TencentDB for MongoDB Practical Tutorial

Contents

Practical Tutorial
Optimizing Indexes to Break Through Read/Write Performance Bottlenecks

Last updated：2025-02-08 11:09:28

Indexes play a crucial role in the query performance of MongoDB databases. Meeting user query requirements with minimal indexes can greatly enhance database performance and reduce storage costs. This article introduces a series of index optimization analysis processes to help you solve database read/write performance bottlenecks.

Anomalies

For daily operation and maintenance, log in to the MongoDB console, click the instance ID to enter the **instance details** page, where you can view the following information.

- Select the **System Monitoring** tab to check the instance's monitoring data.
  - It was found that the CPU consumption of the cluster Mongod nodes was high, with CPU utilization often approaching 90% or even 100%.
  - The number of disk reads and writes per second remained high, resulting in high IO consumption, with single-node IO resource consumption accounting for 60% of the entire server.
- Select the **Database Management** tab, then select the **Slow Log Query** tab to view the slow logs.
  - The instance has a large number of slow logs, which include many different types of find and update requests, reaching thousands per second during peak periods.
  - The slow logs vary in type, with numerous query conditions, and all slow queries have matching indexes. The details are as follows.

```
Mon Aug  2 10:34:24.928 I COMMAND  [conn10480929] command xxx.xxx command: find { find: "xxx",
filter: { $and: [ { alxxxId: "xxx" }, { state: 0 }, { itemTagList: { $in: [ xx ] } }, {
persxxal: 0 } ] }, limit: 3, maxTimeMS: 10000 } planSummary: IXSCAN { alxxxId: 1.0,
itemTagList: 1.0 } keysExamined:1650 docsExamined:1650 hasSortStage:0 cursorExhausted:1
keyUpdates:0 writeConflicts:0 numYields:15 nreturned:3 reslen:8129 locks:{ Global: {
acquireCount: { r: 32 } }, Database: { acquireCount: { r: 16 } }, Collection: { acquireCount: {
r: 16 } } } protocol:op_command 227ms

Mon Aug  2 10:34:22.965 I COMMAND  [conn10301893] command xx.txxx command: find { find:
"txxitem", filter: { $and: [ { itxxxId: "xxxx" }, { state: 0 }, { itemTagList: { $in: [ xxx ] }
}, { persxxal: 0 } ] }, limit: 3, maxTimeMS: 10000 } planSummary: IXSCAN { alxxxId: 1.0,
itemTagList: 1.0 } keysExamined:1498 docsExamined:1498 hasSortStage:0 cursorExhausted:1
keyUpdates:0 writeConflicts:0 numYields:12 nreturned:3 reslen:8039 locks:{ Global: {
acquireCount: { r: 26 } }, Database: { acquireCount: { r: 13 } }, Collection: { acquireCount: {
r: 13 } } } protocol:op_command 158ms
```

Cause Analysis

Analyzing the slow logs, it was found that the query requests all used the { alxxxId: 1.0, itemTagList: 1.0 } index. The keysExamined for this index was 1498 rows, and the docsExamined was 1498 rows, but the number of returned documents was only nreturned = 3 rows. This means that only 3 rows of data met the conditions, but 1498 rows of data and indexes were scanned. It is evident that the key reason affecting read/write performance is the unreasonable index setting.

Note:
keysExamined indicates the number of index entries scanned. docsExamined represents the number of document entries scanned. The larger the keysExamined and docsExamined, the more it indicates that there is no index or the index has low selectivity. Note:

Index Optimization Process

Step 1: Collecting User Data Patterns

Common Business Queries, Update SQL, Are As Follows:

```
Based on AlxxxId (user id) + itxxxId (single or multiple)
Querying Count Based on AlxxxId
Paging query based on AlxxxId through time period (createTime), some queries concatenate state and
other fields
Combined query based on AlxxxId, ParentAlxxxId, parentItxxxId, state
Query data based on ItxxxId (single or multiple)
Combined query based on AlxxxId, state, updateTime
Combined query based on AlxxxId, state, createTime, totalStock (stock quantity)
Based on AlxxxId (user id) + itxxxId (single or multiple) + any other field combination
Query based on AlxxxId, digitalxxxrmarkId (watermark id), state
Query based on AlxxxId, itemTagList (tag id), state, etc.
Query based on AlxxxId + itxxxId (single or multiple) + any other field
Other queries
```

Common Business Count Query SQL, Are As Follows:

```
AlxxxId, state, persxxal combination
AlxxxId, state, itemType combination
AlxxxId (user id) + itxxxId (single or multiple) + any other field combination
```

Step 2: Getting the Existing Index of a Cluster

Use db.xxx.getindex() to get the index information of the table. The query is complex with many indexes, totaling 30 indexes, as shown below.

```
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : -1, "persxxal" : 1, "srcItxxxId" : -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" : 1, "srcItxxxId"
: -1 }
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal" : -1 }
{ "srcItxxxId" : 1 }
{ "createTime" : 1 }
{ "itexxxList.boyunState" : -1, "itexxxList.wozhituUploadServerId": -1, "itexxxList.photoQiniuUrl" :
1, "itexxxList.sourceType" : 1 }
{ "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.sourceType" : 1,
"itexxxList.wozhituUploadServerId" : -1 }
{ "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.photoQiniuUrl" : 1 }
{ "itemResxxxIdList" : 1  }
```

Step 3: Index Optimization

Deleting unused indexes

MongoDB supports obtaining the hit count of each index through the index statistics command, as follows:

```
> db.xxxxx.aggregate({"$indexStats":{}})
{ "name" : "alxxxId_1_parentItxxxId_1_parentAlxxxId_1", "key" : { "alxxxId" : 1, "parentItxxxId" : 1,
"parentAlxxxId" : 1 },"host" : "TENCENT64.site:7014", "accesses" : { "ops" :
NumberLong(11236765),"since" : ISODate("2020-08-17T06:39:43.840Z") } }
```

The field meaning explanation is as follows:

- **name**: Index name, statistics are collected for this index name.
- **ops**: Index hit count, i.e., the number of times this index is used as a query request hit in all queries. If the hit count is 0 or very low, it indicates that this index is rarely chosen as the optimal index and can be considered a useless index.

Use this index statistics command to get the hit count for all indexes as shown below. If the hit count is 0 or very low, directly delete it. Also, if the business has been running for a while and ops is less than 10000, delete it. A total of 11 useless indexes can be deleted, leaving 30 − 11 = 19 useful indexes.

```
db.xxx.aggregate({"$indexStats":{}})
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : -1, "persxxal" : 1, "srcItxxxId" : -1 }
"ops" : NumberLong(88518502)
{ "alxxxId" : 1, "image" : 1 }                               "ops" : NumberLong(293104)
{ "itexxxList.vidxxCheck" : 1, "itemType" : 1, "state" : 1 }     "ops" : NumberLong(0)
{ "alxxxId" : 1, "state" : -1, "newsendTime" : -1, "itxxxId" : -1, "persxxal" : 1 }
"ops" : NumberLong(33361216)
{ "_id" : 1 }                                                  "ops" : NumberLong(3987)
 { "alxxxId" : 1, "createTime" : 1, "checkStatus" : 1 }       "ops" : NumberLong(20042796)
{ "alxxxId" : 1, "parentItxxxId" : -1, "state" : -1, "updateTime" : -1, "persxxal" : 1, "srcItxxxId"
: -1 }                  "ops" : NumberLong(43042796)
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "updateTime" : -1, "persxxal" : -1 }
"ops" : NumberLong(3042796)
{ "itxxxId" : -1 }       "ops" : NumberLong(38854593)
{ "srcItxxxId" : -1 }                                    "ops" : NumberLong(0)
{ "createTime" : 1 }                                   "ops" : NumberLong(62)
{ "itexxxList.boyunState" : -1, "itexxxList.wozhituUploadServerId" : -1, "itexxxList.photoQiniuUrl" :
1, "itexxxList.sourceType" : 1 }     "ops" : NumberLong(0)
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : -1 }                   "ops" :
NumberLong(140238342)
{ "itxxxId" : -1 }                  "ops" : NumberLong(38854593)
{ "alxxxId" : 1, "parentItxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }     "ops" :
NumberLong(132237254)
{ "alxxxId" : 1, "videoCover" : 1 }         { "ops" : NumberLong(2921857)
{ "alxxxId" : 1, "itemType" : 1 }           { "ops" : NumberLong(457)
{ "alxxxId" : 1, "state" : -1, "itemType" : 1, "persxxal" : 1, " itxxxId " : 1 }         "ops" :
NumberLong(68730734)
{ "alxxxId" : 1, "itxxxId" : 1 }        "ops" : NumberLong(232360252)
{ "itxxxId" : 1, "alxxxId" : 1 }        "ops" : NumberLong(145640252)
{ "alxxxId" : 1, "parentAlxxxId" : 1, "state" : 1 }           "ops" : NumberLong(689891)
{ "alxxxId" : 1, "itemTagList" : 1 }                      "ops" : NumberLong(2898693682)
{ "itexxxList.photoQiniuUrl" : 1, "itexxxList.boyunState" : 1, "itexxxList.sourceType" : 1,
"itexxxList.wozhituUploadServerId" : 1 }         "ops" : NumberLong(511303207)
{ "alxxxId" : 1, "parentItxxxId" : 1, "state" : 1 }                 "ops" : NumberLong(0)
{ "alxxxId" : 1, "parentItxxxId" : 1, "updateTime" : 1 }           "ops" : NumberLong(0)
{ "updateTime" : 1 }                                           "ops" : NumberLong(1397)
{ "itemPhoxxIdList" : -1 }         "ops" : NumberLong(0)
{ "alxxxId" : 1, "state" : -1, "isTop" : 1 }        "ops" : NumberLong(213305)
{ "alxxxId" : 1, "state" : 1, "itemResxxxIdList" : 1, "updateTime" : 1 }       "ops" :
NumberLong(2591780)
{ "alxxxId" : 1, "state" : 1, "itexxxList.photoQiniuUrl" : 1}  "ops" : NumberLong(23505)
{ "itexxxList.qiniuStatus" : 1, "itexxxList.photoNetUrl" : 1, "itexxxList.photoQiniuUrl" : 1 }
"ops" : NumberLong(0)
{ "itemResxxxIdList" : 1  }                  "ops" : NumberLong(7)
```

Deleting duplicate indexes

- Index duplication caused by query order. Different business development wrote two indexes as follows. Through analysis, the purpose of these two SQL indexes is the same, so creating either one is sufficient.

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

- Index duplication caused by the leftmost principle matching. { itxxxId:1, alxxxId:1 } and { itxxxId:1 } are two indexes, and { itxxxId:1 } is the duplicate index.
- Index duplication caused by containment relationship.

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

There are the following three queries for these three indexes:

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

These queries all contain common fields, so they can be merged into one index to meet the queries of these two types of SQL. The merged index is as follows:

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

After merging and cleaning duplicate indexes, the following 2 indexes can be retained.

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

Analyzing the uniqueness of indexes, removing overlapping indexes

Analyzing the combination of various field modules in the table data, it is found that the fields alxxxId and itxxxId are high-frequency fields. Analyzing schema information and randomly sampling a portion of the data, it is found that the combination of these two fields is unique. Confirmed, any combination of these two fields represents a unique piece of data. Therefore, the following indexes can be merged into one index: { itxxxId:1, alxxxId:1 }.

```
{ "alxxxId" : 1, "state" : -1, "updateTime" : -1, "itxxxId" : 1, "persxxal" : 1, "srcItxxxId" : -1 }
{ "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-equivalent queries

From the previous 30 indexes, it can be seen that some indexes are time type fields, such as createTime and updateTime. Confirmed, these fields are used for various range queries. Range queries belong to non-equivalent queries. If the range query field appears before other fields in the index, the subsequent fields cannot use the index, as shown below.

```
db.collection.find({{ "alxxxId" : xx, "parentItxxxId" : xx, "state" : xx, "updateTime" : {$gt:
xxxxx}, "persxxal" : xxx, "srcItxxxId" : xxx }     })

db.collection.find({{ "alxxxId" : xx, "state" : xx, "parentItxxxId" : xx, "updateTime" : {$lt:
xxxxx}, "persxxal" : xxx}     })
```

Both queries contain the updateTime field and perform range queries. All fields other than updateTime are equality queries. Fields to the right of updateTime cannot use the index. In the first index, the persxxal and srcItxxxId fields cannot match the index, and in the second index, the persxxal field cannot match the index.
The user sets the following two indexes for these two queries.

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

These two index fields are basically the same and can be optimized into one index to ensure more fields can match the index.

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

Removing indexes for low-frequency query fields

When deleting useless indexes, clean up indexes with hit counts below 10,000. However, some indexes have relatively low hit counts (only hundreds of thousands) compared to high-frequency hit counts (tens of billions). These relatively low-frequency hit indexes are as follows, containing image and videocover fields.

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

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

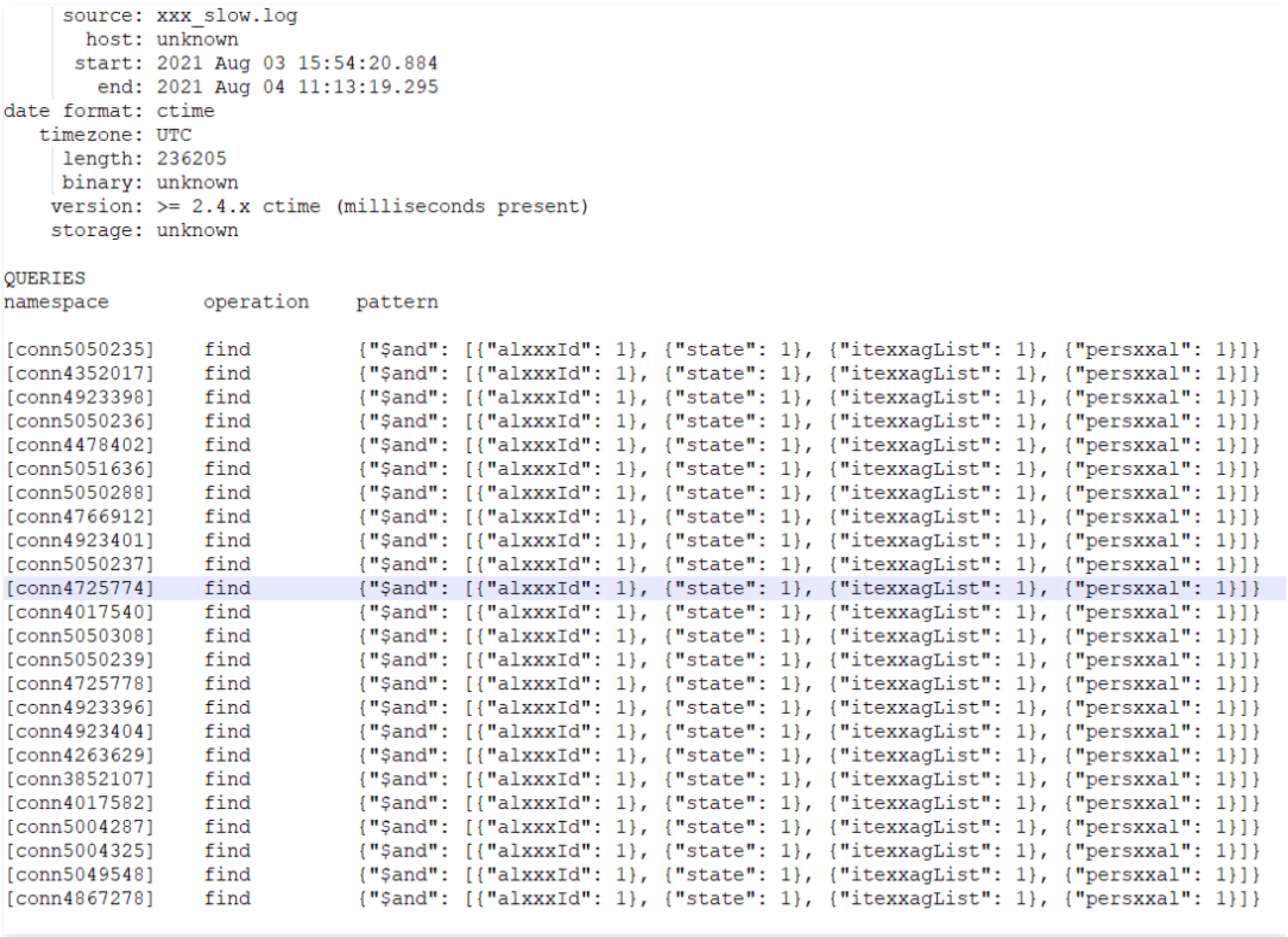
```
Mon Aug  2 10:56:46.533 I COMMAND  [conn5491176] command xxxx.tbxxxxx command: count { count:
"xxxxx", query: { alxxxId: "xxxxxx", itxxxId: "xxxxx", image: "http:/xxxxxxxxxxx/xxxxx.jpg" },
limit: 1 } planSummary: IXSCAN { itxxxId: 1.0,alxxxId:1.0 } keyUpdates:0 writeConflicts:0 numYields:1
reslen:62 locks:{ Global: { acquireCount: { r: 4 } }, Database:  { acquireCount: { r: 2 } },
Collection: { acquireCount: { r: 2 } } } protocol:op_query 4ms

Mon Aug  2 10:47:53.262 I COMMAND  [conn10428265] command xxxx.tbxxxxx command: find { find: "xxxxx",
filter: { $and: [ { alxxxId: "xxxxxxx" }, { state: 0 }, { itemTagList: { $size: 0 } } ] }, limit: 1,
singleBatch: true } planSummary: IXSCAN { alxxxId: 1, videoCover: 1 } keysExamined:128
docsExamined:128 cursorExhausted:1 keyUpdates:0 writeConflicts:0 numYields:22 nreturned:0 reslen:108
```

```
locks:{ Global:{ acquireCount: { r: 46 } }, Database: { acquireCount: { r: 23 } }, Collection: {
acquireCount: { r: 23 } } } protocol:op_command 148ms
```

- Image field: By analyzing the logs, it can be found that the image in user requests is combined with alxxxId and itxxxId for queries, and the combination of alxxxId and itxxxId is unique. Since the image field is not indexed at all, the { "alxxxId" : 1, "ixxxge" : 1 } index can be deleted.
- videoCover field: By analyzing the logs, it is found that the query conditions do not include videoCover. Only some queries match the { alxxxId: 1, videoCover: 1 } index, and keysExamined, docsExamined, and nreturned are different. It can be confirmed that only the alxxxId index field is actually matched. Therefore, the { alxxxId: 1, videoCover: 1 } index can also be deleted.

Analyzing high-frequency query logs and adding the optimal index for high-frequency queries

Log in to the MongoDB console, lower the slow log latency threshold in the **Slow Log Query** tab, and use the mtools tool to analyze queries over a period of time to obtain the following hot query information:



```
     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 high-frequency hot queries account for almost 99% of the queries. By analyzing the logs for these queries, the following information is obtained.

```
Mon Aug  2 10:47:58.015 I COMMAND  [conn4352017] command xxxx.xxx command: find { find: "xxxxx",
filter: { $and: [ { alxxxId:"xxxxx" }, { state: 0 }, { itemTagList: { $in: [ xxxxx ] } }, { persxxal:
0 } ] }, projection: { $sortKey: { $meta: "sortKey" } },  sort: { updateTime: 1 }, limit: 3,
maxTimeMS: 10000 } planSummary: IXSCAN { alxxxId: 1.0, itexxagList: 1.0 } keysExamined:1327
docsExamined:1327 hasSortStage:1 cursorExhausted:1 keyUpdates:0 writeConflicts:0 numYields:23
```

```
nreturned:3 reslen:12036 locks:{ Global: { acquireCount: { r: 48 } }, Database: { acquireCount: { r:
24 } }, Collection: { acquireCount: { r: 24 } } } protocol:op_command 151ms
```

Analysis of the logs shows that this high-frequency query matches the { alxxxId: 1.0, itexxagList: 1.0 } index. There is a significant difference between the number of scanned data rows and the number of final returned data rows, with 1327 rows scanned and only 3 rows retrieved.
This index is not the optimal index. This high-frequency query is a four-field equality query, but only two fields are indexed. The index can be optimized to the following index: { alxxxId: 1.0, itexxagList: 1.0, persxxal: 1.0, stat: 1.0 }.
Additionally, the logs show that this high-frequency query also includes a sort and limit. The original SQL for the entire query is as follows:

```
db.xxx.find({ $and: [ { alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [ xxxx ] } },{
persxxal: 0 } ] }).sort({updateTime:1}).limit(3)
```

This query model is an ordinary multi-field equality query + sort query + limit. The optimal index for this type of query could be one of the following two indexes:

- Index 1: Index for ordinary multi-field equality query
  Analyzing its query conditions:

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

  All four fields in this SQL are equality queries. Create the optimal index based on the degree of dispersion, placing the most dispersed field on the leftmost side. The optimal index can be as follows:

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

  If this index is chosen as the optimal index, the entire ordinary multi-field equality query + sort query + limit query will execute as follows:
  - Use the { alxxxId: 1.0, itexxagList: 1.0, persxxal: 1.0, stat: 1.0 } index to find all data that meets the conditions { $and: [{ alxxxId: "xxxx" }, { state: 0 }, { itexxagList: { $in: [xxxx] } }, { persxxal: 0 }] }.
  - Perform memory sorting on the data that meets the conditions.
  - Take the top three sorted data.
- Index 2: Equality Query + Optimal Index for Sort
  The sort query includes limit. Identify the high-frequency sort SQL as follows:

  ```
  { $and: [ { alxxxId:"xxxx" }, { state: 0 }, { itexxagList: { $in: [ xxxx ] } }, { persxxal: 0 } ]
  }.sort({updateTime :1}).limit(10)
  ```

  This query is an ultra-high-frequency query. It is recommended to add the following index for such SQL:

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

Step 4: Sorting Out the Final Retained Indexes

Through the above optimization, retain the following indexes.

```
{ "itxxxId" : 1, "alxxxId" : 1 }
{ "alxxxId" : 1, "state" : 1, "digitalxxxrmarkId" : 1, "updateTime" : 1 }
{ "alxxxId" : 1, "state" : -1,  "parentItxxxId" : 1, "persxxal" : -1, "updateTime" : 1 } { "alxxxId"
: 1, "itexxList.photoQiniuUrl" : 1, }
{ "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 }
```

Benefits After Index Optimization

- CPU resource savings over 90%
  CPU peak consumption reduced from over 90% to less than 10% after optimization
- Disk IO resource savings of 85%
  Disk IO consumption reduced from 60% – 70% to less than 10%
- Disk storage cost savings of 20%
  Each index corresponds to a disk index file, reduced from 30 indexes to 8, resulting in a real disk consumption reduction of about 20% for data + indexes
- Slow logs reduced by 99%
  Before index optimization, there were thousands of slow logs per second, reduced to dozens after optimization

Fixing High CPU Utilization in MongoDB Instance Based on DBbrain

Last updated：2025-02-08 11:09:59

Problem Description

In daily operations, if the CPU utilization of a MongoDB database is too high, it can easily cause system anomalies. For example, the read/write rate slows down, connections are exhausted, and timeouts increase. A large number of access timeouts can also trigger the client to repeatedly reconnect for authentication, which may eventually lead to a database "avalanche." High CPU utilization scenarios are common in production MongoDB databases. This issue is generally caused by SQL anomalies, high traffic, memory sorting, statements without indexes, or improper use of indexes.
When the database performs operations such as query and modification, the CPU will first request data from the storage engine cache:

- If the engine cache has the target data, the CPU will execute the computing task and return the result, which may involve actions requiring high CPU usage such as sorting.
- If the cache does not have the target data, it will trigger the action of fetching data from the disk.

Solution

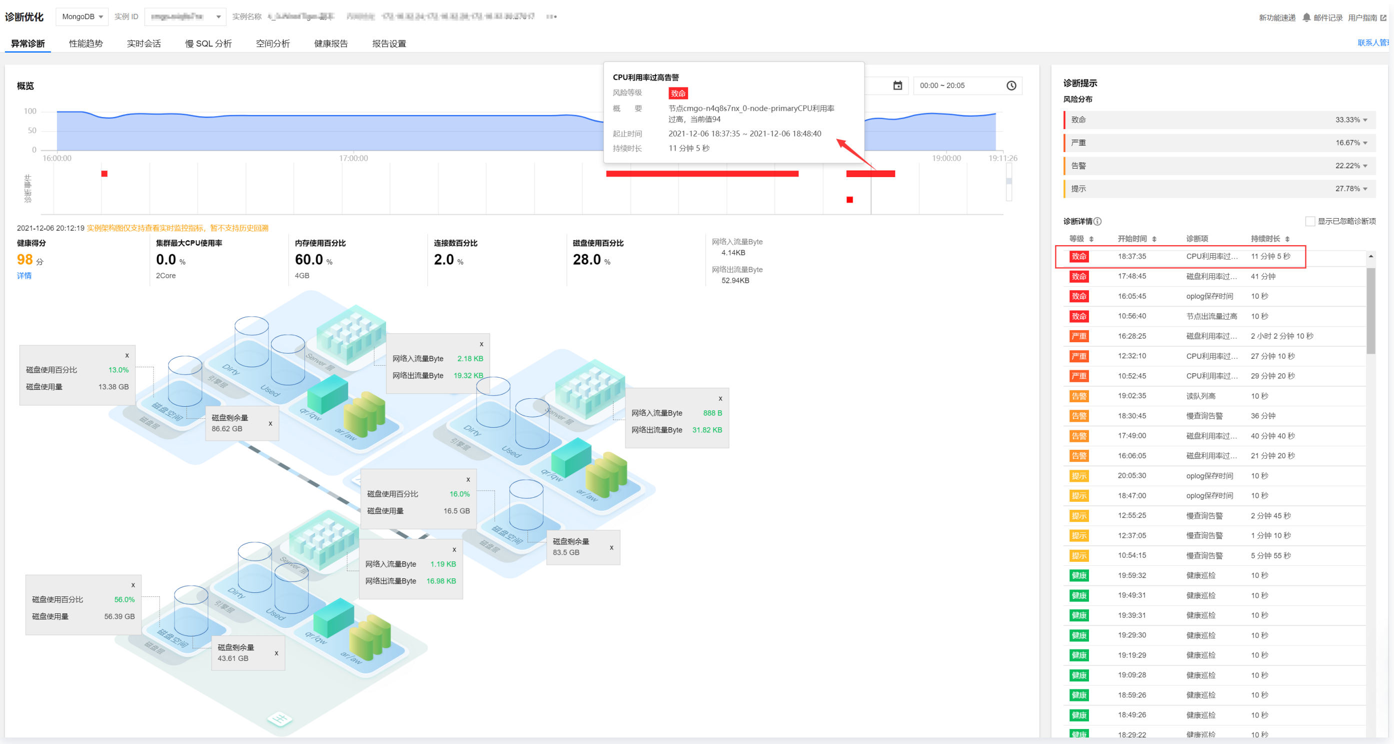
DBbrain's exception diagnosis feature can easily locate the problem of high CPU utilization, determine the time when the problem occurs, find the specific SQL statement that causes the problem, and give suggestions for fix. Then, you can leverage DBbrain's slow SQL optimization feature based on the suggestions to accurately analyze the statement and avoid similar problems.

- Exception diagnosis: 7 * 24-hour exception detection and diagnosis, providing real-time optimization suggestions.
- Slow SQL analysis: It analyzes slow SQL statements of the current instance and provides corresponding optimization suggestions.
- Real-time session: It displays ongoing operations in the current production database for you to handle abnormal operations.

Option 1 (recommended): Use the "exception diagnosis" feature to troubleshoot database exceptions

The exception diagnosis feature can proactively locate and perform optimization for failures, with no Ops experience required. It can find abnormal situations with high CPU utilization. Based on TencentDB for MongoDB Ops experts' many years of experience and combined with machine learning, big data, and intelligent analysis algorithms, this feature also quickly duplicates the capabilities of senior database experts to empower your MongoDB databases for smart Ops. It can discover almost all exceptions and failures in MongoDB production databases in real time.
The steps are as shown in the example below:

1. Log in to the DBbrain console and select **Diagnosis and Optimization** on the left sidebar. On the displayed page, select the **Exception Diagnosis** tab.


诊断优化 MongoDB 实例 ID [redacted] 实例名称 [redacted] [redacted]

异常诊断 性能趋势 实时会话 慢 SQL 分析 空间分析 健康报告 报告设置

新功能速递 邮件记录 用户指南 联系人管理

概览

CPU利用率过高告警
风险等级 致命
概 要 节点cmgo-n4q8s7nx_0-node-primaryCPU利用率过高，当前值94
起止时间 2021-12-06 18:37:35 ~ 2021-12-06 18:48:40
持续时长 11 分钟 5 秒

00:00 ~ 20:05

2021-12-06 20:12:19 实例架构图仅支持查看实时监控指标，暂不支持历史回溯

健康得分	集群最大CPU使用率	内存使用百分比	连接数百分比	磁盘使用百分比	网络入流量Byte 4.14KB
98 分	0.0 %	60.0 %	2.0 %	28.0 %	网络出流量Byte 52.94KB
详情	2Core	4GB			

诊断提示

风险分布

致命	33.33%
严重	16.67%
告警	22.22%
提示	27.78%

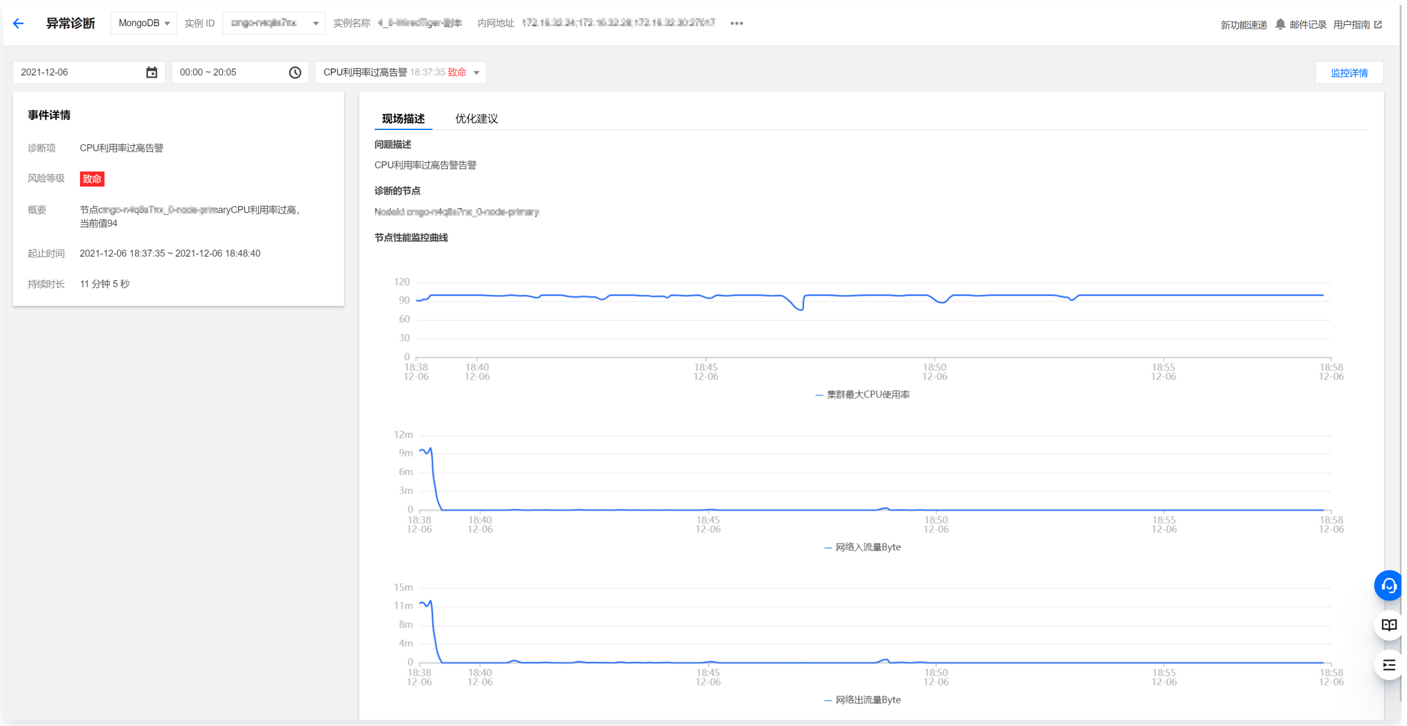
诊断详情 显示已忽略诊断项

等级	开始时间	诊断项	持续时长
致命	18:37:35	CPU利用率过...	11 分钟 5 秒
致命	17:48:45	磁盘利用率过...	41 分钟
致命	16:05:45	oplog保存时间	10 秒
致命	10:56:40	节点出流量过高	10 秒
严重	16:28:25	磁盘利用率过...	2 小时 2 分钟 10 秒
严重	12:32:10	CPU利用率过...	27 分钟 10 秒
严重	10:52:45	CPU利用率过...	29 分钟 20 秒
告警	19:02:35	读队列高	10 秒
告警	18:30:45	慢查询告警	36 分钟
告警	17:49:00	磁盘利用率过...	40 分钟 40 秒
告警	16:06:05	磁盘利用率过...	21 分钟 20 秒
提示	20:05:30	oplog保存时间	10 秒
提示	18:47:00	oplog保存时间	10 秒
提示	12:55:25	慢查询告警	2 分钟 45 秒
提示	12:37:05	慢查询告警	1 分钟 10 秒
提示	10:54:15	慢查询告警	5 分钟 55 秒
健康	19:59:32	健康巡检	10 秒
健康	19:49:31	健康巡检	10 秒
健康	19:39:31	健康巡检	10 秒
健康	19:29:30	健康巡检	10 秒
健康	19:19:29	健康巡检	10 秒
健康	19:09:28	健康巡检	10 秒
健康	18:59:26	健康巡检	10 秒
健康	18:49:26	健康巡检	10 秒
健康	18:29:22	健康巡检	10 秒

2. Select (enter or search for) an instance ID in the top-left corner to switch to the target instance.
3. On this page, select **Real Time** or **History** and select the time period to be queried. If any failures exist in this period, the overview information will be displayed in **Diagnosis Prompt** on the right.
4. Click **View Details** in the **Real-Time/Historical Diagnosis** tab or the target diagnosis item in **Diagnosis Prompt** to access the diagnosis details page.

- Event overview: Includes the diagnosis item name, time range, risk level, duration, and overview.
- Description: Includes symptom snapshots and performance trends of the exception event or health check event.
- Intelligent Analysis: Analyzes the root cause of the performance exception to help you locate the specific operation.
- Optimization Suggestion: Displays optimization suggestions.

5. Select the **Optimization Suggestion** tab to view the optimization suggestions provided by DBbrain for the failure.


异常诊断 MongoDB 实例 ID [redacted] 实例名称 [redacted] 内网地址 [redacted]

新功能速递 邮件记录 用户指南

2021-12-06 00:00 ~ 20:05 CPU利用率过高告警 18:37:35 致命

监控详情

事件详情

诊断项	CPU利用率过高告警
风险等级	致命
概要	节点[redacted]CPU利用率过高，当前值94
起止时间	2021-12-06 18:37:35 ~ 2021-12-06 18:48:40
持续时长	11 分钟 5 秒

现场描述 优化建议

问题描述

CPU利用率过高告警告警

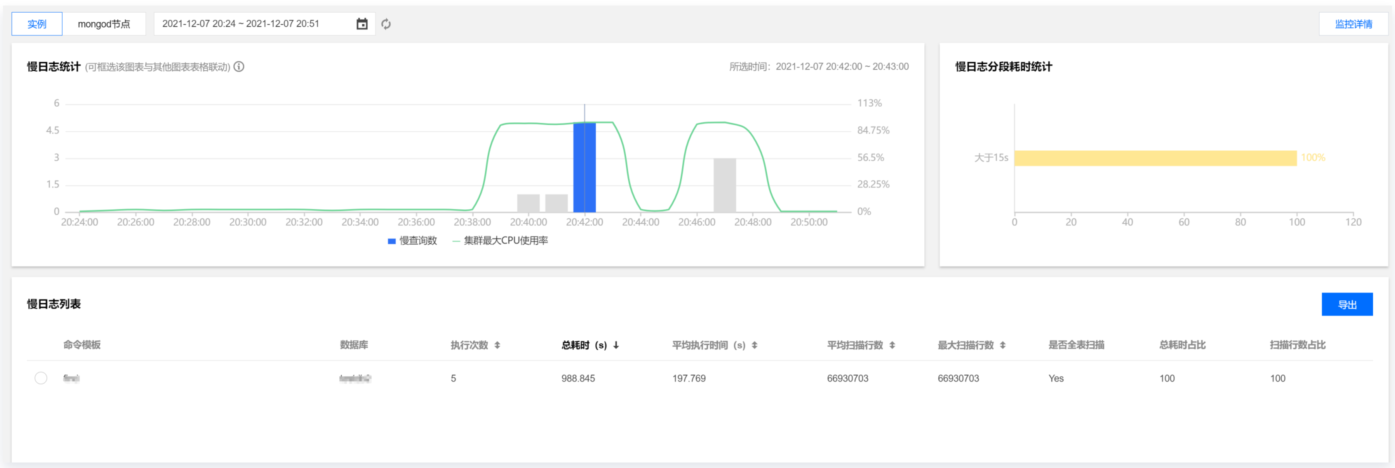
诊断的节点

[redacted]

节点性能监控曲线

Option 2: Use the "slow SQL analysis" feature to troubleshoot databases/tables leading to high CPU utilization

1. Log in to the DBbrain console and select **Diagnosis and Optimization** on the left sidebar. On the displayed page, select the **Slow SQL Analysis** tab.
2. Select (enter or search for) an instance ID in the top-left corner to switch to the target instance.
3. On the page, select the query time. If there are slow SQL statements for this instance during that period, the SQL statistics will display the time points and counts of slow SQL occurrences in a bar chart. Click the bar chart to display all corresponding slow SQL information (SQL after template aggregation) in the list below, and the SQL time distribution for that period will be shown on the right.



慢日志列表

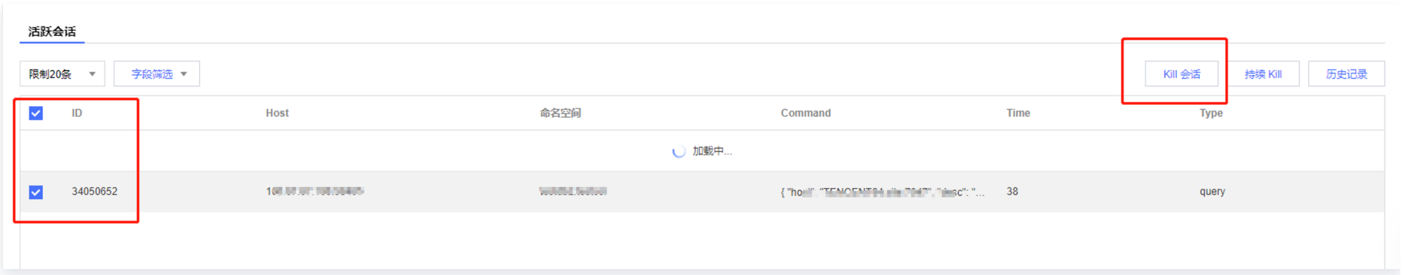
	命令模板	数据库	执行次数	总耗时 (s)	平均执行时间 (s)	平均扫描行数	最大扫描行数	是否全表扫描	总耗时占比	扫描行数占比
○	[redacted]	[redacted]	5	988.845	197.769	66930703	66930703	Yes	100	100

4. You can identify and filter SQL statement execution data in the SQL statement list in the following way:
   4.1 Sort the SQL statements by average time consumed (or maximum time consumed) in descending order. Focus on the SQL statements with higher time consumption. Do not sort the statements by total time consumed, as the data may be affected by a high number of executions.
   4.2 Then, check the numbers of returned rows and scanned rows.
   - If there is an SQL statement with the same **number of returned rows** and **number of scanned rows**, it is very likely that the full table has been queried and returned.
   - If there are several SQL statements with a large number of scanned rows but no or few returned rows, it means that the system generated a lot of logical and physical reads. If the volume of the data to be queried is too high and memory is insufficient, the request will generate many physical I/O requests and consume lots of I/O resources. Too many logical reads will occupy too many CPU resources, resulting in high CPU utilization.

Option 3: Use the "real-time session" feature to kill slow SQL statements

The MongoDB kernel records the current executing currentop information. The real-time session feature of DBbrain allows you to see all ongoing operations in the database and supports killing specified sessions. You can directly kill time-consuming SQL statements to free up CPU, disk I/O, and other resources. Additionally, it provides a continuous kill feature, which can continuously kill sessions based on conditions or information. In case of database blockage, you can use the continuous kill feature for emergency handling.

- Get session information and kill the session.
- Click **Diagnosis and optimization** > **Real-Time Session** > **Active Session**, select the session to be killed, and execute the kill operation.



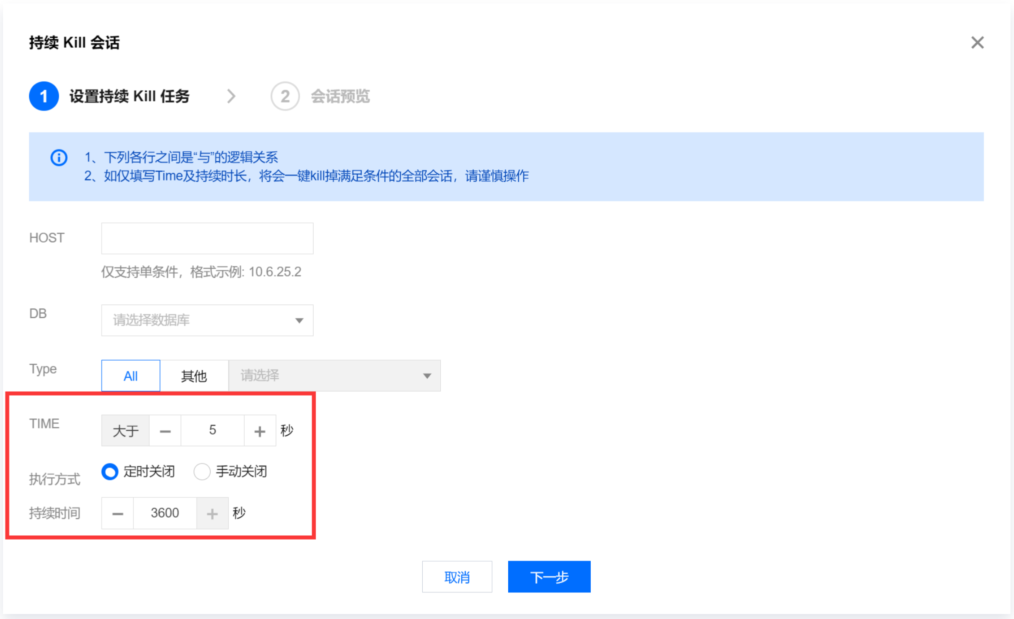
活跃会话

限制20条 字段筛选

Kill 会话 持续 Kill 历史记录

ID	Host	命名空间	Command	Time	Type
加载中...					
34050652	[redacted]	[redacted]	{ "ho[redacted]"TENCENT[redacted]", "desc": "...	38	query

- Continuously kill sessions.
  - The feature of continuously killing sessions can be configured based on dimensions such as DB, HOST, Type, and TIME. Two trigger mechanisms: "timeout auto-exit" and "manual closure".



持续 Kill 会话

1 设置持续 Kill 任务 > 2 会话预览

1、下列各行之间是"与"的逻辑关系
2、如仅填写Time及持续时长，将会一键kill掉满足条件的全部会话，请谨慎操作

HOST:
仅支持单条件，格式示例: 10.6.25.2

DB: 请选择数据库

Type: [✓ All] 其他 请选择

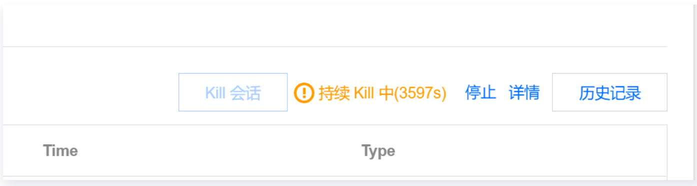
TIME: 大于 5 秒

执行方式: [✓ 定时关闭] 手动关闭

持续时间: 3600 秒

取消 下一步

  - To stop continuously killing sessions, click **Stop** to close scheduled tasks before the timeout or terminate manually triggered tasks.



Kill 会话 持续 Kill 中(3597s) 停止 详情 历史记录

Time	Type

Troubleshooting and Solutions for Mongos Load Imbalance in Sharded Cluster

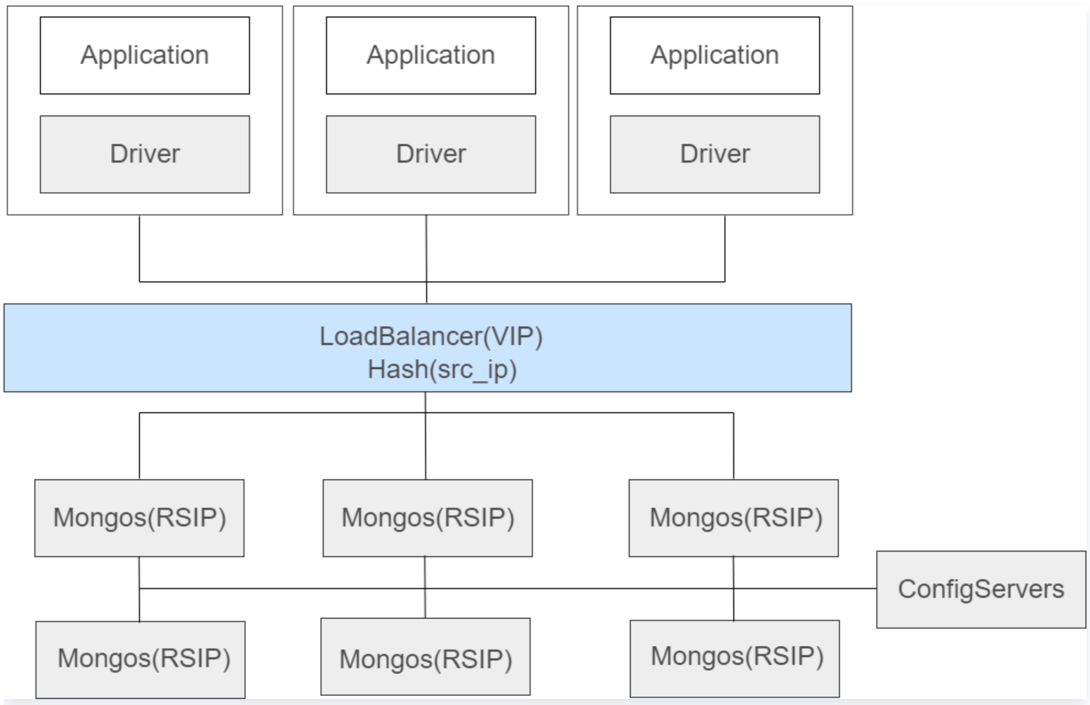
Last updated: 2025-02-08 11:10:27

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 source IP hash policy. 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.

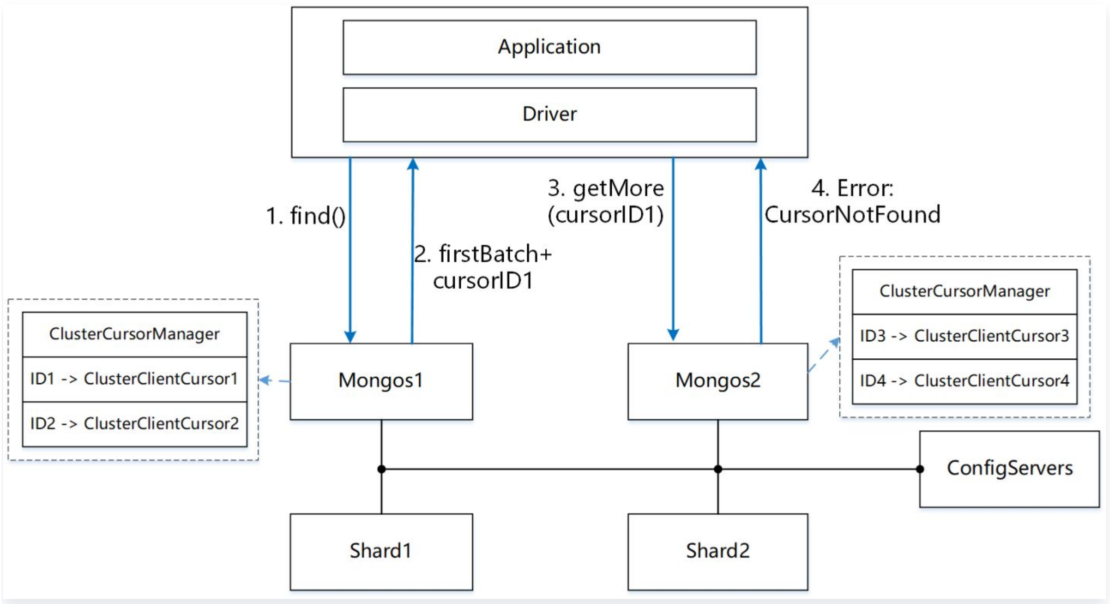


Batch scanning getMore issue

When MongoDB cannot return the find results at once, it will return the first batch of data and cursorID first. The client iterates the remaining data through this cursorID by continuously sending getMore requests. Therefore, a batch scanning request may correspond to one find request and multiple getMore requests, associated through the cursorID.

- Each mongos node maintains a global ClusterCursorManager in memory, which uses a HashMap to maintain the mapping between cursorID and ClusterClientCursor. The cursorID is an int64 random number, and the ClusterClientCursor maintains information such as the execution plan and current status of the request.
- If the query results cannot be returned at once (e.g., exceeding the 16MB limit), a non-zero cursorID will be generated, and this ID and the ClusterClientCursor itself will be registered to the ClusterCursorManager. If the client needs subsequent results, it can carry the previously returned cursorID to make getMore requests. The mongos will find the previously cached ClusterClientCursor, continue executing the query plan, and return the results. The ID and cursor information exist independently on each mongos node.

Therefore, it is necessary to ensure that the find and its associated getMore requests are sent to the same mongos node. If the getMore request is sent to a different mongos node, a CursorNotFound error will be returned because the cursor cannot be found, as shown below.



Transaction operation issue

MongoDB 4.2 supports distributed transactions, allowing users to connect to mongos nodes to initiate transaction operations. Multiple read and write operations can be performed between startTransaction and commitTransaction/abortTransaction. Mongos records metadata such as logicalSessionId and txnId for each request in the transaction to maintain context. Therefore, the design of MongoDB requires that all operations in a transaction be executed on the same mongos.

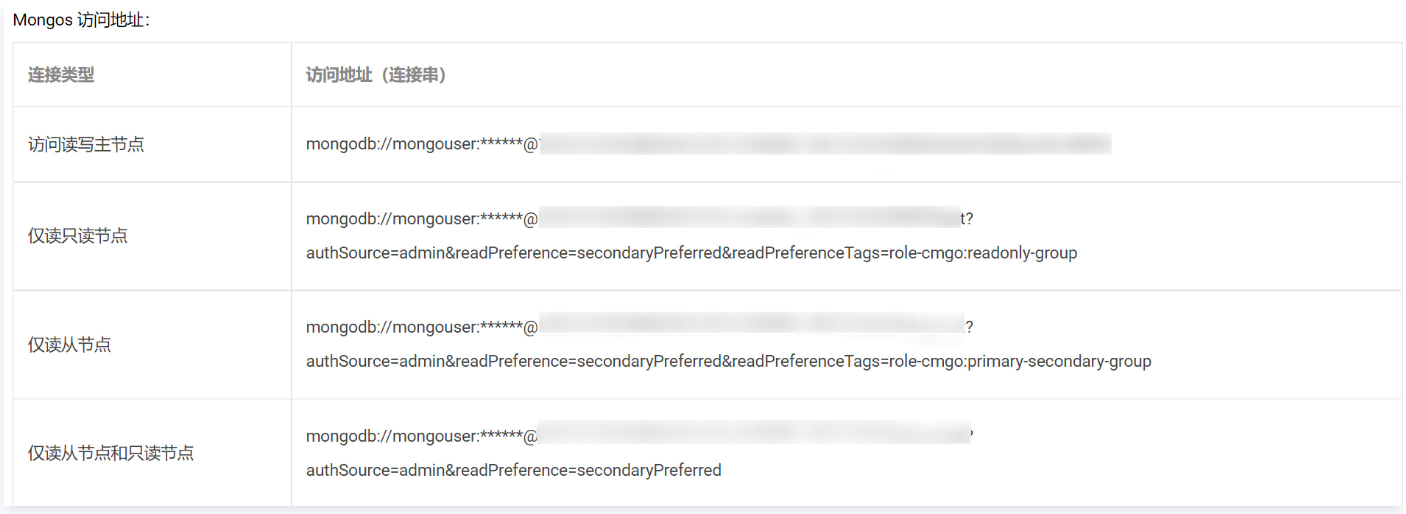
TencentDB for MongoDB load balancing policy

Considering the getMore issue in batch scanning and transaction operation issues, the TencentDB for MongoDB load balancing hash policy balances and distributes traffic based on the access IP (usually CVM): requests from the same source IP will be directed to the same mongos, ensuring that getMore() and transaction operations are performed 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 activation, in the **Network Configuration** section of the **Instance Details** page in the console, you can view the mongos access address, displaying connection strings for different connection types. Each connection string configures all mongos nodes of the instance, controlling the type of access nodes through parameters such as authSource, readPreference, and readPreferenceTags, as shown below.



Mongos 访问地址：

连接类型	访问地址（连接串）
访问读写主节点	mongodb://mongouser:******@[redacted]
仅读只读节点	mongodb://mongouser:******@[redacted]t? authSource=admin&readPreference=secondaryPreferred&readPreferenceTags=role-cmgo:readonly-group
仅读从节点	mongodb://mongouser:******@[redacted]? authSource=admin&readPreference=secondaryPreferred&readPreferenceTags=role-cmgo:primary-secondary-group
仅读从节点和只读节点	mongodb://mongouser:******@[redacted] 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.

Sharded Cluster Usage Considerations

Last updated：2025-02-08 11:10:48

A sharded cluster is the distributed version of MongoDB. Compared with replica sets, sharded clusters evenly distribute data across different shards, which not only greatly increases the data capacity limit of the entire cluster but also distributes the read/write pressure across different shards, effectively solving the performance bottleneck issue of replica sets. However, the architecture of sharded clusters is more complex. This document focuses on the considerations 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.

- **Range-Based**
  Advantages: Shard key range query performance is good, read performance is good.
  Disadvantages: Data distribution may be uneven, hotspots exist.
- **Hash-Based**
  Advantages: Data distribution is even, write performance is good, suitable for high concurrency scenarios such as logs and IoT.
  Disadvantages: Range query efficiency is low.
- **Zone/Tag-Based**
  If the data has some natural distinctions, such as tags based on region or time, data can be distinguished based on tags.
  Advantages: Data distribution is more reasonable.

Choosing a 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.
- **Avoid monotonous increase or decrease**
  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	Methodology
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 shard key modification results in moving the document to another shard, you cannot specify multiple shard key modifications in batch operations; the batch size must be `1` . If shard key modification does not result in moving the document to another shard, you can specify multiple shard key modifications in batch operations.

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.

Introduction to Sharded Cluster Balance and Related Parameters

In a sharded cluster, MongoDB divides data into chunks. The balancer background process is responsible for chunk migration to balance the load across shard servers. Each chunk contains a portion of data, and the creation and migration of chunks result in balancing.

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.
Below are the commands for managing balancing. If some commands cannot be executed due to insufficient permissions, please submit a ticket to contact us for assistance.

- **Checking whether balancing is enabled for MongoDB sharded cluster**

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

  You can also check the balance status by running sh.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 version does not support sh.startBalancer(), run the following command to reopen balance:

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

- **Balancing on a collection**
  - disable balance for a collection:

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

  - enable balance for a collection:

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

  - Check whether balancing is enabled on a collection:

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

Methods for Data Import and Export by Connecting MongoDB Based on CVM

Last updated: 2025-02-08 11:11:12

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 Connect Instances.

Note:
Note: The local database mainly stores configuration information of replica sets, oplog, and other metadata; the admin database mainly stores information about users and roles. To prevent data confusion and authentication failures, TencentDB for MongoDB prohibits importing local and admin databases into instances.

Export and Import Commands

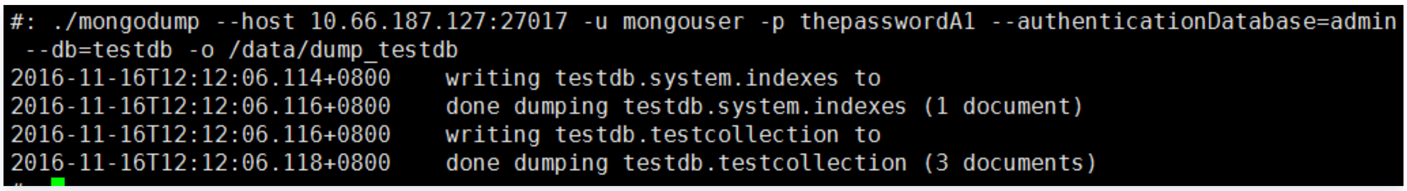
MongoDB provides two sets of official tools for data import and export:
- mongodump and mongorestore
- mongoexport and mongoimport

mongodump and mongorestore

For whole database export and import, mongodump and mongorestore are usually used. This combination operation handles data in BSON format and has higher efficiency for large-scale dump and restore.
- The export command for mongodump is as follows:

```
mongodump --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticationDatabase=admin -
-db=testdb -o /data/dump_testdb
```

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

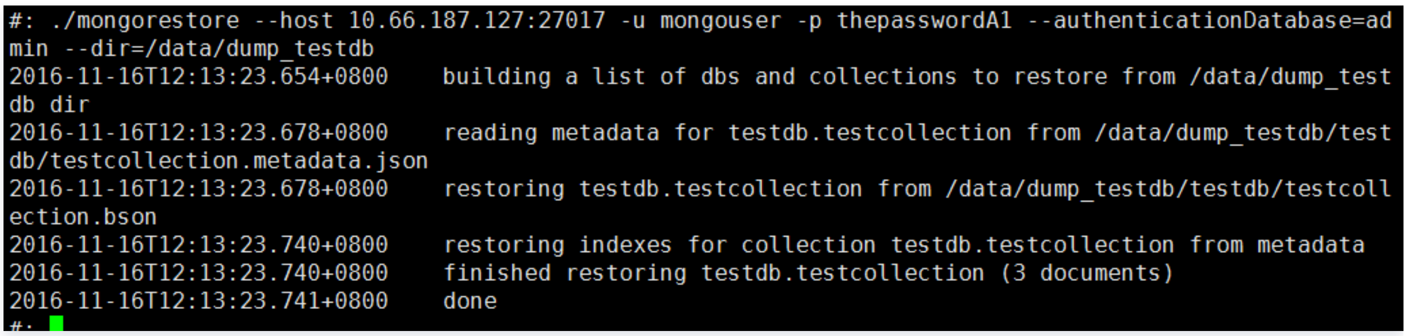


```
#: ./mongodump --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticationDatabase=admin
 --db=testdb -o /data/dump_testdb
2016-11-16T12:12:06.114+0800    writing testdb.system.indexes to
2016-11-16T12:12:06.116+0800    done dumping testdb.system.indexes (1 document)
2016-11-16T12:12:06.116+0800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

  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

When performing single collection export and import, mongoexport and mongoimport are usually used. This combination operation's data is in JSON format, which has higher readability.

- The export command for mongoexport is as follows:

```
mongoexport --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticationDatabase=admin
--db=testdb --collection=testcollection -o /data/export_testdb_testcollection.json
```

  Additionally, you can also add the -f parameter to specify the required fields and the -q parameter to specify a query condition to limit the data to be exported.
- The import command for mongoimport is as follows:

```
mongoimport --host 10.66.187.127:27017 -u mongouser -p thepasswordA1 --authenticationDatabase=admin
--db=testdb --collection=testcollection2 --file=/data/export_testdb_testcollection.json
```

Parameter Description for Multiple Authentication Methods

In the Connection Sample, it is explained that TencentDB for MongoDB default provides two usernames, "rwuser" and "mongouser", which respectively support "MONGODB-CR" and "SCRAM-SHA-1" authentication methods.

- For "mongouser" and all new users created in the console, operate according to the above example when using export and import command tools.
- For "rwuser", you need to add the parameter "--authenticationMechanism=MONGODB-CR" in each command.

Sample for mongodump:

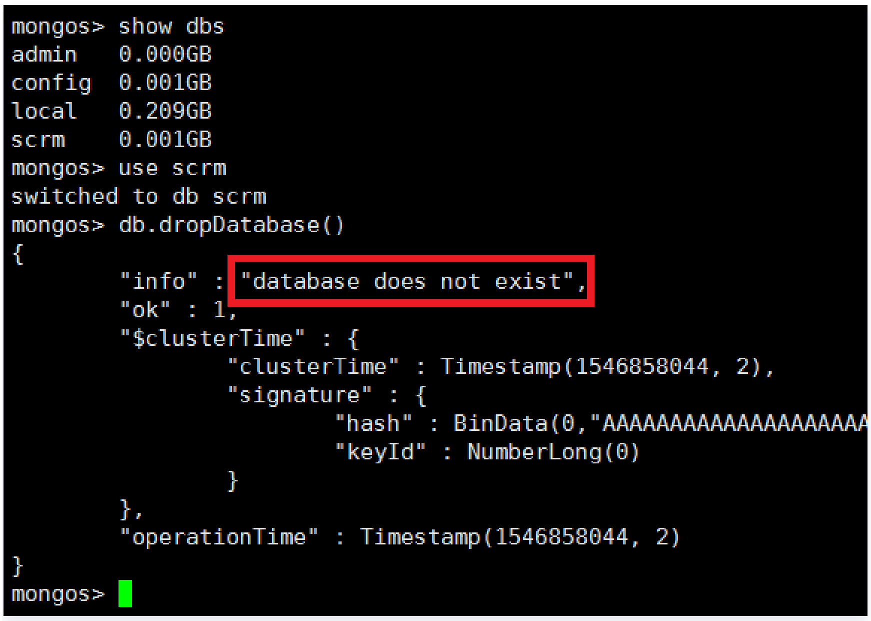
```
mongodump --host 10.66.187.127:27017 -u rwuser -p thepasswordA1 --authenticationDatabase=admin --
authenticationMechanism=MONGODB-CR --db=testdb -o /data/dump_testdb
```

Error Resolution Method for Repeatedly Creating and Deleting the Same Name Database in MongoDB 3.6

Last updated：2025-02-08 11:12:14

Problem Description

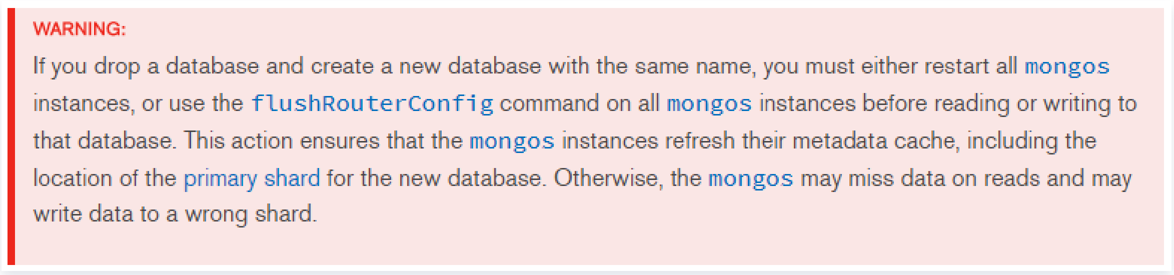
If you repeatedly drop a database and then create a database with the same name in MongoDB 3.6, you may encounter an error 'database does not exist' when reading or dropping the database, as shown in the figure below:



```
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 problem, which may be caused by mongos not updating its metadata cache. For more information, please see the official description. As shown in the figure below:



WARNING:

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

There are two solutions, please choose one of them:

1. Restart mongos in the instance list in the console.
2. Or run the flushRouterConfig command as instructed.

Troubleshooting MongoDB Connection Failures

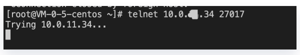
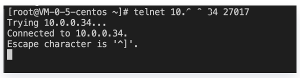
Last updated：2025-02-08 11:12:37

Scene Description

Use a CVM instance to connect to TencentDB for MongoDB via the auto-allocated intranet address. For specific connection methods, see Connecting to TencentDB for MongoDB Instance. If the connection fails, refer to the table below for troubleshooting and solutions.

Troubleshooting and Resolution

No.	Possible Causes	Troubleshooting Methods	Solution

1	The CVM and MongoDB intranet are not interconnected.
	• The CVM and database are not within the same VPC. The CVM and database must be within the same account and VPC, or within the basic network, for the internal network to directly interconnect.
	• Security group configuration error.
	　○ To use a CVM to connect to MongoDB, configure outbound rules in the CVM security group. If the target configuration of the outbound rules is not 0.0.0.0/0 and the protocol port is not ALL, add the IP and port of MongoDB to the outbound rules.
	　○ To allow the specified
		1. Log in to the CVM console and view the CVM network information in the **Instance Configuration** of the instance list.
		2. Log in to the MongoDB console and view the MongoDB network information in the instance list. For specific operations, see Viewing Instance Details.
		3. Compare whether CVM and MongoDB belong to the same network.
		4. Log in to CVM, use `telnet 10.x.x.34 27017` to confirm whether the MongoDB network port is accessible.
		　○ Connectivity issue, as shown in the figure below.
		
		`[root@VM-0-5-centos ~]# telnet 10.0.[redacted].34 27017`
		`Trying 10.0.11.34...`
		　○ Connection link is successful, as shown in the figure below.
		
		`[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 following situations may cause connection failure due to network issues.
			• CVM uses VPC, while MongoDB uses basic network. It is recommended to switch MongoDB from basic network to VPC network. Please refer to Switching Instance Network.
			• CVM uses basic network, while MongoDB uses VPC. It is recommended to switch CVM from basic network to VPC network. See Switching Private Network Service.
			• CVM and MongoDB are in the same region but belong to different VPC networks. It is recommended to migrate MongoDB to the VPC network where CVM is located. See Switching Instance Network.
			• CVM and MongoDB are in different regions and belong to different VPC networks. It is recommended to establish a Cloud Connect Network between the two VPC networks.
			• CVM and MongoDB have different accounts and belong to different VPC networks. It is recommended to establish a Cloud Connect Network between the two VPC networks.

			Incorrect CVM security group configuration 1. Log in to the Security Group Console, find the security group bound to the CVM in the security group list, click the security group name to enter the details page of the security group bound to the CVM. 2. Select the **Outbound Rules** tab, and click **Add Rules.** 　○ **Type** select **Custom.** 　○ **Target** fill in the IP address (range) of your MongoDB. 　○ **Protocol Port** fill in the MongoDB internal network port.

	CVM to connect to the MongoDB instance, configure inbound rules in the MongoDB security group. If the source configuration of the inbound rules is not 0.0.0.0/0 and the protocol port is not ALL, add the IP and port of the CVM to the inbound rules.		○ **Policy** select **Allow.** **MongoDB security group configuration error** 1. Log in to the security group console, find the security group bound to the MongoDB instance in the security group list, click the security group name, and enter the MongoDB bound security group details page. 2. Select the **inbound rules** tab, click **add rule.** Fill in the IP address (range) you allow to connect and the port information (MongoDB intranet port) that needs to be opened, and select allow. ○ **Type** select Custom. ○ **Source** fill in your CVM's IP address (range). ○ **Protocol Port** fill in the MongoDB internal network port. ○ **Policy** select **Allow.**
2	Incorrect username or password.	Log in to the CVM, connect to the database instance, and it prompts incorrect account or password. For example, if it prompts `Error: Authentication failed` , it means the username or password is incorrect.	Log in to the MongoDB console, on the **Instance Details** page, select the **Database Management** tab, go to the **Account Management** page, view all database account information, and reset the password. For specific operations, see Account Management.
3	The database access password contains special characters such as % and @.	The password contains special characters % and @. The driver or mongoshell client did not automatically escape these special characters, causing a connection string address conflict and resulting in incorrect username or password. It prompts `Password cannot properly be URL decoded` or `Error: Authentication failed` .	Process the special characters in the access password according to the following escape rules: • Exclamation mark "!" : escape to %21 • At "@" : escape to %40 • Warning sign "#" : escape to %23 • Percentage sign "%" : escape to %25 • Caret "^" : escape to %5e • Asterisk "*" : escape to %2a • Left parenthesis "(" : escape to %28 • Right parenthesis ")" : escape to %29 • Underscore "_" : escape to %5f For example, if the original password is: `^%@132121a` , the escaped password should be: `^%25%40132121a` .
4	Outdated mongoshell version	Log in to CVM and run `mongo --version` to confirm version information.	To ensure authentication success, install Mongo Shell 3.0 or above. For installation steps, see official documentation.
5	The authentication library is not correctly used in the client connection string.	• Users created in the console: Check if the connection string configured in the client program	• Log in to the TencentDB for MongoDB console, in the **instance details** page under the **network configuration** section, copy the URI format connection string for the default account. For other accounts, modify the authentication

	• Users created in the console: TencentDB for MongoDB uses the admin database as the authentication database for login. In the URI, "/admin" must be added after the port to specify the authentication database. After authentication, switch to the specific business database for read and write operations. • Users created in the command line: Directly specify the corresponding database for authentication. For example, for users created under the test database, specify the authentication database as test when logging in.	includes "/admin" or authSource=admin. • Users created in the command line: Please check if the authentication database in the connection string is the correct database name.	database to the correct one before attempting to connect. For specific operations, refer to Connecting to TencentDB for MongoDB Instance. • If the above methods still do not solve the problem, you can also contact aftersales via online consultation.

6	There are operations blocking other requests. If a foreground index creation operation (background option set to false) is performed during a busy business period, it will block all other operations, causing requests to be locked until the index creation is completed. For specific index creation methods, refer to the MongoDB Official Website.	The business side should investigate the index creation methods on their own.	Create indexes in the background. However, creating indexes in the background has its costs and may result in longer index creation times. For specific index creation options, refer to the MongoDB Official Website. Meanwhile, you can use the `currentOp` command to check the progress of the current index creation. The specific command is as follows:
			`db.currentOp(`
			`    {`
			`      $or: [`
			`        { op: "command",`
			`"query.createIndexes": { $exists: true } },`
			`        { op: "insert", ns:`
			`/\.system\.indexes\b/ }`
			`      ]`
			`    }`
			`    )`

7	Check if the customer connection count	Log in to the MongoDB console, go to the database management page, select the	For solutions for high connection usage rate, see Troubleshooting High Connection Utilization.

	has reached the limit. Each instance has a connection limit, and exceeding this limit will prevent connections.	**Connection Management** tab, and check the instance's **Maximum Connections**, **Real-time Connection Count**, and **Connection Usage Rate**. For specific operations, see Connection Management.	

Note:
Users created in the official website console have admin as the authentication database, so they need to specify the authentication database as admin when logging in. Users created via the command line, such as users created under the test database, need to specify the authentication database as test when logging in.

Performance Fine-Tuning
Analysis and Troubleshooting of High Connection Utilization

Last updated: 2025-02-08 11:13:13

Scene Description

- The service model of MongoDB is one-thread-per-connection, where each network connection is handled by an individual thread. When there are too many network connections, excessive threads will lead to increased context switching overhead and memory overhead. Establishing a connection and authentication for each request will greatly affect performance. Therefore, limiting the number of instance connections and standardizing and releasing connections in a timely manner after use are necessary conditions to ensure database stability.
- Log in to the MongoDB console, and on the **system monitoring** page, view the trend change chart of the instance monitoring metrics **connection percentage**. Connection percentage refers to the ratio of the current number of connections in the cluster to the maximum connection count. If the maximum connection count is reached, it will cause the connection response to slow down or even connection failure. Therefore, when the connection usage rate exceeds 85%, please perform timely troubleshooting.

Troubleshooting and Resolution

No.	Possible Causes	Troubleshooting Methods	Solution
1	If using a connection pool, a large number of connection resources may be occupied due to improper configuration of connection parameters. For the meaning of connection pool parameters, please refer to connection pool usage recommendations.	Please check whether the client connection pool configuration parameters are suitable for the business scenario.	Refer to the connection pool usage recommendations for connection pool parameter configuration.
2	There are many connections without actual business requests on the business side.	Use the **TencentDB for DBbrain** (DBbrain) **diagnosis and optimization** feature to check the **Real-Time Session** page for client information of all connections on the business side to see if they are actual real business connections.	For unnecessary connections, you can directly perform the kill operation on the **Real-Time Session** page in the **diagnosis and optimization** feature of **TencentDB for DBbrain** (DBbrain) for cleanup. For specific operations, refer to diagnosis and optimization.
3	There are many slow queries, and connections are occupied without being released.	1. Use the **TencentDB for DBbrain** (DBbrain) **diagnosis and optimization** feature to query all current slow log records and execution information statistics and views on the **Slow SQL Analysis** page. For details, refer to diagnosis and optimization. 2. Log in to the MongoDB console, click the instance ID to enter the **Instance Details** page,	For slow queries, optimize indexes. • Use the Index Recommendation feature of TencentDB for DBbrain (DBbrain) to select the optimal index. • Refer to the best practices in Optimizing Indexes to Break Through Read/Write Performance Bottlenecks to improve database performance.

		select the **Database Management** tab, click **Querying slow log**, and perform abstract investigation to check specific slow query information. For specific query methods, refer to Manage slow logs.	
4	Connection leakage, there are unreleased connections.	Restarting mongos instance will interrupt all connections of the instance at the moment of restart, but the business can directly reconnect. Therefore, there is no continuous impact on business. If the number of business connections increases rapidly and the connection usage rate 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 specific operations, please refer to Restart Instance.	• Directly increase connections in the console to temporarily resolve sudden business situations. For specific operations, please refer to Connection Management. • Adjust instance specifications. For replica set instances, increase the CPU and memory configuration of Mongod to simultaneously increase the maximum connection number of the instance. For specific information, please refer to Adjust the mongod node specification. For sharded clusters, increase the specification of Mongos nodes or increase the number of shards. For specific operations, please refer to Adjust the mongos node specification and Adjusting Shard Quantity.
5	Sudden increase in business volume, current instance quota is insufficient.	Please refer to the troubleshooting methods in item 2.	

Connection Pool Usage Recommendations

Taking go language as an example, the parameters that need to be configured when the client connects to the database through the connection pool are explained. For more information, see the table below. For other language types, find the corresponding connection pool parameters for configuration. For more parameter information on different language types, refer to the MongoDB Official Website.

Parameters	Unit	<0>Parameter meaning<0>	Configuration Suggestion
maxPoolSize	Quantities	Configure the maximum connection quantity each client can request in the connection pool.	The product of this parameter and the number of clients must be less than the maximum connections of the instance to avoid excessive number of connections and connection failure.
minPoolSize	Quantities	Configure the minimum number of connections each client can request in the connection pool.	The product of this parameter and the number of clients should be less than the maximum connections of the instance to avoid creating too many connections during a business surge, which may lead to insufficient supply of backend instance resources.
socketTimeoutMS	Millisecond	Configure the timeout for sending and receiving sockets to wait for a response. The	It is recommended to set it according to the actual business scenario to avoid the client never receiving a response packet from the server after a MongoDB server-side exception causes a primary-secondary

		default is 0, meaning no timeout.	switch, which would result in this invalid connection resource being occupied.
maxIdleTimeMS	Millisecond	Configure the maximum time an idle connection exists before being deleted or closed.	It is recommended to adjust the business to within 1 hour to avoid idle connections occupying connection resources.
heartbeatFrequencyMS	Millisecond	Configure the frequency of client sending heartbeat to server. It is used for the client to regularly check the survival status of backend database connection.	It is recommended to set it within 10s to identify server operating status at the first time and avoid invalid connections.

FAQs

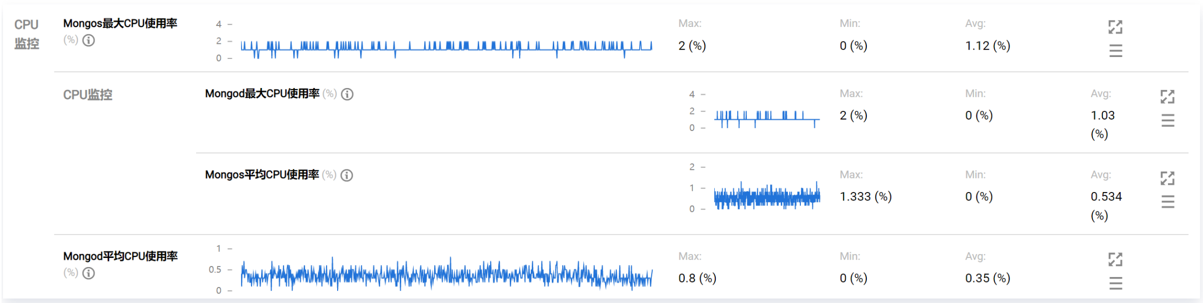
Connection limit exceeded, please increase connection count or restart instance to resolve. For specific information, refer to solutions for connection limit exceeded.

Solutions for High CPU Utilization

Last updated：2025-02-08 11:13:51

Problem Description

For daily operation and maintenance, log in to the TencentDB for MongoDB console, click the instance ID to enter the **Instance Details** page, select the **System Monitoring** tab, check the monitoring metrics details of the instance, and find that the database **CPU monitoring** metrics stay significantly high.



Cause Analysis and Solutions

No.	Possible Causes	Analysis of Causes	Troubleshooting Methods	Solution
1	Frequent Short Connections	The instance's large resource consumption in handling frequent short connections leads to high CPU utilization and a high number of connections, yet the QPS (queries per second) is not meeting expectations.	Utilize the **Diagnosis and optimization** feature of TencentDB for DBbrain to analyze the real-time session statistics and data of the database instance, confirming whether there is a phenomenon of sudden increase in connections. For the specific query method, please refer to Real-Time Session.	Adjust non-persistent connections to persistent connections. For more parameter information on different language types, please refer to MongoDB Official Website.
2	There are many unexpected requests on the business side, and requests take a long time.	Many unexpected requests or being maliciously requested may cause excessive CPU resource waste.	Please use the **DBbrain** (TencentDB for DBbrain, DBbrain) **Diagnosis and optimization** feature to troubleshoot all client requests on the business side on the **Real-Time Session** page to determine whether it is actually needed for the business.	• For unexpected requests, you can directly perform the Kill operation on the **DBbrain** (TencentDB for DBbrain, DBbrain) **Diagnosis and optimization** page, in the **Real-Time Session** section for cleanup. For detailed operations, refer to Real-Time Session. • In the **DBbrain** (TencentDB for DBbrain, DBbrain) **Diagnosis and optimization** page, create SQL throttling tasks in the **SQL Throttling** section. You can set the SQL type, maximum concurrency, throttling duration, and SQL keywords to control the database request volume and SQL concurrency, ensuring service availability. For detailed operations

				and application cases, refer to SQL Throttling.
3	There are high complexity requests.	High complexity requests may cause excessive CPU resource consumption for the following reasons. • Sorting operation: Sorting during the query process itself consumes significant CPU resources. • Full table scan: If the database does not use indexes, the request may perform a full table scan, putting significant pressure on CPU and IO resources, leading to performance degradation. • Unreasonable indexes: This may result in a large number of physical reads and logical reads, occupying significant CPU resources due to insufficient covered fields or unreasonable order of index fields.	High complexity query requests are usually time-consuming and may generate slow logs. You can use the DBbrain's **diagnosis and optimization** feature for **Slow SQL Analysis**. In the slow log information list, check for relevant keywords. For specific operations, see Slow SQL Analysis. • Sorting related keywords: Sort, hasSortStage, etc. • Full table scan keywords ○ COLLSCAN: indicates that a full-collection scan is performed for the query. ○ docsExamined: represents document scan entries. The larger the docsExamined value, the more document entries are scanned, consuming more CPU resources. • Unreasonable index keywords ○ IXSCAN: indicates that an index scan is performed. ○ keysExamined: specifies index scan entries. The larger the keysExamined value, the more entries are scanned, consuming more CPU resources.	For slow queries, optimize indexes. • You can use **DBbrain** Index Recommendation to choose the optimal index. • Refer to the best practices in Optimizing Indexes to Break Through Read/Write Performance Bottlenecks to improve database performance.
4				
5				
6	Business increase, insufficient resources	Increase in business volume, data scale growth, insufficient CPU specifications of the current instance.	Log in to the MongoDB console, click **Instance ID** to enter the **instance details** page, select the **system monitoring** tab, and view the instance's total request metrics QPS to check if the requests	Adjust the instance configuration specifications. • Mongod node: Upgrade the CPU and memory configuration of the mongod. For specific information, see Adjust the mongod node specification.

			per second for the node are significantly high.	• Mongos node: If Mongos reaches a bottleneck, upgrade the Mongos node specification or increase the number of Mongos nodes. For specific operations, see Adjust the Mongos node specification and Adding Mongos Node.

Memory Tunning Method

Last updated：2025-02-08 11:14:16

Memory optimization is not simply about reducing memory usage. It aims to ensure that memory usage is sufficient and stable while maintaining system performance, achieving an optimal balance solution between machine resources and performance. This article explains the reasons for memory usage based on the common categories of memory overhead in TencentDB for MongoDB, and provides troubleshooting methods and solutions to help you optimize database performance in a timely manner.

Storage Engine Memory

TencentDB for MongoDB limits the cache size of the WiredTiger storage engine to 60% of the actual instance memory specification size. If the storage engine cache approaches 95% of the limit, it indicates a high instance load. If the proportion of dirty data in the storage engine cache reaches 20%, it will block user threads.

Troubleshooting Methods

- In MongoDB Shell, you can check the usage of engine memory with the `db.serverStatus().wiredTiger.cache` command. The value after `bytes currently in the cache` in the returned information indicates the size of the cached data in bytes. As shown in the figure below:

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

- In the **DBbrain** performance trend MongoStatus feature, you can view the real-time proportion of dirty data in the storage engine cache. For specific information, see mongostat.

Usage recommendations

- If the storage engine cache dirty data continuously rises above 20%, handle according to the following steps:
  a. control the amount of data written per unit time.
  b. improve the instance Mongod node memory specification.
  c. increase the number of threads for cleaning dirty data. The higher the number of threads, the more instance resources it consumes. Please carefully adjust the appropriate value. The default is `threads_max=4, threads_min=1`, set as follows:

```
db.runCommand({"setParameter":1, "wiredTigerEngineRuntimeConfig":"eviction=
(threads_max=8,threads_min=4)"})
```

> **Note:**
> Adjust the number of threads for cleaning dirty data through db.runCommand, only applicable to replica set architecture version 4.0 and later.

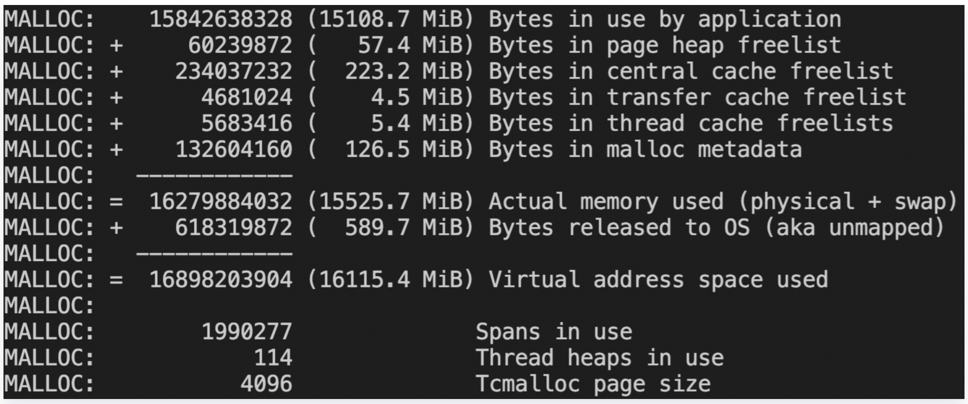
- If the storage engine Cache Used continuously rises above 95%, follow these steps:
  a. Analyze the slow logs in the database through **DBbrain**'s Slow SQL Analysis.
  b. improve the instance Mongod node memory specification.
  c. Increase the number of threads for cleaning up dirty data.

Connection and Request Memory

- Each connection requires a corresponding request thread for processing. Too many request threads can cause frequent context switching, leading to increased memory overhead. Each thread can use up to 1MB of thread stack, typically ranging from tens of KB to a few KB.
- Whenever a request command is received, a request context is created. During the process, many temporary buffers may be allocated, such as request packets, response packets, and sorted temporary buffers. These buffers are released at the end of the request. However, this release only returns them to the memory allocator `tcmalloc`. `tcmalloc` first returns these buffers to its own cache and then gradually returns them to the operating system. In many cases, the reason for high memory usage is that `tcmalloc` fails to return memory to the operating system in time, causing memory to reach up to tens of GB.

Troubleshooting Methods

- The memory size not returned to the operating system by `tcmalloc` can be viewed using the command `db.serverStatus().tcmalloc.tcmalloc.formattedString` or `db.serverStatus().tcmalloc`. As shown in the figure below, use `db.serverStatus().tcmalloc.tcmalloc.formattedString` to query memory information. The `Bytes in use by application` indicates the actual memory consumption by the Mongod node, and `Bytes in page heap freelist` is the memory not returned 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 MongoDB console, and on the **System Monitoring** page, view the trend change chart of the instance monitoring metric **connection percentage**. Connection percentage refers to the proportion of the current cluster's connections to the maximum number of connections.

Usage recommendations

1. Adjust memory recovery rate.
   - `tcmallocReleaseRate` is a parameter to control memory release. It specifies how much memory should be released when memory usage exceeds a certain proportion. The higher the value of `tcmallocReleaseRate`, the faster MongoDB releases memory, but it may also impact performance. When configuring `tcmallocReleaseRate`, adjust according to actual situation to balance memory usage and performance needs. Execute the following command to configure the value of `tcmallocReleaseRate`, which ranges from [1,10].
   - `tcmallocAggressiveMemoryDecommit` set to **1** can quickly recover memory directly. The execution method is as follows:

   ```
   db.adminCommand( { setParameter: 1, tcmallocReleaseRate: 5.0 } )
   db.runCommand( { getParameter: 1, tcmallocAggressiveMemoryDecommit:1} )
   ```

   Note:
   `tcmallocReleaseRate` is applicable to TencentDB for MongoDB version 4.2 and above.

2. Control the number of concurrent connections.
   It is recommended to create a maximum of 100 persistent connections in the database. By default, the MongoDB Driver can establish 100 connection pools with the backend. When there are many clients, reduce the connection pool size of each client. Generally, it is recommended to control the total number of persistent connections to within 1000. If the connection percentage is high, refer to analysis and solutions for high connection usage anomaly for optimization.

3. reduce memory overhead per request.

   For example, reduce collection scans and memory sorting by creating indexes. For creating indexes, refer to MongoDB official Indexes.
4. upgrade memory specifications.

   If memory usage rate continues to rise with an appropriate number of connections, it is recommended to upgrade memory specifications to avoid memory overflow and significant performance degradation due to excessive cache clearing.
   - improve the CPU and memory configuration of Mongod. For details, refer to Adjust the mongod node specification.
   - Sharded cluster, you can increase the number of shards. For specific operations, refer to Adjusting Shard Quantity.

Metadata Information Memory

TencentDB for MongoDB maintains some metadata for each collection, such as index information, number of documents, storage engine options, etc. Managing this information will occupy certain system resources. If there is a large amount of in-memory metadata for collections and indexes, it will occupy a large amount of memory. Especially in versions prior to TencentDB for MongoDB 4.0, a large number of file handles are opened during full logical backups but are not returned to the operating system in time, causing memory to rise rapidly. Additionally, in lower versions of TencentDB for MongoDB, deleting a large number of collections may not remove file handles, which can also cause memory leaks.

Troubleshooting Methods

- To count the number of collections and indexes in TencentDB for MongoDB, use the following commands respectively:

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

- To query the number of collections, the execution example is as follows:

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

- To query the number of indexes for each collection, the execution example is as follows:

```
mongos> db.getCollectionNames().forEach(function(col) {     print("Indexes for " + col + ": " +
db.getCollection(col).getIndexes().length); })
Indexes for nba: 3
Indexes for t1: 1
Indexes for test: 1
Indexes for user: 1
```

Usage recommendations

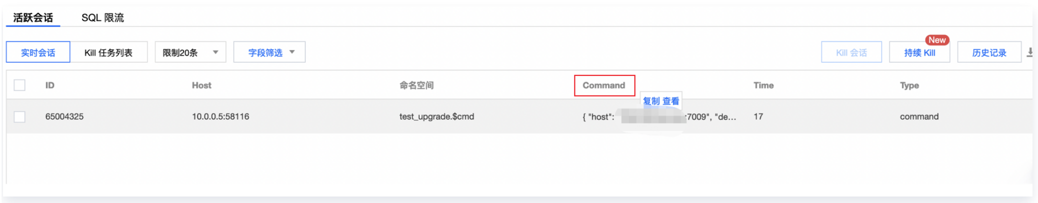
In MongoDB, it is recommended to keep the number of collections within thousands. Otherwise, the performance of MongoDB in managing these collections will significantly decline.

Memory Consumption During Index Creation

MongoDB may occupy a large amount of memory when creating indexes. Under normal business data writing, the secondary node maintains a buffer of about 256M for data playback. Before MongoDB 4.2, when the primary node created indexes in a non-background mode, the backend would playback the operation records of index creation in a serial manner, potentially consuming up to 500M of memory. After MongoDB 4.2, the background option was deprecated by default, meaning indexes are created in the foreground method. However, it allows secondary nodes to parallel playback the primary node's recorded operations for index creation, speeding up the index creation process. At the same time, this will use a certain amount of memory overhead on the secondary nodes. Therefore, creating multiple indexes simultaneously may lead to instance memory overflow. Additionally, secondary nodes may consume more memory during data playback, especially after the primary node completes index creation.

Troubleshooting Methods

On the **DBbrain** Real-Time Session page, under the **Active Session** tab, check if there are multiple processes adding indexes. As shown in the figure below, you can find the CreateIndex key fields in the **Command** column.



ID	Host	命名空间	Command	Time	Type
65004325	10.0.0.5:58116	test_upgrade.$cmd	{ "host": [redacted] :7009", "de...	17	command

Usage recommendations

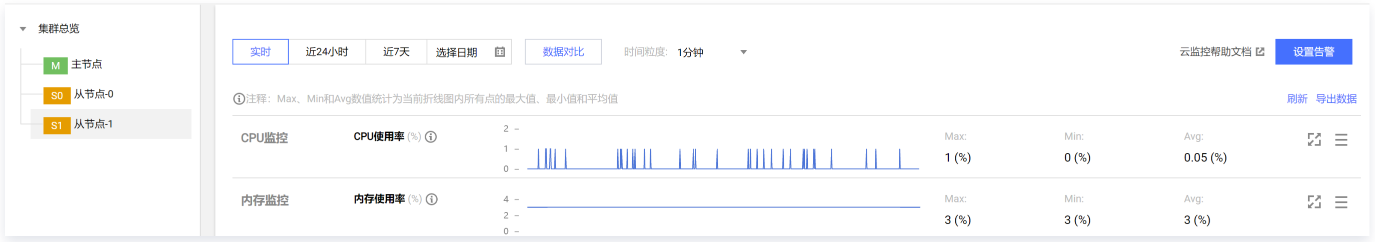
- Avoid creating multiple large indexes simultaneously during peak periods. You can spread out the creation time of the indexes.
- Try to create indexes using the background method to avoid occupying too much memory. For specific index creation methods, refer to the MongoDB official website .
- If there is insufficient memory, consider increasing system memory. For specific operations, refer to Adjust the mongod node specification .
- Consider using more efficient index types, such as text search indexes or geospatial indexes, to reduce memory usage. Use the Index Recommendation feature in DBbrain to select the optimal index.
- For larger collections, consider using sharding technology to distribute the storage of data and indexes, reducing memory pressure on a single instance.

Memory Generated by Logical Backup and Primary-Secondary Switch

- Logical backups usually generate large data scans, leading to high memory usage. If logical backups are performed on secondary nodes or hidden nodes, the memory usage of secondary nodes will be significantly higher than that of the primary node.
- Primary-secondary node switch: the original primary node becomes a secondary node, resulting in the current secondary node having significantly higher memory usage than the current primary node.

Troubleshooting Methods

1. Log in to the MongoDB console and check the **Task Management** page to see if there are any ongoing backup tasks. The currently executing tasks are usually at the top of the task list. Click **Operation** in the **Task Details** column to confirm if it is a **Logical Backup**. For detailed operations, refer to Task Management .
2. On the **Task Management** page, search for **Task Type** and **Switch Primary Node** to confirm if a primary-secondary node switch has been performed on the business side.
3. Click the **System Monitoring** tab, and in the left cluster overview dropdown menu, view the **Memory Usage** of the primary and secondary nodes separately.



时间粒度: 1分钟

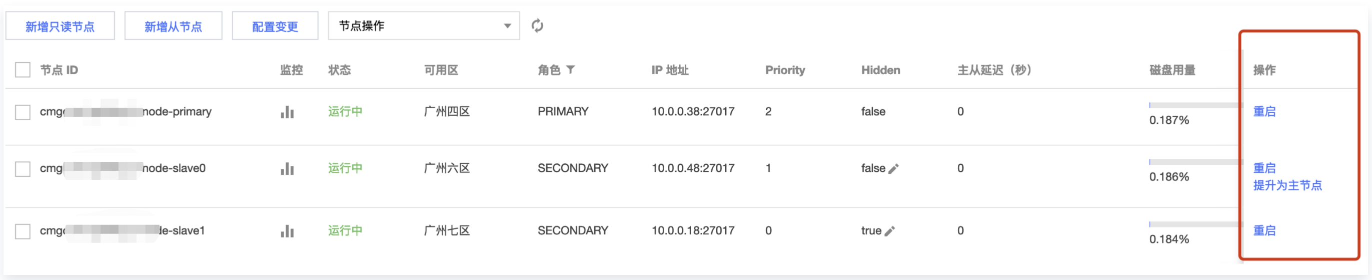
注释：Max、Min和Avg数值统计为当前折线图内所有点的最大值、最小值和平均值

		Max:	Min:	Avg:
CPU监控	CPU使用率 (%)	1 (%)	0 (%)	0.05 (%)
内存监控	内存使用率 (%)	3 (%)	3 (%)	3 (%)

Usage recommendations

- Physical backup is performed through physical file copy, thus consuming relatively less memory during the backup process. If high memory usage on secondary nodes is a concern, choose the physical backup method. For the backup methods supported by each version, refer to Data Backup.

- If the business side has not set to read from the secondary node after the master–slave switch, please restart the corresponding secondary node in the console to release memory. If there are requests to the secondary node, it is recommended to perform this operation during low traffic periods.



节点 ID	监控	状态	可用区	角色	IP 地址	Priority	Hidden	主从延迟（秒）	磁盘用量	操作
cmg[redacted]node-primary		运行中	广州四区	PRIMARY	10.0.0.38:27017	2	false	0	0.187%	重启
cmg[redacted]node-slave0		运行中	广州六区	SECONDARY	10.0.0.48:27017	1	false	0	0.186%	重启 提升为主节点
cmg[redacted]le-slave1		运行中	广州七区	SECONDARY	10.0.0.18:27017	0	true	0	0.184%	重启

Cause Analysis and Solutions for High Latency Due to Increased Slow Queries

Last updated：2025-02-08 11:14:41

Phenomenon Description

Slow logs record queries that exceed the threshold query time. When there are many slow queries in the system, it can lead to decreased system performance, longer response times, and even system crashes. Therefore, it is necessary to optimize slow queries, reduce their number, and improve system performance.
Log in to the TencentDB for MongoDB console, click the instance ID to enter the **Instance Details** page, select the **System Monitoring** tab, and check the instance's monitoring data. You will find that the database latency monitoring metrics have significantly increased. Latency monitoring metrics mainly reflect the time from when a request reaches the access layer to when the request is processed and returned to the client. For specific monitoring metrics, refer to the Monitoring Overview.

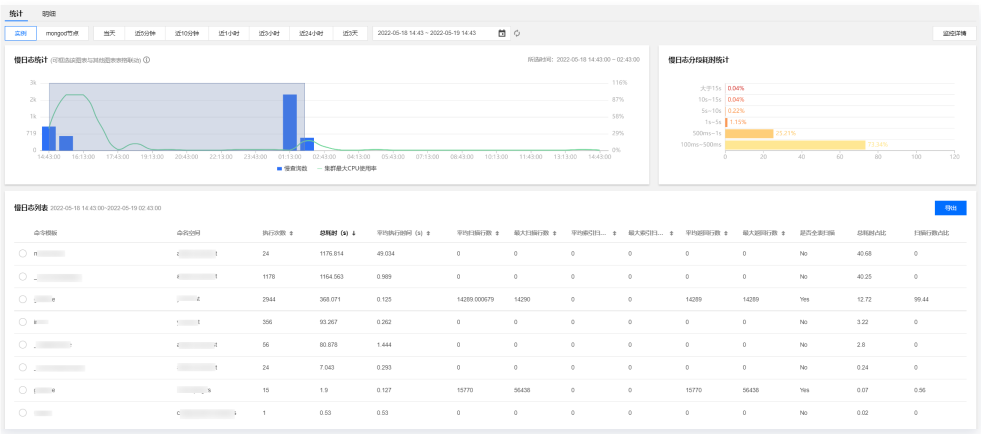
Possible Reasons

- Using the $lookup operator query without an index or with an index that does not support the query requires a complete scan of the entire database, resulting in very low retrieval efficiency.
- Some documents in your collection have many large array fields that are time-consuming to search and index, causing a high system load.

Analyzing Slow Queries

Analyzing Slow SQL with TencentDB for DBbrain to Troubleshoot Slow Queries (Recommended)

TencentDB for DBbrain (DBbrain) is a cloud database autonomy service provided by Tencent Cloud for database performance optimization, security, and management. The Slow SQL Analysis for MongoDB is specifically used to analyze slow logs generated during MongoDB operations. The diagnostic data is intuitive and easy to find, as shown in the figure below. For more information, see Slow SQL Analysis.



慢日志列表 2022-05-18 14:43:00~2022-05-19 02:43:00

命令模板	命名空间	执行次数	总耗时 (s)	平均执行时间 (s)	平均扫描行数	最大扫描行数	平均索引扫...	最大索引扫...	平均返回行数	最大返回行数	是否全表扫描	总耗时占比	扫描行数占比
[redacted]	[redacted]	24	1176.814	49.034	0	0	0	0	0	0	No	40.68	0
[redacted]	[redacted]	1178	1164.563	0.989	0	0	0	0	0	0	No	40.25	0
[redacted]	[redacted]	2944	368.071	0.125	14289.000679	14290	0	0	14289	14289	Yes	12.72	99.44
[redacted]	[redacted]	356	93.267	0.262	0	0	0	0	0	0	No	3.22	0
[redacted]	[redacted]	56	80.878	1.444	0	0	0	0	0	0	No	2.8	0
[redacted]	[redacted]	24	7.043	0.293	0	0	0	0	0	0	No	0.24	0
[redacted]	[redacted]	15	1.9	0.127	15770	56438	0	0	15770	56438	Yes	0.07	0.56
[redacted]	[redacted]	1	0.53	0.53	0	0	0	0	0	0	No	0.02	0

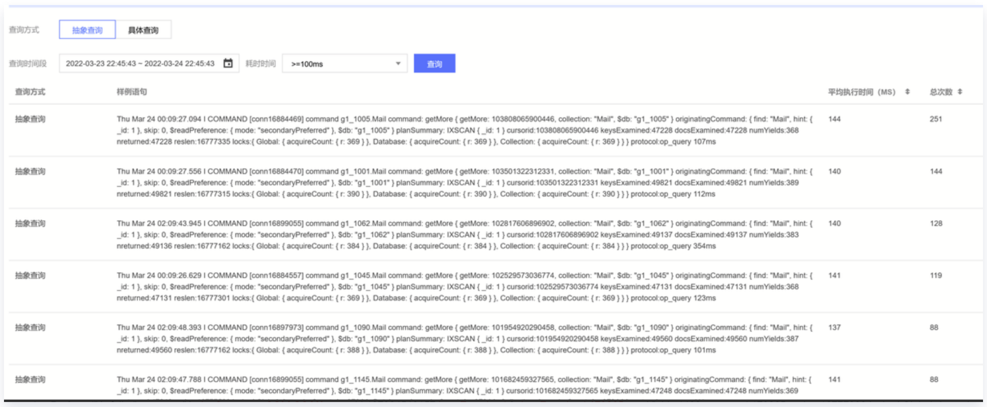
Analyzing Slow Queries Based on MongoDB Console's Slow Log

Before MongoDB is connected to TencentDB for DBbrain (DBbrain), you can obtain slow logs from the MongoDB console and analyze key fields one by one to troubleshoot the causes of slow queries.

Retrieving slow logs

1. Log in to the TencentDB for MongoDB console.
2. In the left navigation bar, select **MongoDB** from the drop-down list, then choose **Replica Set Instance** or **Sharded Instance**. The operations for replica set instances and sharded instances are similar.
3. Select Region at the top of the Instance List page on the right.
4. In the Instance List, find the target instance.

5. Click the Target Instance ID to enter **Instance Details**page.
6. Select the **Database Management** tab, then select the **Slow Log Query** tab.
7. In the **Slow Log Query** tab, analyze slow logs. The system logs operations executed for more than 100 milliseconds and retains slow logs for 7 days. It also supports downloading slow log files. For specific operations, see Manage slow logs.
   - **Abstract query**: Statistical value after fuzzy processing of query conditions. Here you can see slow query statistics sorted by average execution duration. We recommend optimizing the Top 5 requests first.
   - **Specific query**: Records complete user execution requests, including: execution plan, number of scanned rows, execution duration, and some lock wait information.



查询方式	样例语句	平均执行时间（MS）	总次数
抽象查询	Thu Mar 24 00:09:27.094 I COMMAND [conn16884469] command g1_1005.Mail command: getMore { getMore: 103808065900446, collection: "Mail", $db: "g1_1005" } originatingCommand: { find: "Mail", hint: { _id: 1 }, skip: 0, $readPreference: { mode: "secondaryPreferred" }, $db: "g1_1005" } planSummary: IXSCAN { _id: 1 } cursorid:103808065900446 keysExamined:47228 docsExamined:47228 numYields:368 nreturned:47228 reslen:16777335 locks:{ Global: { acquireCount: { r: 369 } }, Database: { acquireCount: { r: 369 } }, Collection: { acquireCount: { r: 369 } } } protocol:op_query 107ms	144	251
抽象查询	Thu Mar 24 00:09:27.556 I COMMAND [conn16884470] command g1_1001.Mail command: getMore { getMore: 103501322312331, collection: "Mail", $db: "g1_1001" } originatingCommand: { find: "Mail", hint: { _id: 1 }, skip: 0, $readPreference: { mode: "secondaryPreferred" }, $db: "g1_1001" } planSummary: IXSCAN { _id: 1 } cursorid:103501322312331 keysExamined:49821 docsExamined:49821 numYields:389 nreturned:49821 reslen:16777315 locks:{ Global: { acquireCount: { r: 390 } }, Database: { acquireCount: { r: 390 } }, Collection: { acquireCount: { r: 390 } } } protocol:op_query 112ms	140	144
抽象查询	Thu Mar 24 02:09:43.945 I COMMAND [conn16899055] command g1_1062.Mail command: getMore { getMore: 102817606896902, collection: "Mail", $db: "g1_1062" } originatingCommand: { find: "Mail", hint: { _id: 1 }, skip: 0, $readPreference: { mode: "secondaryPreferred" }, $db: "g1_1062" } planSummary: IXSCAN { _id: 1 } cursorid:102817606896902 keysExamined:49137 docsExamined:49137 numYields:383 nreturned:49136 reslen:16777162 locks:{ Global: { acquireCount: { r: 384 } }, Database: { acquireCount: { r: 384 } }, Collection: { acquireCount: { r: 384 } } } protocol:op_query 354ms	140	128
抽象查询	Thu Mar 24 00:09:26.629 I COMMAND [conn16884557] command g1_1045.Mail command: getMore { getMore: 102529573036774, collection: "Mail", $db: "g1_1045" } originatingCommand: { find: "Mail", hint: { _id: 1 }, skip: 0, $readPreference: { mode: "secondaryPreferred" }, $db: "g1_1045" } planSummary: IXSCAN { _id: 1 } cursorid:102529573036774 keysExamined:47131 docsExamined:47131 numYields:368 nreturned:47131 reslen:16777301 locks:{ Global: { acquireCount: { r: 369 } }, Database: { acquireCount: { r: 369 } }, Collection: { acquireCount: { r: 369 } } } protocol:op_query 123ms	141	119
抽象查询	Thu Mar 24 02:09:48.393 I COMMAND [conn16897973] command g1_1090.Mail command: getMore { getMore: 101954920290458, collection: "Mail", $db: "g1_1090" } originatingCommand: { find: "Mail", hint: { _id: 1 }, skip: 0, $readPreference: { mode: "secondaryPreferred" }, $db: "g1_1090" } planSummary: IXSCAN { _id: 1 } cursorid:101954920290458 keysExamined:49560 docsExamined:49560 numYields:387 nreturned:49560 reslen:16777162 locks:{ Global: { acquireCount: { r: 388 } }, Database: { acquireCount: { r: 388 } }, Collection: { acquireCount: { r: 388 } } } protocol:op_query 101ms	137	88
抽象查询	Thu Mar 24 02:09:47.788 I COMMAND [conn16899055] command g1_1145.Mail command: getMore { getMore: 101682459327565, collection: "Mail", $db: "g1_1145" } originatingCommand: { find: "Mail", hint: { _id: 1 }, skip: 0, $readPreference: { mode: "secondaryPreferred" }, $db: "g1_1145" } planSummary: IXSCAN { _id: 1 } cursorid:101682459327565 keysExamined:47248 docsExamined:47248 numYields:369	141	88

Analyzing key fields in slow logs

View key fields in slow logs. For the meaning of common key fields, see the table below. For more field descriptions, see MongoDB Official Website .

Key Fields	Field Description
command	Indicates the operation request recorded in the slow log.
COLLSCAN	Indicates that the query performed a full table scan. If the number of scanned rows is less than 100, the speed of full table scan will also be fast.
IXSCAN	Indicates an index scan. The specific index used will be output after this field. A table may have multiple indexes, and if the index here does not meet expectations, consider optimizing the index or modifying the query statement using hint().
keysExamined	Refers to the number of index entries scanned. "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	Used to describe the summary information of query execution. Each MongoDB query generates an execution plan that contains detailed information about the query, such as the index used, scanned document count, query execution time, etc. For example: "planSummary" : "IXSCAN { a: 1, _id: -1 }" indicates that MongoDB used an index scan (IXSCAN) query plan. Specifically, it used the index named "a" and the default "_id" index, scanning the "a" index in ascending order (1). This is a common query plan, indicating that the query used an index to retrieve the required data.
numYield	This field indicates the number of times the lock is yielded during the operation. When an operation needs to wait for certain resources (e.g., disk I/O or locks), it may relinquish CPU control so that other operations can continue execution. This process is called "yielding." A higher numYield value usually indicates a higher system load, as operations require more time to complete. Typically, document search operations (query, update, and delete) can yield locks. They only yield their locks if other operations are queued waiting for the locks held by this operation. By optimizing the number of yields in the system, you can improve system concurrency and throughput, minimize lock competition, and enhance system stability and reliability.

nreturned	Refers to the number of documents returned by the query request. The larger this value, the more rows are returned. If the keysExamined value is large and nreturned returns few documents, it indicates that the index needs optimization.
millis	The time consumed from the start to the end of the MongoDB operation. The larger this value, the slower the execution.
IDHACK	Used to accelerate queries 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 technique to speed up the operation. Specifically, the IDHACK technique can leverage the special nature of the _id field to convert the query or update operation into a more efficient one. 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 scanning the entire collection.
FETCH	This field indicates the number of documents read from the disk by MongoDB when executing a query operation. When executing a query operation, MongoDB reads matching documents from the 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. This is because MongoDB can utilize indexes and other techniques to minimize the number of documents read from the disk, thereby improving query performance.

Solution:

Cleaning Slow Queries

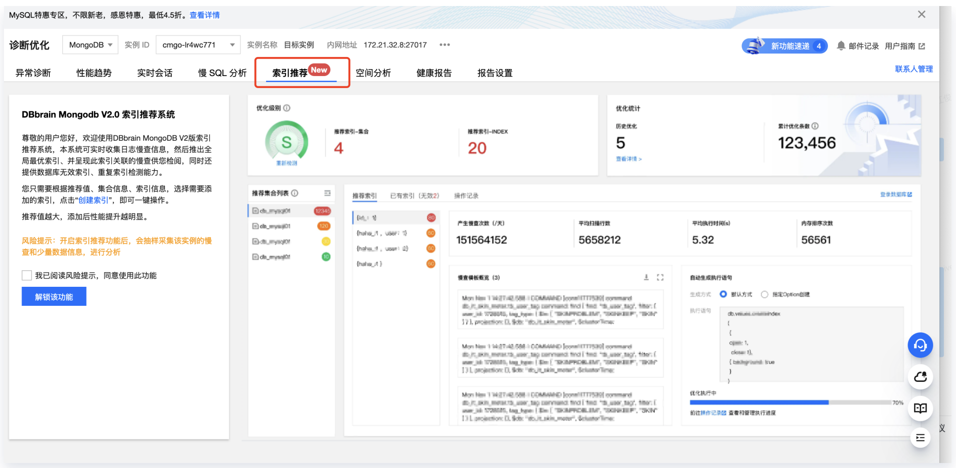
1. Select **Database Management** > **Slow Query Management** tab, and the list will display the requests being executed by the current instance (including requests of secondary nodes). You can click **Batch Kill** to kill unnecessary slow query requests. For specific operations, see Manage Slow Logs.
2. For unexpected requests, you can directly perform the Kill operation in **DBbrain** (TencentDB for DBbrain, DBbrain) under **Diagnosis and Optimization** on the **Real-Time Session** page for cleanup. For specific operations, see Real-Time Session.

SQL Throttling

For TencentDB for MongoDB 4.0, on the **DBbrain** (TencentDB for DBbrain, DBbrain) **Diagnosis and Optimization SQL Throttling** page, create an SQL throttling task, set SQL type, maximum concurrency, throttling duration, and SQL keywords to control the request volume and SQL concurrency of the database, ensuring service availability. For specific operations and application cases, see SQL Throttling.

Using Indexes

- If based on DBbrain (TencentDB for DBbrain, DBbrain) slow SQL analysis, check if the scan rows are large in the slow log list, indicating large scan requests or long-running requests.
  - If slow queries caused by full table scans increase, create indexes to reduce collection scans and memory sorting. For creating indexes, see MongoDB official Indexes.
  - If an index is used but the index scan rows are 0 while the scan rows are greater than 0, the index needs optimization. Use the Index Recommendation feature of DBbrain (TencentDB for DBbrain, DBbrain) to select the optimal index. Index Recommendation collects real-time slow log information for automatic analysis, proposes the globally optimal index, and ranks them by performance impact. The higher the recommendation value, the more significant the performance improvement after the operation.



MySQL特惠专区，不限新老，感恩特惠，最低4.5折。查看详情

诊断优化　MongoDB　实例 ID　cmgo-lr4wc771　实例名称　目标实例　内网地址　172.21.32.8:27017

DBbrain Mongodb V2.0 索引推荐系统

尊敬的用户您好，欢迎使用DBbrain MongoDB V2版索引推荐系统，本系统可实时收集日志慢查信息，然后推出全局最优索引、并呈现此索引关联的慢查供您检阅，同时还提供数据库无效索引、重复索引检测能力。

您只需要根据推荐值、集合信息、索引信息，选择需要添加的索引，点击“创建索引”，即可一键操作。

推荐值越大，添加后性能提升越明显。

风险提示：开启索引推荐功能后，会抽样采集该实例的慢查和少量数据信息，进行分析

我已阅读风险提示，同意使用此功能

优化级别: S　推荐索引-集合: 4　推荐索引-INDEX: 20

优化统计　历史优化: 5　累计优化条数: 123,456

产生慢查次数（/天）: 151564152　平均扫描行数: 5658212　平均执行时间(s): 5.32　内存排序次数: 56561

- If based on slow log analysis, handle it according to the following situations.
  - keysExamined = 0, while docsExamined > 0, and planSummary is COLLSCAN, indicating a full table scan was performed, causing significant query latency, as shown below. For creating indexes, see MongoDB official Indexes.
  - If keysExamined > 0 and docsExamined > 0, with planSummary being IXSCAN, it indicates that some query conditions or returned fields are not included in the index, requiring index optimization. Please use the Index Recommendation feature of TencentDB for DBbrain (DBbrain) to select the optimal index.
  - For key fields where keysExamined > 0 and docsExamined = 0 with planSummary as IXSCAN, it means the query conditions or returned fields are already covered by the index. If the value of keysExamined is large, it is recommended to optimize the order of index fields or add more appropriate indexes for filtering. For more information, see Optimizing Indexes to Break Through Read/Write Performance Bottlenecks.

```
Thu Mar 24 01:03:01.099 I COMMAND [conn8976420] command tcoverage.ogid_mapping_info command: getMore { getMore: 107924518157, collection: "ogid_mapping_info", $db: "tcoverage" } originatingCommand: { find: "ogid_mapping_info", skip: 0, $readPreference: { mode: "secondaryPreferred" }, $db: "tcoverage" } planSummary: COLLSCAN cursorid:107924518157 keysExamined:0 docsExamined:179370 numYields:1401 nreturned:179370 reslen:16777323 locks:{ Global: { acquireCount: { r: 1402 } }, Database: { acquireCount: { r: 1402 } }, Collection: { acquireCount: { r: 1402 } } } protocol:op_query 102ms
```

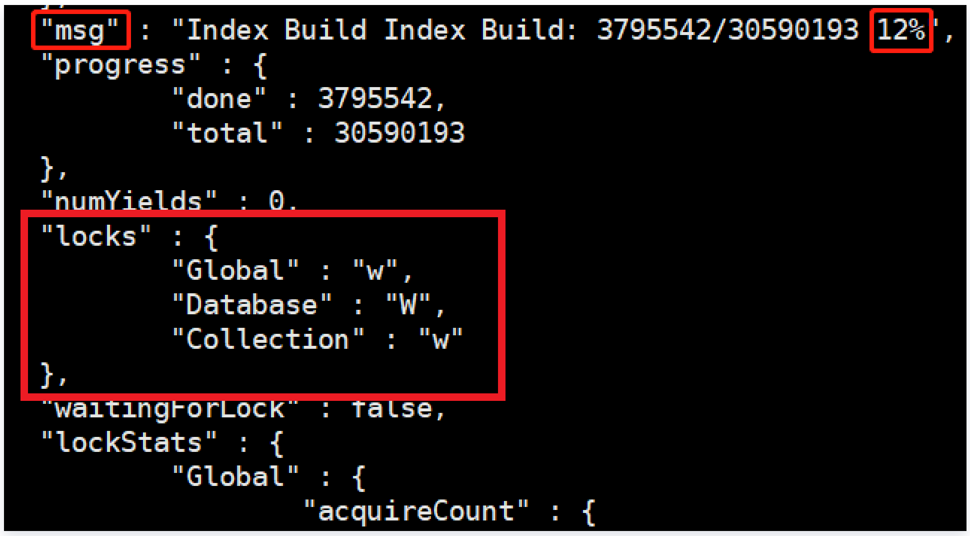
- If MongoDB is version 4.2 or earlier and there is no problem with the index used for business queries, check whether an index is created in the foreground during peak business hours and change it to the background mode.

> **Notes:**
> - Foreground index creation: In MongoDB versions before 4.2, the default method for creating an index on a collection is the foreground method, which sets the background option parameter to false. This operation will block all other operations until the index creation is completed in the foreground.
> - Creating indexes in the background: Setting the background option to true allows MongoDB to continue providing read-write operation service during index creation. However, creating indexes in the background may lead to longer index creation times. For specific methods of creating indexes, please refer to the MongoDB Official Website.

Create an index in the background, and use the `currentOp` command to check the current index creation progress. The specific command is as follows.

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

The return is shown below, the `msg` field indicates the current index creation progress, and the `locks` field represents the lock type of the operation. For more information on locks, refer to the MongoDB Official Website.



```
 "msg" : "Index Build Index Build: 3795542/30590193 12%",
 "progress" : {
         "done" : 3795542,
         "total" : 30590193
 },
 "numYields" : 0,
 "locks" : {
         "Global" : "w",
         "Database" : "W",
         "Collection" : "w"
 },
 "waitingForLock" : false,
 "lockStats" : {
         "Global" : {
                 "acquireCount" : {
```

Restarting an Instance

If there are no slow operations on mongod, but the request latency is high, the problem may be caused by a high mongos load. There are many causes for this; for example, a large number of connections are established in a short time, or data in multiple shards needs to be aggregated. In these cases, you can restart mongos. For specific operations, refer to Restarting an Instance.

Notes:
All instance connections will be interrupted at the moment of restarting mongos, but the business can be directly reconnected.

Expanding Specifications

If the problem still cannot be resolved, the instance load will exceed normal processing capacity. Pay attention to the system monitoring of read and write requests and latency metrics, and compare with the stress test data in Performance Data. If the deviation is too large, it is recommended to upgrade the configuration immediately to avoid affecting normal business.

- Mongod node: Upgrade the CPU and memory configuration of mongod. For more information, see Adjust the mongod node specification.
- Mongos node: If mongos reaches a bottleneck, upgrade the mongos node specification or increase mongos quantity. For specific operations, see Adjust the mongos node specification and Adding Mongos Node.

If slow logs accumulate too quickly and the problem cannot be resolved, please contact after-sales support or Submit a Ticket.

Troubleshooting Excessive Connections

Last updated: 2025-02-08 11:15:01

Phenomenon Description

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

Possible Reasons

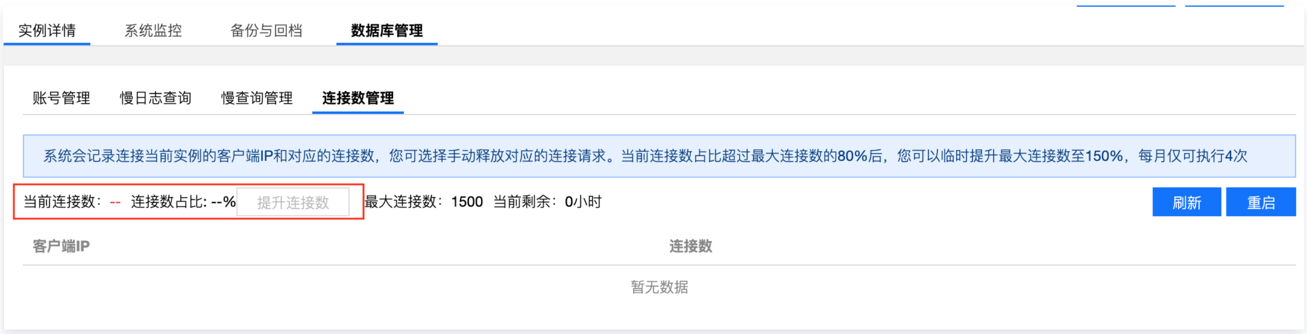
- There are connection leaks, improper client coding, unreasonable connection pool configuration, or many unreleased connections.
- There is a large number of concurrent application requests, and the current database specification cannot meet the current demand, and the configured upper limit of connections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

Processing Procedures

Method 1: Increasing the Number of Connections

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

> **Notes:**
> - 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 after increasing the number of connections to 150%, contact after-sales service or submit a ticket for assistance.



系统会记录连接当前实例的客户端IP和对应的连接数，您可选择手动释放对应的连接请求。当前连接数占比超过最大连接数的80%后，您可以临时提升最大连接数至150%，每月仅可执行4次

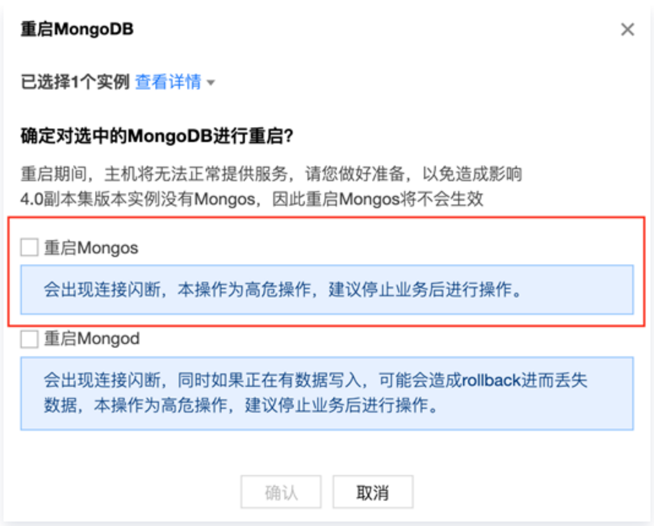
当前连接数：-- 连接数占比: --% 最大连接数：1500 当前剩余：0小时

客户端IP	连接数
暂无数据	

Option 2. Restart the instance

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

> **Notes:**
> - Replica set instances on v4.0 don't have mongos.
> - Restarting mongod is a high-risk operation and will cause a momentary disconnection. If data is being written at the same time, it may cause rollback and lead to data loss. By default, this feature is not enabled. If you need to enable it, please contact after-sales service or submit a ticket.



重启MongoDB

已选择1个实例 查看详情

确定对选中的MongoDB进行重启?

重启期间，主机将无法正常提供服务，请您做好准备，以免造成影响
4.0副本集版本实例没有Mongos，因此重启Mongos将不会生效

☐ 重启Mongos

会出现连接闪断，本操作为高危操作，建议停止业务后进行操作。

☐ 重启Mongod

会出现连接闪断，同时如果正在有数据写入，可能会造成rollback进而丢失数据，本操作为高危操作，建议停止业务后进行操作。

Option 3: Optimize the business-side service

Optimize the service. For specific troubleshooting methods, refer to Analysis and Solutions for High Connection Usage Anomaly.

Solution for High Disk Space Utilization

Last updated：2025-02-08 11:15:22

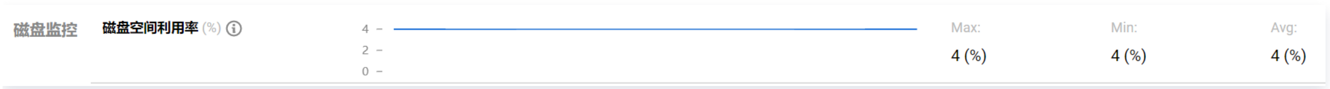
Overview

MongoDB's disk primarily stores data and indexes, as well as some system files and log files. Disk space utilization is a very important monitoring indicator. When the disk space is fully used, the MongoDB instance will be unable to continue writing new data, which will cause instance failure and stop working. Therefore, monitoring disk utilization and taking timely measures to free up disk space is key to ensure MongoDB instance runs normally.

Viewing Disk Space Usage

Quick View of Monitoring Indicators

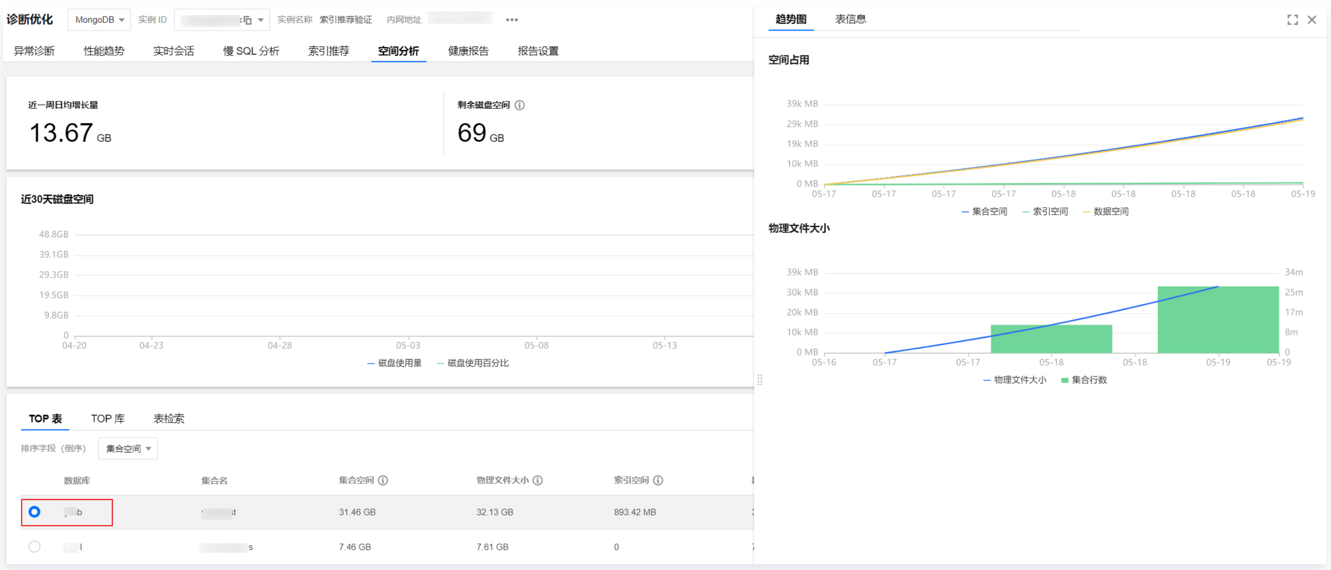
Log in to the MongoDB console, and on the **System Monitoring** tab, you can view the trend chart of **disk space utilization**. For specific operations, see Viewing Monitoring Data.



Max: 4 (%)
Min: 4 (%)
Avg: 4 (%)

Detailed Analysis of Disk Space Usage

- Log in to the MongoDB console, select **Diagnosis and Optimization** on the left navigation bar, and then select the **Space Analysis** tab. You can use the space analysis feature of TencentDB for DBbrain (DBbrain) to further analyze the disk space usage details of the database, including database collection space, index space, physical file space size, database size, data proportion, number of collection rows, and other analysis data and views. For specific operations, see Space Analysis.



近一周日均增长量
13.67 GB

剩余磁盘空间
69 GB

数据库	集合名	集合空间	物理文件大小	索引空间
[redacted]b	[redacted]t	31.46 GB	32.13 GB	893.42 MB
[redacted]l	[redacted]s	7.46 GB	7.61 GB	0

- Analyze disk space usage using MongoDB's own commands `db.stats()` and `db.$collection_name.stats()`. For specific information, see the table below.

Command Analysis	Command Meaning
db.stats()	This command is used to obtain statistical information of the current database. Executing the `db.stats()` command will return a document containing various information about the current database, such as database name, data size, index size, number of collections, etc.
db.collection.stats()	This command is used to get the statistical information of the specified collection. Executing the `db.collection.stats()` command will return a document containing

	various information about the specified collection, such as: collection names, number of documents, data size, number of indexes, etc.
db.collection.storageSize()	This command is used to get the storage size occupied by the specified collection. Executing the `db.collection.storageSize()` command will return the storage size of the specified collection in bytes. The returned storage size includes the space occupied by data and indexes in the collection, but does not include other overhead of the MongoDB instance, such as log files and temporary files.
db.collection.totalIndexSize()	This command is used to get the storage size occupied by all indexes of the specified collection. Executing the `db.collection.totalIndexSize()` command will return the storage size occupied by all indexes of the specified collection in bytes. The returned storage size 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 get the total storage size occupied by the specified collection. Executing the `db.collection.totalSize()` command will return the total storage size occupied by the specified collection in bytes. The returned storage size includes the space occupied by data and indexes in the collection, as well as other overhead of the MongoDB instance, such as log files and temporary files.

Analysis

TencentDB for MongoDB uses the WiredTiger engine by default. When deleting documents, disk space is not immediately reclaimed. When new data is inserted, MongoDB reuses the previously occupied space instead of occupying additional new disk space. As delete operations increase, fragmentation also increases. The following code can be used to view the fragmentation rate of all collections in the specified databases at once.
When the instance disk space utilization reaches 80%~85% or more, you can avoid the risk of space full by reducing the actual occupied space of the database or expanding the storage space.

```
Function to generate and check fragmentation rate
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 for reuse'];
        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) / totalStorageSize).toFixed(2)
+ "%");
}
##Check fragmentation rate of all collections in the specified databases
use xxxdb
db.getCollectionNames().forEach((c) => {print("\n\n" + c);
getCollectionDiskSpaceFragRatio(db.getName(), c);});
```

Solution

High Disk Utilization and High Fragmentation Rate

- TencentDB for MongoDB version 4.4 and above

  If disk utilization is high (generally above 80%~85%) and fragmentation rate is high (generally above 25%, making reclamation beneficial), and TencentDB for MongoDB is version 4.4 and above with a replica set architecture, use the command `db.runCommand({compact:"collectionName"})` to compress the specified collection document to free up disk space. Here, `collectionName` is the name of the collection, please replace it according to the actual situation.

  > **Note:**
  > When MongoDB performs the Compact operation, it compresses the entire database. During this period, operations such as the creation and deletion of collections and indexes will be blocked, while other operations (such as queries) will not be affected, but there will be a load impact, and requests may experience latency. It is recommended to perform this operation during the business off-peak period.

- TencentDB for MongoDB version before 4.4

  If disk utilization is high (generally above 80%~85%) and fragmentation rate is high (generally above 25%, making reclamation beneficial), and TencentDB for MongoDB is version below 4.4, it is not recommended to perform `compact` as it will block all requests to the instance and may cause Compact index invalidation. The solution is as follows:
  - Upgrade the version. It is recommended to upgrade to the latest version for better performance and stability. For specific operations, please refer to Version Upgrade.
  - If you do not want to upgrade the version, you can achieve the effect of shrinking space by rebuilding the node through logical migration. There will be multiple interruptions during operation. For specific operations, please contact after-sales service or Submit a Ticket.

High Disk Utilization but Low Fragmentation Rate

If disk utilization is high (generally above 80%~85%) and fragmentation rate is not high, with fragmentation rate below 20%, it is not recommended to perform `compact` operation because MongoDB will reuse this part of the space. In this case, please expand the disk space. For specific operations, please refer to Adjust the mongod node specification.