TencentDB for MongoDB

Best Practice

Product Documentation

Contents

Best Practice

Optimizing Indexes to Break Through Read/Write Performance Bottlenecks

Last updated：2024-01-15 14:49:55

Index is a key factor affecting the MongoDB database query performance. Meeting your query needs with a minimal number of indexes can greatly improve the database performance and reduce the storage costs. This document describes how to analyze and optimize indexes to help you break through the bottleneck in database read/write performance.

Problem Description

In daily Ops, you can log in to the TencentDB for MongoDB console and click the instance ID to enter the **Instance Details** page and view the following information:
Select the **System Monitoring** tab to view the instance monitoring data:
The CPU usage of the mongod node is too high. The CPU utilization is close to 90% or even 100%.
The disk reads/writes per second stays high continuously, and the I/O resource usage of a node accounts for 60% of that of the entire server.
Select **Database Management** > **Slow Log Query** to view slow logs:
The instance has a high number of slow logs, which include many `find` and `update` requests of different types. Thousands of such requests are received per second during peak hours.
Slow logs are of diverse types and have various query conditions, and all slow queries have matching indexes. Below is the log content:

```
Mon Aug  2 10:34:24.928 I COMMAND  [conn10480929] command xxx.xxx command: find { f
Mon Aug  2 10:34:22.965 I COMMAND  [conn10301893] command xx.txxx command: find { f
```

Cause Analysis

By analyzing the slow logs, it is found that all query requests use the `{ alxxxId: 1.0, itemTagList: 1.0 }` index. `keysExamined` and `docsExamined` are both set to `1498` for scan by index; however, the number of returned documents (`nreturned`) is only `3` ; that is, only 3 data entries out of the 1498 rows of data and indexes scanned meet the conditions. As can be seen, the key cause compromising the read/write performance is unreasonable index configuration.

Note:

`keysExamined` and `docsExamined` indicate the numbers of index entries and documents scanned respectively. Larger **keysExamined** and **docsExamined** values indicate that no index is created or the created index is less distinctive.

Index Optimization Process

Step 1. Collect the SQL statements

Common business query and update SQL statements as listed below:

```
Query based on `AlxxxId` (user ID) and `itxxxId` (one or multiple values).
`count` query based on `AlxxxId`.
Paginated query based on `AlxxxId` by time range (`createTime`). In some queries, `
Query based on `AlxxxId`, `ParentAlxxxId`, `parentItxxxId`, and `state`.
Query based on `ItxxxId` (one or multiple values).
Query based on `AlxxxId`, `state`, and `updateTime`.
Query based on `AlxxxId`, `state`, `createTime`, and `totalStock` (number of invent
Query based on `AlxxxId` (user ID), `itxxxId` (one or multiple values), and any oth
Query based on `AlxxxId`, `digitalxxxrmarkId` (watermark ID), and `state`.
Query based on `AlxxxId`, `itemTagList` (tag ID), and `state`.
Query based on `AlxxxId`, `itxxxId` (one or multiple values), and any other fields.
```

```
Other queries
```

Common business statistics count SQL statements as listed below:

```
Query based on `AlxxxId`, `state`, and `persxxal`.
Query based on `AlxxxId`, `state`, and `itemType`.
Query based on `AlxxxId` (user ID), `itxxxId` (one or multiple values), and any oth
```

Step 2. Get the existing indexes of the cluster

Use `db.xxx.getindex()` to get the collection index information. The query is complex, and there are 30 indexes in total as listed below:

```
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : -1, "persxxal" : 1, "
{ "alxxxId" : 1, "image" : 1 }
{ "itexxxList.vidxxCheck" : 1, "itemType" : 1, "state" : 1 }
{ "alxxxId" : 1, "state" : -1, "newsendTime" : -1, "itxxxId" : 1, "persxxal" : 1 }
{ "_id" : 1 }
{ "alxxxId" : 1, "createTime" : -1, "checkStatus" : 1 }
{ "alxxxId" : 1, "parentItxxxId" : -1, "state" : -1, "updateTime" : -1, "persxxal"
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal"
{ "srcItxxxId" : 1 }
{ "createTime" : 1 }
{ "itexxxList.boyunState" : -1, "itexxxList.wozhituUploadServerId": -1, "itexxxList
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : -1 }
{ "itxxxId" : -1 }
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "videoCover" : 1 }
{ "alxxxId" : 1, "itemType" : 1 }
{ "alxxxId" : 1, "state" : -1, "itemType" : 1, "persxxal" : 1, "updateTime" : 1 }
{ "alxxxId" : 1, "itxxxId" : 1 }
{ "itxxxId" : 1, "alxxxId" : 1 }
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "itemTagList" : 1 }
{ "itexxxList.photoQiniuUrl" : 1, "itexxxList.boyunState" : -1, "itexxxList.sourceT
{ "alxxxId" : 1, "parentItxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "parentItxxxId" : 1, "updateTime" : 1 }
{ "updateTime" : 1 }
{ "itemPhoxxIdList" : -1 }
{ "alxxxId" : 1, "state" : -1, "isTop" : 1 }
{ "alxxxId" : 1, "state" : 1, "itemResxxxIdList" : 1, "updateTime" : -1 }
{ "alxxxId" : 1, "state" : -1, "itexxxList.photoQiniuUrl" : 1 }
{ "itexxxList.qiniuStatus" : 1, "itexxxList.photoNetUrl" : 1, "itexxxList.photoQini
{ "itemResxxxIdList" : 1  }
```

Step 3. Optimize indexes

Deleting useless indexes

MongoDB allows you to get the number of hits of each index through the following index statistics command:

```
> db.xxxxx.aggregate({"$indexStats":{}})
{ "name" : "alxxxId_1_parentItxxxId_1_parentAlxxxId_1", "key" : { "alxxxId" : 1, "p
```

The fields are as described below:

name: The name of the index for which to collect statistics.

ops: The number of index hits, i.e., the number of times query requests hit an index. If the value of an index is zero or very small, the index is seldom selected as the optimal index and can be considered useless.

Use the index statistics command to get the numbers of hits of all indexes as shown below. If the value of an index is zero or very small, directly delete the index. In addition, as the business has been operated for a period of time, indexes with an `ops` value smaller than 10,000 can also be deleted. At this point, 30 - 11 = 19 useful indexes are retained.

```
db.xxx.aggregate({"$indexStats":{}})
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : -1, "persxxal" : 1, "
{ "alxxxId" : 1, "image" : 1 }                            "ops" : NumberLong(293104
{ "itexxxList.vidxxCheck" : 1, "itemType" : 1, "state" : 1 }    "ops" : NumberLong(
{ "alxxxId" : 1, "state" : -1, "newsendTime" : -1, "itxxxId" : -1, "persxxal" : 1 }
{ "_id" : 1 }                                             "ops" : NumberLong(3987)
 { "alxxxId" : 1, "createTime" : 1, "checkStatus" : 1 }      "ops" : NumberLong(200
{ "alxxxId" : 1, "parentItxxxId" : -1, "state" : -1, "updateTime" : -1, "persxxal"
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal"
{ "itxxxId" : -1 }       "ops" : NumberLong(38854593)
{ "srcItxxxId" : -1 }                               "ops" : NumberLong(0)
{ "createTime" : 1 }                              "ops" : NumberLong(62)
{ "itexxxList.boyunState" : -1, "itexxxList.wozhituUploadServerId" : -1, "itexxxLis
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : -1 }
{ "itxxxId" : -1 }                 "ops" : NumberLong(38854593)
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }    "ops" :
{ "alxxxId" : 1, "videoCover" : 1 }         { "ops" : NumberLong(2921857)
{ "alxxxId" : 1, "itemType" : 1 }           { "ops" : NumberLong(457)
{ "alxxxId" : 1, "state" : -1, "itemType" : 1, "persxxal" : 1, " itxxxId " : 1 }
{ "alxxxId" : 1, "itxxxId" : 1 }        "ops" : NumberLong(232360252)
{ "itxxxId" : 1, "alxxxId" : 1 }        "ops" : NumberLong(145640252)
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }          "ops" : NumberLong(689
{ "alxxxId" : 1, "itemTagList" : 1 }                    "ops" : NumberLong(28986936
{ "itexxxList.photoQiniuUrl" : 1, "itexxxList.boyunState" : 1, "itexxxList.sourceTy
{ "alxxxId" : 1, "parentItxxxId" : 1, "state" : 1 }                "ops" : NumberLo
{ "alxxxId" : 1, "parentItxxxId" : 1, "updateTime" : 1 }           "ops" : NumberLon
{ "updateTime" : 1 }                                      "ops" : NumberLong(139
{ "itemPhoxxIdList" : -1 }         "ops" : NumberLong(0)
{ "alxxxId" : 1, "state" : -1, "isTop" : 1 }        "ops" : NumberLong(213305)
{ "alxxxId" : 1, "state" : 1, "itemResxxxIdList" : 1, "updateTime" : 1 }       "ops
{ "alxxxId" : 1, "state" : 1, "itexxxList.photoQiniuUrl" : 1}  "ops" : NumberLong(2
{ "itexxxList.qiniuStatus" : 1, "itexxxList.photoNetUrl" : 1, "itexxxList.photoQini
{ "itemResxxxIdList" : 1  }                 "ops" : NumberLong(7)
```

Deleting duplicate indexes

Duplicate indexes caused by the query sequence

Different developers of the business have written two SQL indexes as shown below. Analysis finds that the two indexes have the same purpose, so only one of them is needed.

```
db.xxxx.find({{ "alxxxId" : xxx, "itxxxId" : xxx }})
db.xxxx.find({{ " itxxxId " : xxx, " alxxxId " : xxx }})
```

Duplicate indexes caused by the leftmost match rule

Among the `{ itxxxId:1, alxxxId:1 }` and `{ itxxxId :1}` indexes, `{ itxxxId :1}` is a duplicate.

Duplicate indexes caused by inclusion

```
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
{ "alxxxId" : 1, " state " : 1 }
```

There are three queries for the three indexes:

```
Db.xxx.find({ "alxxxId" : xxx, "parentItxxxId" : xx, "parentAlxxxId" : xxx, "state"
Db.xxx.find({ "alxxxId" : xxx, " parentAlxxxId " : xx, " state " : xxx })
Db.xxx.find({ "alxxxId" : xxx,  " state " : xxx })
```

The queries all contain common fields, so the indexes can be combined into one to serve both types of SQL queries.

Below is the combined index:

```
{ "alxxxId" : 1, " state " : 1, " parentAlxxxId " : 1, parentItxxxId :1}
```

After duplicate indexes are combined and cleared, the following two indexes can be retained:

```
{ itxxxId:1, alxxxId:1 }
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }
```

Analyzing the index uniqueness to remove duplicate indexes

By analyzing the combination of field modules in the collection data, you can find that the `alxxxId` and `itxxxId` fields are frequently used. By analyzing the schema information and extracting random data, you can find that the combination of these two fields are unique.

It is confirmed that any combination of the two fields represents a unique data entry. Therefore, all combinations of the two fields and any other fields are unique, and the following indexes can be combined into `{ itxxxId:1, alxxxId:1 }`.

```
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : 1, "persxxal" : 1, "s
{ "alxxxId" : 1, "state" : -1, "itemType" : 1, "persxxal" : 1, " itxxxId " : 1 }
{ "alxxxId" : 1, "state" : -1, "newsendTime" : -1, "itxxxId" : 1, "persxxal" : 1 }
{ "alxxxId" : 1, "state" : 1, "itxxxId" : 1, "updateTime" : -1 }
{ itxxxId:1, alxxxId:1 }
```

Optimizing useless indexes caused by non-equi query

As can be seen from the above 30 indexes, some are time fields, such as `createTime` and `updateTime`, which are used for various range queries. Range queries are non-equi queries. If range query fields are placed before index fields, index fields will fail to be indexed as shown below:

```
db.collection.find({{ "alxxxId" : xx, "parentItxxxId" : xx, "state" : xx, "updateTi

db.collection.find({{ "alxxxId" : xx, "state" : xx, "parentItxxxId" : xx, "updateTi
```

Both queries contain the `updateTime` field and are range queries. Except the `updateTime` fields, all other fields are equi queries, and fields on the right of `updateTime` cannot use indexes; that is, the `persxxal` and `srcItxxxId` fields of the first index and the `persxxal` field of the second index cannot be matched with indexes.

Set the following two indexes for the two queries:

```
{ "alxxxId" : 1, "parentItxxxId" : -1, "state" : -1, "updateTime" : -1, "persxxal"
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal"
```

As the fields of the two indexes are basically the same, the indexes can be optimized into the following index to ensure that more fields can be matched:

```
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1,  "persxxal" : -1, "updateTime"
```

Removing indexes of infrequently queried fields

Indexes with less than 10,000 hits are removed when you delete useless indexes. However, compared with indexes with billions of hits, some indexes still have a relatively lower number of hits (like hundreds of thousands). Such indexes contain `image` and `videoCover` fields respectively as shown below:

```
{ "alxxxId" : 1, "image" : 1 }          "ops" : NumberLong(293104)
{ "alxxxId" : 1, "videoCover" : 1 }     "ops" : NumberLong(292857)
```

Log in to the TencentDB for MongoDB console. On the **Slow Log Query** tab, lower the slow log latency threshold and analyze the logs of the two queries as shown below:

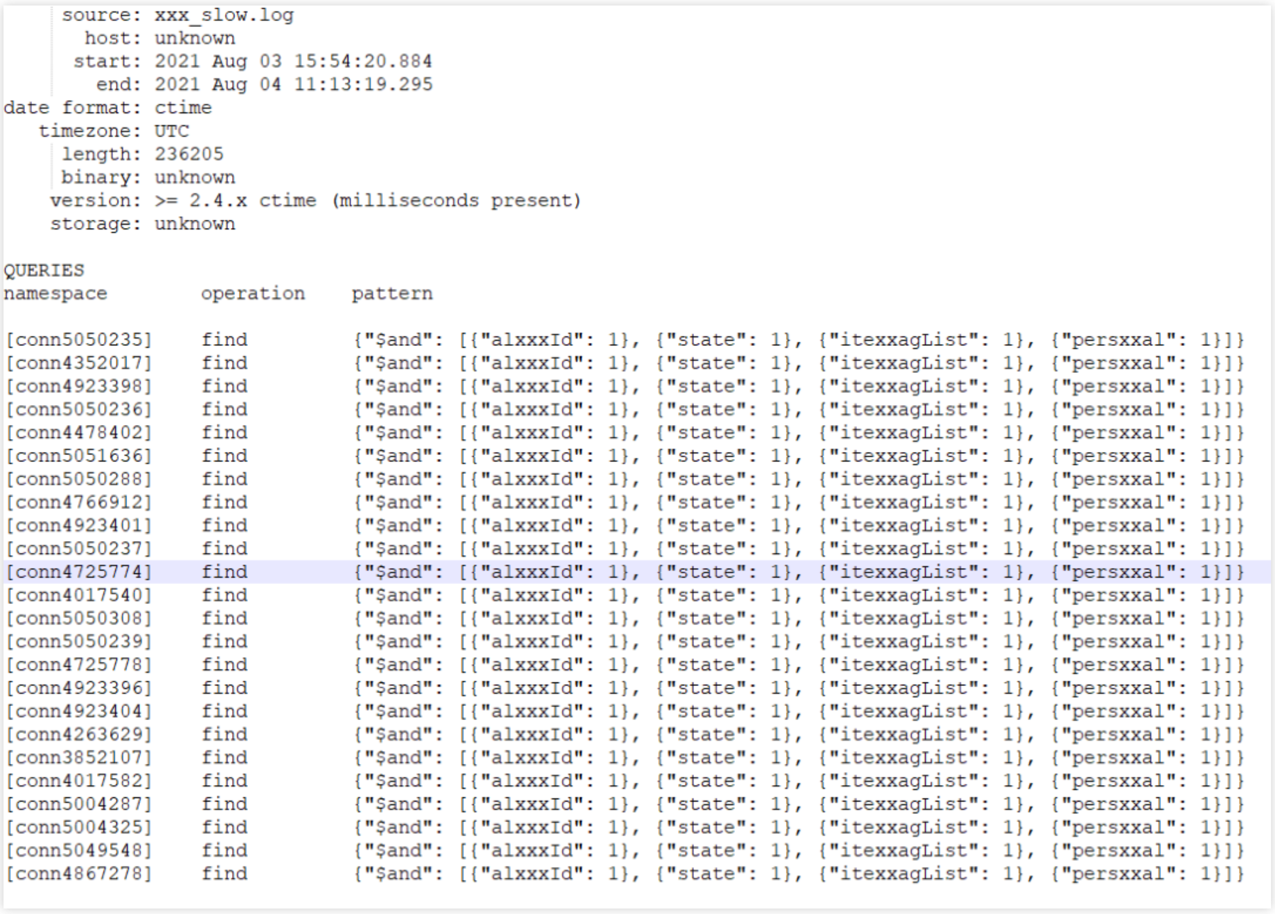
```
Mon Aug  2 10:56:46.533 I COMMAND  [conn5491176] command xxxx.tbxxxxx command: coun

Mon Aug  2 10:47:53.262 I COMMAND  [conn10428265] command xxxx.tbxxxxx command: fin
```

`image` field: It is used together with `alxxxId` and `itxxxId` for combined query. However, the combination of `alxxxId` and `itxxxId` is already unique, and the `image` field is totally not indexed, so the `{ "alxxxId" : 1, "ixxxge" : 1 }` index can be deleted.

`videoCover` field: By analyzing logs, it can be found that `videoCover` is not in the query conditions, only part of queries match the `{ alxxxId: 1, videoCover: 1 }` index, and `keysExamined` and `docsExamined` are different from `nreturned` . Therefore, it can be confirmed that only the `alxxxId` index field is matched, and the `{ alxxxId: 1, videoCover: 1 }` index can also be deleted.

Analyzing frequent queries in logs to add optimal index

Log in to the TencentDB for MongoDB console. On the **Slow Log Query** tab, lower the slow log latency threshold. Use mtools to analyze queries for a period of time, and you can get the following information about frequent queries:



```
     source: xxx_slow.log
       host: unknown
      start: 2021 Aug 03 15:54:20.884
        end: 2021 Aug 04 11:13:19.295
date format: ctime
   timezone: UTC
     length: 236205
     binary: unknown
    version: >= 2.4.x ctime (milliseconds present)
    storage: unknown

QUERIES
namespace        operation    pattern

[conn5050235]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4352017]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923398]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050236]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4478402]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5051636]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050288]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4766912]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923401]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050237]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4725774]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4017540]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050308]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5050239]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4725778]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923396]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4923404]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4263629]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn3852107]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4017582]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5004287]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5004325]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn5049548]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
[conn4867278]    find         {"$and": [{"alxxxId": 1}, {"state": 1}, {"itexxagList": 1}, {"persxxal": 1}]}
```

These frequent queries account for more than 99% of queries. By analyzing their logs, you can get information similar to the following:

```
Mon Aug  2 10:47:58.015 I COMMAND  [conn4352017] command xxxx.xxx command: find { f
```

As can be seen from the log, the frequent query matches the `{ alxxxId: 1.0, itexxagList: 1.0 }` index, and there is a huge difference between the numbers of scanned data rows and returned rows: 1327 vs. 3.
The index is sub-optimal. The frequent query is a four-field equi query, and only two fields are indexed. In this case, you can optimize the index as follows: `{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0}` .
In addition, the log also shows that the frequent query actually has a `sort` and a `limit` . Below is the entire raw query SQL statement:

```
db.xxx.find({ $and: [ { alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [ xxx
```

The query model consists of a common multi-field equi query, `sort` query, and `limit` , so the optimal index of the query can be one of the following two indexes:

Index 1: Index for a common multi-field equi query

Analyze the query conditions:

```
{ $and: [ { alxxxId:"xxx" }, { state: 0 }, { itexxagList: { $in: [ xxxx ] } }, { pe
```

All four fields of the SQL statement are equi queries. Create the optimal index based on the hash, and place fields from left to right by value hash. You can get the following optimal index:

```
{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0}
```

If you use the index as the optimal index, the execution process of the entire common multi-field equi query, `sort` query, and `limit` is as detailed below:

Use the `{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0}` index to find all data entries meeting the conditions specified by `{ $and: [ { alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [ xxxx ] } }, { persxxal: 0 } ] }`.

Perform memory sorting on the data entries meeting the conditions.

Get the first three data entries after sorting.

Index 2: Optimal index of the equi query and `sort`

The `sort` query has a `limit`. Find the frequent sorting SQL statement, which is as shown below:

```
{ $and: [ { alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [ xxxx ] } }, { p
```

As the query is extremely frequent, we recommend you add the following index to such SQL statements:

```
{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0, updateTime : 1}
```

Step 4. Sort out the final indexes to be retained

The following indexes are retained after the above optimization steps:

```
{ "itxxxId" : 1, "alxxxId" : 1 }
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : 1 }
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "persxxal" : -1, "updateTime"
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1"parentItxxxId" : 1}
{ alxxxId: 1.0, itexxagList: 1.0 , persxxal:1.0, stat:1.0, updateTime:1}
{ "alxxxId" : 1, "createTime" : -1}
```

Index Optimization Benefits

CPU resource usage is reduced by over 90%.
After optimization, the peak CPU utilization is reduced from over 90% to below 10%.
Disk I/O resource usage is reduced by over 85%.
Disk I/O utilization is reduced from 60%–70% to below 10%.
Disk storage costs are reduced by 20%.
Each index has an index file in the disk. After 30 indexes are reduced to 8, the final actual disk usage of data entries and indexes is reduced by about 20%.
Slow logs are reduced by 99%.
Before index optimization, thousands of slow logs are generated per second. After optimization, only dozens are generated per second.

Troubleshooting Mongos Load Imbalance in Sharded Cluster

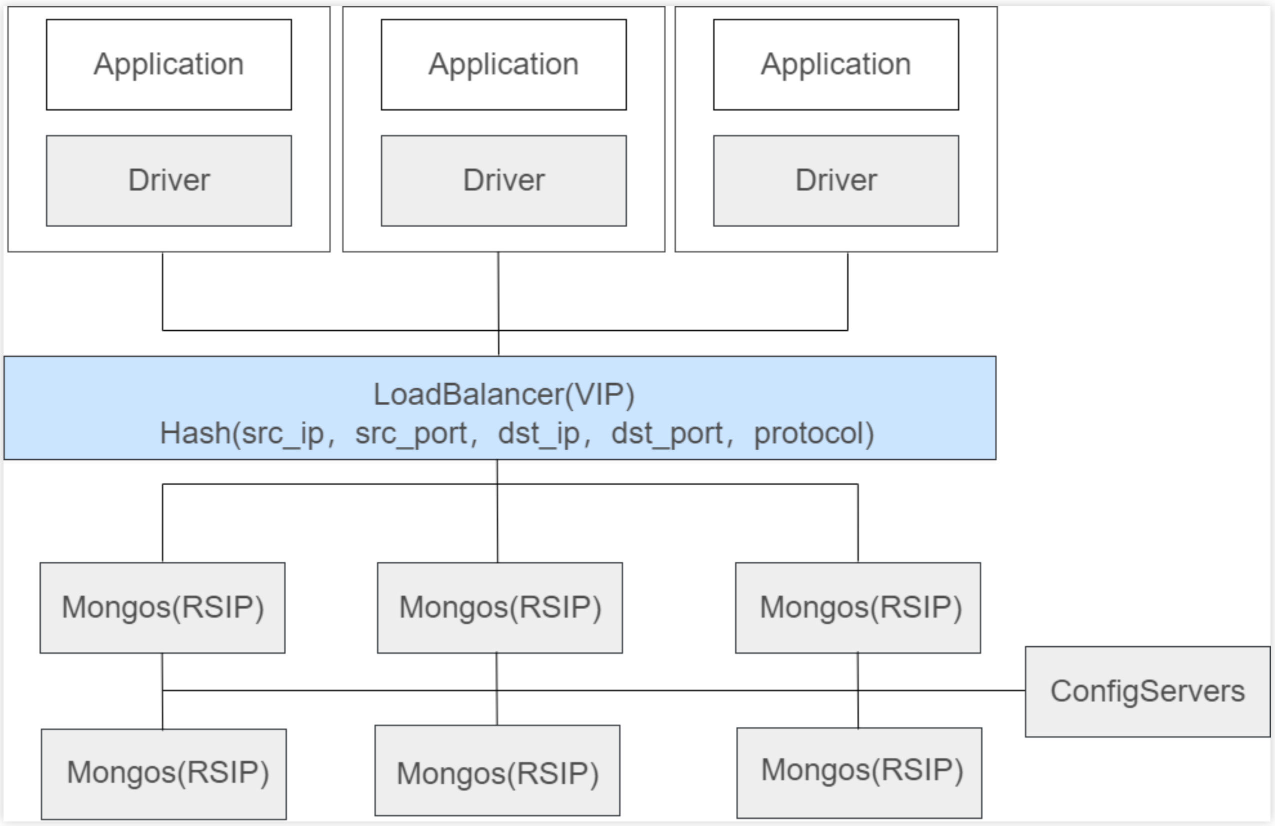
Last updated：2024-01-15 14:49:56

Business Background

In a TencentDB for MongoDB sharded cluster, multiple mongos nodes are available for receiving connection query requests from all client applications, routing the requests to the corresponding shards in the cluster, and splicing the received responses back to the clients. You can connect the cluster to a load balancer through one or multiple VIPs, so that the traffic is automatically distributed among the mongos nodes to increase the data processing capacity, throughput, availability, and flexibility of the network.

How Load Balancing Works

A user program connects to a load balancer (through VIP) to block multiple real server IPs (RSIPs). The load balancing service of TencentDB for MongoDB routes different request source IPs to different mongos nodes through a 5-tuple hash policy (source IP, source port, target IP, target port, and communication protocol). In case of an RSIP change on the backend, an automated process will be initiated to change the mappings between the VIP and RSIPs, which is easy and imperceptible to the business.



`getMore` problem during batch scan

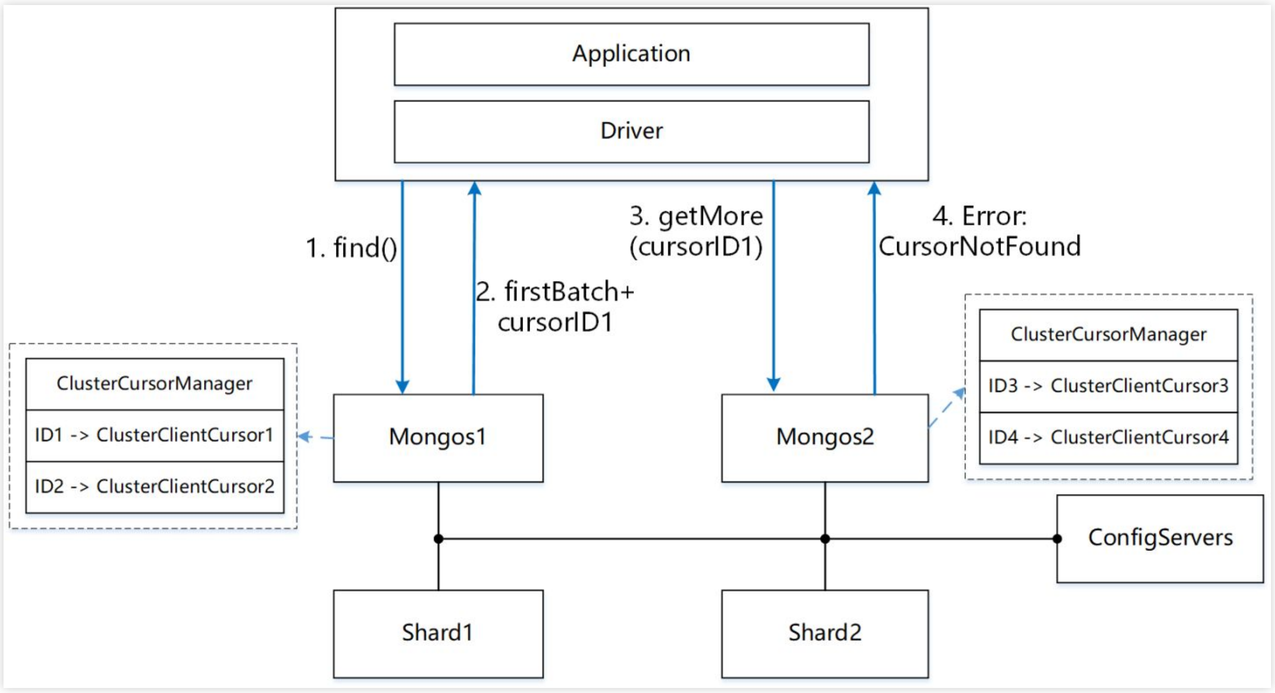
If MongoDB could not return all the `find` results at a time, it will first return the first batch of data and `cursorID`, through which the client constantly calls `getMore` to iterate the remaining data. Therefore, a batch scan request may correspond to one `find` request and multiple `getMore` requests and associate the `find` request with returned `getMore` results through `cursorID`.

Each mongos node maintains a global ClusterCursorManager in the memory and maintains the mapping between `cursorID` and `ClusterClientCursor` through HashMap. `cursorID` is a random int64 number, and `ClusterClientCursor` maintains the execution plan, current status, and other information of a request.

If the query result cannot be returned at a time (for example, it exceeds the limit of 16 MB), a `cursorID` other than `0` will be generated, which will be registered in the ClusterCursorManager together with `ClusterClientCursor` itself.

If the client needs subsequent results, it can send `getMore` requests carrying the returned `cursorID` value, and mongos will find the cached ClusterClientCusor, continue to execute the query plan, and return subsequent results. The ID and cursor information independently exist on each mongos node.

Therefore, be sure to send the `find` request and the associated `getMore` requests to the same mongos node. If a `getMore` request is sent to a different mongos node, the cursor cannot be found, and the `CursorNotFound` error will be returned:



Transaction operation issue

MongoDB 4.2 supports distributed transactions, so you can connect to a mongos node to initiate transaction operations. You can perform multiple read/write operations between `startTransaction` and `commitTransaction` / `abortTransaction` . mongos records the metadata carried in each request in a transaction such as `logicalSessionId` and `txnId` to maintain the context. Therefore, MongoDB is designed to guarantee that each operation in a transaction is sent to the same mongos node.

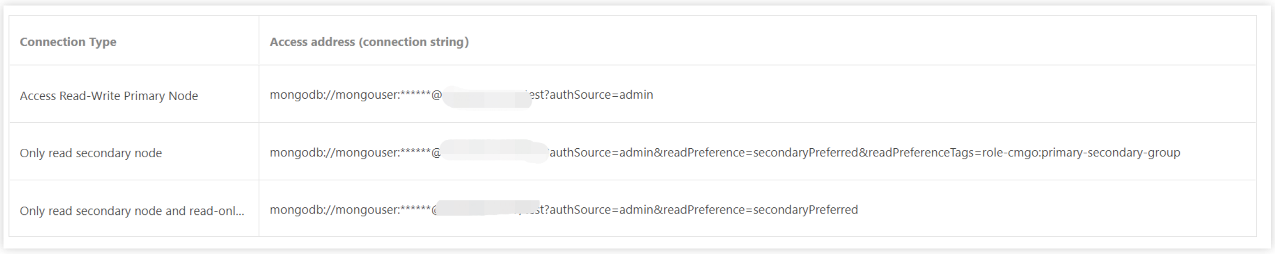
TencentDB for MongoDB load balancing policy

To address the `getMore` problem during batch scan and transaction operation issue, the TencentDB for MongoDB load balancing hash policy balances traffic based on the IP information of the accessing client (generally a CVM instance); that is, all requests from the same source IP will be routed to the same mongos node, which ensures that `getMore` and transaction operations are processed in the same context.
This policy works well for the production environment with many access IPs. However, when there are only a few access IPs, particularly in stress test scenarios, it tends to cause mongos load imbalance.

Solution to mongos Load Imbalance

If you don't want to use the default TencentDB for MongoDB load balancing policy, you can enable the mongos access address. Under the current VIP of the instance, the system will bind different vports to different mongos nodes, so you can flexibly control the distribution of mongos requests. In addition, if a mongos node fails, the system will bind

its VIP and vport to a new mongos process, which will not change the VIP and vport or affect the original load balancer address. For detailed directions, see Enabling Mongos Access Address.
After enabling the mongos access address, you can view it in **Access Address** in the **Network Configuration** section on the **Instance Details** page in the console, which displays connection strings of different connection types. Each connection string is configured with all mongos nodes in the instance, and the `authSource`, `readPreference`, and `readPreferenceTags` parameters are used to determine which type of node to access:



Connection Type	Access address (connection string)
Access Read-Write Primary Node	mongodb://mongouser:******@[redacted]est?authSource=admin
Only read secondary node	mongodb://mongouser:******@[redacted]authSource=admin&readPreference=secondaryPreferred&readPreferenceTags=role-cmgo:primary-secondary-group
Only read secondary node and read-onl...	mongodb://mongouser:******@[redacted]st?authSource=admin&readPreference=secondaryPreferred

To implement traffic balancing, you can directly copy a connection string and configure it in the application used to connect the client to the database SDK to access the corresponding mongos nodes. For more information on the connection method, see Connecting to TencentDB for MongoDB Instance. However, as a connection string is long, proceed with caution.
Note that if you configure all mongos nodes in a connection string, after you adjust the number of mongos nodes in the instance, you need to update the connection string in the application.

Considerations for Using Shard Clusters

Last updated：2024-01-15 14:49:55

A sharded cluster is distributed MongoDB database architecture. Compared with replica sets, sharded clusters evenly distribute data across shards, which not only greatly increases the capacity, but also distributes the read/write workload across shards, effectively solving the performance bottleneck of replica sets. The trade off is increased complexity in architecture. This document lists some issues you should take note of when using TencentDB for MongoDB sharded clusters.

Sharded Cluster Components

A MongoDB sharded cluster consists of the following components:
shard: each shard contains a subset of the sharded data, and can be deployed as a replica set.
mongos: the mongos acts as a query router, providing an interface between client applications and the sharded cluster.
config servers: config servers store metadata and configuration settings for the cluster, including permission and authentication configurations.

Sharding Strategies and Performance Impact

MongoDB sharded clusters supports 3 sharding strategies for data distribution: ranged sharding, hashed sharding, and zone/tag-based sharding. Each sharding strategy results in different performances when used in different functions.

Ranged sharding
Advantages: good performance in shard key range-based query and read
Disadvantages: possibly uneven data distribution with hot spots
Hashed sharding
Advantages: even data distribution, good write performance, and suitable for high concurrency use cases such as logging and Internet of Things
Disadvantages: low range-based query efficiency
Zone/tag-based sharding
Data which has a natural distinction, such as geographical or time distinction, can be distinguished by tags.
Advantages: good data distribution

Choosing Shard Key

The shard key is a field in a document used for routing queries.
The choice of shard key can have a great impact on sharding efficiency due to the following factors:
Cardinality
Choose a shard key with high cardinality. A shard key with low cardinality will have a small number of available values, which constrains the maximum number of chunks. As the data increases, the chunk size grows, making it difficult to migrate chunks in horizontal scaling.
For example: if you use age as the cardinality, there will only be at most 100 available shard key values. As data increases, a chunk will soon store too much data and grow beyond the specified chunk size and becoming a jumbo chunk. Such a chunk cannot be migrated, resulting in uneven data distribution and performance bottlenecks.
Distribution
Choose a shard key whose values are distributed evenly; otherwise, some chunks may contain huge volumes of data, which will result in uneven data distribution and performance bottlenecks.
Shard key-based query
Use the shard key as the query condition. Mongos can locate the specific shard according to the shard key; otherwise, mongos needs to distribute the query to all shards and wait for their responses.
Monotonically changing shard keys (not recommended)
A monotonically increasing shard key leads to fewer migrations of data, but all new inserts are routed to the last chunk which has to keep migrating as it grows. The same problem will occur when a monotonically decreasing shard key is used.
Consider all of the above factors when choosing a shard key to reduce the negative effects of chunk migration and optimize overall performance.

Modifying shard key value

Prior to MongoDB 4.2, the shard key field value of a document cannot be changed.
Starting from MongoDB 4.2, unless the shard key field is an immutable `_id` field, you can update its value in the following methods:

Command	Method

update with multi: false	db.collection.replaceOne()
	db.collection.updateOne()
	db.collection.update() with multi: false

findAndModify	db.collection.findOneAndReplace()
	db.collection.findOneAndUpdate()
	db.collection.findAndModify()

-	db.collection.bulkWrite()
	Bulk.find.updateOne()

	If the shard key modification causes the document to be moved to another shard, you cannot specify multiple shard keys for batch modification; that is, the batch size must be `1` . Otherwise, you can specify multiple shard keys for batch modification.

Notes on shard key modification:
You must perform it in a transaction or on mongos in a retryable write mode. Do not perform it directly on shards.
You must include an equality condition in the complete shard key of the query filter. For example, if you use `{country:1, userid:1}` as the shard key in a shard collection, to update the document shard key, you must include `country:, userid:` in the query filter. You can also include other fields in the query as needed.

Balancing and Related Parameters

MongoDB sharded cluster partitions data into chunks. The background process `balancer` monitors the number of chunks on each shard and migrates the chunks between shards to balance the load on each shard server.
Note:
The system creates an initial chunk, and the chunk size is 64 MB by default.
Because chunk migrations have an impact on cluster read/write performance, you can set the balancing window to avoid the impact during business peak, or run commands to disable balancing.
Commands to manage balancing are described as follows. If you do not have the permission to run the commands, submit a ticket for further assistance.
Checking whether balancing is enabled for MongoDB sharded cluster

```
mongos> sh.getBalancerState()
true
```

You can also run `sh.status()` to check balancing status.

Checking whether there are data in migration

```
mongos> sh.isBalancerRunning()
false
```

Setting the balancing window

Modify the balancing window:

```
db.settings.update(
   { _id: "balancer" },
   { $set: { activeWindow : { start : "<start-time>", stop : "<stop-time>" } } },
   { upsert: true }
)
```

Delete the balancing window:

```
use config
db.settings.update({ _id : "balancer" }, { $unset : { activeWindow : true } })
```

Disabling balancing

By default, the balancer can migrate a chunk whenever it needs to be migrated. You can run the following commands to disable balancing:

```
sh.stopBalancer()
sh.getBalancerState()
```

Run the following commands to query whether a migration process is running after balancing is disabled:

```
use config
while( sh.isBalancerRunning() ) {
          print("waiting...");
          sleep(1000);
}
```

Enabling balancing

Run the following command to enable balancing again:

```
sh.setBalancerState(true)
```

When the driver does not support `sh.startBalancer()` , run the following commands to enable balancing again:

```
use config
db.settings.update( { _id: "balancer" }, { $set : { stopped: false } } , { upsert:
```

Balancing on a collection

Disable balancing on a collection:

```
sh.disableBalancing("students.grades")
```

Enable balancing on a collection:

```
sh.enableBalancing("students.grades")
```

Check whether balancing is enabled on a collection:

```
db.getSiblingDB("config").collections.findOne({_id : "students.grades"}).noBalance
```

Sample of Reading and Writing Data in MongoDB Instance

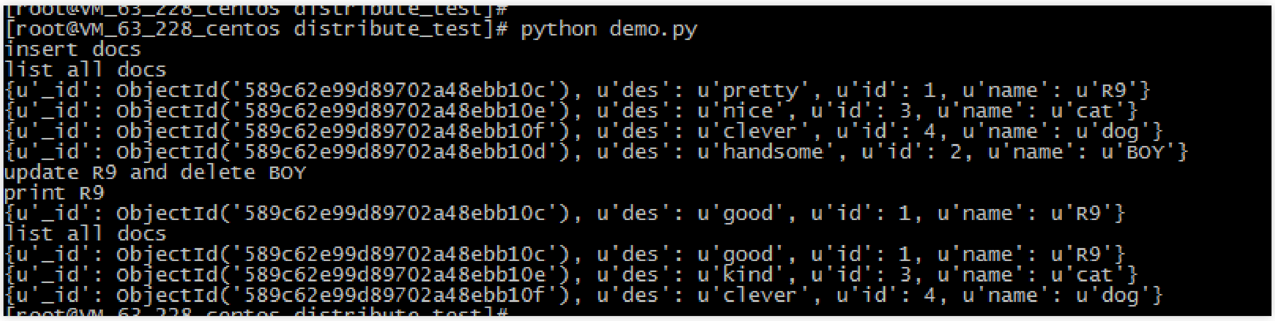
Last updated：2024-01-15 14:49:55

This document uses Python sample code to demonstrate the basic data read/write operations in a TencentDB for MongoDB sharded cluster. Create a sharded cluster instance in the console first and then add the following codes in the service:

Sample code:

```
#!/usr/bin/python
import pymongo
import random


mongodbUri = 'mongodb://mongouser:1234567a@10.66.153.111:27017/admin'

client = pymongo.MongoClient(mongodbUri)
db = client.test

if 'num' in db.collection_names():
    db.drop_collection('num')

#create database and shardkey,shardkey is name
db_admin=client.admin
db_admin.command('enableSharding', 'test')
db_admin.command('shardCollection', 'test.num', key = {'name':1})

#insert data
print 'insert docs'
db.num.insert_one({'id':1, 'name':'R9', 'des':'pretty'})
db.num.insert_one({'id':2, 'name':'BOY', 'des':'handsome'})
db.num.insert_one({'id':3, 'name':'cat', 'des':'nice'})
db.num.insert_one({'id':4, 'name':'dog', 'des':'clever'})
print 'list all docs'
for i in db.num.find(): print i

#insert update doc
print 'update R9 and delete BOY'
db.num.update_one({"name":"R9"},{"$set":{"des":"good"}})
db.num.delete_one({"name":"BOY"})
db.num.update_one({"id":3}, {"$set":{"des":"kind"}})

print 'print R9'
for i in db.num.find({"name":"R9"}): print i
print 'list all docs'
for i in db.num.find(): print i
```

Execution result:



```
[root@VM_63_228_centos distribute_test]#
[root@VM_63_228_centos distribute_test]# python demo.py
insert docs
list all docs
{u'_id': ObjectId('589c62e99d89702a48ebb10c'), u'des': u'pretty', u'id': 1, u'name': u'R9'}
{u'_id': ObjectId('589c62e99d89702a48ebb10e'), u'des': u'nice', u'id': 3, u'name': u'cat'}
{u'_id': ObjectId('589c62e99d89702a48ebb10f'), u'des': u'clever', u'id': 4, u'name': u'dog'}
{u'_id': ObjectId('589c62e99d89702a48ebb10d'), u'des': u'handsome', u'id': 2, u'name': u'BOY'}
update R9 and delete BOY
print R9
{u'_id': ObjectId('589c62e99d89702a48ebb10c'), u'des': u'good', u'id': 1, u'name': u'R9'}
list all docs
{u'_id': ObjectId('589c62e99d89702a48ebb10c'), u'des': u'good', u'id': 1, u'name': u'R9'}
{u'_id': ObjectId('589c62e99d89702a48ebb10e'), u'des': u'kind', u'id': 3, u'name': u'cat'}
{u'_id': ObjectId('589c62e99d89702a48ebb10f'), u'des': u'clever', u'id': 4, u'name': u'dog'}
[root@VM_63_228_centos distribute_test]#
```

Methods for Importing and Exporting Data Based on CVM Connected with MongoDB

Last updated：2024-05-07 10:11:25

You can use a CVM instance to connect to TencentDB for MongoDB for data import and export. Be sure to use the latest MongoDB client suite. For detailed directions, see Connecting to TencentDB for MongoDB Instance.

Note:

The `local` database mainly stores metadata such as configuration information of the replica set and oplog, and the `admin` database mainly stores information such as users and roles. In order to prevent data disorder and authentication failures, TencentDB for MongoDB prohibits importing `local` and `admin` databases into an instance.

Export and Import Commands

MongoDB provides two sets of official tools for data import and export:

mongodump and mongorestore

mongoexport and mongoimport

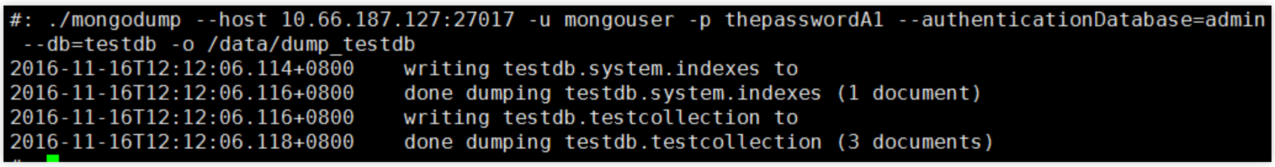
mongodump and mongorestore

mongodump and mongorestore are generally used to export and import an entire database, as they manipulate data in BSON format, which is more efficient when a large number of `dump` and `restore` operations are performed.

The export command for mongodump is as follows:

```
mongodump --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authentication
```

If the following information is output, the command is executed successfully:

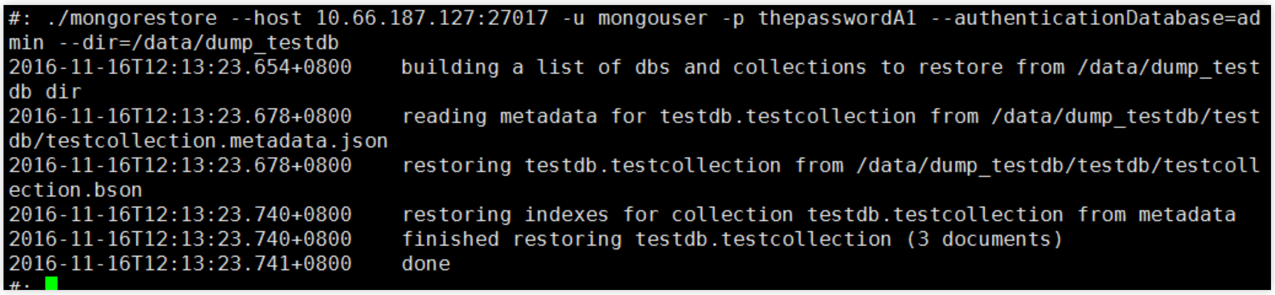


```
#: ./mongodump --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticationDatabase=admin
 --db=testdb -o /data/dump_testdb
2016-11-16T12:12:06.114+0800    writing testdb.system.indexes to
2016-11-16T12:12:06.116+0800    done dumping testdb.system.indexes (1 document)
2016-11-16T12:12:06.116+0800    writing testdb.testcollection to
2016-11-16T12:12:06.118+0800    done dumping testdb.testcollection (3 documents)
```

The import command for mongorestore is as follows:

```
mongorestore --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticat
```

If the following information is output, the command is executed successfully:



```
#: ./mongorestore --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticationDatabase=ad
min --dir=/data/dump_testdb
2016-11-16T12:13:23.654+0800    building a list of dbs and collections to restore from /data/dump_test
db dir
2016-11-16T12:13:23.678+0800    reading metadata for testdb.testcollection from /data/dump_testdb/test
db/testcollection.metadata.json
2016-11-16T12:13:23.678+0800    restoring testdb.testcollection from /data/dump_testdb/testdb/testcoll
ection.bson
2016-11-16T12:13:23.740+0800    restoring indexes for collection testdb.testcollection from metadata
2016-11-16T12:13:23.740+0800    finished restoring testdb.testcollection (3 documents)
2016-11-16T12:13:23.741+0800    done
#:
```

mongoexport and mongoimport

mongoexport and mongoimport are generally used to export and import a single set, as they manipulate data in JSON format, which features a higher readability.

The export command for mongoexport is as follows:

```
mongoexport --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticati
```

In addition, you can also include the `-f` parameter to specify a desired field or the `-q` parameter to specify a query condition so as to restrict the data to be exported.

The import command for mongoimport is as follows:

```
mongoimport --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticati
```

Authentication Methods and Parameters

As described in the connection sample, TencentDB for MongoDB provides two usernames `rwuser` and `mongouser` by default to support the `MONGODB-CR` and `SCRAM-SHA-1` authentication methods, respectively.

For `mongouser` and all new users created in the console, follow the above directions to use the import and export tools.
For `rwuser` , the parameter `--authenticationMechanism=MONGODB-CR` should be included in each command.
Sample for mongodump:

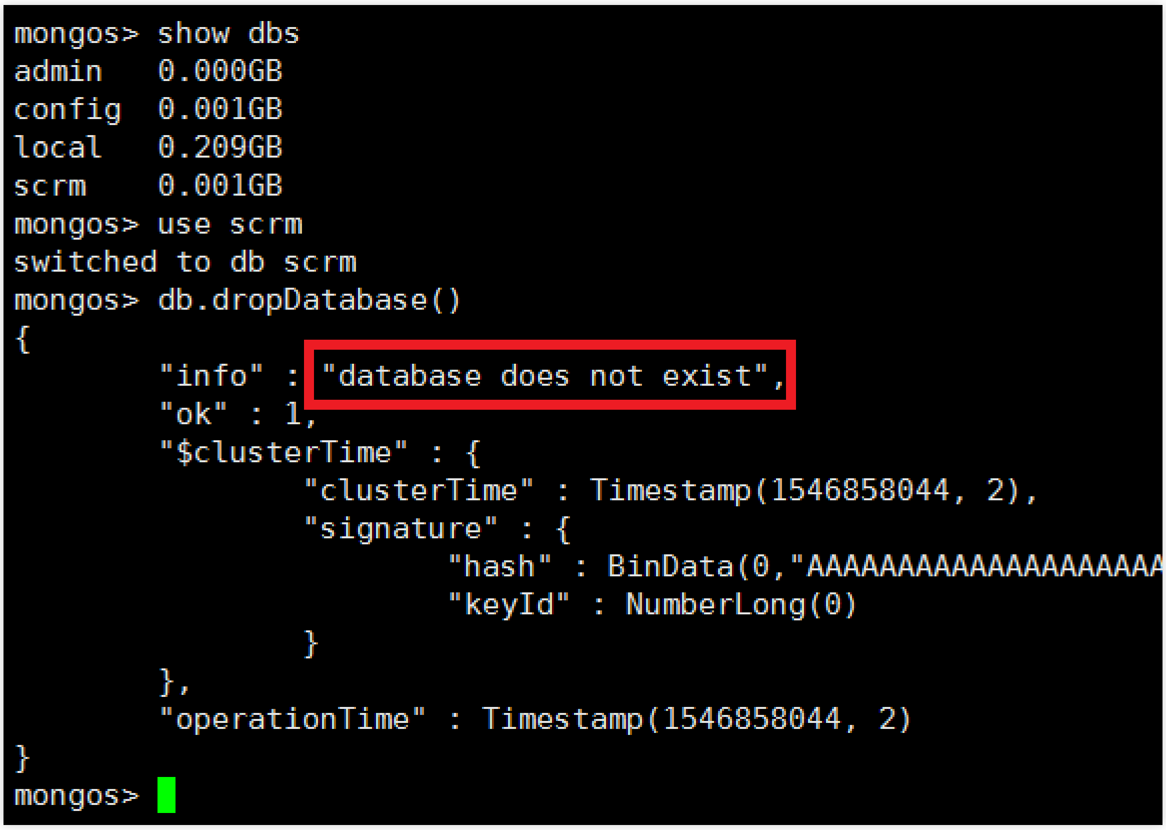
```
mongodump --host 10.66.187.127:27017 -u rwuser -p thepasswordA1 --authenticationDat
```

What to Do for Errors of Repeated Instance Creation and Deletion of Databases with the Same Names?

Last updated：2024-05-07 21:09:14

Problem Description

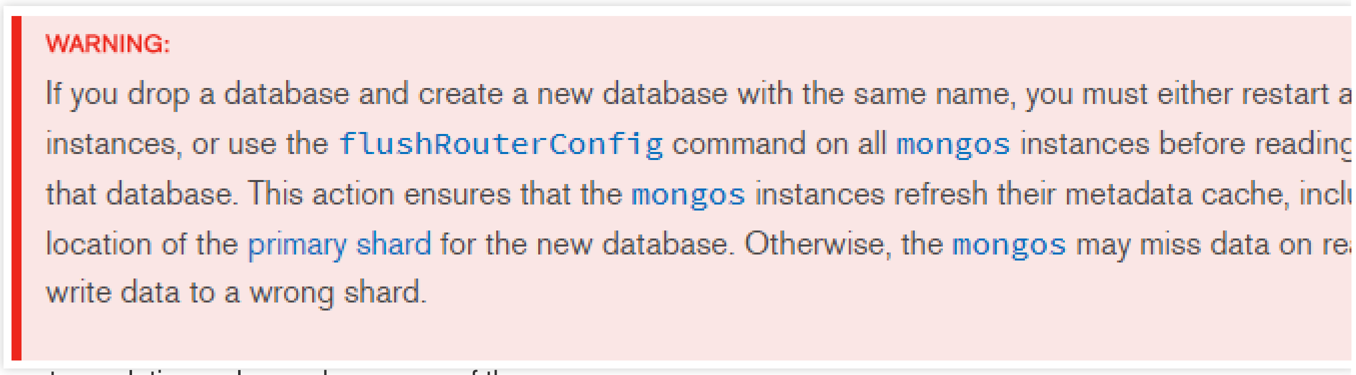
When repeatedly dropping a database and then creating a database with the same name in MongoDB 3.6, errors such as 'database does not exist' may occur during read-write operations or when dropping the database, as shown below:



```
mongos> show dbs
admin   0.000GB
config  0.001GB
local   0.209GB
scrm    0.001GB
mongos> use scrm
switched to db scrm
mongos> db.dropDatabase()
{
        "info" : "database does not exist",
        "ok" : 1,
        "$clusterTime" : {
                "clusterTime" : Timestamp(1546858044, 2),
                "signature" : {
                        "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAA
                        "keyId" : NumberLong(0)
                }
        },
        "operationTime" : Timestamp(1546858044, 2)
}
mongos>
```

Solution

This is a common issue that may arise if mongos has not refreshed its metadata cache. For more information, please refer to the official description as illustrated below:



WARNING:

If you drop a database and create a new database with the same name, you must either restart a
instances, or use the `flushRouterConfig` command on all `mongos` instances before reading
that database. This action ensures that the `mongos` instances refresh their metadata cache, incl
location of the primary shard for the new database. Otherwise, the `mongos` may miss data on re
write data to a wrong shard.

There are two solutions; please choose one of them:

1. Restart mongos, which can be done in the Console Instance List.
2. Alternatively, run the flushRouterConfig command, which comes with a detailed explanation.

Troubleshooting MongoDB Connection Failures

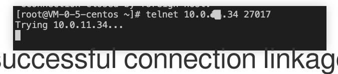
Last updated：2024-04-12 14:15:50

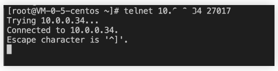
Overview

You can use a CVM instance to connect to the automatically assigned private network address of a TencentDB for MongoDB instance as instructed in Connecting to TencentDB for MongoDB Instance. If connection fails, troubleshoot the problem as follows:

Troubleshooting

No.	Possible Cause	Troubleshooting Method	Solution

1	The CVM and TencentDB for MongoDB instance cannot be interconnected over the private network.
	The CVM and TencentDB for MongoDB instance are not in the same VPC. To interconnect over the private network, the two instances must be under the same account and in the same VPC.
	The security group is incorrectly configured.
		1. Log in to the CVM console and view the CVM network information in **Instance Configuration** in the instance list.
		2. Log in to the TencentDB for MongoDB console and view the MongoDB network information in the instance list. For more information, see Viewing Instance Details.
		3. Check whether the CVM and TencentDB for MongoDB instances are in the same network.
		4. Log in to the CVM instance and run `telnet 10.x.x.34 27017`. Check whether the TencentDB for MongoDB Network port can be accessed normally.
		The failed connection linkage is as shown below:
		
		`[root@VM-0-5-centos ~]# telnet 10.0.[redacted].34 27017`
		`Trying 10.0.11.34...`
		The successful connection linkage is as shown below:
			The following conditions may cause a connection failure due to network issues.
			The CVM instance is in a VPC, but the TencentDB for MongoDB instance is in the classic network. We recommend that you switch the network type of the MongoDB instance to VPC. For more information, see Switching Instance Network.
			The CVM instance is in the classic network, but the TencentDB for MongoDB instance is in a VPC. We recommend that you switch the network type of the CVM instance from classic network to VPC. For more information, see Switching to VPC.
			The CVM and TencentDB for MongoDB instances are in different VPCs in the same region. We recommend that you migrate the MongoDB instance to the VPC of the CVM instance. For more information, see Switching Instance Network.

	To use the CVM instance to access the MongoDB instance, you need to configure an outbound rule in the security group of the CVM instance. If the target of the outbound rule isn't "0.0.0.0/0" and the protocol port isn't "ALL", the IP and port of the MongoDB instance should be added to the outbound rule.
	To use the CVM instance to access the MongoDB instance, you need to configure an inbound rule in the security group of the MongoDB instance. If the source of the inbound rule isn't "0.0.0.0/0" and the protocol port isn't "ALL", the IP and port of the CVM instance should be added to the inbound rule.
		
		`[root@VM-0-5-centos ~]# telnet 10.[redacted]34 27017`
		`Trying 10.0.0.34...`
		`Connected to 10.0.0.34.`
		`Escape character is '^]'.`
			The CVM and TencentDB for MongoDB instances are in different VPCs in different regions. We recommend that you create a CCN instance between the two VPCs.
			The CVM and TencentDB for MongoDB instances are in different VPCs under different accounts. We recommend that you create a CCN instance between the two VPCs.

			The security group of the CVM instance is incorrectly configured. 1. Go to the Security Group page in the CVM console, find the security group bound to the CVM instance in the security group list, and click its name to enter its details page. 2. On the **Outbound rule** tab, click **Add Rule**. **Type**: Select **Custom**. **Target**: Enter the IP or IP range of your MongoDB instance. **Protocol Port**: Enter the private network port of the MongoDB instance. **Policy**: Select **Allow**. **The security group of the MongoDB instance is incorrectly configured.** 1. Log in to the Security Group console, find the security group bound to the MongoDB instance in the security group list, and click its name to enter its details page. 2. On the **Inbound rule** tab, click **Add Rule**. Enter the allowed IP address/range and port, then select **Allow**. **Type**: Select **Custom**. **Source**: Enter the IP or IP range of your CVM instance. **Protocol Port**: Enter the private network port of the MongoDB instance. **Policy**: Select **Allow**.
2	The username or	When you try to log in and access a	Log in to the TencentDB for MongoDB

	password is incorrect.	database instance, the CVM instance displays an error message like `Error: Authentication failed` , indicating that the account name or password is incorrect.	console, go to the **Instance Details** page, select the **Database Management** tab to enter the **Account Management** page, and view the information of all accounts of the current database or reset passwords. For detailed directions, see Account Management.
3	Database access password contains symbols such as % and @.	The driver or client such as MongoShell cannot automatically escape % and @ in the password, causing conflicts between these symbols and connection string address. This results in incorrect username or password. The error message may show as 'Password cannot properly be URL decoded' or 'Error: Authentication failed`.	Special characters in the access password should be handled according to the following escape rules: Exclamation mark "!": escaped as %21 at "@": escaped as %40 Warning Sign "#": escaped as %23 Percent sign "%": escaped as %25 Caret "^": escaped as %5e Asterisk "*": escaped as %2a Left parenthesis "(": escaped as %28 Right parenthesis ")": escaped as %29 Underscore "_": escaped as %5f For example, if the original password is ^%@132121a, then the password after escape should be: ^%25%40132121a.
4	The mongo shell version is too old.	Log in to the CVM instance and run `mongo --version` to view the version information.	To ensure successful authentication, install mongo shell 3.0 or later. For detailed directions, see Install MongoDB.
5	The authentication database is not correctly used in the connection string of the client. For users created in the console: TencentDB for MongoDB uses the `admin` database as the authentication	For users created in the console: Check whether the connection string configured in the client program contains `/admin` or `authSource=admin` . For users created on the command line: Check whether the authentication database in the connection string has the correct database name.	Log in to the TencentDB for MongoDB console. In the **Network Configuration** section on the **Instance Details** page, directly copy the URI connection string of the default account. For other accounts, change it to the correct authentication database and try connecting again. For more information, see Connecting to TencentDB for MongoDB Instance. If the problem persists, submit a ticket for assistance.

	database during login authentication, so the port in a URI must be followed by `/admin` to specify it. After authentication, you can switch to a specific business database for reads/writes. For users created on the command line: Directly specify the corresponding authentication database. For example, for users created under the `test` database, specify `test` as the authentication database.		

6	There are operations that block other requests. For example, if an index creation operation is performed in the foreground during peak hours (with the `background` option being `false`), the operation will block all other	Check the index creation method.	Create indexes in the background. However, it takes more time to create indexes in this way. For specific options for index creation, see db.collection.createIndex(). You can also run the `currentOp` command to check the progress of index creation. The specific command is as follows:
			`db.currentOp(`
			`    {`
			`      $or: [`
			`        { op: "command",`
			`"query.createIndexes": {`
			`$exists: true } },`
			`        { op: "insert", ns:`

	operations and cause requests to be locked until the index is created in the foreground. For options to create an index, see Index Builds on Populated Collections.		`/\\.system\\.indexes\\b/ }`
			`]`
			`}`
			`)`

7	Check whether the number of connections to the client has reached the upper limit. Each instance has a limit on the number of connections, and if it is exceeded, new connections cannot be made.	Log in to the TencentDB for MongoDB console, go to the **Database Management** page, select the **Manage Connection** tab, and view the instance's **Max Connections**, **Real-time connections**, and **Connection Utilization**. For more information, see Connection Management.	To troubleshoot high connection utilization, see Troubleshooting High Connection Utilization.

Note:

The authentication database for users created in the console is the `admin` database, so the users need to specify `admin` as the authentication database during login. The users created with the command line, such as those created under the `test` database, need to specify `test` as the authentication database.

Performance Fine-Tuning

Troubleshooting High Connection Utilization

Last updated：2024-01-15 14:49:55

Problem description

The MongoDB service is provided in a mode where each network connection is processed by a single thread (one-thread-per-connection). Too many network connections generate too many threads, which will increase context switch and memory overheads. Establishing connections and performing authentication for each request greatly affect performance. Therefore, limiting the number of connections to an instance and releasing connections promptly after use is a prerequisite for ensuring the database stability.

Log in to the TencentDB for MongoDB console, select the **System Monitoring** page, and view the trend chart of the instance monitoring metric **Connection Percentage**. This metric refers to the ratio of the number of connections in the current cluster to the maximum number of connections. If the maximum number of connections is reached, the connection response will slow down, and even connections will fail. Therefore, when the connection utilization exceeds 85%, check and handle it in time.

Troubleshooting

No.	Possible Cause	Troubleshooting Method	Solution
1	If a connection pool is used, a lot of connection resources may be used due to the unreasonable configuration of connection parameters.	Check whether the configuration parameters of the client connection pool are suitable for the business scenario.	Configure connection pool parameters as detailed below.
2	There are many connections without actual service requests on the business side.	Use **DBbrain's performance optimization** feature to check whether the connections from clients to the business are actually needed by the business on the **Real-Time Session** page.	Use **DBbrain's performance optimization** feature to directly kill unnecessary clients on the **Real-time Session** page. For detailed directions, see Performance Optimization.

3	There are a high number of slow queries that occupy connections.	1. Use **DBbrain's performance optimization** feature to query the records, execution statistics, and views of all current slow logs in the database on the **Slow SQL Analysis** page. For more information, see Performance Optimization. 2. Log in to the TencentDB for MongoDB console, click the instance ID to enter the **Instance Details** page, select the **Database Management** tab, click **Slow Log Query**, and select **Query statistics** to query slow queries. For detailed directions, see Slow Log Management.	For slow queries, perform index optimization. Use DBbrain's index recommendation feature to select the optimal index. Improve the database performance as instructed in Optimizing Indexes to Break Through Read/Write Performance Bottlenecks.
4	There are unreleased connections due to connection leaks.	Restart the instance. All instance connections will be interrupted at the moment of restarting mongos, but the business can be directly reconnected. Therefore, restarting mongos will not continuously affect the business. If the number of business connections increases rapidly and the connection utilization reaches 100% again after the restart, it indicates that the business does have a large number of valid connections and there are no connection leaks. For detailed	Directly increase the number of connections in the console to temporarily sustain business surges. For detailed directions, see Connection Management. Adjust the instance configuration. For replica set instances, upgrading the CPU and memory specifications of mongod can increase the maximum number of connections of the instance. For more information, see Adjusting Mongod Node Specification. For sharded clusters, upgrade the mongos node specification or add more shards as instructed in Adjusting Mongos Node Specification or Adjusting Shard Quantity respectively.

		directions, see Restarting Instance	
5	The current instance quota becomes insufficient as the business surges.	Refer to item 2.	

Suggestions on connection pool usage

The following takes the Go language as an example to describe the parameters that need to be configured when the client connects to the database through a connection pool. For other programming languages, find the corresponding connection pool parameters for configuration. For more information, see Start Developing with MongoDB.

Parameter	Unit	Description	Configuration Suggestion
maxPoolSize	-	Configure the maximum number of connections that each client can apply for in the connection pool.	The product of this parameter and the number of clients must be less than the maximum number of connections of the instance; otherwise, if there are excessive connections, new connections cannot be established.
minPoolSize	-	Configure the minimum number of connections that each client can apply for in the connection pool.	The product of this parameter and the number of clients must be less than the maximum number of connections of the instance; otherwise, if many new connections need to be established during business surges, backend instance resources will become insufficient.
socketTimeoutMS	ms	Configure the wait timeout period for sending and receiving sockets. The default value is `0` , indicating not to time out.	Set this parameter according to your actual business scenario so as to avoid that when a MongoDB server failure causes a primary/secondary switch, the client will not be able to receive the packet of the server response, resulting in prolonged use of resources by this invalid connection.
maxIdleTimeMS	ms	Configure the maximum time an idle connection exists before it is deleted or closed.	Set this parameter to be within 1 hour to avoid idle connections occupying connection resources all the time.

heartbeatFrequencyMS	ms	Configure the frequency at which the client sends heartbeats to the server. This parameter is used by the client to periodically check the status of the connection to the backend database.	Set this parameter to be within 10s, so that the status of the server can be checked as soon as possible to avoid generating invalid connections.

Troubleshooting High CPU Utilization

Last updated：2024-01-15 14:49:55

Problem Description

The **CPU monitoring** metrics stay high on the **System Monitoring** tab on the **Instance Details** page of an instance in the TencentDB for MongoDB console.

Troubleshooting

No.	Possible Cause	Cause Analysis	Troubleshooting Method	Solution
1	Non-persistent connections are established frequently.	A large number of instance resources are used to process frequent non-persistent connections, resulting in a high CPU utilization and many connections. However, the QPS (cluster access requests per second) doesn't meet expectations.	Use DBbrain's **performance optimization** feature to intelligently analyze the real-time session statistics and data of the database instance and check whether there is a sudden increase in the number of connections. For detailed directions, see Real-Time Session.	Change non-persistent connections to persistent connections. For more information on parameters for different programming languages, see Start Developing with MongoDB.
2	There are many unexpected time-consuming requests on the business side.	Many unexpected or malicious requests may lead to excessive waste of CPU resources.	Use **DBbrain's performance optimization** feature to check whether the requests from all connected clients to the business are actually needed by the business on the **Real-Time Session** page.	Use **DBbrain's performance optimization** feature to directly kill unexpected requests on the **Real-time Session** page. For detailed directions, see Real-Time Session. On the **Performance Optimization** page in the **DBbrain** console, select the **SQL Throttling** tab, and create

				SQL throttling tasks to control the database requests and SQL concurrency by setting the **SQL Type**, **Max Concurrency**, **Throttling Duration**, and **SQL Keyword**. For detailed directions and use cases, see SQL Throttling.
3	There are highly complex requests.	Highly complex requests may cause excessive usage of CPU resources for the following reasons. Sorting operation: Sorting during the query itself will consume a lot of CPU resources. Full-collection scan: If the database does not have an index, the request may perform a full-collection scan, putting a high pressure on CPU and IO resources and resulting in performance drop. Unreasonable index: A large number of physical reads and logical reads may be caused due to insufficiently covered fields or unreasonably ordered index fields, using more CPU resources.	Highly complex query requests are usually time-consuming and generate slow logs accordingly. You can use DBbrain's **performance optimization** feature to perform **slow log analysis** and check whether there are relevant keywords in the slow log list. For detailed directions, see Slow SQL Analysis. Sorting keywords: Sort, hasSortStage, etc. Full-collection scan keywords: `COLLSCAN` indicates that a full-collection scan is performed for the query. `docsExamined` indicates the number of documents scanned. The larger the `docsExamined` value, the more the documents scanned, and the more the CPU resources used. Unreasonable index keywords:	For slow queries, perform index optimization. Use **DBbrain's** index recommendation feature to select the optimal index. Improve the database performance as instructed in Optimizing Indexes to Break Through Read/Write Performance Bottlenecks.
4				
5				

			`IXSCAN` indicates that an index scan is performed. `keysExamined` indicates the number of index entries scanned. The larger the `keysExamined` value, the more the index entries scanned, and the more the CPU resources used.	
6	There are insufficient resources as the business grows.	The CPU specification of the current instance is insufficient as the business grows and the data volume increases.	Log in to the TencentDB for MongoDB console and click the ID of the target instance to enter the **Instance Details** page. Then, select the **System Monitoring** tab and check whether the **Total Requests** (QPS) metric of the instance is obviously higher.	Adjust the instance specifications. Mongod node: Upgrade the CPU and memory configuration of mongod as instructed in Adjusting Mongod Node Specification. Mongos nodes: If mongos hits the bottleneck, upgrade the mongos node specification or add more mongos nodes as instructed in Adjusting Mongos Node Specification or Adding Mongos Node respectively.

Memory Tunning Method

Last updated：2024-01-15 14:49:55

Memory tuning aims to achieve an optimal balance between machine resources and performance by providing a sufficient and stable memory usage to ensure normal system performance rather than simply reducing the memory usage. This document describes why and how TencentDB for MongoDB memory is occupied. It also provides troubleshooting methods and solutions for you to tune the database performance in time.

Storage Engine Memory

The cache size of WiredTiger (TencentDB for MongoDB storage engine) cannot exceed 60% of the actual requested instance memory specification. If the storage engine cache reaches 95% of the limit, the instance load will be high. If the dirty data cache percentage reaches 20% in the storage engine, user threads will be blocked.

Troubleshooting

You can view the engine memory utilization by running the `db.serverStatus().wiredTiger.cache` command in MongoDB Shell. The size of the cached data is indicated by the value after the returned `bytes currently in the cache` as shown below:

```
{
    ......
    "bytes belonging to page images in the cache":6511653424,
    "bytes belonging to the cache overflow table in the cache":65289,
    "bytes belonging to page images in the cache":8563140208,
    "bytes dirty in the cache cumulative":NumberLong("369249096605399"),
    ......
}
```

You can view the percentage of the dirty data cached in the storage engine in real time on the MongoStatus feature of the **DBbrain** performance trends. For more information, see mongostat.

Usage suggestions

If Cache Dirty of the storage engine stays above 20%, you can troubleshoot as follows:

A. Control the volume of data written per unit time.

B. Expand instance memory as instructed in Adjusting Mongod Node Specification.

C. Increase the number of threads for cleaning dirty data. The higher the number of threads, the more instance resources will be consumed. Select the value with caution. `Threads_max=4,threads_min=1` is set by default, but you can adjust it as follows:

```
db.runCommand({"setParameter":1, "wiredTigerEngineRuntimeConfig":"eviction=(threads
```

Note:

You can adjust the number of threads for cleaning the dirty data through `db.runCommand`, but this operation is only suitable for replica set architecture 4.0 and later.

If Cache Used of the storage engine stays above 95%, you can troubleshoot as follows:

a. Analyze the slow log in the database by using **DBbrain's** Slow SQL Analysis.

B. Expand instance memory as instructed in Adjusting Mongod Node Specification.

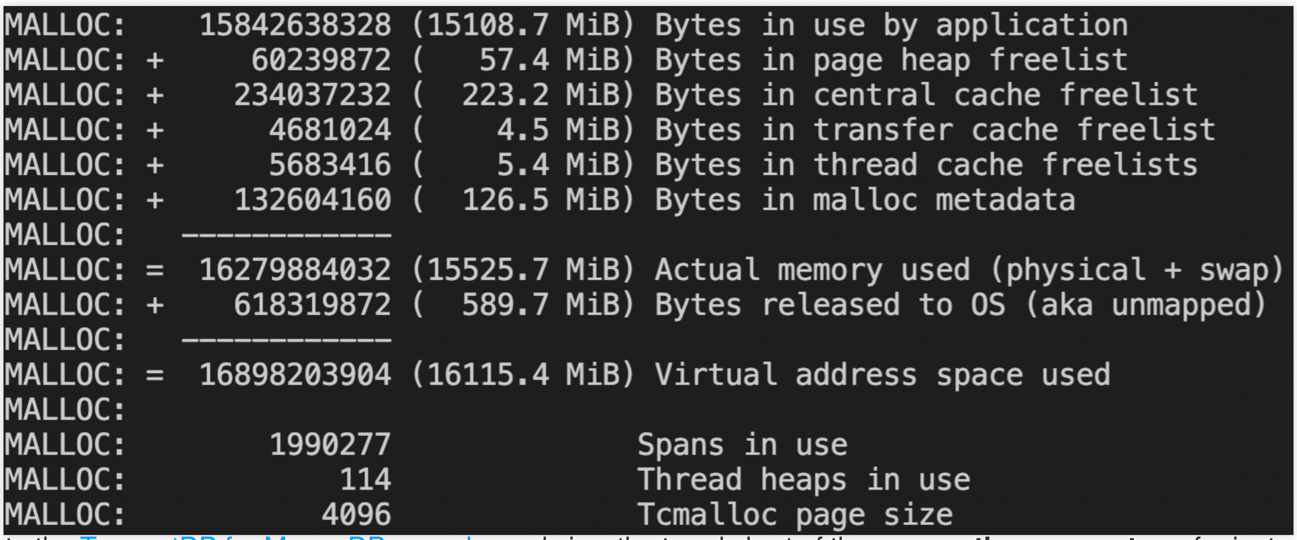
c. Increase the number of threads for cleaning dirty data.

Memory of Connection and Request

Each connection needs one request thread. But excessive request threads will cause frequent context switch, leading to high memory overload. Each thread can use up to 1 MB thread stack. Generally, the memory ranges from a few KB to dozens of KB.
When a request command is received, a request context will be created, and many temporary buffers may be allocated throughout the process, such as request packets, response packets, and sorted temporary buffers. When the request ends, these buffers will be released to the memory allocator `tcmalloc` which will send them to their cache and return them to the operating system gradually. In many cases, the failure of **'tcmalloc'** to return the memory to the operating system timely is a common cause of high memory usage, which can reach tens of GB.

Troubleshooting

For memory unreturned to the system by `tcmalloc` , you can view it
by running the command `db.serverStatus().tcmalloc.tcmalloc.formattedString` or `db.serverStatus().tcmalloc` . You can query the memory information by using `db.serverStatus().tcmalloc.tcmalloc.formattedString` as prompted below. `Bytes in use by application` refers to the memory actually consumed by the Mongod node, and `Bytes in page heap freelist` refers to the memory unreturned to the operating system.


```
MALLOC:     15842638328 (15108.7 MiB) Bytes in use by application
MALLOC: +      60239872 (   57.4 MiB) Bytes in page heap freelist
MALLOC: +     234037232 (  223.2 MiB) Bytes in central cache freelist
MALLOC: +       4681024 (    4.5 MiB) Bytes in transfer cache freelist
MALLOC: +       5683416 (    5.4 MiB) Bytes in thread cache freelists
MALLOC: +     132604160 (  126.5 MiB) Bytes in malloc metadata
MALLOC:   ------------
MALLOC: =   16279884032 (15525.7 MiB) Actual memory used (physical + swap)
MALLOC: +     618319872 (  589.7 MiB) Bytes released to OS (aka unmapped)
MALLOC:   ------------
MALLOC: =   16898203904 (16115.4 MiB) Virtual address space used
MALLOC:
MALLOC:         1990277              Spans in use
MALLOC:             114              Thread heaps in use
MALLOC:            4096              Tcmalloc page size
```
Log in to the TencentDB for MongoDB console, and view the trend chart of the **connection percentage** for instance monitoring metric on the **System Monitoring** page. Connection percentage refers to the ratio of current connections to the maximum connections.

Usage suggestions

1. Repossess the memory in time.
The `tcmallocReleaseRate` parameter is used to control the release of the memory, whichspecifies the amount of memory to release when the memory usage reaches a ratio. The higher the value of `tcmallocReleaseRate` ,

the faster MongoDB will release memory, but it will also have an impact on performance. You need to make adjustment when configuring `tcmallocReleaseRate` as required. By doing so, you can reach a balance between memory usage and performance requirements. You can run the following commands to set the value of `tcmallocReleaseRate` to a number between 1 and 10.
When `diagnosticDataCollectionVerboseTCMalloc` is `true` , you can repossess the memory directly in the following method:

```
db.adminCommand( { setParameter: 1, tcmallocReleaseRate: 5.0 } )
db.runCommand( { setParameter: 1, diagnosticDataCollectionVerboseTCMalloc:true}
```

Note:
`tcmallocReleaseRate` is suitable for TencentDB for MongoDB replica set instance 4.2 and later.
`diagnosticDataCollectionVerboseTCMalloc` is suitable for TencentDB for MongoDB replica set instance 4.4 and later.

2. Control the number of concurrent connections.
Up to 100 persistent connections can be created in the database, and 100 connection pools will be created between MongoDB Driver and the backend by default. If there are many clients, you need to reduce their connection pool size. We recommend that you keep the number of long connections established with the entire database below 1000. When the connection percentage is high, see Troubleshooting High Connection Utilization.

3. Reduce the memory overhead per request.
For example, you can reduce collection scans and memory sorting by creating indexes. To create an index, see MongoDB Indexes.

4. Upgrade the memory specification.
If the memory utilization continues to rise when the number of connections is appropriate, we recommend that you upgrade the memory specifications to avoid drastic decline in system performance due to memory overflow and large cache purge.
Upgrade CPU and memory configuration of Mongod. For more information, see Adjusting Mongod Node Specification.
You can add shards for the sharded cluster. For detailed directions, see Adjusting Shard Quantity.

Metadata Memory

TencentDB for MongoDB maintains some metadata for each collection, such as index information, number of documents, and storage engine options. The management of this information will occupy some system resources. If there is a large amount of memory metadata in your collection or index, a lot of memory will be occupied. Especially in versions earlier than TencentDB for MongoDB 4.0, a large number of file handles will be opened during full logical backup. If they are not returned to the operating system in time, the memory usage will increase rapidly. Furthermore, in lower versions of TencentDB for MongoDB, file handles may not be deleted after a large number of collections are deleted, which may also cause memory leaks.

Troubleshooting

To count the number of collections and indexes in MongoDB, use the following commands separately:

```
use <database>;
db.getCollectionNames().filter(function(c) { return !c.startsWith("system."); }).le
db.getCollectionNames().forEach(function(col) {     print("Indexes for " + col + ":
```

Query the number of collections as prompted below:

```
mongos> db.getCollectionNames().filter(function(c) { return !c.startsWith("system."
4
```

Query the index number of each collection as follows:

```
mongos> db.getCollectionNames().forEach(function(col) {     print("Indexes for " +
Indexes for nba: 3
Indexes for t1: 1
Indexes for test: 1

Indexes for user: 1
```

Usage suggestions

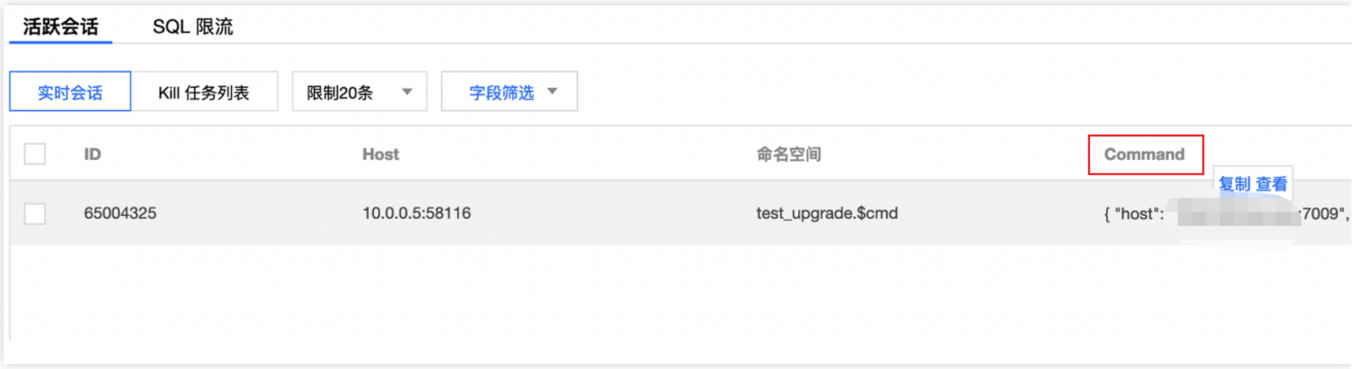
We recommend that you keep the number of collections within a few thousand for MongoDB. Otherwise, MongoDB will experience a significant performance drop when managing these collections.

Memory Usage in Index Creation

A large amount of memory may be occupied when index is created for MongoDB. In the normal data writing, the secondary node will maintain a `buffer` of about 256 MB for data playback. For versions earlier than MongoDB 4.2, an index is created by the primary node in a non-background way, which will be played back serially by the backend. The playback may consume up to 500 MB of memory. But for versions later than MongoDB 4.2, `background` option is disused by default, and the secondary node is allowed to parallelly play back the index creation process, incurring more memory usage. Therefore, if multiple indexes are created at the same time, instance memory overflow may occur. Moreover, the secondary node may consume more memory during data replay process, especially after the index is created for the primary node.

Troubleshooting

On the Real-Time Session page of **DBbrain**, check whether there are many processes of adding indexes. You can query the key fields of `CreateIndex` in the **Command** list as prompted below.



ID	Host	命名空间	Command
65004325	10.0.0.5:58116	test_upgrade.$cmd	{ "host": [redacted] 7009",

Usage suggestions

Don’t create multiple large indexes simultaneously during peak hours.

Create an index in a background way to avoid high memory usage.

If the memory is insufficient, increase the system memory.

To reduce the memory usage, we recommend that you use more effective index types, such as text search index and geospatial index. Use DBbrain's index recommendation feature to select the optimal index.

For larger collections, use sharding technology to disperse the storage of data and indexes. By doing so, the memory pressure of a single instance will be reduced.

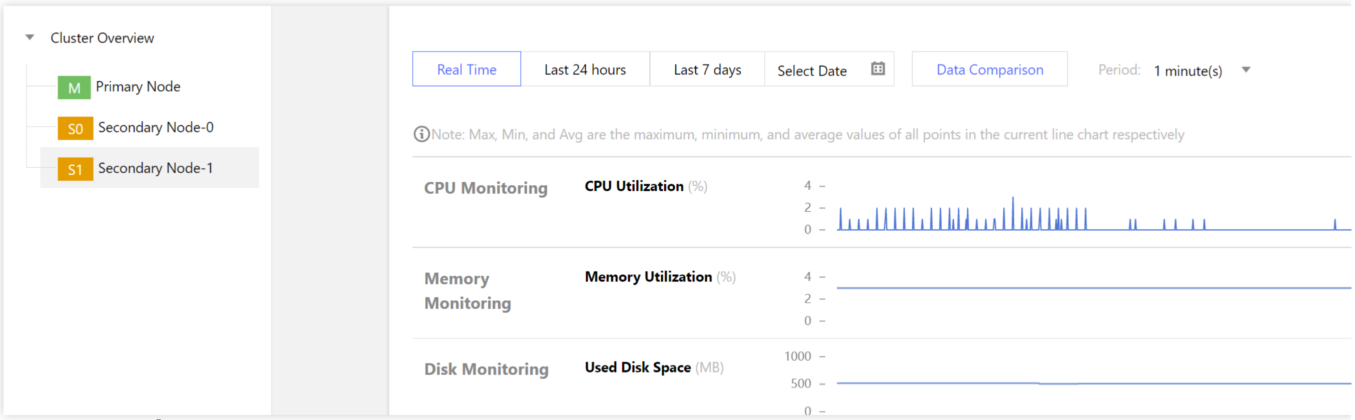
Memory Produced in Logical Backup and Primary-Secondary Switch

Logical backup generally will produce large number of data scans, causing high memory usage. If you select secondary node or hidden node for logical backup, the memory used by secondary node will be evidently higher than that of the primary node.

In primary-secondary switch, the original primary node will be switched as the secondary node. Thus, the memory usage of the current secondary node is higher than that of the current primary node.

1. Troubleshooting
2. Log in to the MongoDB console, and check whether there is an running backup task on the **Task Management** page. The task being executed will be displayed on the top of the task list. Click **Task Details** in the **Operation** column, and check whether **Logical Backup** is selected. For detailed direction, see Task Management.
3. In **Task Type** on the **Task Management** tab, search for **Promote to Primary Node** to confirm whether the primary-secondary switch is performed.

Click **System Monitoring** tab, in the drop-down list on the left sidebar, view the memory utilization of the primary and secondary node.



Usage suggestions

As physical backup is performed by copying the physical file, memory usage is not so high. If your business has high requirements on the node memory, select physical backup mode. For more information, see Data Backup.

If reading a secondary node is not set after the primary-secondary switch, restart the corresponding secondary node in the console to free up the memory. If the secondary node has a request, perform the operation during off-peak hours.

Cause Analysis and Solutions for Increased Slow Queries and Elevated Latency

Last updated：2024-05-07 16:15:02

Phenomenon Description

A slow query log is used to record queries that surpass a predefined execution time threshold. When there is a large volume of slow queries in the system, it can lead to decreased system performance, longer response times, and may even cause a system crash. Therefore, it is necessary to optimize slow queries, reduce their quantity, and improve system performance.
Log in to TencentDB for MongoDB console. Click Instance ID to enter the **Instance Details** page, select the **System Monitoring** tab, and examine the instance's monitoring data. It has been observed that latency-related monitoring metrics have notably increased. The latency monitoring metrics mainly reflect the duration from when a request arrives at the access layer until it completes processing and is returned to the client. For specific monitoring items, see Monitoring Overview.

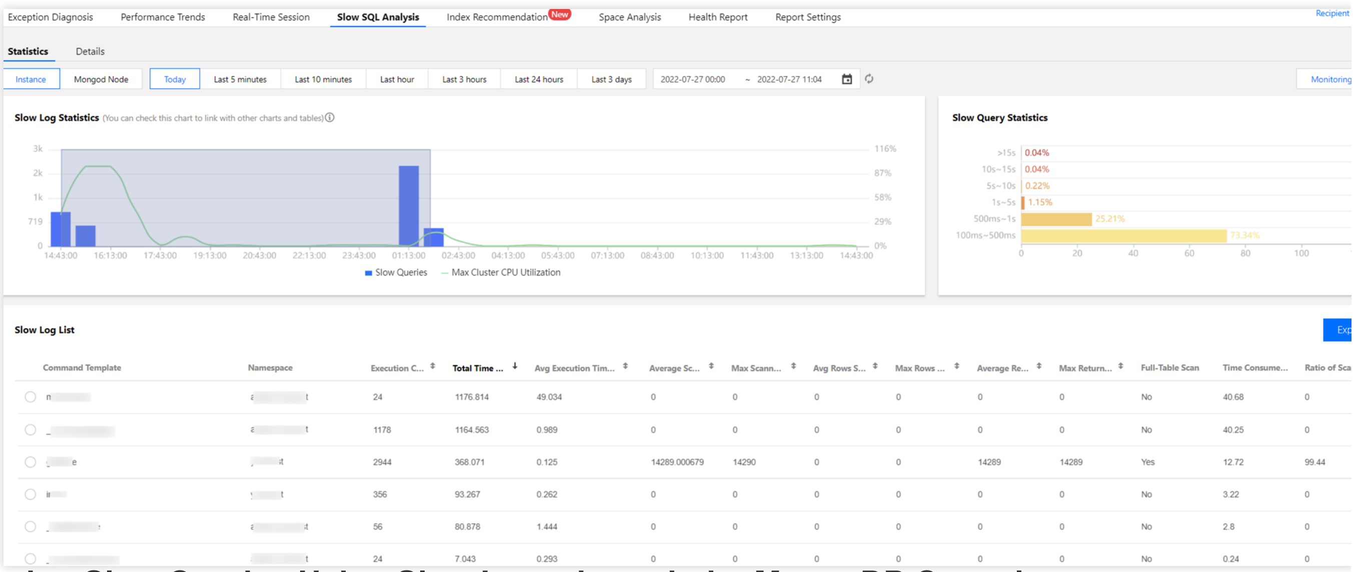
Possible Causes

Query through the $lookup operator without using an index or using an unsupported index. It requires traversing the entire database for a full scan, leading to low retrieval efficiency.
Documents within certain collections contain numerous large array fields that are extensively searched and indexed, leading to excessively large datasets being queried and indexed, consequently causing high system loads.

Analyzing Slow Queries

Slow SQL Analysis based on TencentDB for DBbrain (DBbrain) to Troubleshoot Slow Queries (Recommended)

Launched by Tencent Cloud, DBbrain is a cloud-based database management service for database performance optimization, security, and management. Specifically, it offers slow SQL analysis for MongoDB, aiming at analyzing slow logs generated during MongoDB operations. The diagnostic data is intuitive and easy to find, as shown in the following figure. For more information, see Slow SQL Analysis.



Slow Log List

	Command Template	Namespace	Execution C...	Total Time ...	Avg Execution Tim...	Average Sc...	Max Scann...	Avg Rows S...	Max Rows ...	Average Re...	Max Return...	Full-Table Scan	Time Consume...	Ratio of Sca
○	n[redacted]	a[redacted]t	24	1176.814	49.034	0	0	0	0	0	0	No	40.68	0
○	_[redacted]	a[redacted]t	1178	1164.563	0.989	0	0	0	0	0	0	No	40.25	0
○	[redacted]e	[redacted]t	2944	368.071	0.125	14289.000679	14290	0	0	14289	14289	Yes	12.72	99.44
○	ir[redacted]	[redacted]t	356	93.267	0.262	0	0	0	0	0	0	No	3.22	0
○	_[redacted]	a[redacted]t	56	80.878	1.444	0	0	0	0	0	0	No	2.8	0
○	_[redacted]	[redacted]t	24	7.043	0.293	0	0	0	0	0	0	No	0.24	0

Analyzing Slow Queries Using Slow Logs through the MongoDB Console

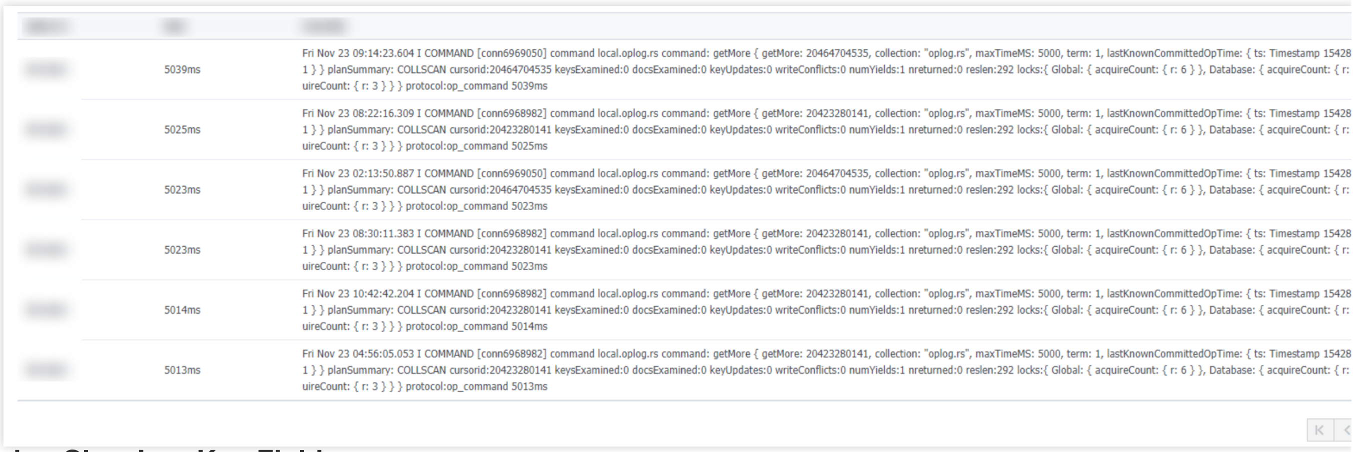
Before integrating MongoDB with DBbrain, you can retrieve slow logs in the MongoDB console and analyze key fields one by one to identify the reasons for slow queries.

Retrieveing Slow Logs

1. Log in to the MongoDB console.
2. In the drop-down list of **MongoDB** in the left sidebar, select either **Replica Set Instance** or **Sharded Instance**. The operations for both types of instances are similar.
3. Above the instance list on the right, select the region.
4. In the instance list, find the target instance.
5. Click the target instance ID to enter the **Instance Details** page.
6. Select the **Database Management** tab, and then select the **Slow Log Query** tab.
7. On the **Slow Log Query** tab, analyze the slow logs. The system will record operations executed over 100 milliseconds. Slow logs are retained for 7 days. And downloading slow log files is supported. For detailed operations, see Manage Slow Logs.

Abstract query: After the query conditions have undergone fuzzy processing, this presents statistical data on slow queries sorted by their average execution time. It is recommended to initially focus on optimizing the top 5 requests.

Detailed query: Records the full details of user-executed requests, including: the execution plan, the number of rows scanned, the execution duration, as well as certain lock waiting and related information.



[redacted]	[redacted]	[redacted]
[redacted]	5039ms	Fri Nov 23 09:14:23.604 I COMMAND [conn6969050] command local.oplog.rs command: getMore { getMore: 20464704535, collection: "oplog.rs", maxTimeMS: 5000, term: 1, lastKnownCommittedOpTime: { ts: Timestamp 15428 1 } } planSummary: COLLSCAN cursorid:20464704535 keysExamined:0 docsExamined:0 keyUpdates:0 writeConflicts:0 numYields:1 nreturned:0 reslen:292 locks:{ Global: { acquireCount: { r: 6 } }, Database: { acquireCount: { r: uireCount: { r: 3 } } } protocol:op_command 5039ms
[redacted]	5025ms	Fri Nov 23 08:22:16.309 I COMMAND [conn6968982] command local.oplog.rs command: getMore { getMore: 20423280141, collection: "oplog.rs", maxTimeMS: 5000, term: 1, lastKnownCommittedOpTime: { ts: Timestamp 15428 1 } } planSummary: COLLSCAN cursorid:20423280141 keysExamined:0 docsExamined:0 keyUpdates:0 writeConflicts:0 numYields:1 nreturned:0 reslen:292 locks:{ Global: { acquireCount: { r: 6 } }, Database: { acquireCount: { r: uireCount: { r: 3 } } } protocol:op_command 5025ms
[redacted]	5023ms	Fri Nov 23 02:13:50.887 I COMMAND [conn6969050] command local.oplog.rs command: getMore { getMore: 20464704535, collection: "oplog.rs", maxTimeMS: 5000, term: 1, lastKnownCommittedOpTime: { ts: Timestamp 15428 1 } } planSummary: COLLSCAN cursorid:20464704535 keysExamined:0 docsExamined:0 keyUpdates:0 writeConflicts:0 numYields:1 nreturned:0 reslen:292 locks:{ Global: { acquireCount: { r: 6 } }, Database: { acquireCount: { r: uireCount: { r: 3 } } } protocol:op_command 5023ms
[redacted]	5023ms	Fri Nov 23 08:30:11.383 I COMMAND [conn6968982] command local.oplog.rs command: getMore { getMore: 20423280141, collection: "oplog.rs", maxTimeMS: 5000, term: 1, lastKnownCommittedOpTime: { ts: Timestamp 15428 1 } } planSummary: COLLSCAN cursorid:20423280141 keysExamined:0 docsExamined:0 keyUpdates:0 writeConflicts:0 numYields:1 nreturned:0 reslen:292 locks:{ Global: { acquireCount: { r: 6 } }, Database: { acquireCount: { r: uireCount: { r: 3 } } } protocol:op_command 5023ms
[redacted]	5014ms	Fri Nov 23 10:42:42.204 I COMMAND [conn6968982] command local.oplog.rs command: getMore { getMore: 20423280141, collection: "oplog.rs", maxTimeMS: 5000, term: 1, lastKnownCommittedOpTime: { ts: Timestamp 15428 1 } } planSummary: COLLSCAN cursorid:20423280141 keysExamined:0 docsExamined:0 keyUpdates:0 writeConflicts:0 numYields:1 nreturned:0 reslen:292 locks:{ Global: { acquireCount: { r: 6 } }, Database: { acquireCount: { r: uireCount: { r: 3 } } } protocol:op_command 5014ms
[redacted]	5013ms	Fri Nov 23 04:56:05.053 I COMMAND [conn6968982] command local.oplog.rs command: getMore { getMore: 20423280141, collection: "oplog.rs", maxTimeMS: 5000, term: 1, lastKnownCommittedOpTime: { ts: Timestamp 15428 1 } } planSummary: COLLSCAN cursorid:20423280141 keysExamined:0 docsExamined:0 keyUpdates:0 writeConflicts:0 numYields:1 nreturned:0 reslen:292 locks:{ Global: { acquireCount: { r: 6 } }, Database: { acquireCount: { r: uireCount: { r: 3 } } } protocol:op_command 5013ms

Analyzing Slow Log Key Fields

View key fields in the slow logs. For the meanings of common key fields, see the table below. For more field descriptions, see MongoDB official website.

Key Fields	Description
command	Indicates the request operation recorded in the slow log.
COLLSCAN	Indicates that a full table scan was performed for the query. If the number of scanned rows is below 100, the speed of the full table scan will also be fast.
IXSCAN	Indicates that an index scan was performed. The specific index used will be listed after this field. A table may have multiple indexes. When the index here does not meet expectations, it should be considered to optimize the index or restructure the query statement with hint().
keysExamined	Indicates the scanning of index entries. "keysExamined": 0, # The number of index keys scanned by MongoDB for the operation is 0.
docsExamined	Indicates the number of documents scanned in the collection.
planSummary	Indicates the summary information of query execution. Each MongoDB query will generate an execution plan, which contains detailed information about the query, such as the used index, the number of documents scanned, the execution time of the query, etc. For example, "planSummary": "IXSCAN { a: 1, _id: -1 }" indicates a MongoDB query plan using index scan (IXSCAN). Specifically, it uses the index named a and the default "_id" index, scanning the a index in ascending order (1). This is a common query plan, indicating the query used the index to retrieve the required data.
numYield	Indicates the number of times an operation yields locks during its execution. When an operation needs to wait for certain resources (such as disk I/O or locks), it may relinquish CPU control, allowing other operations to continue. This process is referred to as "yielding". The higher the value of numYield, the heavier the system load is generally indicated, as operations take more time to complete. Typically,

	operations involving document searches (like querying, updating, and deleting) can yield locks. An operation can only yield its lock when other operations are queued waiting for the lock it holds. By optimizing the number of yields in the system, concurrency performance and throughput can be improved, and lock contention reduced, thereby enhancing system stability and reliability.
nreturned	Indicates the number of documents returned by a query request. The larger this value, the more rows are returned. If the keysExamined value is large but nreturned returns few documents, it indicates that the index may need optimization.
millis	Indicates the time it takes for a MongoDB operation to complete from start to finish. The larger this value, the slower the execution.
IDHACK	Used to accelerate query or update operations. In MongoDB, each document has an _id field, which is a unique identifier. In some cases, if the query or update operation includes the _id field, MongoDB can use the IDHACK technology to speed up the operation. Specifically, IDHACK technology can take advantage of the unique properties of the _id field to convert query or update operations into more efficient ones. For example, if the query condition is an exact match of the _id value, IDHACK can directly use the _id index to find the document, without needing to scan the entire collection.
FETCH	Indicates the number of documents MongoDB reads from disk during the execution of a query operation. During a query operation, MongoDB reads matching documents from disk based on query conditions and index information. The FETCH field records the number of documents read during this process. Generally, the smaller the value of the FETCH field, the better the query performance. Because MongoDB can make full use of technologies like index to reduce the number of times documents are read from disk, thereby improving query performance.

Solutions

Cleaning Up Slow Queries

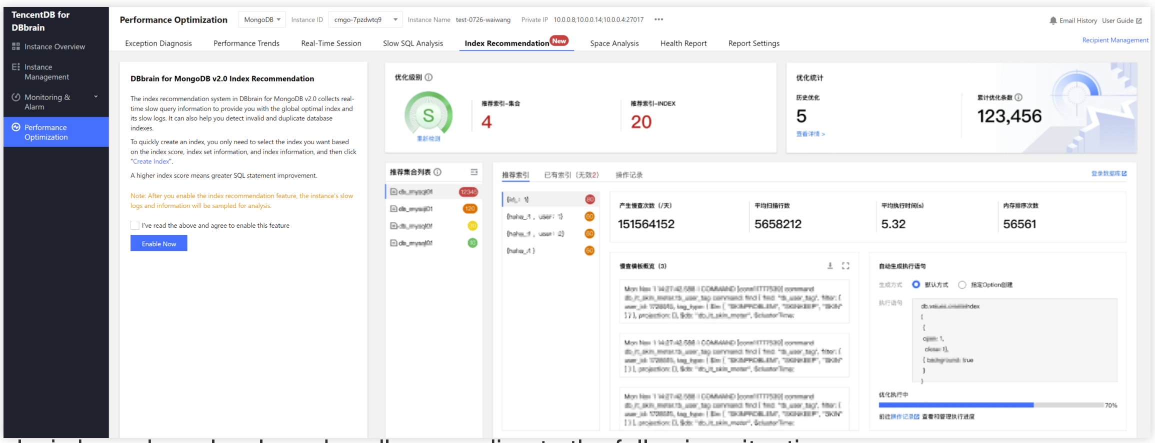
1. Select the **Database Management** > **Slow Query Management** tab. The list will display the current instance's executing requests (including those from secondary nodes). You can click **Batch Kill** to kill unwanted slow query requests. For detailed operations, see Slow Log Management.
2. For unexpected requests, you can directly perform a kill operation in the **Real-Time Session** page of the **Diagnostic Optimization** in **DBbrain** (TencentDB for DBbrain, DBbrain) to clean them up. For detailed operations, see Real-Time Session.

SQL Throttling

For TencentDB for MongoDB 4.0, in the **SQL Throttling** page of **Diagnostic Optimization** in the **DBbrain**, you can create an SQL throttling task to autonomously set SQL types, maximum concurrency, throttling duration, and SQL keywords to control the database's request volume and SQL concurrency, thereby achieving service availability. For detailed operations and application cases, see SQL Throttling.

Using Index

For slow SQL analysis based on the DBbrain, pay attention to the number of rows scanned in the slow log list. A large number may indicate requests with large scans or long-running requests.
An increase in slow queries due to full table scans can be reduced by creating indexes to lessen collection scans, memory sorting, etc. For index creation, see the official MongoDB Indexes.
If an index is used, and the number of index scans is 0, while the number of actual rows scanned is greater than 0, this indicates that the index needs optimization. Through the Index Recommendation of DBbrain, choose the optimal index. Index recommendations are generated by automatic analysis through real-time log slow query data collection. They offer the globally optimal indexes and rank them by the impact on performance. A larger recommendation value indicates more significant performance improvements after implementation.



DBbrain for MongoDB v2.0 Index Recommendation

The index recommendation system in DBbrain for MongoDB v2.0 collects real-time slow query information to provide you with the global optimal index and its slow logs. It can also help you detect invalid and duplicate database indexes.

To quickly create an index, you only need to select the index you want based on the index score, index set information, and index information, and then click "Create Index".

A higher index score means greater SQL statement improvement.

Note: After you enable the index recommendation feature, the instance's slow logs and information will be sampled for analysis.

☐ I've read the above and agree to enable this feature

For analysis based on slow logs, handle according to the following situations.
keysExamined = 0, while docsExamined > 0, and planSummary is COLLSCAN, indicating a full table scan that significantly delays queries, as shown below. For index creation, see the official MongoDB Indexes.
keysExamined > 0, while docsExamined > 0, and planSummary is IXSCAN, indicating that some query conditions or returned fields are not included in the index, necessitating index optimization. Please use the Index Recommendation of DBbrain to choose the optimal index.
For key field keysExamined > 0, while docsExamined = 0 and planSummary is IXSCAN, indicating that the query conditions or returned fields are already covered by the index. If keysExamined value is high, consider optimizing the order of fields in the index or adding a more suitable index for filtering. For more information, see Index Optimization to Solve Read/Write Performance Bottleneck.

```
Thu Mar 24 01:03:01.099 I COMMAND [conn8976420] command tcoverage.ogid_mapping_info
```

If you are using a MongoDB version earlier than 4.2 and have confirmed there is no issue with the index used for business queries, check whether an index was created in foreground mode during peak business hours. Switch to background mode.

Note:

Index creation in foreground mode: Before MongoDB version 4.2, the default mode for index creation in a collection was foreground. It sets the background option's value to false. This operation blocks all other operations until the index creation in the foreground is completed.

Index creation in background mode: Set the background option's value to true. It allows MongoDB to continue providing read-write operation services during the index creation process. However, creating an index in the background may extend the time it takes to create the index. For specific methods of creating an index, visit TencentDB for MongoDB's official website.
To create an index in background mode, use the `currentOp` command to view the current progress of the index creation. The specific command is as shown below.

```
db.adminCommand(
    {
      currentOp: true,
      $or: [
        { op: "command", "command.createIndexes": { $exists: true }  },
        { op: "none", "msg" : /^Index Build/ }
      ]
    }
)
```

Return as shown in the following figure. The `msg` field indicates the current progress of the index creation. The `locks` field indicates the lock type for the operation. For more information on locks, visit TencentDB for MongoDB's official website.

Troubleshooting Excessive Connections

Last updated：2024-05-07 10:18:49

Problem Description

If the number of connections exceeds the upper limit, you can use the connection management and restart features in the console and optimize the service for troubleshooting.

Possible Causes

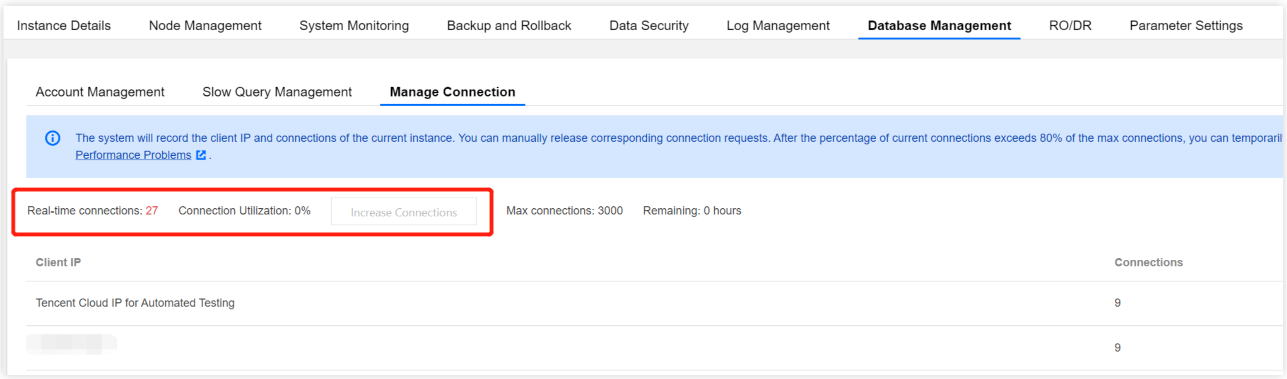
There are connection leaks, improper client coding, unreasonable connection pool configuration, or many unreleased connections.
There is a large number of concurrent application requests, and the configured upper limit of connections is insufficient, so the current database specification cannot sustain such requests.

Troubleshooting

Option 1. Increase the connections

1. Log in to the TencentDB for MongoDB console and click an instance ID to enter the instance management page.
2. Select **Database Management** > **Manage Connection** and view the source IPs and number of connections for service troubleshooting.

Note:
If the number of connections reaches or exceeds 80% of the upper limit and affects the establishment of new connections, you can click **Increase Connections** in the console to increase the maximum number of connections to 150% of the original limit for the next 6 hours.
If the problem persists, contact the aftersales service or submit a ticket for assistance.


Instance Details Node Management System Monitoring Backup and Rollback Data Security Log Management **Database Management** RO/DR Parameter Settings

Account Management Slow Query Management **Manage Connection**

The system will record the client IP and connections of the current instance. You can manually release corresponding connection requests. After the percentage of current connections exceeds 80% of the max connections, you can temporaril
Performance Problems .

Real-time connections: 27 Connection Utilization: 0% Increase Connections Max connections: 3000 Remaining: 0 hours

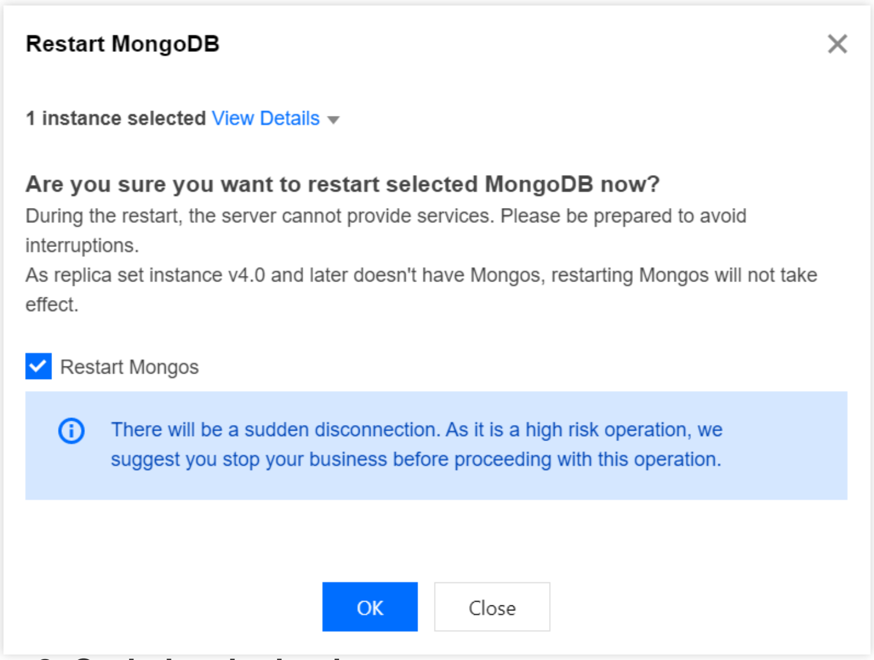
Client IP	Connections
Tencent Cloud IP for Automated Testing	9
	9

Option 2. Restart the instance

1. Log in to the TencentDB for MongoDB console and click an instance ID to enter the instance management page.
2. Select **Database Management** > **Manage Connection**. You can click **Restart** on the right to restart mongos to fix the problem.

Note:

Replica sets on v4.0 don't have mongos.
Restarting the mongod is highly risky and will cause a momentary disconnection, during which if data is written, rollback may be triggered, leading to data loss. Therefore, the restart feature is not enabled by default. If you need to enable it, contact the aftersales service or submit a ticket for assistance.


Restart MongoDB

1 instance selected View Details

Are you sure you want to restart selected MongoDB now?
During the restart, the server cannot provide services. Please be prepared to avoid interruptions.
As replica set instance v4.0 and later doesn't have Mongos, restarting Mongos will not take effect.

✓ Restart Mongos

There will be a sudden disconnection. As it is a high risk operation, we suggest you stop your business before proceeding with this operation.

Option 3. Optimize the business

To enhance the performance of services from the business perspective, it is essential to conduct a thorough investigation. For detailed investigative techniques, see Troubleshooting High Connection Utilization.

Solutions for High Disk Space Utilization

Last updated：2024-05-07 16:17:38

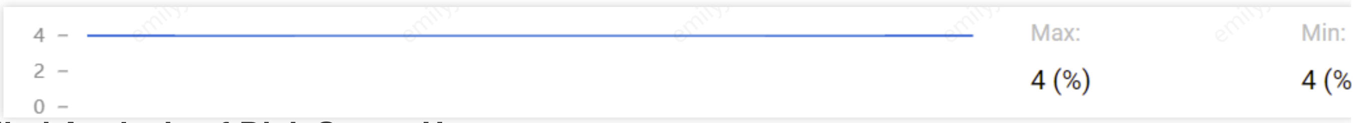
Overview

The disk in MongoDB mainly stores data and indexes, as well as some system files and log files. Disk space usage rate is a very important monitoring metrics. When disk space is fully used, the MongoDB instance will be unable to continue writing new data, leading to instance failure and shutdown. Therefore, monitoring disk usage and promptly taking measures to free up disk space is key to ensuring normal operation of the MongoDB instance.

Viewing Disk Space Usage

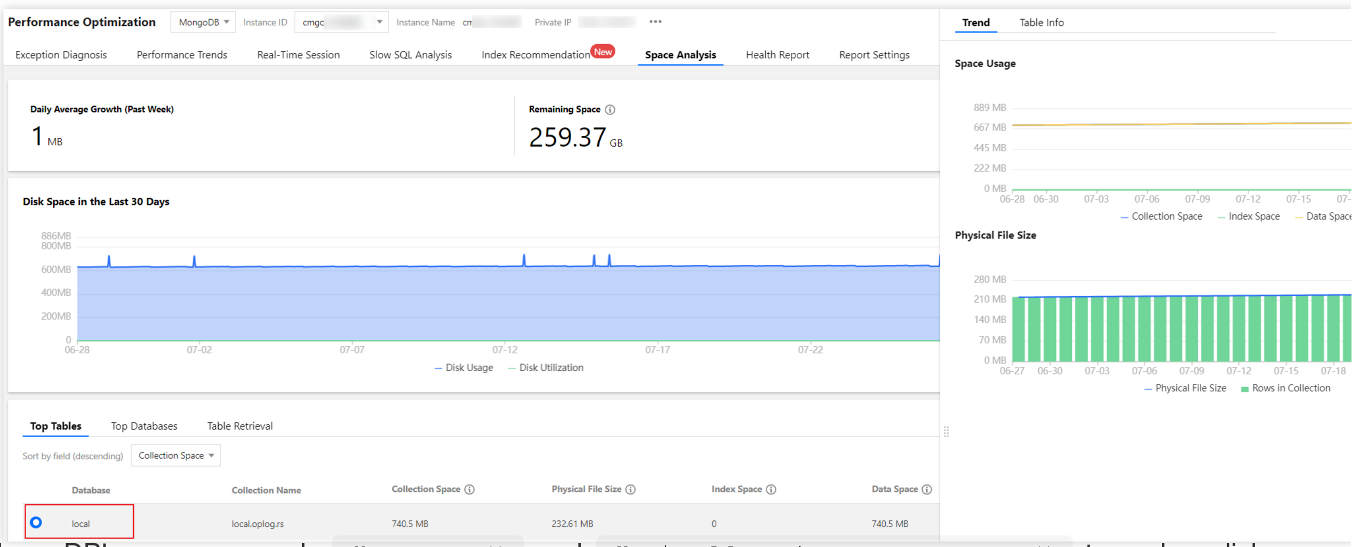
Quickly Viewing Monitoring Metrics

Log in to the MongoDB console. On the **System Monitoring** tab, you can view the trend view of **Disk Space Usage Rate**. For specific operations, see View Monitoring Data.



Detailed Analysis of Disk Space Usage

Log in to the MongoDB console. In the left sidebar, choose **Performance Optimization**, and then choose the **Space Analysis** tab. Through the Space Analysis of DBbrain (TencentDB for DBbrain), you can further analyze the details of disk space usage for the database, including database collection space, index space, physical file size, database size, data proportion, number of rows in collections, and other analytical data and views. For specific operations, see Space Analysis.



Daily Average Growth (Past Week): 1 MB

Remaining Space: 259.37 GB

Sort by field (descending): Collection Space

	Database	Collection Name	Collection Space	Physical File Size	Index Space	Data Space
●	local	local.oplog.rs	740.5 MB	232.61 MB	0	740.5 MB

Use MongoDB's own commands `db.stats()` and `db.$collection_name.stats()` to analyze disk space usage. For detailed information, see the following table.

Analysis Command	Command Definition
db.stats()	This command is used to retrieve the statistical information of the current database. Executing the `db.stats()` command returns a document containing various information about the current database. For example: database name, data size, index size, number of collections, etc.
db.collection.stats()	This command is used to retrieve the statistical information of a specified collection. Executing the `db.collection.stats()` command returns a document containing various information about the specified collection. For example: collection name, number of documents, data size, number of indexes, etc.
db.collection.storageSize()	This command is used to retrieve the storage space size occupied by a specified collection. Executing the `db.collection.storageSize()` command returns the storage space size occupied by the specified collection, in bytes. The storage space size returned by this command includes the space occupied by data and indexes in the collection, but does not include other overheads of the MongoDB instance, such as log files and temporary files.
db.collection.totalIndexSize()	This command is used to retrieve the storage space size occupied by all indexes of a specified collection. Executing the `db.collection.totalIndexSize()` command returns the storage space size occupied by all indexes of the specified collection, in bytes. The storage space size returned by this command does not include the space occupied by data in the collection, only the space occupied by all indexes of the collection.
db.collection.totalSize()	This command is used to retrieve the total storage space size occupied by a specified collection. Executing the `db.collection.totalSize()`

	command returns the total storage space size occupied by the specified collection, in bytes. The storage space size returned by this command includes the space occupied by data and indexes in the collection, as well as other overheads of the MongoDB instance, such as log files and temporary files.

Problem Analysis

TencentDB for MongoDB by default uses the WiredTiger engine. When documents are deleted, disk space is not directly reclaimed. However, when new data is inserted, MongoDB will reuse the previously occupied space instead of consuming additional new disk space. As more deletions occur, fragmentation will increase. The following code allows you to view the fragmentation rate of all collections in a specified database at once.
When the disk space usage rate of the instance reaches 80% to 85% or higher, the risk of running out of space can be avoided by either reducing the actual space occupied by the database or expanding the storage space.

```
## Creating a Function for Querying the Fragmentation Rates
function getCollectionDiskSpaceFragRatio(dbname, coll) {
    var res = db.getSiblingDB(dbname).runCommand({
        collStats: coll
    });
    var totalStorageUnusedSize = 0;
    var totalStorageSize = res['storageSize'] + res['totalIndexSize'];
    Object.keys(res.indexDetails).forEach(function(key) {
        var size = res['indexDetails'][key]['block-manager']['file bytes available
        print("index table " + key + " unused size: " + size);
        totalStorageUnusedSize += size;
    });
    var size = res['wiredTiger']['block-manager']['file bytes available for reuse']
    print("collection table " + coll + " unused size: " + size);
    totalStorageUnusedSize += size;
    print("collection and index table total unused size: " + totalStorageUnusedSize
    print("collection and index table total file size: " + totalStorageSize);
    print("Fragmentation ratio: " + ((totalStorageUnusedSize * 100.0) / totalStorag
}
## Specifying a Database for Querying the Fragmentation Rates of All Collections
use xxxdb
db.getCollectionNames().forEach((c) => {print("\\n\\n" + c); getCollectionDiskSpace
```

Solutions

High Disk Usage Rate and High Fragmentation Rates

For TencentDB for MongoDB versions 4.4 and later.

If the disk usage rate is high (generally exceeding 80% to 85%), and the fragmentation rates are also high (where it becomes beneficial to perform space reclamation if they exceed 25%), for TencentDB for MongoDB running on a replica set architecture with versions 4.4 and later, use the command `db.runCommand({compact:"collectionName"})` to compress documents within the specified collection, thereby releasing disk space. In this command, replace `collectionName` with the actual name of the targeted collection.

Note:

When MongoDB performs a Compact operation, it compresses the whole database. Thus, during this process, operations like creating and deleting collections, as well as creating and deleting indexes, will be temporarily blocked. Meanwhile, other operations, such as querying, will not be impacted directly, but there may be performance implications leading to increased latency in requests. It is recommended to execute this operation during business off-peak hours.

For TencentDB for MongoDB versions earlier than 4.4.

If the disk usage rate is high (generally exceeding 80% to 85%), and the fragmentation rate is also high (where it becomes beneficial to perform space reclamation if they exceed 25%), for TencentDB for MongoDB running versions below 4.4, it is not recommended to execute the `compact` operation. This is because doing so will block all incoming requests to the instance and might potentially result in issues where the Compact process does not effectively manage indexes. The solutions are as follows:

Upgrade the version. It is recommended to upgrade to the latest version for better performance and stability. For specific operations, see Version Upgrade.

If you choose not to upgrade, you can perform a logical migration to rebuild nodes in order to attain space shrinkage. This process may result in multiple temporary disconnections. For specific operations, contact after-sales support or

Submit a Ticket.

High Disk Usage Rate but Low Fragmentation Rates

If the disk usage rate is high (generally exceeding 80% to 85%), but the fragmentation rate is relatively low (less than 20%), it is not recommended to perform the `campact` operation, as MongoDB inherently reuses this space. In such circumstances, consider expanding the disk space. For specific operations, see Change Mongod Node Configuration Specifications.