Tencent Cloud TCHouse-D

Practical Tutorial

Product Documentation

Contents

Practical Tutorial

Basic Feature Usage

Last updated：2024-07-31 09:16:24

Create a table

Choosing a Data Model

The Doris data model is currently divided into three categories: AGGREGATE KEY, UNIQUE KEY, and DUPLICATE KEY. In all three models, data is sorted by KEY.

1. AGGREGATE KEY

When AGGREGATE KEYs are the same, the new and old records are aggregated. Currently supported aggregate functions include SUM, MIN, MAX, and REPLACE.

The AGGREGATE KEY model can aggregate data in advance and is suitable for reporting and multidimensional analysis business.

```
CREATE TABLE site_visit
(
 siteid      INT,
 city        SMALLINT,
 username    VARCHAR(32),
 pv BIGINT   SUM DEFAULT '0'
)
AGGREGATE KEY (siteid, city, username)
DISTRIBUTED BY HASH (siteid) BUCKETS 10;
```

2. UNIQUE KEY

When UNIQUE KEYs are the same, the new record overwrites the old record. Currently, the implementation of UNIQUE KEYs is the same as the REPLACE aggregation method of AGGREGATE KEYs, and the two are essentially the same. It is suitable for analytical businesses requiring updates.

```
CREATE TABLE sales_order
(
     orderid     BIGINT,
     status      TINYINT,
     username    VARCHAR(32),
     amount      BIGINT DEFAULT '0'
)
UNIQUE KEY(orderid)
DISTRIBUTED BY HASH(orderid) BUCKETS 10;
```

3. DUPLICATE KEY

Only the sort column is specified, and the same rows will not be merged. It is suitable for analytical businesses where advanced data aggregation is not required.

```
CREATE TABLE session_data
(
    visitorid   SMALLINT,
    sessionid   BIGINT,
    visittime   DATETIME,
    city        CHAR(20),
    province    CHAR(20),
    ip          varchar(32),
    brower      CHAR(20),
    url         VARCHAR(1024)
)
DUPLICATE KEY(visitorid, sessionid)
DISTRIBUTED BY HASH(sessionid, visitorid) BUCKETS 10;
```

Large Wide Tables and Star Schema

When the business side creates a table, in order to adapt to the front-end business, they often do not distinguish between dimension information and metric information, but define the Schema as a large wide table. For Doris, these large, such large wide tables often do not perform as well as one expects.

There are many fields in the Schema, and there may be more key columns in the aggregation model, which will increase the columns to be ordered during the import process.

The update of dimension information will be reflected in the entire table, and its update frequency directly affects query efficiency.

During use, users should use Star Schema to distinguish between dimension tables and metric tables whenever possible. Frequently updated dimension tables can also be placed in MySQL external tables. If there are only a few updates, they can be placed directly in Doris. When storing dimension tables in Doris, you can set more replicas of the dimension tables to improve the performance of Join.

Partitioning and Bucketing

Doris supports two-level partition storage. The first level is partition, which currently supports two types: RANGE partition and LIST partition. The second level is HASH bucket.

1. Partition

Partitions are used to divide data into different intervals, which can be logically understood as dividing a base table into multiple sub-tables. Data can be easily managed by partition, for example, deleting data is faster with partition.

1.1 RANGE Partition

In business, most users choose to partition by time.

1.2 LIST Partition

In business, users can choose cities or other enumeration values for partitioning.

2. HASH Bucket

Data is divided into different buckets by hash value.

It is recommended to use columns with high discrimination for bucketing to avoid data skew.

To facilitate data recovery, it is recommended to restrict the size of a single bucket to 10 GB. You should consider the number of buckets when creating a table or adding partitions. Different numbers of buckets can be designated to different partitions.

Sparse Indexes and Bloom Filters

Doris stores data in order and creates a sparse index for the data based on the order of the data. The index granularity is block (1,024 rows).

For the sparse index, a prefix of fixed length in schema is chosen as the index content. Currently in Doris, a prefix of 36 bytes is chosen as the index.

When creating a table, it is recommended to place the commonly used filter fields in the query in front of Schema. The more distinctive and frequently used query fields are, the earlier they are placed.

There is one thing needing attention, that is, the varchar type field. A varchar type field can only be the last field of a sparse index. The index is truncated at varchar, so if varchar appears first, the index may be less than 36 bytes long. For details, see Data Model, ROLLUP and Prefix Index.

In addition to sparse indexes, Doris also provides bloomfilter indexes, which have a significant filtering effect on columns with high discrimination. Considering that varchar cannot be placed in a sparse index, you can create a bloomfilter index.

Materialized Views (Rollup)

Rollup can essentially be understood as a materialized index of the Base Table. When creating a Rollup, you can select certain columns in the Base Table as Schema. The order of fields in Schema can also be different from that in the Base Table.

A Rollup can be created in the following situations:

1. The data in the Base Table is not highly aggregated.

This is usually because that there are highly discriminate fields in the Base Table. At this time, you can consider selecting certain columns and creating a Rollup.

For example, for the `site_visit` table:

```
site_visit(siteid, city, username, pv)
```

SiteID may result in low data aggregation. If the business side often needs to count PVs by cities, a Rollup with only city and PV can be created:

```
ALTER TABLE site_visit ADD ROLLUP rollup_city(city, pv);
```

2. The prefix index in the Base Table cannot be hit.

This is generally because that the Base Table creation method cannot cover all query modes. At this time, you can consider adjusting the column order and creating a Rollup.

For example, for the session_data table:

```
session_data(visitorid, sessionid, visittime, city, province, ip, brower, url)
```

If you want to analyze visits by browser and province in addition to visitorid, you can create a separate Rollup.

```
ALTER TABLE session_data ADD ROLLUP rollup_brower(brower,province,ip,url) DUPLICATE
```

Schema Change

In Doris, there are currently three ways to perform Schema Changes: Sorted Schema Change, Direct Schema Change, and Linked Schema Change.

1. Sorted Schema Change

The sorting method of the column has been changed, and the data needs to be re-sorted. For example, when a sorted column is deleted, the fields need to be re-sorted.

```
ALTER TABLE site_visit DROP COLUMN city;
```

2. Direct Schema Change: No re-sorting is required, but a data transformation is required. For example, modify the column type, add a column to a sparse index, and so on.

```
ALTER TABLE site_visit MODIFY COLUMN username varchar(64);
```

3. Linked Schema Change: No need to convert data, it can be completed directly. For example, add a column.

```
ALTER TABLE site_visit ADD COLUMN click bigint SUM default '0';
```

It is recommended to consider the Schema when creating a table, so that the process of Schema Change can be accelerated.

Advanced Features Usage

Last updated：2024-07-31 09:16:39

This document will introduce some advanced features.

Table Structure Change

You can use the ALTER TABLE command to modify the table Schema, including the following changes:

Add columns.

Drop columns.

Modify column types.

Change column order.

Here are some examples.

The Schema of the original table1 is as follows:

```
+----------+-------------+------+-------+---------+-------+
| Field    | Type        | Null | Key   | Default | Extra |
+----------+-------------+------+-------+---------+-------+
| siteid   | int(11)     | No   | true  | 10      |       |
| citycode | smallint(6) | No   | true  | N/A     |       |
| username | varchar(32) | No   | true  |         |       |
| pv       | bigint(20)  | No   | false | 0       | SUM   |
+----------+-------------+------+-------+---------+-------+
```

We add a new column uv with the type BIGINT, aggregation type SUM, and a default value of 0:
`ALTER TABLE table1 ADD COLUMN uv BIGINT SUM DEFAULT '0' after pv;`
After successful submission, you can check the job progress with the following command:
`SHOW ALTER TABLE COLUMN;`
When the job status is FINISHED, it indicates that the job is complete.
After the ALTER TABLE is completed, you can use `DESC TABLE` to view the latest Schema.

```
mysql> DESC table1;
+----------+-------------+------+-------+---------+-------+
| Field    | Type        | Null | Key   | Default | Extra |
+----------+-------------+------+-------+---------+-------+
| siteid   | int(11)     | No   | true  | 10      |       |
| citycode | smallint(6) | No   | true  | N/A     |       |
| username | varchar(32) | No   | true  |         |       |
| pv       | bigint(20)  | No   | false | 0       | SUM   |
| uv       | bigint(20)  | No   | false | 0       | SUM   |
+----------+-------------+------+-------+---------+-------+
5 rows in set (0.00 sec)
```

You can use the following command to cancel the currently executing job: `CANCEL ALTER TABLE COLUMN FROM table1` .

For more help, see `HELP ALTER TABLE` .

Rollup

Materialized means its data is physically stored independently, and **index** implies that Rollup can adjust column order to increase the hit rates of the prefix index and reduce key columns to enhance data aggregation.

Here are some examples.

The Schema of the original table1 is as follows:

```
+-----------+-------------+------+-------+---------+-------+
| Field     | Type        | Null | Key   | Default | Extra |
+-----------+-------------+------+-------+---------+-------+
| siteid    | int(11)     | No   | true  | 10      |       |
| citycode  | smallint(6) | No   | true  | N/A     |       |
| username  | varchar(32) | No   | true  |         |       |
| pv        | bigint(20)  | No   | false | 0       | SUM   |
| uv        | bigint(20)  | No   | false | 0       | SUM   |
+-----------+-------------+------+-------+---------+-------+
```

For table1, the detailed data uses siteid, citycode, and username as a composite key to aggregate the pv field. If the business team frequently needs to view the total pv for each city, you can create a rollup with only citycode and pv.
`ALTER TABLE table1 ADD ROLLUP rollup_city(citycode, pv);`
After successful submission, you can check the job progress with the following command: `SHOW ALTER TABLE ROLLUP;` . When the job status is FINISHED, it indicates that the job is complete.
After the Rollup is created, you can use `DESC table1 ALL` to view the Rollup information of the table.

```
mysql> desc table1 all;
+-------------+-----------+-------------+------+-------+---------+-------+
| IndexName   | Field     | Type        | Null | Key   | Default | Extra |
+-------------+----------+-------------+------+-------+---------+-------+
| table1      | siteid   | int(11)     | No   | true  | 10      |       |
|             | citycode | smallint(6) | No   | true  | N/A     |       |
|             | username | varchar(32) | No   | true  |         |       |
|             | pv       | bigint(20)  | No   | false | 0       | SUM   |
|             | uv       | bigint(20)  | No   | false | 0       | SUM   |
|             |          |             |      |       |         |       |
| rollup_city | citycode | smallint(6) | No   | true  | N/A     |       |
|             | pv       | bigint(20)  | No   | false | 0       | SUM   |
+-------------+----------+-------------+------+-------+---------+-------+
8 rows in set (0.01 sec)
```

You can use the following command to cancel the currently executing job: `CANCEL ALTER TABLE COLUMN FROM table1`

After the Rollup is created, queries do not need to specify the Rollup to be used. The system will automatically determine whether to use the Rollup. You can check whether the Rollup is being used by executing the command `EXPLAIN your_sql;`.

For more help, see `HELP ALTER TABLE`.

Query Data Tables

Memory Limit

To prevent a user's query from potentially consuming too much memory, Memory limits are applied to query. By default, a single query task will use no more than 2 GB of memory on any BE node.

If users encounter a `Memory limit exceeded` error, it usually means the memory limit has been exceeded. When the memory exceeds the limit, users should try to resolve it by optimizing their SQL statements. If the 2GB memory is insufficient, you can manually set memory parameters.

Display query memory limits:

```
mysql> SHOW VARIABLES LIKE "%mem_limit%";
+---------------+-------------+
| Variable_name | Value       |
+---------------+-------------+
| exec_mem_limit| 2147483648  |
+---------------+-------------+
1 row in set (0.00 sec)
```

The unit of `exec_mem_limit` is byte. You can use the `SET` command to change the value of `exec_mem_limit`. For example, set it to 8GB.

```
SET exec_mem_limit = 8589934592;
```

```
mysql> SHOW VARIABLES LIKE "%mem_limit%";
+---------------+-------------+
| Variable_name | Value       |
+---------------+-------------+
| exec_mem_limit| 8589934592 |
+---------------+-------------+
1 row in set (0.00 sec)
```

Note

The above modification is at the session level and is only effective within the current connection session. Disconnecting and reconnecting will revert it to the default value.
If you need to modify the global variable, you can set it as follows: `SET GLOBAL exec_mem_limit = 8589934592;` . After the settings are completed, disconnect the session and log in again for the parameter to take effect permanently.

Query Timeout

The current default query timeout is set to a maximum of 300 seconds. If a query is not complete within 300 seconds, the Doris system will cancel the query. Users can customize the timeout for their applications using this parameter to implement a blocking method similar to wait(timeout).
View Current Timeout Setting:

```
mysql> SHOW VARIABLES LIKE "%query_timeout%";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| QUERY_TIMEOUT | 300   |
+---------------+-------+
1 row in set (0.00 sec)
```

Change the timeout to 1 minute:

`SET query_timeout = 60;`

Note

The current timeout check interval is 5 seconds, so timeouts shorter than 5 seconds may not be very accurate.
The above modification is also at the session level. You can use `SET GLOBAL` to apply the change globally.

Broadcast/Shuffle Join

The system's default method for implementing a Join is to filter the small table based on conditions, broadcast it to each node containing the large table, create an in-memory Hash table, and then stream the large table's data for a Hash Join. But if the filtered data from the small table cannot fit into memory, the Join operation will fail. This typically results in a memory limit exceeded error.
If you encounter this situation, it is recommended to explicitly specify a Shuffle Join, also known as a Partitioned Join. This involves executing Hash both the small table and the large table based on the Join key, and then performing a distributed Join. This approach distributes the memory consumption across all computing nodes in the cluster.
Doris will automatically attempt a Broadcast Join. If the estimated size of the small table is too large, it will automatically switch to a Shuffle Join. Note that if Broadcast Join is explicitly specified under these conditions, Doris will still automatically switch to Shuffle Join.

```
mysql> select sum(table1.pv) from table1 join table2 where table1.siteid = 2;
+--------------------+
| sum(table1.pv) |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.20 sec)
```

Using Broadcast Join (Explicitly Specified):

```
mysql> select sum(table1.pv) from table1 join [broadcast] table2 where table1.sitei
+--------------------+
| sum(table1.pv) |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.20 sec)
```

Using Shuffle Join:

```
mysql> select sum(table1.pv) from table1 join [shuffle] table2 where table1.siteid
+--------------------+
| sum(table1.pv)  |
+--------------------+
|                 10 |
+--------------------+
1 row in set (0.15 sec)
```

Query Retries and High Availability

When multiple FE nodes are deployed, users can implement high availability for Doris by deploying a Cloud Load Balancer over the FE nodes.

Here are some high availability solutions:

First Solution

Implement retries and use a Cloud Load Balancer in your application layer code. For example, if a connection fails, it will automatically retry on another connection. Application layer code retries require the application to configure multiple Doris front-end node addresses.

Second Solution

If you use the MySQL JDBC connector to connect to Doris, you can use the JDBC automatic retry mechanism:

```
jdbc:mysql://[host:port],[host:port].../[database][?propertyName1][=propertyValue1]
```

Third Solution

Applications can connect to a MySQL Proxy deployed on the same machine as the application, achieving the desired result by configuring the Failover and Load Balance features of the MySQL Proxy.

`http://dev.mysql.com/doc/refman/5.6/en/mysql-proxy-using.html`

Resource Specification Selection and Optimization Suggestions

Last updated：2024-08-02 12:44:13

This document will introduce how to choose the instance specifications of Tencent Cloud TCHouse-D and provide optimization suggestions when resources are insufficient.

Note:

For different types of businesses, it is recommended to configure resource isolation policies or split clusters, for example, one cluster for real-time report business, and one cluster for real-time risk control business.

When a business supports multiple ToB tenants simultaneously, it is recommended to isolate resources or split clusters based on the actual situation to reduce mutual interference, for example, providing SaaS services for 200 tenants simultaneously, spliting into 4 clusters, and each supporting 50 tenants.

Resource Specifications and Adaptation Scenes

When purchasing a Tencent Cloud TCHouse-D cluster, you need to select the computing resource specifications and storage resource specifications of the FE node and BE node, and choose whether to enable high availability.

Resource Specifications and Recommended Scenes

Model Type	Compute Node Specifications	Recommended Storage Type	Recommended Scenes
Standard	4-core 16 GB	High Performance Cloud Disk SSD Enhanced SSD Cloud Disk	Limited to **POC feature testing or personal learning use**, mainly used to experience and test product capabilities
	8-core 32 GB	High Performance Cloud Disk SSD Enhanced SSD Cloud Disk	Recommended for the **test environment**, supporting medium data scale and rather complex data analysis
	16-core 64 GB	High Performance Cloud Disk SSD Cloud Disk Enhanced SSD Cloud Disk	Recommended for the **production environment**, supporting data analysis of larger scale and more complex scenes, as well as high concurrency scenes

	32 cores and above	High Performance Cloud Disk SSD Enhanced SSD Cloud Disk	Recommended for the production environment, supporting large-scale, highly complex data analysis, high concurrency, and other scenes

High Availability and Node Quantity Suggestions

Scene	High Availability Selection	Recommended Minimum Number of FE Nodes	Recommended Minimum Number of BE Nodes
POC feature testing	Non-high availability	1	3
Production scenario (query high availability)	Read high availability	3 FE nodes at least	3 BE nodes at least, on-demand scaling
Production scenario (query-write high availability)	Read-write high availability	5 FE nodes at least	3 BE nodes at least, on-demand scaling
Cross-AZ high availability scenario	Read-write high availability + 3 AZ deployment	5 FE nodes at least	3 BE nodes at least, scaling in increments of 3

Examples of Resource Specification Selection

Note:

The following content is for reference only. The performance may vary greatly in different business scenes.

1. **Scene 1: Product feature verification and simple data analysis**

FE: High availability not enabled, single node, 4-core 16 GB

BE: 3 nodes, 4-core 16 GB per node

2. **Scene 2: Simple query of small- to medium-sized data, such as hundreds of GB of data, less than 1,000 QPS**

FE: High availability not enabled, single node, 8-core 32 GB.

BE: 3 nodes, 8-core 32 GB per node

3. **Scene 3: Production scene, TB-level data volume, involving complex queries such as multi-table join and GROUP BY**

FE: High availability enabled, 3 nodes, 16-core 64 GB per node

BE: 3 nodes, 16-core 64 GB per node

4. **Scene 4: Production business, TB-level data volume, complex queries, and a large number of high-concurrency point queries**

FE: High availability enabled, 3 nodes, 16-core 64 GB per node

BE: 6 nodes, 16-core 64 GB per node

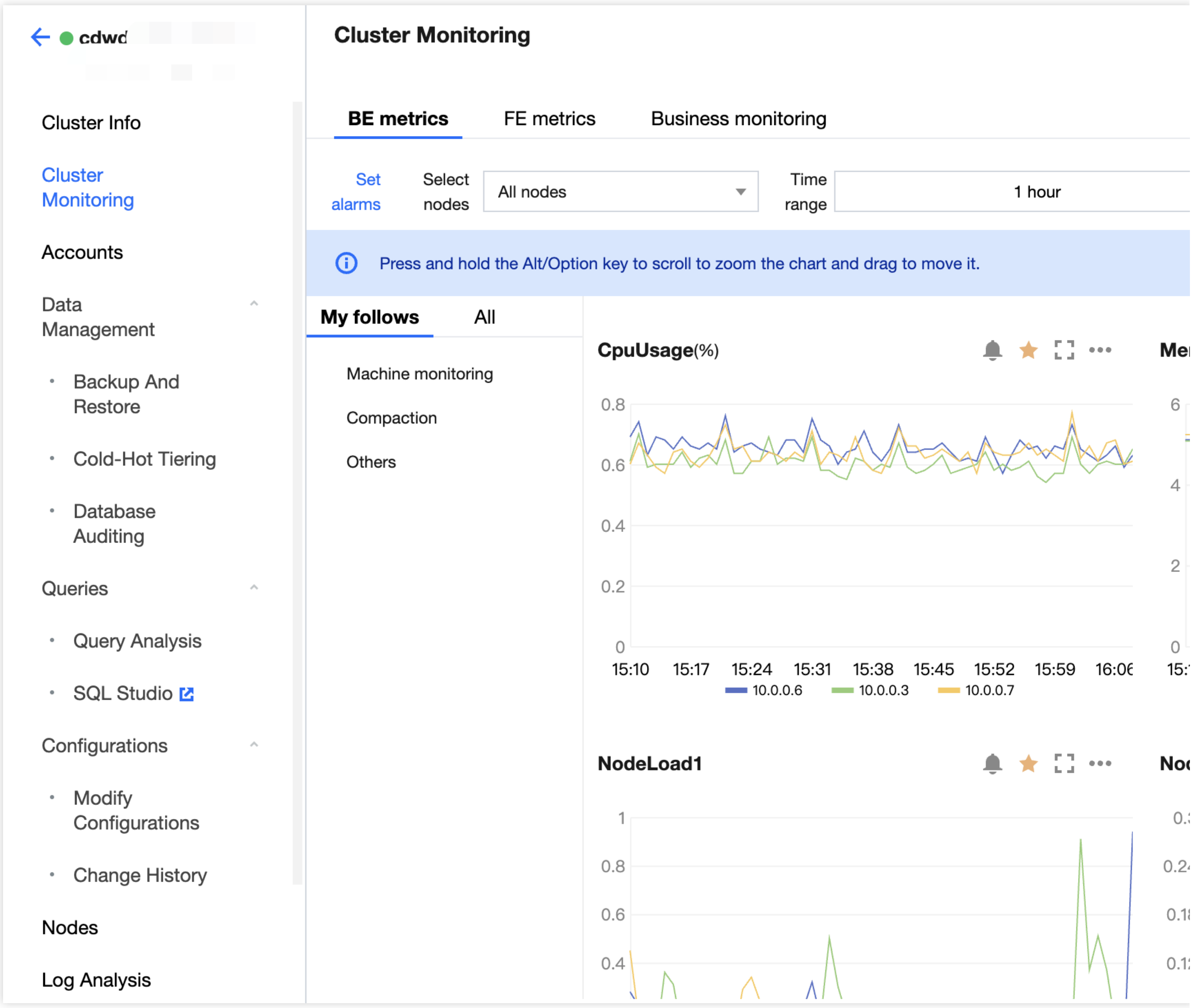
Resource Monitoring and Optimization Suggestions

Operations such as large-scale data import, data query, concurrent query, and multi-taweweweqweqweqweqeble join will cause a large amount of CPU and memory usage. If the CPU/memory utilization continues to exceed 85%, the cluster will become unstable. It is recommended to optimize the business or change the configuration.wewaeqeadwadasdwadasdwawdaw

Resource Usage Monitoring

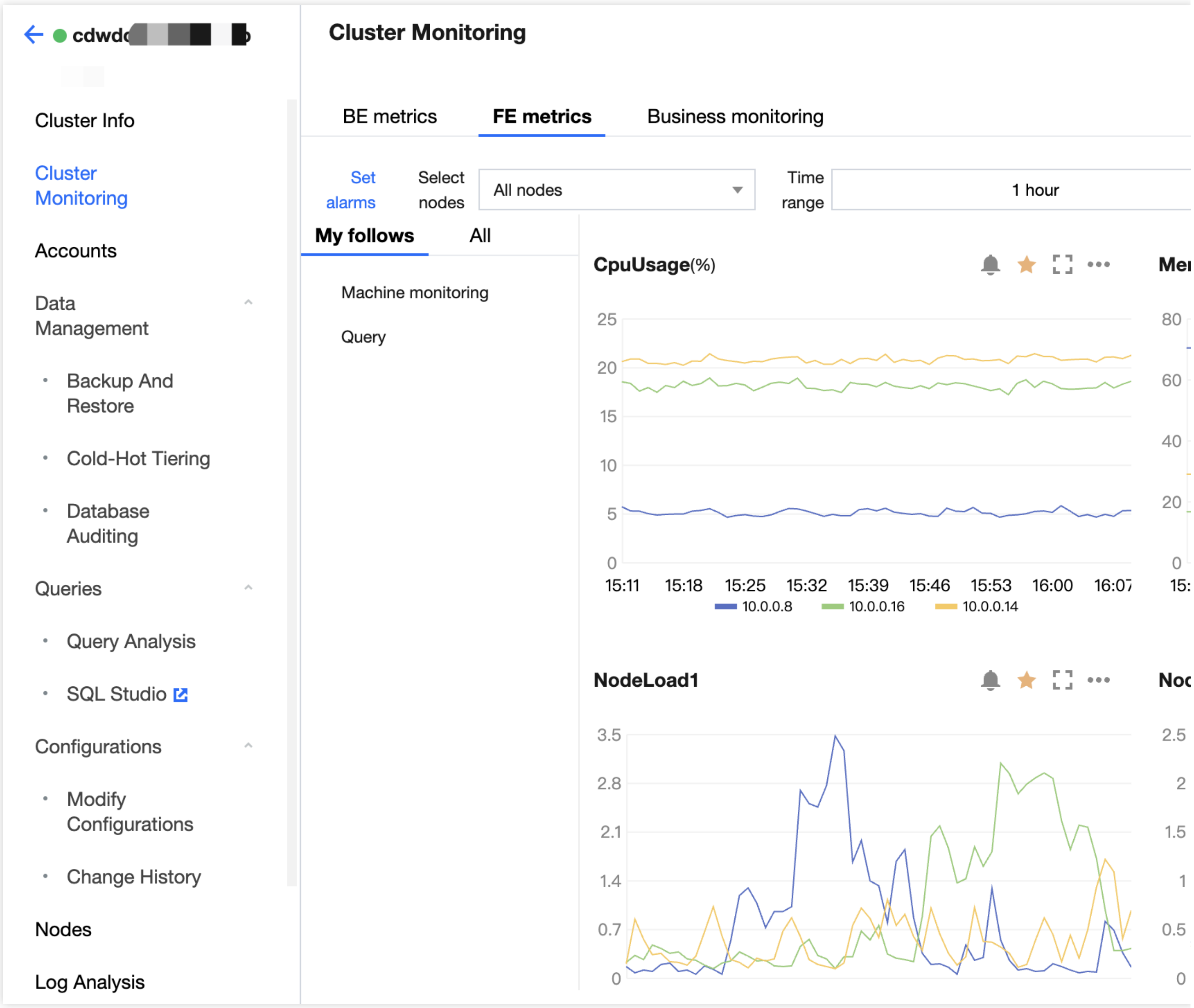
You can go to **Cluster Management> Cluster Monitoring** to check the CPU and memory usage of each BE and FE node, as shown in the following figure.

Cluster Monitoring > BE metrics



Press and hold the Alt/Option key to scroll to zoom the chart and drag to move it.

Cluster Monitoring > FE metrics



Resource Scale-out Suggestions

When the CPU and memory usage of FE and BE exceeds 85% continuously, you need to consider upgrade or scale out resources.

Note:

The main reasons for the high CPU and memory usage of FE and BE are as follows:

High usage of FE CPU: Multiple concurrent queries and a large number of complex queries.

High usage of FE memory: Too much metadata (unreasonable partitioning) and frequent table deletion.

High usage of BE CPU: Large amounts of data imported and large amounts of complex queries (such as aggregate queries).

High usage of BE memory: Large amounts of data imported and large amounts of complex queries (such as aggregate queries).

Common Scenes	Resource Consumption Performance	Optimization Suggestions for the Usage Continuously Exceeding 85%

Too much data continuously imported	The CPU and memory of FE and BE will be highly used.	If the bottleneck is FE: Vertical upgrade is recommended. If the bottleneck is BE: Vertical upgrade is recommended.
Frequent point checks/high concurrency	The CPU of FE and BE will be highly used.	If the bottleneck is FE: Vertical upgrade is recommended. If the bottleneck is BE: Vertical upgrade is recommended.
Frequent metadata changes and deletions	The memory of FE will be highly used.	It is recommended to upgrade FE vertically and increase memory.
Many multi-table join/aggregation queries	The CPU and memory of BE will be highly used.	It is recommended to horizontally scale out BE. Vertical upgrade is also an option.
Data multi-concurrency writing	The CPU and memory of BE will be highly used.	It is recommended to horizontally scale out BE. Vertical upgrade is also an option.

Cluster Scaling Must-Knows

Operation Type	Must-Knows
Scale-out	During the horizontal scale-out process, system reading and writing are still possible, but there may be some jitters. The operation takes about 5 to 15 minutes. Choose to perform it during non-business peak hours. When both the amount of data storage and the amount of queries increase relatively, horizontal scale-out is the preferred option.
Scale-in	Only one type of nodes can be selected for scale-in operation at a time, such as FE scale-in only or BE scale-in only. FE scale-in: Multiple FE nodes can be scaled in at one time. BE scale-in: Scaling in multiple BE nodes at one time may result in data loss or be time-consuming. It is recommended to scale the node in one by one. During the scale-in process, system reading and writing are still possible, but there may be some jitters.
Vertical upgrade/downgrade	The scale up/down system cannot be read or written. Computing specifications can be upgraded or downgraded; storage specifications can only be upgraded. The results of specification adjustment are effective for all nodes in a cluster.

Business Optimization Suggestions

Optimization Type	Optimization Instructions
Usage recommendations	If you often perform point queries on a column and the column has a high cardinality, it is recommended to create a Bloom filter index on this column. If you often perform fixed-mode aggregate queries on a table, it is recommended to create a materialized view on this table. It is recommended to divide partitions and buckets reasonably according to business scenes to avoid excessive FE memory usage due to too many partitions and buckets. For SQL queries of general data exploration, if not all data is needed, it is recommended to add a limit number for the records returned, which can also speed up the query. It is recommended to use CSV for data import and avoid JSON format.
Try-to-avoid	Avoid select * queries. Avoid enabling profiles globally (This will result in more resource consumption. It is recommended to enable profiles for the demanding SQL statements). When creating a table: Avoid enabling merge_on_write (this feature is not yet mature). When creating a table: Avoid enabling auto bucket (this feature is not yet mature). When creating a table: Avoid opening a dynamic Schema table (this feature is not yet mature). Avoid the join of multiple large tables. When multiple large tables are joined: Every two large tables can be joined through Colocation Join. Or use pre-aggregate tables and indexes to speed up queries.
Parameter optimization	When a SQL statement involves multiple concurrent operations, it is recommended to increase the `parallel_fragment_exec_instance_num` parameter. The default value of this parameter is 200. It can be increased by multiples (such as 400 and 800). It is recommended to control it within 2,000. It is recommended to control the compaction speed. If the monitoring metric `base_compaction_score` exceeds 200 and continues to rise (for details, see the Cluster Monitoring-BE Indicators-BE page), you can increase the compaction_task_num_per_disk parameter configuration (the system default is 2, which can be increased to 4 or greater).

Naming Specifications and Limits to the Database and Data Table

Last updated：2024-07-31 09:17:12

This document will introduce the naming specifications and some limitations for creating and changing databases and tables in Tencent Cloud TCHouse-D.

Naming Specifications

Database

Name: The table name must start with a letter or an underscore. It can contain letters, numbers, and underscores, with a length of 1 to 64 characters.
Description: It can contain up to 2,048 characters.
Limitations: The same data link cannot have identical database names.

Data Table/View

Name: The table name must start with a letter or an underscore. It can contain letters, numbers, and underscores, with a length of 1 to 64 characters.
Description: It can contain up to 2,048 characters.
Limitations: The same database cannot have identical data table names.

Attribute Column

Name: The column name must start with a letter or an underscore. It can contain letters, numbers, and underscores, with a length of 1 to 256 characters.
Description: It can contain up to 256 characters.
Limitations: The same data table cannot have identical data column names.

Partition

Partition feld name: Length should be 1 to 256 characters.
Usage recommendations:

1. When the single table data volume is below 200 million records, for convenience, you can choose not to set partitions and use bucketing instead;
2. When the partition field has a continuous data range (such as date and ID), it is recommended to choose Range partition;

3. When the data has discrete values (such as country, region, and status), it is recommended to choose LIST partition.

Bucketing

Bucket field name: Length should be 1 to 256 characters

Usage recommendations:

1. The bucket key can be one or multiple; multiple keys ensure a more balanced data distribution, while a single key is easier to match.
2. Selection of bucket columns: generally choose fields with high differentiation/cardinality and even hash to avoid data skew; they should also be frequently used fields to improve query efficiency;
3. The number of buckets created should not be too many or too few; it is recommended to keep each bucket between 1-10 GB.

Use Limits

Quantity Limit of Database/Table/Column/Partition

Restriction Item	Quantity Limit
Number of catalogs that can be created	20
Number of databases per cluster	10,000
Number of databases per user	1,000
Number of data tables per cluster	100,000
Number of data tables per user	10,000
Number of data tables per database	4,096
Number of fields per table	2,048
Number of partitions per table	10,000
Number of views per cluster	10,000
Maximum view nesting level	10
Number of materialized views per cluster	10,000

Query Limits

Restriction Item	Explanation	Supplement
Daily data query volume	No limit	-
Number of tables in a single SQL query	32	-
Single SQL query timeout	900 seconds	-
Maximum number of files per external table	No limit	-

DML/DML Limits

Restriction Item	Explanation	Supplement
Simultaneous schema changes per cluster	20	-
Simultaneous schema changes per table	1	-
Data write concurrency per table (per second)	10	High-frequency writes are not recommended; it is advised to batch write appropriately: the total number of imports per table should not exceed 20 times per minute.
Data write concurrency for the same key per table (per second)	1	The same key in one table cannot be updated simultaneously; otherwise, it may cause data inaccuracy.

Table Design and Data Import

Last updated：2024-07-31 09:17:29

How to choose a data model?

For data that is already aggregated, either the Aggregate or Duplicate model can be used.
For detail data that has already been cleaned, if you need to perform aggregation for querying statistical values, choose the Aggregate model.
For detail data that has already been cleaned, if you need to perform detailed queries, choose the Duplicate model (which does not include pre-aggregation, resulting in lower aggregation query efficiency. You can enhance aggregation query performance through materialized views).
If there is **a unique Key** and you need to deduplicate by Key, choose the **Unique Model**.

How to set up partitions?

Choose partition fields with values that are as evenly distributed as possible. It is recommended to use fields such as dates or IDs for partitioning.

How to set up replicas?

The number of replicas can be set to 3 (it can also be set to 2, but setting it to 1 is not recommended as it may lead to data unavailability during subsequent rolling upgrades).

How to set up buckets?

Use Keys with a uniform Hash distribution as bucket Keys to avoid data skewing.
The number of buckets should not be too many or too few. It is recommended to keep each bucket between 1 and 10 GB in size. For small tables, a few buckets are usually sufficient.
The bucket Key can be one or multiple keys. Using multiple keys ensures more balanced data distribution, while a single key is easier to match (a single bucket key should generally be a Key with high cardinality).

How to choose field formats?

For each field in the table, prioritize using **Integer** types instead of string types. This greatly improves query and version merge efficiency.
Use decimal for floating-point numbers instead of double or float.

Must-Knows for Data Import

For real-time imports, it is recommended to use Stream Load, and for offline imports, Broker Load is preferred. The basic principle of import is batch loading: **Reduce concurrency and try to import as much data as possible in a single instance** to minimize merge costs and avoid impacting read efficiency. (For example, the total number of

imports per minute should not exceed 20 times. Considering various types of concurrency, high-frequency imports are currently not suitable). **Too many small files can severely affect query efficiency**.
When importing data, pay special attention to **filtering NULL values** for fields used as Hash keys to avoid data skew.
Too many small files can severely affect query efficiency.
When data is imported, make sure to **specify the partitions to be imported**. Otherwise, importing data into large tables may easily fail.
It is recommended to use CSV for data import and avoid JSON format.

Must-Knows for Schema Changes

To ensure cluster stability and meet business requirements, we recommend thoroughly evaluating field types before creating a table. When the table Schema needs to be modified, we only recommend the **Add Column** operation. We guarantee that new columns will be added as quickly as possible.

Must-Knows for Data Clearing

If you need to clear partition data, it is recommended to prioritize the **truncate** operation (which is equivalent to dropping and then creating the partition) instead of the delete operation. The delete operation significantly impacts query performance.

Other

For unfamiliar operations, it is best to type help keyword in the command line for guidance (e.g., help stream load to understand how to perform real-time data import). Alternatively, you can Submit Ticket to get more detailed assistance with your issue.

Query Optimization

Last updated：2024-07-31 09:17:42

Basic Query Optimization

When querying a partition table, be sure to include the **partition field**.Explain + SQL can help users analyze and query data in several partitions and tablets.
It is best if the query SQL condition can hit the partition Key and bucket Key.
It is best if the query SQL condition can hit the prefix index.
Since Doris is a column-based database, when there are enough fields to query, the performance may be worse than row-based storage. It is recommended to select specific fields instead of * as much as possible when querying, and add a **limit number** at the end of the query.
When performing a Select operation, **avoid writing function(column) = "xxxx" as much as possible;** otherwise, the strength of Doris system in pushing down predicates cannot be exerted. The left side shall be the column name and the right side shall be a constant value that can be calculated and flattened.
Avoid using "or, union all" in queries as much as possible. In most scenes, consider using **in instead of "or".** .
For SQL queries of general data exploration, if not all data is needed, it is recommended to add a limit number for the records returned, which can also speed up the query.

Join Optimization

Shuffle mode optimization: Efficiency is **Colocate join** > **Bucket Shuffle** > **Shuffle** > **BroadCast**. For details, see Bucket Shuffle Join.
RuntimeFilter : In a join query, in addition to the join condition, there are other filtering conditions on the right.

Use Rollup.

The query cannot cover the prefix index of the base table. The prefix index is formed by adjusting the Key order through Rollup.
Perform Key filtering aggregation on the Aggregate table.

Using Materialized View

If you often perform fixed-mode aggregate queries on a table, it is recommended to create a materialized view on this table.
It can be used in all scenes supported by Rollup.
An additional aggregation is formed for the Duplicate table.
For details, see Materialized View.

Index Optimization

Bitmap index: Select a column with a relatively small value cardinality [100-100,000], where the query condition hits the column.
BloomFilter index: If you often perform precise point queries on a column and the column has a high cardinality, it is recommended to create a Bloom filter index on this column.

Use cache.

PageCache: This configuration is enabled by default.
SqlCache: This configuration is disabled by default. The effect is better when the concurrency is high and the query result set is small.

Suggested Usage to Avoid

Last updated：2024-07-31 09:17:57

Suggested Scenes to Avoid

Avoid large-scale periodic scheduling of offline/batch ETL jobs (insert into select / create table as select) in production clusters, particularly when running both offline and online businesses within the same cluster. Offline jobs can consume significant resources, impacting the stability and performance of online businesses.

Note:

It is recommended to isolate offline and online business on different clusters, or to complete offline processing with Spark first, followed by writing the data to Doris.

Avoid executing insert into one by one: Each insert into in Doris is a transaction, and inserting data row by row can cause concurrency to exceed the upper limit of transactions.

Note:

It is recommended to batch the data, such as executing insert into dozens or hundreds of rows at a time, to reduce write pressure.

1.2 Kernel Version: Try to avoid using complex data types (e.g., MAP, ARRAY, STRUCT).

1.2 Kernel Version: Support for complex data types is not fully developed, and some write and query operations might cause errors.

Suggested Queries to Avoid

Try to avoid using select * queries on tables with many columns and large amounts of data.

Avoid enabling the profile globally (this can result in significant resource overhead, so it is recommended to enable the profile only for specific SQL statements that need it).

Try to avoid joining multiple large tables.

Note:

To deal with multiple large table joins, it is recommended to join large tables in pairs using Colocation Join, or to use pre-aggregated tables, indexes, etc., to speed up queries.

Suggested Features to Avoid

1.2 Kernel Version: Avoid enabling merge_on_write (this feature is not yet fully developed).

1.2 Kernel Version: Avoid enabling Light scheme change (this feature is not yet fully developed).