console	Views the execution plan and cost. Standard input and output are supported.
5	Query results	Displays, copies, exports, edits, and searches for query results.

For details about how to use Data Studio, see section **About Data Studio**.

4.1.3 Database Schema Convertor

Background

After customers switch to Huawei databases, database migration is required, which includes user data migration and application SQL script migration.

The migration of application SQL scripts is a complex, risky, and time-consuming process.

Description

Database Schema Convertor is a command-line tool running on the Linux or Windows OS. It is dedicated to providing customers with simple, fast, reliable application SQL script migration services. It parses SQL scripts into ones that compatible with GaussDB(DWS).

Database Schema Convertor does not require a connection to databases, and can perform migration offline. The tool also displays the status of a migration process and logs the errors that occur during the process, helping quickly locate faults.

Database Schema Convertor supports migration from Teradata and Oracle to GaussDB(DWS). It now supports the following objects:

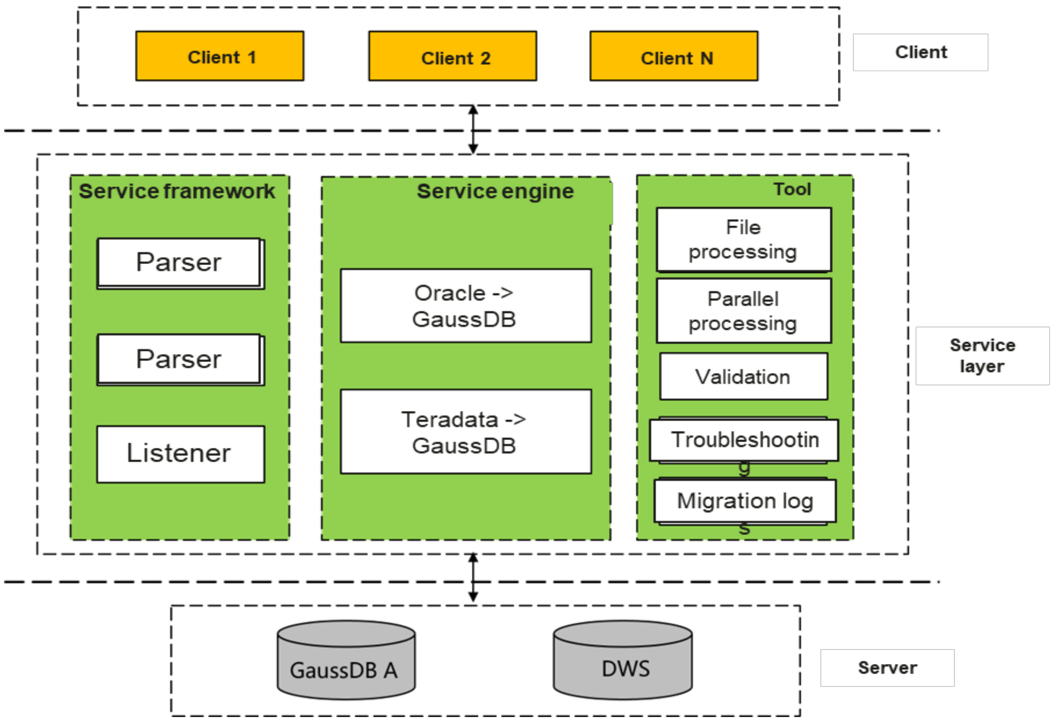
- General objects: SQL schemas and SQL queries
- Objects supported only by Oracle: PL/SQL objects
- Objects supported only by Teradata: Perl files containing **BTEQ** and **SQL_LANG** scripts

Target user groups

- Database administrators
- Database migration engineers

Technical Principles

Figure 4-3 Database Schema Convertor architecture



Database Schema Convertor supports the following types of source and target databases:

No.	Function	Description
1	Teradata Perl Migration	Migrating Teradata Perl files to GaussDB A, GaussDB 300, and GaussDB(DWS)
2	Teradata SQL Migration	Migrating Teradata SQL to GaussDB A, GaussDB 300, and GaussDB(DWS)
3	Oracle SQL Migration	Migrating Oracle SQL to GaussDB A, GaussDB 300, and GaussDB(DWS)
4	Oracle(beta) SQL Migration	Migrating Oracle SQL to GaussDB A, GaussDB 300, and GaussDB(DWS)

For details about how to use Database Schema Convertor, see the section **DSC: SQL Syntax Migration Tool**.

4.1.4 Data Admin Service (DAS)

Description

Data Admin Service (DAS) is a one-stop platform that allows you to manage databases on a web console. It offers database development, O&M, and intelligent diagnosis, making it easy for you to use and maintain databases. The Standard Edition provides a best-in-class experience. You no longer need to install clients locally for a visualized operation experience. Diverse database development functions are available, including data and table structure synchronization, online editing, and intelligent prompts for SQL input.

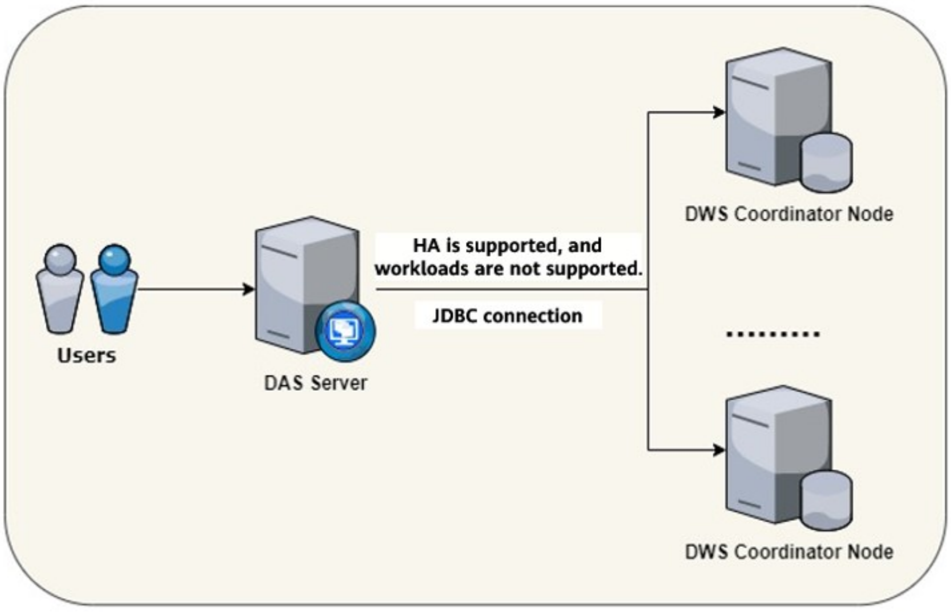
Version Requirements: The GaussDB(DWS) version must be 8.0.1 or later, and the Agent version must be 8.1.3.101 or later.

DAS provides the following functions for GaussDB(DWS) developers:

- Browsing and managing database objects, such as databases, tables, views, stored procedures, sequences, and triggers.
- Editing and executing PL/SQL statements
- Managing SQL execution records

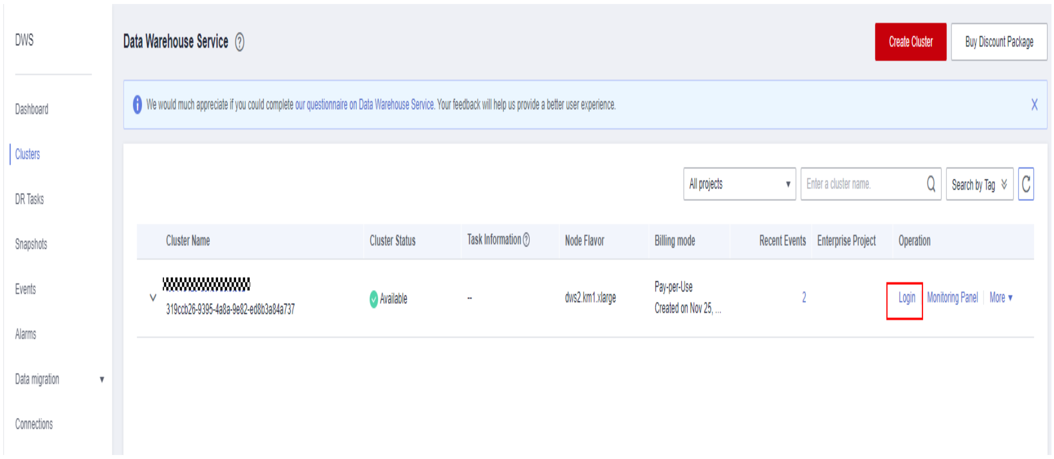
Technical Principles

Figure 4-4 Interaction between DAS and GaussDB(DWS)



As shown in the preceding figure, DAS uses the B/S structure and communicates with the GaussDB(DWS) database through the JDBC driver.

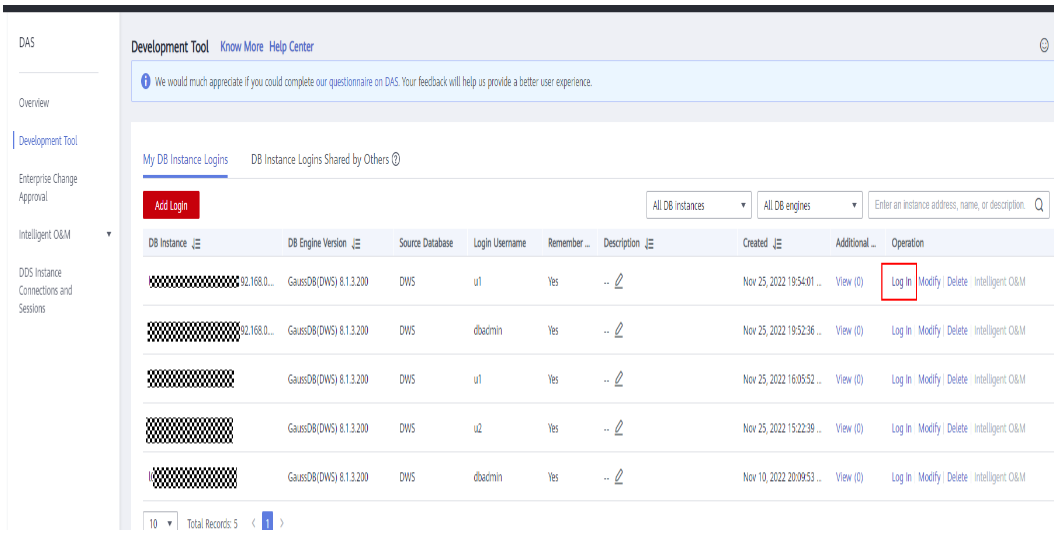
Figure 4-5 GaussDB(DWS) entry



We would much appreciate if you could complete our questionnaire on Data Warehouse Service. Your feedback will help us provide a better user experience.

Cluster Name	Cluster Status	Task Information	Node Flavor	Billing mode	Recent Events	Enterprise Project	Operation
[redacted] 319ccb26-9395-4a8a-9e82-ed8b3a84a737	Available	--	dws2.km1.xlarge	Pay-per-Use Created on Nov 25, ...	2		Login Monitoring Panel More

Figure 4-6 DAS entry

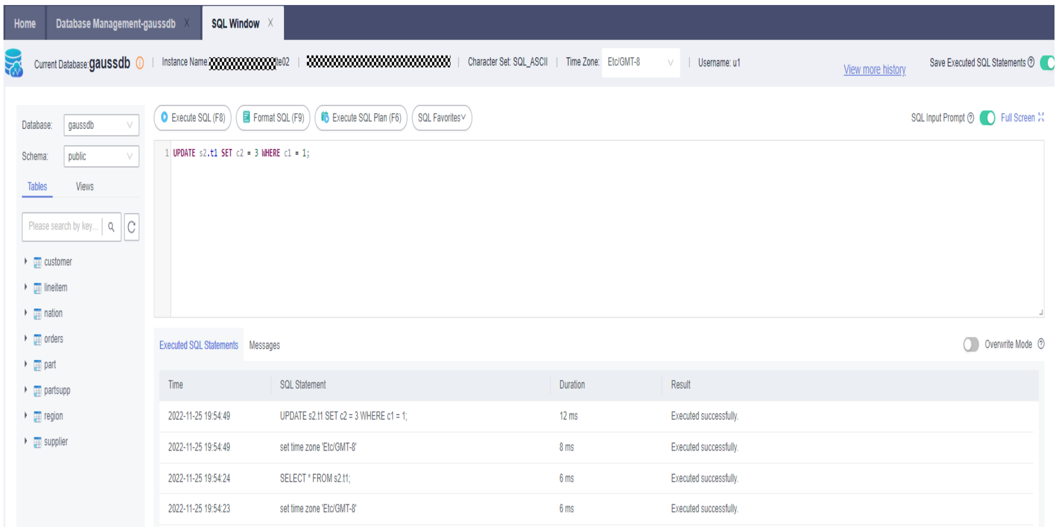

DAS
Overview
Development Tool
Enterprise Change Approval
Intelligent O&M
DDS Instance Connections and Sessions
Development Tool
Know More
Help Center
My DB Instance Logins
DB Instance Logins Shared by Others
Add Login
All DB instances
All DB engines
Enter an instance address, name, or description.

We would much appreciate if you could complete our questionnaire on DAS. Your feedback will help us provide a better user experience.

DB Instance	DB Engine Version	Source Database	Login Username	Remember ...	Description	Created	Additional ...	Operation
[redacted]92.168.0...	GaussDB(DWS) 8.1.3.200	DWS	u1	Yes	--	Nov 25, 2022 19:54:01 ...	View (0)	Log In | Modify | Delete | Intelligent O&M
[redacted]92.168.0...	GaussDB(DWS) 8.1.3.200	DWS	dbadmin	Yes	--	Nov 25, 2022 19:52:36 ...	View (0)	Log In | Modify | Delete | Intelligent O&M
[redacted]	GaussDB(DWS) 8.1.3.200	DWS	u1	Yes	--	Nov 25, 2022 16:05:52 ...	View (0)	Log In | Modify | Delete | Intelligent O&M
[redacted]	GaussDB(DWS) 8.1.3.200	DWS	u2	Yes	--	Nov 25, 2022 15:22:39 ...	View (0)	Log In | Modify | Delete | Intelligent O&M
[redacted]	GaussDB(DWS) 8.1.3.200	DWS	dbadmin	Yes	--	Nov 10, 2022 20:09:53 ...	View (0)	Log In | Modify | Delete | Intelligent O&M

10 Total Records: 5 1

Figure 4-7 DAS-SQL execution page



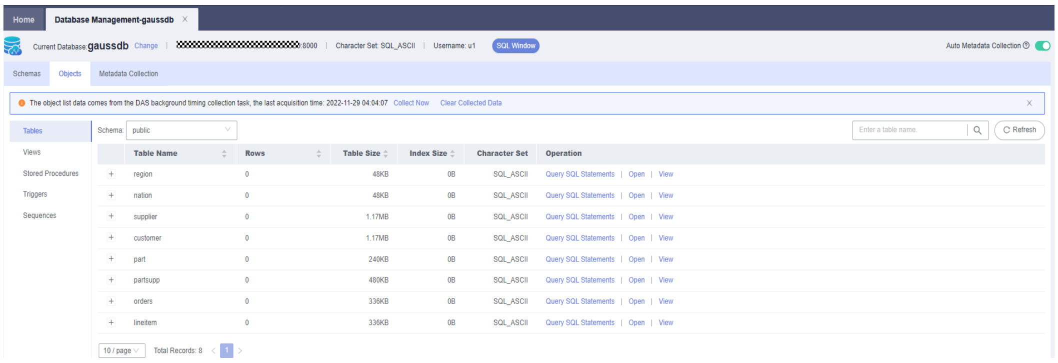
Current Database: gaussdb | Instance Name: [redacted]e02 | [redacted] | Character Set: SQL_ASCII | Time Zone: Etc/GMT-8 | Username: u1

Database: gaussdb
Schema: public

```
UPDATE s2.t1 SET c2 = 3 WHERE c1 = 1;
```

Time	SQL Statement	Duration	Result
2022-11-25 19:54:49	UPDATE s2.t1 SET c2 = 3 WHERE c1 = 1;	12 ms	Executed successfully.
2022-11-25 19:54:49	set time zone 'Etc/GMT-8'	8 ms	Executed successfully.
2022-11-25 19:54:24	SELECT * FROM s2.t1;	6 ms	Executed successfully.
2022-11-25 19:54:23	set time zone 'Etc/GMT-8'	6 ms	Executed successfully.

Figure 4-8 DAS-database object page



[redacted]:8000 | Character Set: SQL_ASCII | Username: u1

The object list data comes from the DAS background timing collection task, the last acquisition time: 2022-11-29 04:04:07

Schema: public

	Table Name	Rows	Table Size	Index Size	Character Set	Operation
+	region	0	48KB	0B	SQL_ASCII	Query SQL Statements | Open | View
+	nation	0	48KB	0B	SQL_ASCII	Query SQL Statements | Open | View
+	supplier	0	1.17MB	0B	SQL_ASCII	Query SQL Statements | Open | View
+	customer	0	1.17MB	0B	SQL_ASCII	Query SQL Statements | Open | View
+	part	0	240KB	0B	SQL_ASCII	Query SQL Statements | Open | View
+	partsupp	0	480KB	0B	SQL_ASCII	Query SQL Statements | Open | View
+	orders	0	336KB	0B	SQL_ASCII	Query SQL Statements | Open | View
+	lineitem	0	336KB	0B	SQL_ASCII	Query SQL Statements | Open | View

10 / page Total Records: 8 < 1 >

Table 4-1 Description

No.	Function	Description
1	Managing database objects	Manages database objects such as databases, schemas, tables, columns, constraints, indexes, views, stored procedures, triggers, and sequences.
2	Creating and executing stored procedures/ functions	Creates and executes stored procedures/ functions
3	SQL editor	SQL statement execution, formatting, execution plan, SQL prompt, and common SQL management
4	Managing SQL execution records	Manages execution records of non-sensitive SQL statements.

For details about how to use DAS, see **Using DAS to Connect to a Cluster**.

4.2 Database Monitoring Tool

Background

As the IT infrastructure, databases must be stable, high-throughput, and low-latency. To achieve the preceding objectives, ensure that the database cluster can continuously provide stable and fast services for users. A comprehensive database monitoring tool is required to help database O&M personnel monitor database running in real time and detect, locate, and handle exceptions in a timely manner.

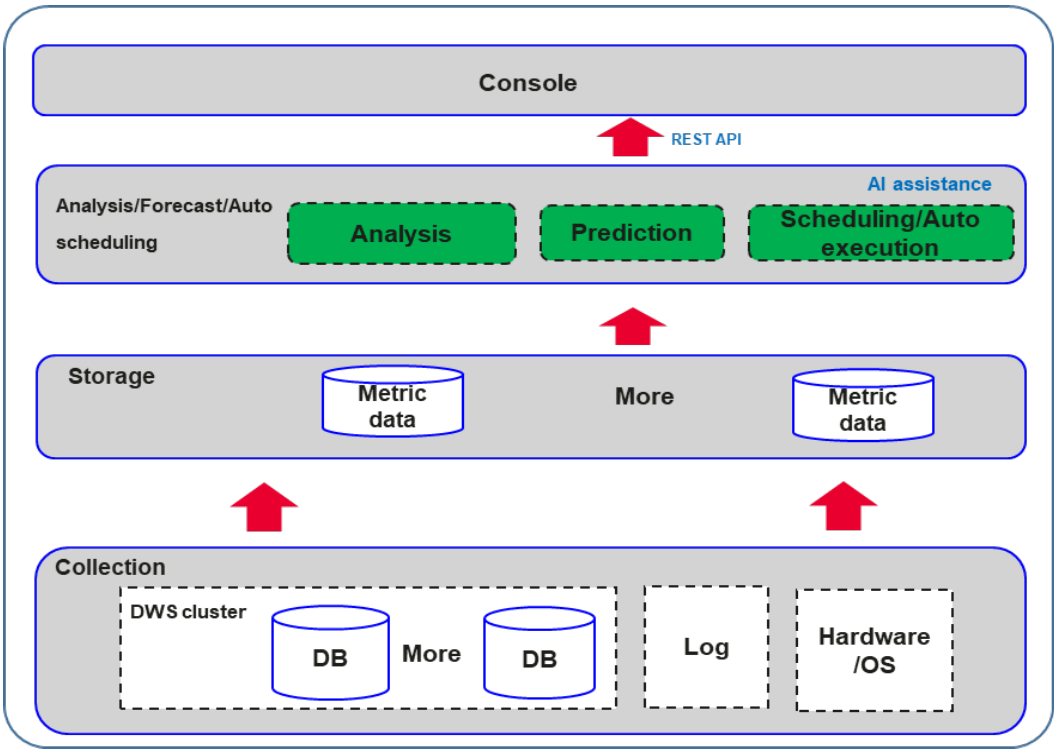
The Database Monitor Service (DMS) is a native database monitoring service of GaussDB(DWS). It provides comprehensive database resource consumption indicators and database service execution indicators. DMS provides a visual tool to monitor the real-time and historical running status of database clusters, helping you detect, locate, and rectify faults.

Database Monitoring Principles

DMS uses a three-layer structure to monitor the GaussDB(DWS) database cluster:

- Collection (dms-agent): The agent is embedded in each node of the database cluster to collect the cluster running status, collect the original data of cluster running, and report the data.
- Storage (dms-collection): receives data reported by the agent and saves the data to the metric database.
- Analysis (dms-monitoring): A large amount of data in the monitoring database is used, and the monitoring view is displayed to users through aggregation calculation.

Figure 4-9 DMS architecture



Database Monitoring Metrics

Table 4-2 describes the database monitoring metrics. For more information, see **Database Monitoring Overview**

Table 4-2 GaussDB(DWS) monitoring metrics

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
Cluster Overview	Cluster Status	Status of a cluster.	Normal/ Abnormal / Degraded	30s
	Nodes	Number of available nodes and total number of nodes (Available/Total) in a cluster.	≥ 0	60s
	CNs	Number of CNs in a cluster.	≥ 0	60s
	Databases	Number of created databases in a cluster.	≥ 0	90s
Resource Consumption	CPU Usage	Average real-time CPU usage of all nodes in a cluster.	0% to 100%	30s
	Memory Usage	Average real-time memory usage of all nodes in a cluster.	0% to 100%	30s
	Disk Usage	Average real-time disk usage of all nodes in a cluster.	0% to 100%	30s
	Disk I/O	Average real-time disk I/O of all nodes in a cluster.	≥ 0 KB/s	30s
	Network I/O	Average real-time network I/O of all NICs in a cluster.	≥ 0 KB/s	30s
Top 5 Time-Consuming Queries	Query ID	ID of a query, which is automatically generated by the database.	≥ 0	180s
	SQL Statement	Query statement executed by a user.	Character string	180s
	Execution Time	Execution time of a query statement (unit: ms).	≥ 0 ms	180s
Top 5 Queries with Most Data Written to Disk	Query ID	ID of a query, which is automatically generated by the database.	≥ 0	180s
	SQL Statement	Query statement executed by a user.	Character string	180s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Data Written to Disk	Data to be written to disks after a user runs a statement (unit: MB).	≥ 0 MB	180s
Cluster Resource Metrics	CPU Usage	Average CPU usage and skew ratio of all nodes in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Memory Usage	Average memory usage and skew ratio of all nodes in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Disk Usage	Average usage and skew ratio of all disks in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Disk I/O Usage	Average I/O usage and skew rate of all disks in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Network I/O Usage	Average I/O usage and skew rate of all NICs in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
Key Database Metrics	Cluster Status	Cluster running status.	Normal/ Degraded / Abnormal	30s
	Cluster Abnormal CNs	Number of abnormal CNs in the cluster	≥ 0	60s
	Cluster Read-only	Whether the cluster is in the read-only state	Yes/No	30s
	Concurrent Sessions	Number of concurrent sessions in a cluster within a specified period.	≥ 0	30s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Concurrent Queries	Number of concurrent queries in a cluster within a specified period.	≥ 0	30s
Node Monitoring-Overview	Node Name	Name of a node in a cluster.	Character string	30s
	CPU Usage	CPU usage of a host.	0% to 100%	30s
	Memory Usage	Memory usage of a host.	0% to 100%	30s
	Average Disk Usage (%)	Disk usage of a host.	0% to 100%	30s
	IP Address	Service IP address of a host.	Character string	30s
	Disk I/O	Disk I/O of a host (unit: KB/s)	≥ 0 KB/s	30s
	TCP Protocol Stack Retransmission Rate	Retransmission rate of TCP packets per unit time.	0% to 100%	30s
	Status	Running status of a host	Online/ Offline	30s
Node Monitoring-Disks	Node Name	Name of a node in a cluster.	Character string	30s
	Disk Name	Name of a disk on a host.	Character string	30s
	Disk Capacity	Disk capacity of the host (unit: GB)	≥ 0 GB	30s
	Disk Usage	Disk usage of a host.	0% to 100%	30s
	Disk Read Rate	Disk read rate of the host (unit: KB/s)	≥ 0 KB/s	30s
	Disk Write Rate	Disk write rate of the host (unit: KB/s)	≥ 0 KB/s	30s
	I/O Wait Time (await, ms)	Average waiting time for each I/O request (unit: ms)	≥ 0 ms	30s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	I/O Service Time (svctm, ms)	Average processing time for each I/O request (unit: ms)	≥ 0 ms	30s
	I/O Utility (util, %)	Disk I/O usage of a host.	0% to 100%	30s
Node Monitoring-Network	Node Name	Name of a node in a cluster.	Character string	30s
	NIC Name	Name of the NIC on a host.	Character string	30s
	NIC Status	NIC status.	up/down	30s
	NIC Speed	Working rate of a NIC, in Mbit/s.	≥ 0	30s
	Received Packets	Number of received packets of a NIC.	≥ 0	30s
	Sent Packets	Number of sent packets of a NIC.	≥ 0	30s
	Lost Packets Received	Number of received lost packets of a NIC.	≥ 0	30s
	Receive Rate	Number of bytes received by a NIC per unit of time (KB/s).	≥ 0 KB/s	30s
	Transmit Rate	Number of bytes sent by a NIC per unit of time (unit: KB/s)	≥ 0 KB/s	30s
Database Monitoring	Database Name	Name of the database created by a user in a cluster.	Character string	60s
	Usage	Used capacity of the current database (unit: GB).	≥ 0 GB	86400s
	Users	Number of users in the current database.	≥ 0	30s
	Sessions	Number of sessions in the current database.	≥ 0	30s
	Applications	Number of applications in the current database.	≥ 0	30s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Queries	Number of active queries in the current database.	≥ 0	30s
	Scanning Rows	Number of rows returned by the full table scan query in the current database.	≥ 0	60s
	Index Query Rows	Number of rows returned by the index query in the current database.	≥ 0	60s
	Inserted Rows	Number of rows inserted in the current database.	≥ 0	60s
	Updated Rows	Number of rows updated in the current database.	≥ 0	60s
	Deleted Rows	Number of rows deleted from the current database.	≥ 0	60s
	Executed Transactions	Number of transaction executions on the current database.	≥ 0	60s
	Transaction Rollbacks	Number of transactions in the current database that have been rolled back.	≥ 0	60s
	Deadlocks	Number of deadlocks detected in the current database.	≥ 0	60s
	Temporary Files	Number of temporary files created in the current database.	≥ 0	60s
	Temporary File Capacity	Size of temporary files written by the current database, in GB.	≥ 0	60s
Performance Monitoring	Cluster CPU Usage	Historical trend of the average CPU usage and skew of all nodes in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Cluster Memory Usage	Historical trend of the average memory usage and skew of all nodes in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Cluster Disk Usage	Historical trend of the average disk usage and skew of all nodes in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Cluster Disk I/O	Historical trend of the average disk I/O and skew of all disks in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Cluster Network I/O	Historical trend of the average network I/O value and skew of all NICs in the cluster. The formula for calculating the skew is (max-avg)/max.	0% to 100%	30s
	Cluster Status	Historical trend of the cluster status.	Normal/ Abnormal / Degraded	30s
	Cluster Read-only	Historical trend of the cluster read-only status change trend.	Yes/No	30s
	Cluster Abnormal CNs	Historical trend of the number of abnormal CNs in the cluster.	≥ 0	60s
	Cluster Abnormal DNs	Historical trend of the number of abnormal DNs in the cluster.	≥ 0	60s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Cluster CPU Usage of DNs	Historical trends of the average CPU usage and skew ratio changes of all DNs in the cluster. The formula for calculating the skew ratio is (max – avg)/ max.	0% to 100%	60s
	Cluster Sessions	Historical trend of the number of sessions in a cluster.	≥ 0	30s
	Cluster Queries	Historical change trend of the number of queries in the cluster.	≥ 0	30s
	Cluster Deadlocks	Historical trend of the number of deadlocks in a cluster.	≥ 0	60s
	Cluster TPS	Average number of transactions per second of all databases in a cluster. Formula: (delta_xact_commit + delta_xact_rollback)/ current_collect_rate	≥0	60s
	Cluster QPS	Average number of concurrent requests per second of all databases in a cluster. Formula: delta_query_count/ current_collect_rate	≥ 0	60s
	Database Sessions	Historical trend of the number of sessions on a single database in a cluster.	≥ 0	30s
	Database Queries	Historical trend of the number of queries on a single database in a cluster.	≥ 0	30s
	Database Inserted Rows	Historical trend of the number of rows inserted into a single database in a cluster.	≥ 0	60s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Database Updated Rows	Historical trend of the number of updated rows in a single database in a cluster.	≥ 0	60s
	Database Deleted Rows	Historical trend of the number of deleted rows in a single database in a cluster.	≥ 0	60s
	Database Capacity	Historical trend of the capacity in a single database in a cluster.	≥ 0	86400s
Live Session	Session ID	ID of the current session (query thread ID).	Character string	30s
	User Name	Name of the user who executes the current session.	Character string	30s
	Database Name	Name of the database connected to the current session.	Character string	30s
	Session Duration	Duration of the current session (unit: ms).	≥ 0 ms	30s
	Application Name	Name of the application that creates the current session.	Character string	30s
	Queries	Number of SQL statements executed in the current session.	≥ 0	30s
	Latest Query Duration	Duration for executing the previous SQL statement in the current session.	≥ 0 ms	30s
	Client IP Address	IP address of the client that initiates the current session.	Character string	30s
	Connected CN	Connected CN of the current session.	Character string	30s
	Session Status	Execution status of the current session.	Running/ Idle/Retry	30s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
Real-Time Query	Query ID	Query ID of a current query statement, which is a unique identifier allocated by the kernel to each query statement.	Character string	30s
	User Name	Name of the user who submits the current query statement.	Character string	30s
	Database Name	Name of the database corresponding to the current query statement.	Character string	30s
	Application Name	Name of the application corresponding to the current query statement.	Character string	30s
	Workload Queue	Name of the workload queue that carries the current query statement.	Character string	30s
	Submitted	Timestamp when the current query statement is submitted.	Character string	30s
	Blocking Time	Waiting time before the current query statement is executed, in ms.	≥ 0	30s
	Execution Time	Execution time of the current query statement, in ms.	≥ 0	30s
	CPU Time	Total CPU time spent by the current query statement on all DNs, in ms.	≥ 0	30s
	CPU Time Skew	CPU time skew of the current query statement among all DNs.	0% to 100%	30s
	Statement	Query statement that is being executed.	Character string	30s
	Connected CN	Name of the CN that submits the current query statement.	Character string	30s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Client IP Address	IP address of the client that submits the current query statement.	Character string	30s
	Lane	Lane where the current query statement is located.	Fast Lane/ Slow Lane	30s
	Query Status	Query status of the statement that is being executed.	Character string	30s
	Session ID	Session ID of the current query statement, which is a unique identifier allocated by the kernel to each client connection.	Character string	30s
	Queuing Status	Status of the current query execution in the database, indicating whether the query is queued in the workload queue.	Yes/No	30s
Historical Query	Query ID	Query ID of a query statement, which is a unique identifier allocated by the kernel to each query statement.	Character string	180s
	User Name	Name of the user who submits a query statement.	Character string	180s
	Application Name	Application name corresponding to a query statement.	Character string	180s
	Database Name	Name of the database corresponding to a query statement.	Character string	180s
	Workload Queue	Name of the workload queue that carries the current query statement.	Character string	180s
	Submitted	Timestamp when a query statement is submitted.	Character string	180s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Blocking Time	Waiting time before the query statement is executed, in ms.	≥ 0	180s
	Execution Time	Execution time of the query statement, in ms.	≥ 0	180s
	CPU Time	Total CPU time spent by the query statement on all DNs, in ms.	≥ 0	180s
	CPU Time Skew	CPU time skew of a query statement executed on all DNs.	0% to 100%	180s
	Statement	Query statements to be parsed	Character string	180s
Slow Instance Monitoring	Slow Instance	Number of slow instances detected at the current time point.	≥ 0	240s
	Detected	Time when a slow instance is detected for the first time.	Character string	240s
	Node Name	Name of the node where the slow instance is deployed.	Character string	240s
	Instance	Name of an instance.	Character string	240s
	Slow Node Detections (within 24 hours)	Number of times that a slow instance is detected within 24 hours.	≥ 0	240s
Workload Queue Monitoring	Workload Queue	Name of the workload queue in the cluster.	Character string	120s
	CPU Usage	Real-time CPU usage of the workload queue.	0% to 100%	120s
	CPU Resource	CPU usage quotas of the workload queue.	0% to 100%	120s
	Real-Time Concurrent Short Queries	Number of real-time concurrent simple queries in a workload queue.	≥ 0	120s
	Concurrent Short Queries	Concurrent simple query quotas of a workload queue.	≥ 0	120s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Real-Time Concurrent Queries	Number of real-time concurrent complex queries in a workload queue.	≥ 0	120s
	Query Concurrency	Concurrent complex query quotas of a workload queue.	≥ 0	120s
	Storage	Storage quota of the workload queue.	≥ 0	120s
	Disk Usage	Disk usage of the workload queue.	0% to 100%	120s
	Memory	Memory quota of the workload queue.	≥ 0	120s
	Memory Usage	Memory usage of the workload queue.	0% to 100%	120s
Waiting Queries	User	Name of the user of waiting queries	Character string	120s
	Application	Name of the application to be queried.	Character string	120s
	Database	Name of the database to be queried.	Character string	120s
	Queuing Status	Execution status of a query in the database (CCN/CN/DN).	Character string	120s
	Wait Time	Waiting time for a waiting query (unit: ms).	≥ 0 ms	120s
	Workload Queue	Workload queue to which the waiting query belongs.	Character string	120s
	Statement	Query statement for the waiting status.	Character string	120s
Circuit Breaking Queries	Query ID	Query ID of the circuit breaking query statement.	Character string	120s
	Query Statement	Query statement for the circuit breaking status.	Character string	120s
	Blocking Time	Blocking time before the query statement triggers circuit breaking, in ms.	≥ 0	120s

Monitored Object	Metric	Description	Value Range	Monitoring Period (Raw Data)
	Execution Time	Execution time before the query statement triggers circuit breaking, in ms.	≥ 0	120s
	CPU Time	Average CPU time consumed by each DN before the query statement triggers circuit breaking, in ms.	≥ 0	120s
	CPU Skew	Skew rate of CPU time consumed by each DN before the query statement triggers circuit breaking.	0% to 100%	120s
	Exception Handling	Handling method after the query statement triggers circuit breaking.	Abort/ Degrade	120s
	Status	Circuit breaking handling status of a query statement.	Executing / Completed	120s
SQL Tuning	Query ID	IP address of the current query (query logic ID).	Character string	180s
	Database	Name of the database where the current query is executed.	Character string	180s
	Schema Name	Name of the current query schema.	Character string	180s
	User Name	Name of the user who performs the query.	Character string	180s
	Client	Name of the client that initiates the current query.	Character string	180s
	Client IP Address	IP address of the client that initiates the current query.	Character string	180s
	Running Time	Execution time of the current query, in ms.	≥ 0	180s
	CPU Time	CPU time of the current query, in ms.	≥ 0	180s
	Scale-Out Started	Start time of the current query.	Timestamp	180s

Monitored Object	**Metric**	**Description**	**Value Range**	**Monitoring Period (Raw Data)**
	Completed	End time of the current query.	Timestamp	180s
	Details	Details about the current query.	Character string	180s
INODE	Inode Usage	Disk inode usage.	0% to 100%	30s
SCHEMA	Schema Usage	Database schema usage.	0% to 100%	3600s

5 External APIs

Table 6-1 describes external APIs provided by the components of GaussDB(DWS).

Table 5-1 External APIs of the components

Component	Supported API Type
DWS	JDBC and ODBC
Management plane of DWS	REST/SNMP