Document Database Service

Best Practices

Issue	01
Date	2022-08-30

HUAWEI TECHNOLOGIES CO., LTD.

Contents

1 Overview

This document provides best practices for Huawei Cloud Document Database Service (DDS) and guides you through using DDS to best suit your business needs.

Service	Reference	Overview
Document Database Service	**Common Methods for Connecting to a DDS Instance**	This section describes common DDS connection methods.
	How Do Replica Sets Achieve High Availability and Read/Write Splitting?	This section describes how to connect to a replica set instance to achieve high availability.
	Sharding	This section describes how to set cluster shards to improve database performance.
	How Do I Improve DDS Performance by Optimizing SQL Statements?	This section describes DDS usage suggestions.
	How Do I Prevent the Mongos Cache Problem?	This section describes how to avoid the mongos route cache defect of the cluster.
	How Do I Solve the High CPU Usage Issue?	This section describes how to troubleshoot high CPU usage.
	Creating a User and Granting the Read-Only Permission to the User	This section describes how to use IAM to grant read-only permissions to DDS.

2 Common Methods for Connecting to a DDS Instance

This section describes how to connect to a DDS instance using the following three methods:

- Mongo Shell
- Python Mongo
- Java Mongo

Mongo Shell

- Prerequisites
  a. To connect an ECS to a DDS instance, run the following command to connect to the IP address and port of the instance server to test the network connectivity.

     curl *ip:port*

     If the message **It looks like you are trying to access MongoDB over HTTP on the native driver port** is displayed, the ECS and DDS instance can communicate with each other.

  b. Download the client installation package whose version is the same as the instance version from the **MongoDB official website**. Decompress the package, obtain the **mongo** file, and upload it to the ECS.

  c. If SSL is enabled, download the root certificate and upload it to the ECS.

- Connection commands
  – SSL is enabled.

    ./mongo *ip:port* **--authenticationDatabase admin -u** *username* **-p** *password* **--ssl --sslCAFile $***path to certificate authority file* **--sslAllowInvalidHostnames**

  – SSL is disabled.

    ./mongo *ip:port* **--authenticationDatabase admin -u** *username* **-p** *password*

Table 2-1 Parameter description

Parameter	**Description**
ip	If you access an instance from an ECS, *ip* is the private IP address of the instance.
	If you access an instance from a device over a public network, *ip* is the EIP bound to the instance,
port	Database port displayed on the **Basic Information** page. Default value: **8635**
username	Current username
password	Password of the current username
path to certificate authority file	Path of the SSL certificate

- Precautions
  a. If SSL is enabled, the connection command must contain **--ssl** and **--sslCAFile**.
  b. **--authenticationDatabase** must be set to **admin**. If you log in to the database as user **rwuser**, switch to **admin** for authentication.

For details, see **Connecting to an Instance** in *Getting Started with Document Database Service*.

Python Mongo

- Prerequisites
  a. To connect an ECS to a DDS instance, run the following command to connect to the IP address and port of the instance server to test the network connectivity.
     curl *ip*:*port*
     If the message **It looks like you are trying to access MongoDB over HTTP on the native driver port** is displayed, the network connectivity is normal.
  b. Install Python and third-party installation package **pymongo** on the ECS. Pymongo 2.8 is recommended.
  c. If SSL is enabled, download the root certificate and upload it to the ECS.
- Input the connection code.
  – SSL is enabled.

```
import ssl
from pymongo import MongoClient
conn_urls="mongodb://rwuser:rwuserpassword@ip:port/{mydb}?authSource=admin"
connection = MongoClient(conn_urls,connectTimeoutMS=5000,ssl=True,
ssl_cert_reqs=ssl.CERT_REQUIRED,ssl_match_hostname=False,ssl_ca_certs=${path to
certificate authority file})
dbs = connection.database_names()
print "connect database success! database names is %s" % dbs
```

– SSL is disabled.

```
import ssl
from pymongo import MongoClient
conn_urls="mongodb://rwuser:rwuserpassword@ip:port/{mydb}?authSource=admin"
connection = MongoClient(conn_urls,connectTimeoutMS=5000)
dbs = connection.database_names()
print "connect database success! database names is %s" % dbs
```

- Precautions
  a. *{mydb}* is the name of the database to be connected.
  b. The authentication database in the URL must be **admin**. Set **authSource** to **admin**.

Java Mongo

- Prerequisites
  a. To connect an ECS to a DDS instance, run the following command to connect to the IP address and port of the instance server to test the network connectivity.

     curl *ip:port*

     If the message **It looks like you are trying to access MongoDB over HTTP on the native driver port** is displayed, the ECS and DDS instance can communicate with each other.
  b. Download the **MongoDB JAR** package compatible with the instance version by referring to the **MongoDB Compatibility** table.
  c. JDK is installed on the ECS.
  d. If SSL is enabled, download the root certificate and upload it to the ECS.
- Input the connection code.

  Use keytool to generate a trustStore.

  keytool -import -file */var/chroot/mongodb/CA/ca.crt* **-keystore** */home/Mike/jdk1.8.0_112/jre/lib/security/mongostore* **-storetype pkcs12 -storepass** ******

  NOTE

  - **/var/chroot/mongodb/CA/ca.crt** is the root certificate path.
  - **/home/Mike/jdk1.8.0_112/jre/lib/security/mongostore** indicates the path of the generated truststore.
  - ******** is the password of the trustStore.

– SSL is enabled.

```
import java.util.ArrayList;
import java.util.List;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import com.mongodb.MongoClientURI;
import com.mongodb.MongoClientOptions;
public class MongoDBJDBC {
public static void main(String[] args){
    try {
            System.setProperty("javax.net.ssl.trustStore", "/home/Mike/
jdk1.8.0_112/jre/lib/security/mongostore");
```

```
            System.setProperty("javax.net.ssl.trustStorePassword", "****");
            ServerAddress serverAddress = new ServerAddress("ip", port);
            List addrs = new ArrayList();
            addrs.add(serverAddress);
            MongoCredential credential =
MongoCredential.createScramSha1Credential("rwuser", "admin", "!
rwuserPassword".toCharArray());
            List credentials = new ArrayList();
            credentials.add(credential);
            MongoClientOptions opts= MongoClientOptions.builder()
            .sslEnabled(true)
            .sslInvalidHostNameAllowed(true)
            .build();
            MongoClient mongoClient = new MongoClient(addrs,credentials,opts);
            MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb");
            MongoCollection collection =
mongoDatabase.getCollection("testCollection");
            Document document = new Document("title", "MongoDB").
            append("description", "database").
            append("likes", 100).
            append("by", "Fly");
            List documents = new ArrayList();
            documents.add(document);
            collection.insertMany(documents);
            System.out.println("Connect to database successfully");
            } catch (Exception e) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
        }
      }
}
```

Sample codes:

```
javac -cp .:mongo-java-driver-3.2.0.jar MongoDBJDBC.java
java -cp .:mongo-java-driver-3.2.0.jar MongoDBJDBC
```

– SSL is disabled.

```
import java.util.ArrayList;
import java.util.List;
import org.bson.Document;
import com.mongodb.MongoClient;
import com.mongodb.MongoCredential;
import com.mongodb.ServerAddress;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.MongoCollection;
import com.mongodb.MongoClientURI;
import com.mongodb.MongoClientOptions;
public class MongoDBJDBC {
public static void main(String[] args){
      try {
            ServerAddress serverAddress = new ServerAddress("ip", port);
            List addrs = new ArrayList();
            addrs.add(serverAddress);
            MongoCredential credential =
MongoCredential.createScramSha1Credential("rwuser", "admin", "!
rwuserPassword".toCharArray());
            List credentials = new ArrayList();
            credentials.add(credential);
            MongoClient mongoClient = new MongoClient(addrs,credentials);
            MongoDatabase mongoDatabase = mongoClient.getDatabase("testdb");
            MongoCollection collection =
mongoDatabase.getCollection("testCollection");
            Document document = new Document("title", "MongoDB").
            append("description", "database").
            append("likes", 100).
            append("by", "Fly");
            List documents = new ArrayList();
            documents.add(document);
            collection.insertMany(documents);
            System.out.println("Connect to database successfully");
            } catch (Exception e) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
        }
       }
}
```

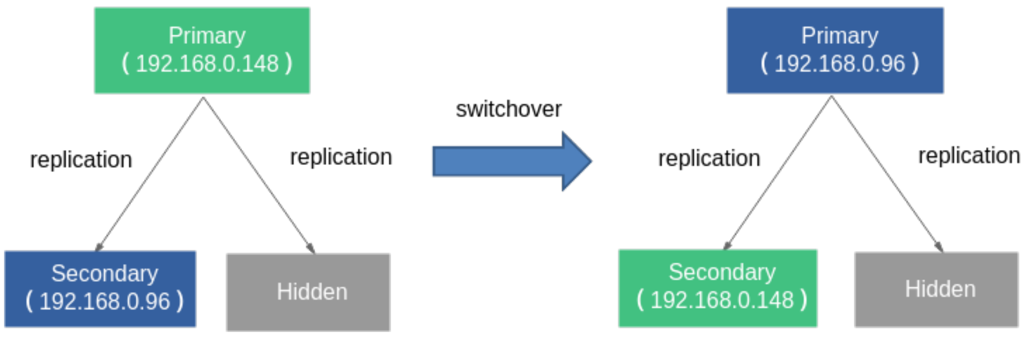
- Precautions
  a. In SSL mode, you need to manually generate the trustStore file.
  b. Change the authentication database to **admin**, and then switch to the service database after authentication.

3 How Do Replica Sets Achieve High Availability and Read/Write Splitting?

DDS replica set instances can store multiple duplicates to ensure data high availability and support the automatic switch of private IP addresses to ensure service high availability. To enhance the read and write performance of your client for connecting to the instance, you can use your client to read different data copies. You are advised to use the recommended method to connect replica set instances. Otherwise, the high availability and high read performance of replica set instances cannot be guaranteed.

The primary node of a replica set instance is not fixed. If the instance settings are changed, or the primary node fails, or primary and secondary nodes are switched, a new primary node will be elected and the previous one becomes a secondary node. The following figure shows the process of a switchover.

Figure 3-1 Primary/Secondary switchover



Connecting to a Replica Set Instance (HA)

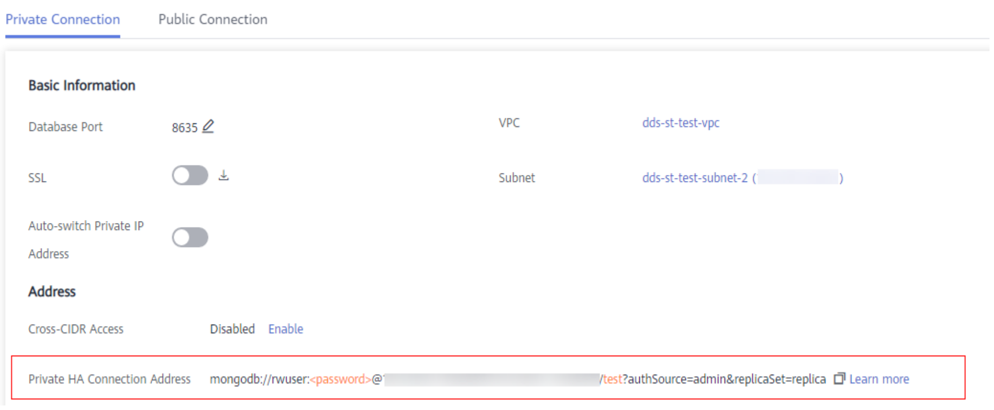
A DDS replica set consists of the primary, secondary, and hidden nodes. The hidden node is invisible to users. Read/Write splitting and HA can be realized only when you connect to the IP addresses and ports of the primary and secondary nodes of the replica set at the same time (in HA mode).

The following describes how to use URL and Java to connect to an instance in HA mode.

Method 1: Using a URL

On the **Instances** page, click the instance name. The **Basic Information** page is displayed. Choose **Connections**. Click the **Private Connection** tab and obtain the connection address of the current instance from the **Private HA Connection Address** field.

Figure 3-2 Obtaining the private HA connection address



Private Connection | Public Connection

Basic Information

Database Port	8635	VPC	dds-st-test-vpc
SSL		Subnet	dds-st-test-subnet-2 ([redacted])
Auto-switch Private IP Address			

Address

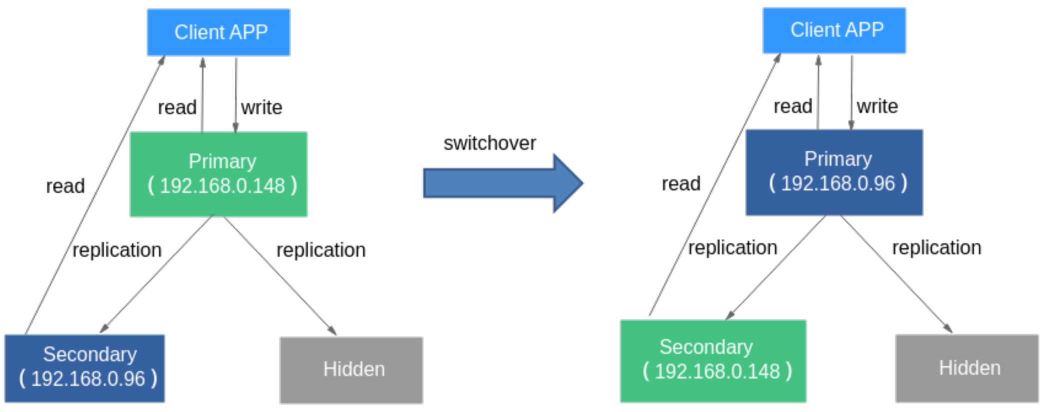
Cross-CIDR Access Disabled Enable

Private HA Connection Address mongodb://rwuser:<password>@[redacted]/test?authSource=admin&replicaSet=replica Learn more

Example: **mongodb://rwuser:****@***192.168.0.148:8635,192.168.0.96:8635***/test?authSource=admin&replicaSet=replica**

In the preceding URL, **192.168.0.148:8635** and **192.168.0.96:8635** are the IP addresses and ports of the primary and secondary nodes, respectively. If you use this address, the connection between your client and the instance can be ensured even when a primary/standby switchover occurs. In addition, using multiple IP addresses and port numbers can enhance the read and write performance of the entire database.

Figure 3-3 Data read and write process



Method 2: Using a Java Driver

Sample code:

```
MongoClientURI connectionString = new MongoClientURI("mongodb://
rwuser:****@192.168.0.148:8635,192.168.0.96:8635/test?authSource=admin&replicaSet=replica");
MongoClient client = new MongoClient(connectionString);
MongoDatabase database = client.getDatabase("test");
MongoCollection<Document> collection = database.getCollection("mycoll");
```

Table 3-1 Parameter description

Parameter	Description
rwuser:****	Username and password for starting authentication
192.168.0.148:8635, 192.168.0.96:8635	IP addresses and ports of the primary and secondary nodes in a replica set instance
test	Name of the database to be connected
authSource=admin	Database username for authentication
replicaSet=replica	Name of the replica set instance type

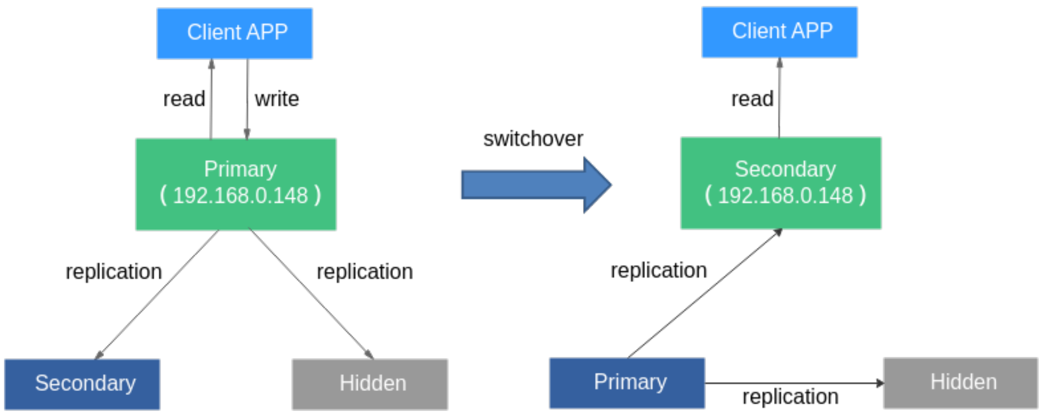
(Not Recommended) Connecting to a Replica Set Instance

Using the Connection Address

mongodb://rwuser:**@***192.168.0.148:8635***/test?authSource=admin&replicaSet=replica**

In the preceding URL, **192.168.0.148:8635** is the IP address and port number of the current primary node. If a switchover occurs or the primary node is changed, the client fails to connect to the replica set instance because the IP address and port of the newly elected primary node is unknown. As a result, the database service becomes unavailable. In addition, read and write operations can only be performed on a fixed primary node, so the read and write performance cannot be improved by adding nodes.

Figure 3-4 Data read and write process



Read/Write Splitting

For details about read/write splitting, see **MongoDB official documents**.

4 Sharding

You can shard a large-size collection for a sharded cluster instance. Sharding distributes data across different machines to make full use of the storage space and compute capability of each shard.

Number of Shards

The following is an example using database **mytable**, collection **mycoll**, and the field **name** as the shard key.

Step 1 Log in to a sharded cluster instance using Mongo Shell.

Step 2 Enable sharding for the databases that belong to the cluster instance.

- Method 1

```
sh.enableSharding("<database>")
```

  Example:

```
sh.enableSharding("mytable")
```

- Method 2

```
use admin
db.runCommand({enablesharding:"<database>"})
```

Step 3 Shard a collection.

- Method 1

```
sh.shardCollection("<database>.<collection>",{"<keyname>":<value> })
```

  Example:

```
sh.shardCollection("mytable.mycoll",{"name":"hashed"})
```

- Method 2

```
use admin
db.runCommand({shardcollection:"<database>.<collection>",key:{"keyname":<value> }})
```

Table 4-1 Parameter description

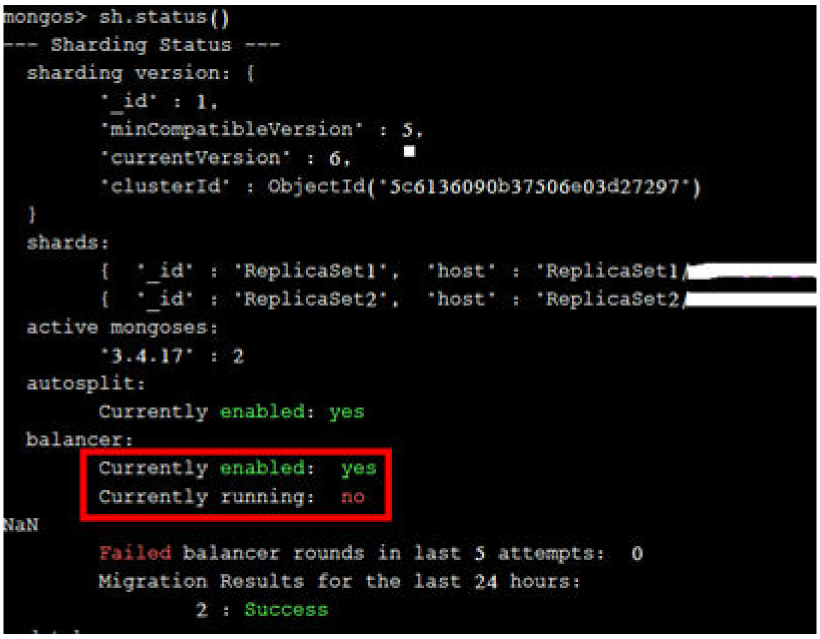
Parameter	Description
<database>	Database name
<collection>	Collection name

Parameter	Description
<keyname>	Shard key Cluster instances are sharded based on the value of this parameter. Select a proper shard key for the collection based on your service requirements. For details, see **Selecting a Shard Key**.
<value>	The sort order based on the range of the shard key. ● 1: Ascending indexes ● -1: Descending indexes ● hashed: indicates that hash sharding is used. Hashed sharding provides more even data distribution across the sharded cluster. For details, see **sh.shardCollection()**.

Step 4 Check the data storage status of the database on each shard.

```
sh.status()
```

Example:



```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        '_id' : 1,
        'minCompatibleVersion' : 5,
        'currentVersion' : 6,
        'clusterId' : ObjectId('5c6136090b37506e03d27297')
  }
  shards:
        {  '_id' : 'ReplicaSet1',  'host' : 'ReplicaSet1/[redacted]
        {  '_id' : 'ReplicaSet2',  'host' : 'ReplicaSet2/[redacted]
  active mongoses:
        '3.4.17' : 2
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
NaN
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                2 : Success
```

----End

Selecting a Shard Key

- **Background**

  Each sharded cluster contains collections as its basic unit. Data in the collection is partitioned by the shard key. Shard key is a field in the collection. It distributes data evenly across shards. If you do not select a proper shard key, the cluster performance may deteriorate, and the sharding statement execution process may be blocked.

Once the shard key is determined it cannot be changed. If no shard key is suitable for sharding, you need to use a sharding policy and migrate data to a new collection for sharding.

- **Characteristics of proper shard keys**
  - All inserts, updates, and deletes are evenly distributed to all shards in a cluster.
  - The distribution of keys is sufficient.
  - Rare scatter-gather queries.

  If the selected shard key does not have all the preceding features, the read and write scalability of the cluster is affected. For example, If the workload of the find() operation is unevenly distributed in the shards, hot shards will be generated. Similarly, if your write load (inserts, updates, and deletes) is not uniformly distributed across your shards, then you could end up with a hot shard. Therefore, you need to adjust the shard keys based on service requirements, such as read/write status, frequently queried data, and written data.

  After existing data is sharded, if the **filter** filed of the update request does not contain shard keys and **upsert:true** or **multi:false**, the update request will report an error and return message "An upsert on a sharded collection must contain the shard key and have the simple collation.".

- **Judgment criteria**

  You can use the dimensions provided in **Table 4-2** to determine whether the selected shard keys meet your service requirements:

Table 4-2 Reasonable shard keys

Identification Criteria	Description
Cardinality	Cardinality refers to the capability of dividing chunks. For example, if you need to record the student information of a school and use the age as a shard key, data of students of the same age will be stored in only one data segment, which may affect the performance and manageability of your clusters. A much better shard key would be the student number because it is unique. If the student number is used as a shard key, the relatively large cardinality can ensure the even distribution of data.
Write distribution	If a large number of write operations are performed in the same period of time, you want your write load to be evenly distributed over the shards in the cluster. If the data distribution policy is ranged sharding, a monotonically increasing shard key will guarantee that all inserts go into a single shard.
Read distribution	Similarly, if a large number of read operations are performed in the same period, you want your read load to be evenly distributed over the shards in a cluster to fully utilize the computing performance of each shard.

Identification Criteria	Description
Targeted read	The mongos query router can perform either a targeted query (query only one shard) or a scatter/gather query (query all of the shards). The only way for the mongos to be able to target a single shard is to have the shard key present in the query. Therefore, you need to pick a shard key that will be available for use in the common queries while the application is running. If you pick a synthetic shard key, and your application cannot use it during typical queries, all of your queries will become scatter/gather, thus limiting your ability to scale read load.

Choosing a Distribution Policy

A sharded cluster can store a collection's data on multiple shards. You can distribute data based on the shard keys of documents in the collection.

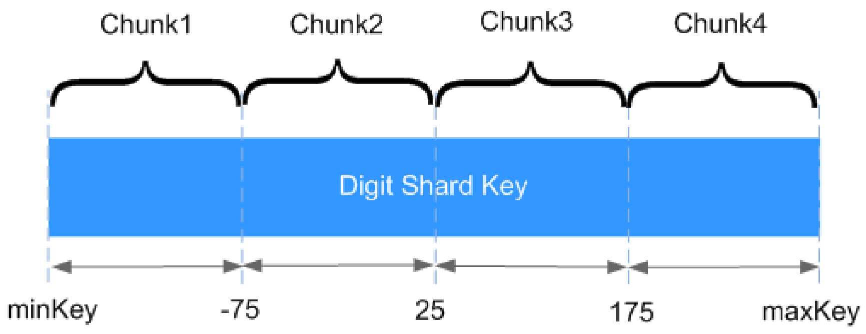
There are two data distribution policies: ranged sharding and hashed sharding. For details, see **Step 3**.

The following describes the advantages and disadvantages of the two methods.

- **Ranged sharding**

  Ranged-based sharding involves dividing data into contiguous ranges determined by the shard key values. If you assume that a shard key is a line stretched out from positive infinity and negative infinity, each value of the shard key is the mark on the line. You can also assume small and separate segments of a line and that each chunk contains data of a shard key within a certain range.

  Figure 4-1 Distribution of data

  As shown in the preceding figure, field **x** indicates the shard key of ranged sharding. The value range is [*minKey,maxKey*] and the value is an integer. The value range can be divided into multiple chunks, and each chunk (usually 64 MB) contains a small segment of data. For example, chunk 1 contains all documents in range [minKey, -75] and all data of each chunk is stored on the same shard. That means each shard containing multiple chunks. In addition, the data of each shard is stored on the config server and is evenly distributed by mongos based on the workload of each shard.

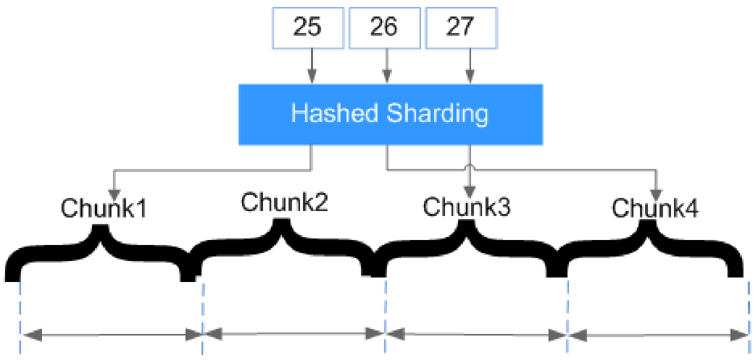
Ranged sharding can easily meet the requirements of query in a certain range. For example, if you need to query documents whose shard key is in range [-60,20], mongos only needs to forward the request to chunk 2.

However, if shard keys are in ascending or descending order, newly inserted documents are likely to be distributed to the same chunk, affecting the expansion of write capability. For example, if **_id** is used as a shard key, the high bits of **_id** automatically generated in the cluster are ascending.

- **Hashed sharding**

  Hashed sharding computes the hash value (64-bit integer) of a single field as the index value; this value is used as your shard key to partition data across your shared cluster. Hashed sharding provides more even data distribution across the sharded cluster because documents with similar shard keys may not be stored in the same chunk.

Figure 4-2 Distribution of data



Hashed sharding randomly distributes documents to each chunk, which fully expands the write capability and makes up for the deficiency of ranged sharding. However, queries in a certain range need to be distributed to all backend shards to obtain documents that meet conditions, resulting in low query efficiency.

5 How Do I Improve DDS Performance by Optimizing SQL Statements?

DDS is inherently a NoSQL database with high performance and strong extensibility. Similar to RDS, such as RDS for MySQL, Microsoft SQL Server, and Oracle, DDS performance may also be affected by database design, statement optimization, and index creation.

The following provides suggestions for improving DDS performance in different dimensions:

Creating Databases and Collections

1. Use short field names to save storage space. Different from an RDS database, each DDS document has its field names stored in the collection. Short name is recommended.
2. Limit the number of documents in a collection to avoid the impact on the query performance. Archive documents periodically if necessary.
3. Each document has a default **_id**. Do not change the value of this parameter.
4. Capped collections have a faster insertion speed than other collections and can automatically delete old data. You can create capped collections to improve performance based on your service requirements.

For details, see **Usage Suggestions** in the *Document Database Service Developer Guide*.

Query

Indexes

1. Create proper number of indexes for frequently queried fields based on service requirements. Indexes occupy some storage space, and the insert and indexing operations consume resources. It is recommended that the number of indexes in each collection should not exceed 5.

   If data query is slow due to lack of indexes, create proper indexes for frequently queried fields.
2. For a query that contains multiple shard keys, create a compound index that contains these keys. The order of shard keys in a compound index is

important. A compound index support queries that use the leftmost prefix of the index, and the query is only relevant to the creation sequence of indexes.

3. TTL indexes can be used to automatically filter out and delete expired documents. The index for creating TTL must be of type date. TTL indexes are single-field indexes.
4. You can create field indexes in a collection. However, if a large number of documents in the collection do not contain key values, you are advised to create sparse indexes.
5. When you create text indexes, the field is specified as **text** instead of **1** or **-1**. Each collection has only one text index, but it can index multiple fields.

Command usage

1. The findOne method returns the first document that satisfies the specified query criteria from the collection according to the natural order. To return multiple documents, use this method.
2. If the query does not require the return of the entire document or is only used to determine whether the key value exists, you can use **$project** to limit the returned field, reducing the network traffic and the memory usage of the client.
3. In addition to prefix queries, regular expression queries take longer to execute than using selectors, and indexes are not recommended.
4. Some operators that contain **$** in the query may deteriorate the system performance. The following types of operators are not recommended in services. $or, $nin, $not, $ne, and $exists.

   NOTE

   - $or: The times of queries depend on the number of conditions. It is used to query all the documents that meet the query conditions in the collection. You are advised to use $in instead.
   - $nin: Matches most of indexes, and the full table scan is performed.
   - $not: The query optimizer may fail to match a specific index, and the full table scan is performed.
   - $ne: Selects the documents where the value of the field is not equal to the specified value. The entire document is scanned.
   - $exists: matches each document that contains the field.

   For more information, see **official MongoDB documents**.

Precautions

1. Indexes cannot be used in operators $where and $exists.
2. If the query results need to be sorted, control the number of result sets.
3. If multiple field indexes are involved, place the field used for exact match before the index.
4. If the key value sequence in the search criteria is different from that in the compound index, DDS automatically changes the query sequence to the same as index sequence.
   – Modification operation

     Modify a document by using operators can improve performance. This method does not need to obtain and modify document data back and forth on the server, and takes less time to serialize and transfer data.

- Batch insert

  Batch insert can reduce the number of times data is submitted to the server and improve the performance. The BSON size of the data submitted in batches cannot exceed 48 MB.

- Aggregated operation

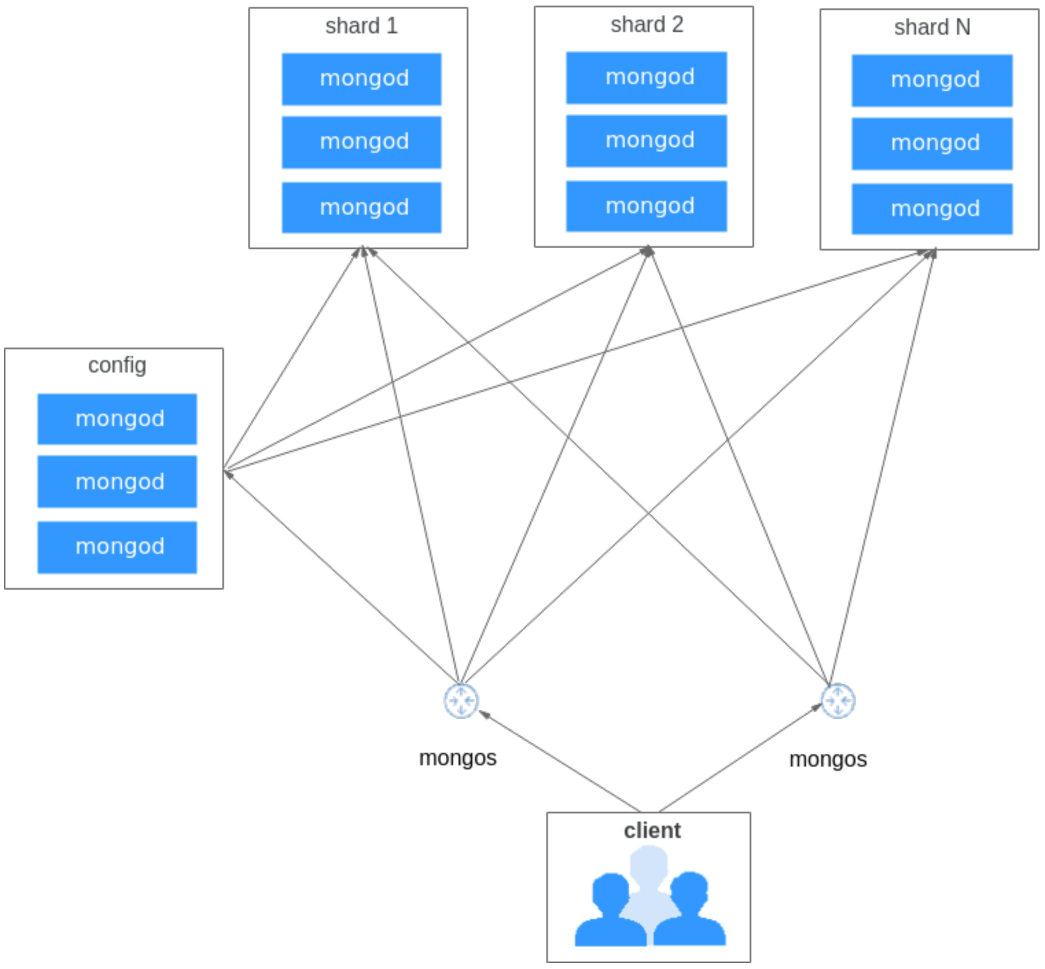
  During aggregation, $match must be placed before $group to reduce the number of documents to be processed by the $group operator.

6 How Do I Prevent the Mongos Cache Problem?

Background

DDS is a document-oriented database service based on distributed file storage, famed for its scalability, high performance, open source, and free mode.

Figure 6-1 DDS cluster architecture



A cluster instance consists of the following three parts:

- Mongos is deployed on a single node. It provides APIs to allow access from external users and shields the internal complexity of the distributed database. A DDS cluster can contain 2 to 12 mongos. You can add them as required.
- Config server is deployed as a replica set. It stores metadata for a sharded cluster. The metadata include information about routes and shards. A cluster contains only one config server.
- Shard server is deployed as a replica set. It stores user data on shards. You can add shard servers in a cluster as required.

Sharding

Sharding is a method for distributing data evenly across multiple shard servers based on a specified shard key. The collection that has a shard key is called sharded collection. If the collection is not sharded, data is stored on only one shard server. DDS cluster mode allows the coexistence of sharded collection and non-sharded collection.

You can run the **sh.shardCollection** command to convert a non-sharded collection into a sharded collection. Before sharding, ensure that the sharding function is enabled on the database where the collections to be sharded are located. You can run the **sh.enableSharding** command to enable the sharding function.

Caching Metadata with mongos

User data is stored in the shard server and metadata is stored in the config server. The route information belongs to metadata and is also stored in the config server. When a user needs to access data through mongos, mongos sends the user's requests to the corresponding shard server according to the route information stored on the config server.

This means that every time the user accesses the data, mongos needs to connect to the config server for the route information, which may affect the system performance. Therefore, a cache mechanism is developed for the mongos to cache the route information of the config server. In this scenario, not only the config server stores the route information, but also the mongos caches the route information.

If no operation is performed on mongos, mongos does not cache any route information. In addition, the route information cached on mongos may not be the latest because the information is only updated in the following scenarios:

- If the mongos is started, it will obtain the latest route information from the config server and caches them locally.
- If the mongos processes the data request for the first time, it will obtain the route information from the config server. After that, the information is cached and can be used directly at the time when it is required.
- Updating route information by running commands on mongos.

NOTE

Only the metadata related to the requested data is updated.

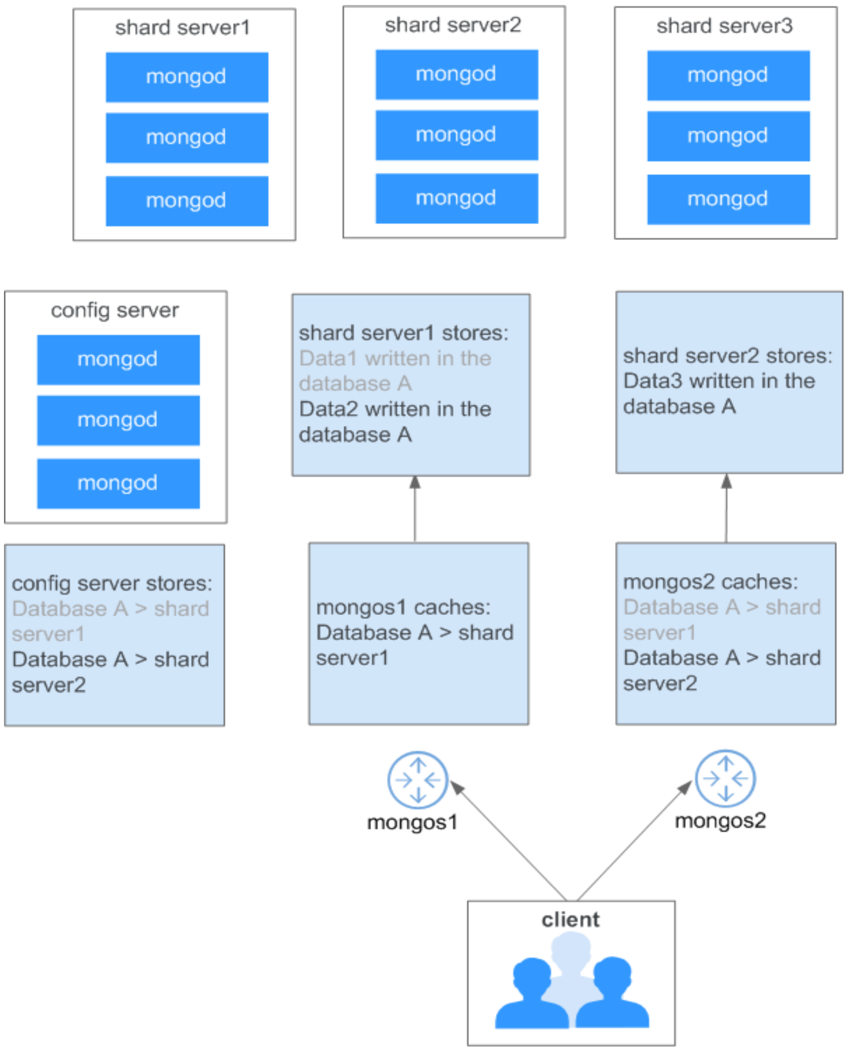
The data to be updated is in the unit of DB.

Scenarios

In the scenario where data is not sharded and multiple mongos exist in a sharded cluster, if data is accessed through different mongos, the cached route information on each mongos may become different. The following shows an example scenario:

1. Create database A with sharding disabled through mongos1. After data1 is written, data1 is allocated to shard server1 for storage. Then, mongos2 is used to query data. Both mongos1 and mongos2 have cached the route information of database A.
2. If database A is deleted through mongos2, the information about database A in the config server and shard server1 is deleted. As a result, mongos1 cannot identify data1 because database A has been deleted.
3. When data2 is written to database A through mongos1, data2 will be stored on shard server1 based on the cached route information but actually database A has been deleted. Then, when data3 is written into database A through mongos2, new information about database A will be generated again on the config server and shard server2 because mongos2 has identified that database A has been deleted.
4. In this case, the route information cached in the mongos1 and mongos2 is inconsistent. mongos1 and mongos2 are associated with different shard servers, and data is not shared between them. As a result, data inconsistency occurs.

Figure 6-2 mongos cache defect scenario


shard server1
mongod
mongod
mongod
shard server2
mongod
mongod
mongod
shard server3
mongod
mongod
mongod
config server
mongod
mongod
mongod
shard server1 stores: Data1 written in the database A Data2 written in the database A
shard server2 stores: Data3 written in the database A
config server stores: Database A > shard server1 Database A > shard server2
mongos1 caches: Database A > shard server1
mongos2 caches: Database A > shard server1 Database A > shard server2
mongos1
mongos2
client

The client queries data through different mongos:

- mongos1: Data2 can be queried, but data3 cannot be queried.
- mongos2: Data3 can be queried, but data2 cannot be queried.

Workaround Suggestion

MongoDB official suggestions: After deleting databases or collections, run **db.adminCommand("flushRouterConfig")** on all mongos nodes to update the route information.

Reference link: **https://docs.mongodb.com/manual/reference/method/db.dropDatabase/index.html#replica-set-and-sharded-clusters**

https://jira.mongodb.org/browse/SERVER-17397

Workaround Suggestion

- For the cluster mode, you are advised to enable the sharding function and then shard the collections in the cluster.

- If the database with sharding disabled is deleted, do not create a database or collection with the same name as the deleted database or collection.

  If you need to create a database or collection with the same name as the deleted database or collection, log in to all the mongos nodes to update the route information before creating the database and collection.

7 How Do I Solve the High CPU Usage Issue?

If the CPU usage is high or close to 100% when you use DDS, data read and write will slow down, affecting your services.

The following describes how to analyze current slow queries. After the analysis and optimization, queries will be processed better and indexes will be used more efficiently.

Analyzing Current Queries

1. Connect to an instance using Mongo Shell.

   To access an instance from the Internet

   For details, see

   – **Connecting to a Cluster Instance over a Public Network**
   – **Connecting to a Replica Set Instance over a Public Network**
   – **Connecting to a Single Node Instance over a Public Network**

   To access an instance that is not publicly accessible

   For details, see

   – **Connecting to a Cluster Instance over a Private Network**
   – **Connecting to a Replica Set Instance over a Private Network**
   – **Connecting to a Single Node Instance over a Private Network**

2. Run the following command to view the operations being performed on the database:

   db.currentOp()

   Command output:

```
{
        "raw" : {
                "shard0001" : {
                        "inprog" : [
                                {
                                        "desc" : "StatisticsCollector",
                                        "threadId" : "140323686905600",
                                        "active" : true,
                                        "opid" : 9037713,
                                        "op" : "none",
                                        "ns" : "",
                                        "query" : {

                                        },
                                        "numYields" : 0,
                                        "locks" : {

                                        },
                                        "waitingForLock" : false,
                                        "lockStats" : {

                                        }
                                },
                                {
                                        "desc" : "conn2607",
                                        "threadId" : "140323415066368",
                                        "connectionId" : 2607,
                                        "client" : "172.16.36.87:37804",
                                        "appName" : "MongoDB Shell",
                                        "active" : true,
                                        "opid" : 9039588,
                                        "secs_running" : 0,
                                        "microsecs_running" : NumberLong(63),
                                        "op" : "command",
                                        "ns" : "admin.",
                                        "query" : {
                                                "currentOp" : 1
                                  },
                                        "numYields" : 0,
                                        "locks" : {

                                        },
                                        "waitingForLock" : false,
                                        "lockStats" : {

                                        }
                                }
                        ],
                        "ok" : 1
                },
              ...
}
```

NOTE

- **client**: IP address of the client that sends the request
- **opid**: unique operation ID
- **secs_running**: elapsed time for execution, in seconds. If the returned value of this field is too large, check whether the request is reasonable.
- **microsecs_running**: elapsed time for execution, in seconds. If the returned value of this field is too large, check whether there is something wrong with the request.
- **op**: operation type. The operations can be query, insert, update, delete, or command.
- **ns**: target collection
- For details, see the **db.currentOp()** command in **official document**.

3. Based on the command output, check whether there are requests that take a long time to process.

   If the CPU usage is low while services are being processed but then becomes high during just certain operations, analyze the requests that take a long time to execute.

   If an abnormal query is found, find the **opid** corresponding to the operation and run **db.killOp(***opid***)** to kill it.

Analyzing Slow Queries

Slow query profiling is enabled for DDS by default. The system automatically records any queries whose execution takes longer than 100 ms to the **system.profile** collection in the corresponding database. You can:

1. Connect to an instance using Mongo Shell.

   To access an instance from the Internet

   For details, see

   – **Connecting to a Cluster Instance over a Public Network**
   – **Connecting to a Replica Set Instance over a Public Network**
   – **Connecting to a Single Node Instance over a Public Network**

   To access an instance that is not publicly accessible

   For details, see

   – **Connecting to a Cluster Instance over a Private Network**
   – **Connecting to a Replica Set Instance over a Private Network**
   – **Connecting to a Single Node Instance over a Private Network**

2. Select a specific database (using the **test** database as an example):

   use test

3. Check whether slow SQL queries have been collected in **system.profile**.

   show collections;

   – If the command output includes **system.profile**, slow SQL queries have been generated. Go to the next step.
```
mongos> show collections
system.profile
test
```
   – If the command output does not contain **system.profile**, no slow SQL queries have been generated, and slow query analysis is not required.
```
mongos> show collections
test
```

4. Check the slow query logs in the database.

   db.system.profile.find().pretty()

5. Analyze slow query logs to find the cause of the high CPU usage.

   The following is an example of a slow query log. The log shows a request that scanned the entire table, including 1,561,632 documents and without using a search index.
```
{
        "op" : "query",
        "ns" : "taiyiDatabase.taiyiTables$10002e",
        "query" : {
                "find" : "taiyiTables",
                "filter" : {
                        "filed19" : NumberLong("852605039766")
                },
                "shardVersion" : [
                        Timestamp(1, 1048673),
                        ObjectId("5da43185267ad9c374a72fd5")
                ],
                "chunkId" : "10002e"
        },
        "keysExamined" : 0,
        "docsExamined" : 1561632,
        "cursorExhausted" : true,
        "numYield" : 12335,
        "locks" : {
                "Global" : {
                        "acquireCount" : {
                                "r" : NumberLong(24672)
                        }
                },
                "Database" : {
                        "acquireCount" : {
                                "r" : NumberLong(12336)
                        }
                },
                "Collection" : {
                        "acquireCount" : {
                                "r" : NumberLong(12336)
                        }
                }
        },
        "nreturned" : 0,
        "responseLength" : 157,
        "protocol" : "op_command",
        "millis" : 44480,
        "planSummary" : "COLLSCAN",
        "execStats" : {
              "stage" :
"SHARDING_FILTER",
                [3/1955]
                "nReturned" : 0,
                "executionTimeMillisEstimate" : 43701,
                "works" : 1561634,
                "advanced" : 0,
                "needTime" : 1561633,
                "needYield" : 0,
                "saveState" : 12335,
                "restoreState" : 12335,
                "isEOF" : 1,
                "invalidates" : 0,
                "chunkSkips" : 0,
                "inputStage" : {
                        "stage" : "COLLSCAN",
                        "filter" : {
                                "filed19" : {
                                        "$eq" : NumberLong("852605039766")
                                }
                        },
                        "nReturned" : 0,
                        "executionTimeMillisEstimate" : 43590,
                        "works" : 1561634,
                        "advanced" : 0,
                        "needTime" : 1561633,
                        "needYield" : 0,
                        "saveState" : 12335,
                        "restoreState" : 12335,
                        "isEOF" : 1,
                        "invalidates" : 0,
                        "direction" : "forward",
                        "docsExamined" : 1561632
                }
        },
        "ts" : ISODate("2019-10-14T10:49:52.780Z"),
        "client" : "172.16.36.87",
        "appName" : "MongoDB Shell",
        "allUsers" : [
                {
                        "user" : "__system",
                        "db" : "local"
                }
        ],
    "user" : "__system@local"
}
```

The following stages can be causes for a slow query:

– **COLLSCAN** involves a full collection (full table) scan.

  When a request (such as query, update, and delete) requires a full table scan, a large amount of CPU resources are occupied. If you find **COLLSCAN** in the slow query log, CPU resources may be occupied.

  If such requests are frequent, create indexes for the fields to be queried.

– **docsExamined** involves a full collection (full table) scan.

  You can view the value of **docsExamined** to check the number of documents scanned. A larger value indicates a higher CPU usage.

– **IXSCAN** and **keysExamined** scan indexes.

  NOTE

  An excessive number of indexes can affect the write and update performance.

  If your application has more write operations, creating indexes may increase write latency.

  You can view the value of **keyExamined** to see how many indexes are scanned in a query. A larger value indicates a higher CPU usage.

  If the index is not appropriate or there are many matching results, the CPU usage may spike and the execution can slow down.

  Example: For the data of a collection, the number of values of the **a** field is small (only **1** and **2**), but the **b** field has more values.

```
{ a: 1, b: 1 }
{ a: 1, b: 2 }
{ a: 1, b: 3 }
......
{ a: 1, b: 100000}
{ a: 2, b: 1 }
{ a: 2, b: 2 }
{ a: 2, b: 3 }
......
{ a: 1, y: 100000}
```

  The following shows how to implement the {a: 1, b: 2} query.

```
db.createIndex({a: 1}): The query is not effective because the a field has too many
same values.
db.createIndex({a: 1, b: 1}): The query is not effective because the a field has too
many same values.
db.createIndex({b: 1}): The query is effective because the b field has a few same
values.
db.createIndex({b: 1, a: 1}): The query is not effective because the a field has a few
same values.
```

  For the differences between {a: 1} and {b: 1, a: 1}, see the **official documents**.

– **SORT** and **hasSortStage** may involve sorting a large amount of data.

  If the value of **hasSortStage** in the **system.profile** collection is **true**, the query request involves sorting. If the sorting cannot be implemented through indexes, the query results are sorted, and sorting is a CPU intensive operation. In this scenario, you need to create indexes for fields that are frequently sorted.

If the **system.profile** collection contains **SORT**, you can use indexing to improve sorting speed.

Other operations, such as index creation and aggregation (combinations of traversal, query, update, and sorting), also apply to the preceding scenarios because they are also CPU intensive operations. For more information about profiling, see **official documents**.

Analysis Capability

After the analysis and optimization of the requests that are being executed and slow requests, all requests use proper indexes, and the CPU usage becomes stable. If the CPU usage remains high after the analysis and troubleshooting, the current instance may have reached the performance bottleneck and cannot meet service requirements. In this case, you can perform the following operations to solve the problem:

1. View monitoring information to analyze instance resource usage. For details, see **Viewing Monitoring Metrics**.
2. Change the DDS instance class or add shard nodes.

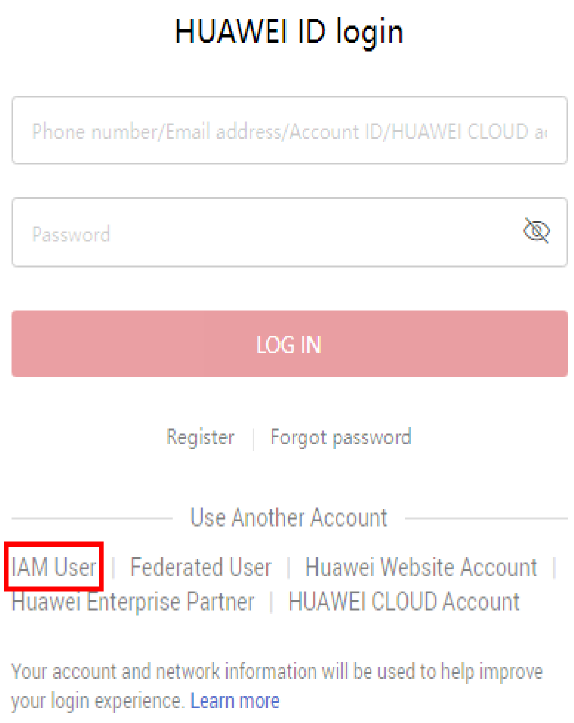
8 Creating a User and Granting the Read-Only Permission to the User

Step 1: Create a User Group and Grant Permissions

Users in the same user group have the same permissions. Users created in IAM inherit permissions from the groups to which they belong. Users created in IAM inherit permissions from the groups they belong to. To create a user group, perform the following steps:

Step 1 Log in to Huawei Cloud using your HUAWEI ID.

Figure 8-1 HUAWEI ID Login



HUAWEI ID login

Phone number/Email address/Account ID/HUAWEI CLOUD a

Password

LOG IN

Register | Forgot password

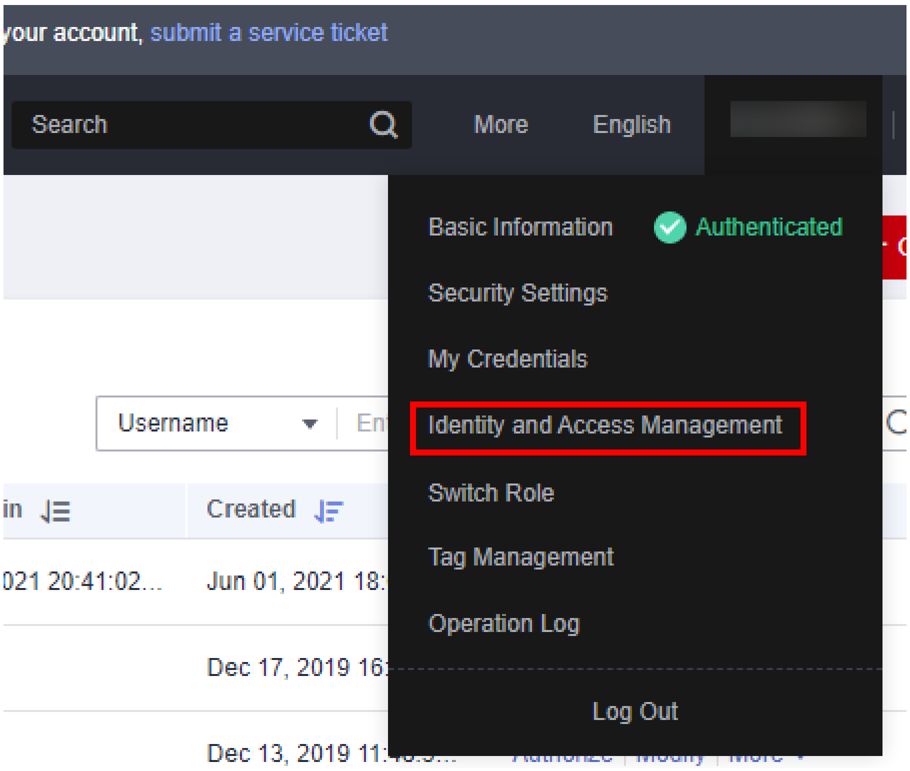
Use Another Account

IAM User | Federated User | Huawei Website Account | Huawei Enterprise Partner | HUAWEI CLOUD Account

Your account and network information will be used to help improve your login experience. Learn more

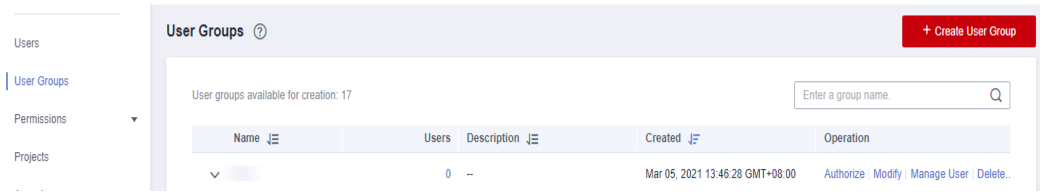
Step 2 On the management console, click the username in the upper right corner and then choose **Identity and Access Management**.

Figure 8-2 Choosing IAM



Step 3 On the IAM console, choose **User Groups** in the navigation pane. Then click **Create User Group**.

Figure 8-3 User group



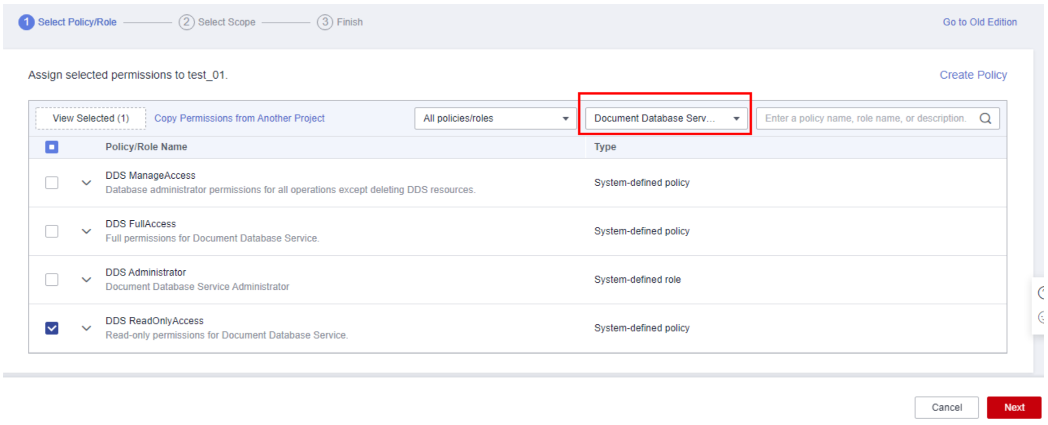
Step 4 Enter a user group name (for example, **test_01**), set the password, and click **OK**.

The user group is then displayed in the user group list.

Step 5 In the user group list, choose **Authorize** in the row that contains the **test_01** user group.

Step 6 Select **Document Database Service** from the drop-down list, select **DDS ReadOnlyAccess**, and click **Next**.

Figure 8-4 Authorization



1 Select Policy/Role
2 Select Scope
3 Finish
Go to Old Edition

Assign selected permissions to test_01.

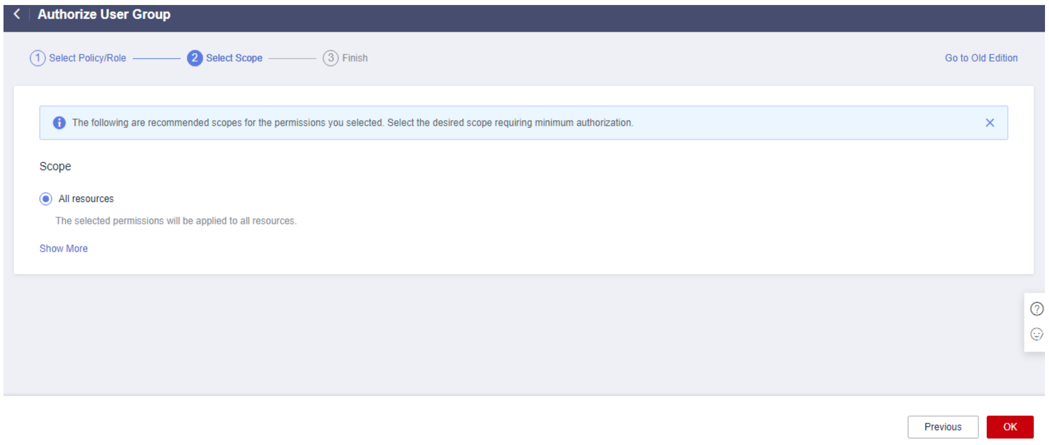
Create Policy
View Selected (1)
Copy Permissions from Another Project
All policies/roles
Document Database Serv...
Enter a policy name, role name, or description.

	Policy/Role Name	Type
☐	DDS ManageAccess Database administrator permissions for all operations except deleting DDS resources.	System-defined policy
☐	DDS FullAccess Full permissions for Document Database Service.	System-defined policy
☐	DDS Administrator Document Database Service Administrator	System-defined role
☑	DDS ReadOnlyAccess Read-only permissions for Document Database Service.	System-defined policy

Step 7 Specify the scope and click **OK**.

- All resources
- Region-specific projects: The selected permissions will be applied to resources in the region-specific projects you select.

Figure 8-5 Specifying the scope



The following are recommended scopes for the permissions you selected. Select the desired scope requiring minimum authorization.

Scope

[✓ All resources]

The selected permissions will be applied to all resources.

----End

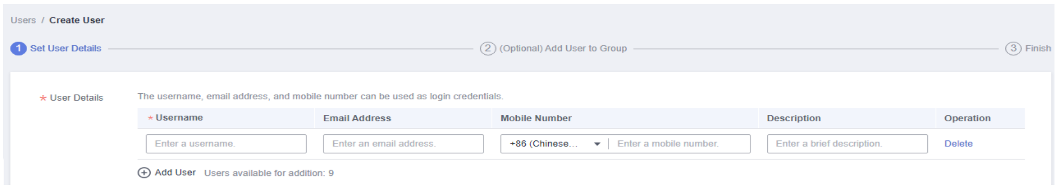
Step 2: Create an IAM User

IAM users can be created for employees or applications of an enterprise. Each IAM user has their own security credentials, and inherits permissions from the groups it is a member of. To create an IAM user, perform the following steps:

Step 1 On the IAM console, choose **Users** in the navigation pane. Then click **Create User**.

Step 2 Specify the user information on the **Create User** page. To create more users, click **Add User**. You can add a maximum of 10 users at a time.

Figure 8-6 Creating a user



Users / **Create User**

1 Set User Details — 2 (Optional) Add User to Group — 3 Finish

* User Details

The username, email address, and mobile number can be used as login credentials.

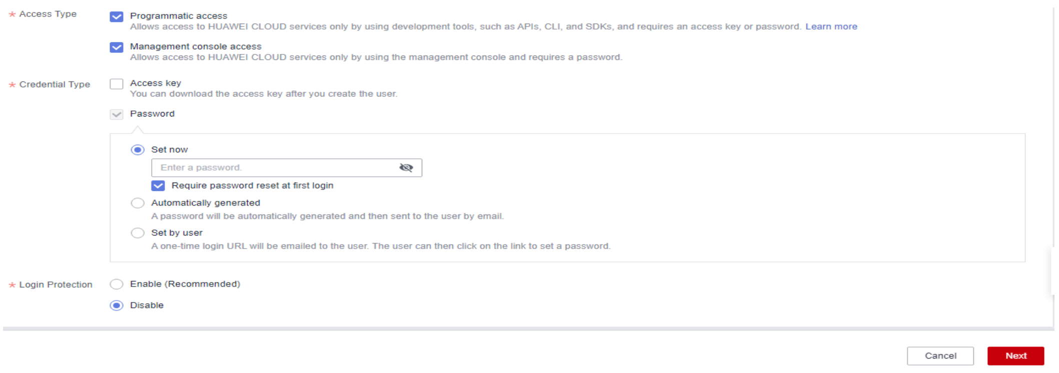
* Username	Email Address	Mobile Number	Description	Operation
Enter a username.	Enter an email address.	+86 (Chinese... Enter a mobile number.	Enter a brief description.	Delete

Add User   Users available for addition: 9

- **Username**: Used for logging in to Huawei Cloud. For this example, enter **James**.
- **Email Address**: Email address bound to the IAM user. This parameter is mandatory if the access type is specified as **Set by user**.
- (Optional) **Mobile Number**: Mobile number bound to the IAM user.
- (Optional) **Description**: Description of the user.

Step 3 Configure required parameters and click **Next**.

Figure 8-7 Configuring user details



* Access Type

[✓] Programmatic access
Allows access to HUAWEI CLOUD services only by using development tools, such as APIs, CLI, and SDKs, and requires an access key or password. Learn more

[✓] Management console access
Allows access to HUAWEI CLOUD services only by using the management console and requires a password.

* Credential Type

[ ] Access key
You can download the access key after you create the user.

[✓] Password

[✓ Set now]
Enter a password.
[✓] Require password reset at first login

Automatically generated
A password will be automatically generated and then sent to the user by email.

Set by user
A one-time login URL will be emailed to the user. The user can then click on the link to set a password.

* Login Protection

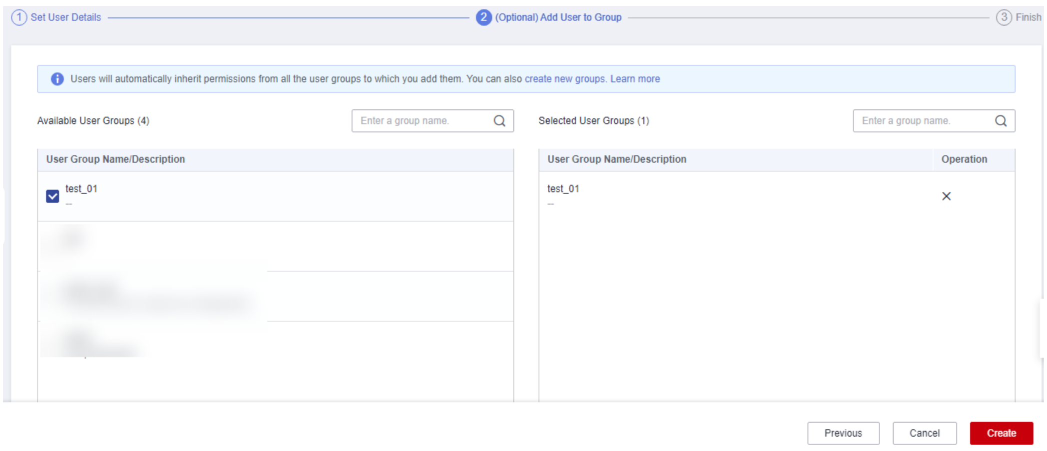
Enable (Recommended)
[✓ Disable]

Table 8-1 Configuration items

Parameter	Description
Access Type	• **Programmatic access**: Select this option to allow the user to access cloud services using development tools, such as APIs, CLI, and SDKs. You can generate an access key or set a password for the user. • **Management console access**: Select this option to allow the user to access cloud services using the management console. You can set or generate a password for the user or request the user to set a password at first login.
Credential Type	• Access key: Download the access key after the user is created. • Password: If you create multiple users, set a password for the users and determine whether to require the users to reset the password at first login. If you create one user, you can select **Automatically generated** and the system automatically generates a login password for the user.

Parameter	Description
Login Protection	To ensure account security, you are advised to select **Enable**.

Step 4 Add the users to user group created in **Step 4** and click **Create User**.



Users will automatically inherit permissions from all the user groups to which you add them. You can also create new groups. Learn more

Available User Groups (4) Enter a group name.

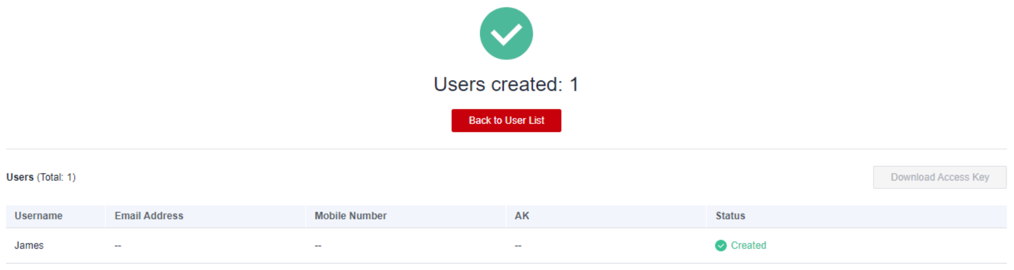
User Group Name/Description
✓ test_01 --

Selected User Groups (1) Enter a group name.

User Group Name/Description	Operation
test_01 --	×

Figure 8-8 Creating a user

Step 5 Check the created users in the user list. If you select **Access key** for **Credential Type**, you can download the access key after you create the user. You can also manage the access keys on the **My Credentials** page.


Users created: 1

Users (Total: 1)

Username	Email Address	Mobile Number	AK	Status
James	--	--	--	✓ Created

Figure 8-9 Viewing the results

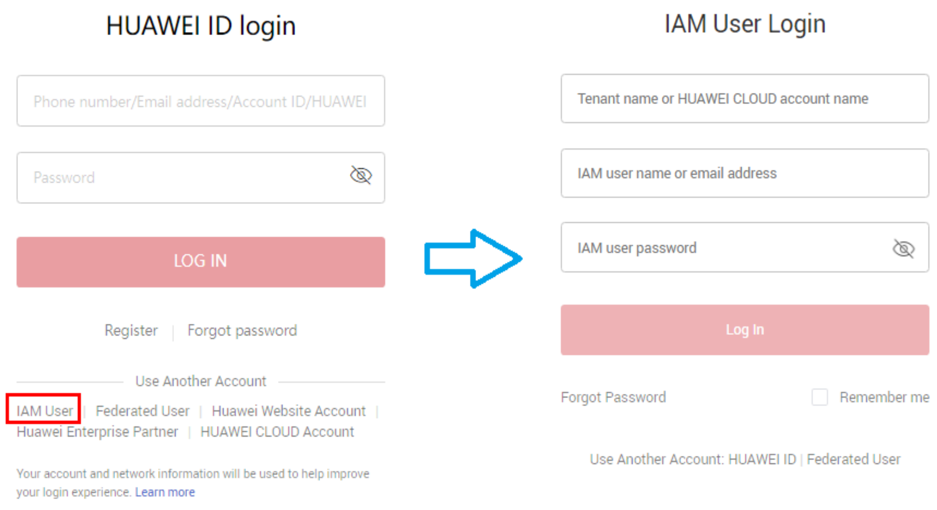
----End

Step 3: Log In and Verify Permissions

After the user is created, use the username and identity credential to log in to Huawei Cloud, and verify that the user has the permissions defined by the **DDS ReadOnlyAccess** policy. For more login methods, see "Create IAM Users and Log In".

Step 1 On the Huawei Cloud login page, click **IAM User** in the lower left corner.

Figure 8-10 IAM user login


HUAWEI ID login
Phone number/Email address/Account ID/HUAWEI
Password
LOG IN
Register | Forgot password
Use Another Account
IAM User | Federated User | Huawei Website Account | Huawei Enterprise Partner | HUAWEI CLOUD Account
Your account and network information will be used to help improve your login experience. Learn more
IAM User Login
Tenant name or HUAWEI CLOUD account name
IAM user name or email address
IAM user password
Log In
Forgot Password
Remember me
Use Another Account: HUAWEI ID | Federated User

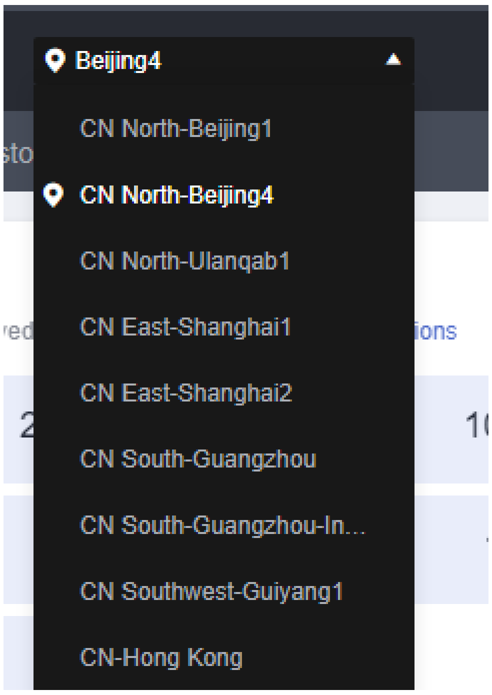
Step 2 Enter the account name, username, and password, and click **Log In**.

- The account name is the name of the Huawei Cloud account that created the IAM user.
- The username and password are those set by the account when creating the IAM user.

If the login fails, contact the entity owning the account to verify the username and password. Alternatively, you can reset the password by following the procedure in "Resetting Password for an IAM User".

Step 3 After successful login, switch to a region where the user has been granted permissions on the management console.

Figure 8-11 Region



Step 4 Choose **Service List** > **Document Database Service**. Then click **Buy DB Instance** on the DDS console. If a message appears indicating insufficient permissions to perform the operation, the **DDS ReadOnlyAccess** policy has already taken effect.

Step 5 Choose any other service in the **Service List**. If a message appears indicating insufficient permissions to access the service, the **DDS ReadOnlyAccess** policy has already taken effect.

----End