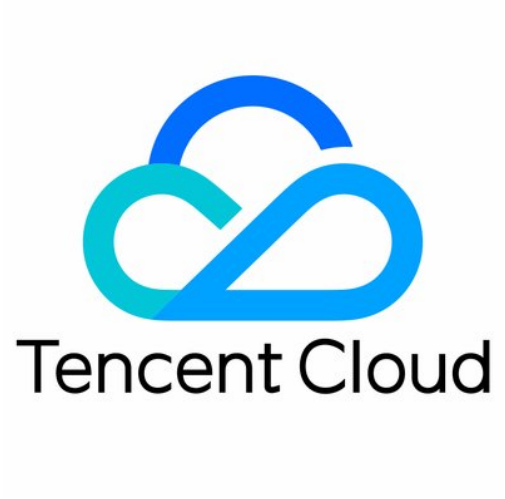
TencentDB for MongoDB

OPS and Development Guide

Product Documentation



Contents

OPS and Development Guide Development Specifications

Last updated : 2021-06-28 10:52:04

Database Design Specifications

Prohibited

Do not create too many tables in a database
In the WiredTiger engine, multiple files need to be created in each collection to store metadata, data, and indexes. Too many small files on the disk will compromise the performance. We recommend you keep the number of tables in one database below 100 and the number of tables in the entire database instance below 2,000.

Recommended

- We recommend you make database names begin with "db". A database name cannot contain more than 64 characters and special symbols other than "_". All letters in the name should be in lowercase.
- We recommend you make collection names begin with "t_". A collection name cannot contain more than 120 characters and special symbols other than "_".

Index Design Specifications

Prohibited

- **Do not create an index without the `background` parameter for a database in the production environment**
  In MongoDB 4.2 or below, the `createIndex()` command is in `foreground` mode by default, where index creation will block all operations on the database and cause business interruption. Therefore, if you need to run `createIndex()` for a business in the production environment, be sure to add the `background` parameter.

  Note :
  `background` must be placed in the `options` parameter of the `createIndex()` command, for example, `db.test.createIndex({a: 1}, {unique:true, background: true})`. Do not separate

different parameters as follows: `db.test.createIndex({a: 1}, {unique:true}, {background: true})`.

- **Place the sort field in the index to avoid out-of-memory (OOM) events caused by a large amount of sorting work by the business in the memory**
  - In MongoDB 4.2 or below, one SQL statement is allowed to use a maximum of 32 MB memory for sorting. If the limit is exceeded, the system will prompt that `Sort operation used more than the maximum 33554432 bytes of RAM`. In this case, you can run `db.runCommand({ getParameter : 1, "internalQueryExecMaxBlockingSortBytes" : 1 } )` to adjust the sort memory size.
    However, it is prohibited to adjust the memory size for a business in the production environment, as this will increase the possibility of database OOM. We recommend you add the sort field to the index so as to implement the sorting feature through the index.
  - In MongoDB 4.4, although the disk sort option is provided to avoid large memory consumption by sorting, we still recommend you use index for sorting.

- **Do not create too many indexes in a table**
  Every time a data entry is inserted into MongoDB, the index will be written. The more the indexes, the higher the overheads incurred by data writes. Therefore, abuse of indexes, such as creating an index for each field even when some fields will not be used for query, is forbidden. We recommend you keep the number of indexes in a table below 10.

Recommended

- **Drop useless indexes periodically**
  During a write operation, indexes will cause extra resource consumption. Therefore, the indexes should be as few as possible.
  For versions above MongoDB 4.4, we recommend you use `hidden index` to hide useless indexes first, confirm that the business is normal, and then drop them.

- **Drop single-column indexes if they are contained in compound indexes according to the left-prefix rule**
  Extra indexes will cause performance waste during write operations.

- **Restrain from using operators such as `$ne/$nin`**
  Like other databases (such as MySQL), `NOT EQUAL` and `NOT IN` operators cannot effectively use indexes, which thus should be avoided as much as possible.

- **Create indexes on fields that are highly distinctive**
  If an index field has low distinction, a large number of rows will be scanned during a query, leading to a low query efficiency, which will have a high impact on the database load. Therefore, the field on which an index is created should have high distinction.

Database Operation Specifications

Prohibited

- **Do not use SQL statements whose execution plan has not been confirmed through `explain()` in the production environment**
  Before using an SQL statement in the production environment, you need to run `explain()` to confirm whether its execution plan meets the expectation; otherwise, a failure may occur.

- **Do not disable authentication in the production environment, especially when the database is accessible over the public network**
  If authentication is disabled, the database will be exposed to everyone, especially when its server is accessible over the public network. We recommend you enable authentication in the production environment. If you have to disable it, be sure to set a firewall rule or IP allowlist.

- **Do not store business data in admin and local databases**
  If the admin database is read/written, a database lock will be added, which will compromise the performance. Data in the local database will be stored locally only but not replicated to the secondary nodes, and if primary-secondary switch occurs, data will be lost. Therefore, do not use the admin and local databases.

- **Do not create a database with the same name as the one dropped through the `db.dropDatabase()` command**
  - In MongoDB 4.0 or below, the official document stipulates that if a database is dropped and then a database with the same name is created, all mongos nodes should be restarted or run the `flushRouterConfig` command before the business reads/writes data.
  - In MongoDB 4.2 or above, all mongos and mongod nodes should be restarted or run the `flushRouterConfig` command.
    Therefore, do not directly run the `db.dropDatabase()` command in the business code and then create a database with the same name. When performing this operation, you must strictly follow the instructions in the official document.

- **Do not abuse `IN` and `OR` operators in high-concurrency and high-performance scenarios**
  `IN` and `OR` conditional clauses need to be converted to multiple queries at the database's underlying layer. Too many `IN` and `OR` operators in high-concurrency and high-performance scenarios will severely increase the request response latency and database load.

- **Do not execute complicated computing operations in the database in high-concurrency and high-performance scenarios**
  MongoDB provides powerful computing capabilities (such as MapReduce), and these features are very developer-friendly and greatly simplify the business logic. However, these operations inevitably require resources. If complicated operations are performed at the database layer in high-concurrency scenarios, the database will experience great pressure, and if it fails, the entire system will also fail.
  We recommend you keep database operations simple, perform complicated computation on servers, and add a cache as appropriate on the database frontend in high-concurrency and high-performance scenarios.

- **Do not directly remove data in batches for businesses in the production environment**
  When the `remove` command is executed in the database, `¥_id` values of the entries that meet the removal conditions will be queried first, and the entries will be removed by `¥_id` one by one and recorded in the `oplog` (an `oplog` entry will be generated for each removed data entry).
  If there are a large number of data entries that meet the removal conditions, the database will experience high pressure, and the primary-secondary delay will surge suddenly.
  For businesses in the production environment, we recommend you drop the collection directly, use a script to remove the entries one by one and control the removal speed, or use TTL indexes as much as possible.

- **Do not customize the `¥_id` field in businesses**
  `¥_id` is the default primary key of MongoDB, which is an auto-incrementing sequence by default. If you customize `¥_id`, but your business cannot ensure that `¥_id` is auto-incrementing, every time data is inserted, the `¥_id` index will inevitably need to adjust the B-tree index, which will bring additional load to the database.

- **Do not configure only one single IP in the connection string in scenarios where a replica set is directly connected to a mongod node. Do not connect a sharded cluster to only one single mongos address (unless mongos is deployed together with the application server)**
  For businesses in the production environment, if only the primary node of a replica set is

connected, once HA occurs in the database, write interruption will occur. If only one single mongos node is connected, the business will be interrupted if the node fails.

- **Do not set write concern to `j:false` for businesses in the production environment**
  Write concern is set to `j:true` by default, which indicates that the result will be returned to the client after the journal log is written by the server. Generally, do not set `j:false` ; otherwise, data may be lost if the process is restarted after a sudden failure.

- **Do not use UPDATE statements without conditions**
  We recommend you keep the default value (`false`) of `multi` to avoid updating the data of the full table due to program bugs (for example, some exceptions cause the `query` parameter to pass in `{}`).

- **Do not read an entire array, update it, and write it back when you only need to update certain elements in it**
  We recommend you use `arrayFileters` to modify elements on an as-needed basis.

Recommended

- **Use local read/write instead of global read/write**
  Use the `$projection` operator as much as possible in query statements to project the required fields. If you want to modify a certain field in the `update` command, we recommend you use `$set` . Do not read the entire document and then write it back entirely after modifying it.

- **Restrain from using the `db.collection.renameCollection()` command in the production environment**
  `renameCollection()` will block all database operations in MongoDB 4.0 and below and block operations on the current and target tables in MongoDB 4.2 and above. Moreover, during execution of `renameCollection()` , problems such as cursor failure, `changeStream` failure, and failure of `mongodump` with the `--oplog` command may occur. Do not directly perform this operation during peak hours in the production environment.

- **Configure the `{w: "majority"}` parameter for write concern for core businesses**
  By default, the configuration of write concern of general drivers is `{w:1}` , that is, a request will be considered successful after its write to the primary node is completed. If the server fails suddenly and the written data has not been replicated to the secondary nodes, this configuration will cause data loss.
  Therefore, for core businesses in the production environment, we recommend you configure `{w:`

`"majority"}` , which specifies that the result will be returned to the client after the data is synced to most nodes. However, the reliability and performance cannot be balanced. If the `{w: "majority"}` configuration is used, the request delay will increase accordingly.

Not recommended

- **Do not use a large number of multi-document transactions in high-performance scenarios unless necessary**
  In MongoDB 4.0 or above, the multi-document transaction feature is provided. However, it is only a supplement to the MongoDB database capabilities. In high-concurrency and high-performance scenarios, comprehensive stress testing must be performed before many multi-document transactions are used.
  Generally, before a multi-document transaction is committed, it will keep a snapshot in the memory, which may consume a large amount of cache and compromise the performance.

- **Restrain from using non-persistent connections**
  The authentication logic of MongoDB is a very complicated computing process, and MongoDB creates a thread for each connection by default. A high number of non-persistent connections will bring high pressure to the database, especially for replica set cluster without mongos. We recommend you use persistent connections. For more information, please see the `ConnectionPool` parameter in the MongoDB URL.

Sharded Cluster Design Specifications

Prohibited

- **Do not use ranged sharding if the `¥_id` field is used as the shard key**
  `id` is an ascending sequence that increases as the data volume increases. If `¥_id` is used as the shard key and ranged sharding is used, the cluster will be constantly balanced when data is inserted.

- **Do not directly connect a sharded cluster to the mongod node to write data**
  A sharded cluster should write data through mongos. Data directly written through mongod has no route information and cannot be accessed.

- **Do not keep the balancer and `autoSplit` configuration disabled for a prolonged time in the production environment**

If the balancer is disabled, the data volume will become uneven across different shards. If `autoSplit` is disabled, jumbo chunks may be generated.

- **Restrain from using queries without a shard key in a sharded table**
  If you query a sharded table without using a shard key, all shards will be scanned, and the results will be aggregated on mongos, which has high performance overheads and is therefore not recommended.

- **Be sure to set a balancer window in the production environment to avoid the impact of balancing on the businesses**
  Balancing will cause high pressure on the database. We recommend you balance the database during off-peak hours.

Recommended

- **Use a field that is highly distinctive as the shard key. Ideally, use the unique primary key as the shard key**
  Suppose you have a collection for storing population information that uses gender as the shard key. This shard key is considered as poorly distinctive, as data entries with the same gender field value will account for half of all data in the collection theoretically.
  If the shard key has low distinction, a large number of data entries may concentrate on certain shards, and such imbalance cannot be solved by adding more shards. Therefore, we recommend you use a highly distinctive field as the shard key.

- **Perform presharding if you want to use hashed sharding, especially for large tables where large amounts of data are frequently inserted**
  Only two chunks are created for each shard by the `shardCollection()` command by default. As the data volume grows, MongoDB needs to constantly balance and run `splitChunk`, which will bring high pressure to the database.
  Therefore, we recommend you perform presharding (by running the `shardCollection` command and specifying the `numInitialChunks` parameter), especially before batch data import into large collections. Each shard supports up to 8,192 chunks.

Not recommended

- **Restrain from using ranged sharding if you have no strong requirements for scanning in the shard key order. Use hashed sharding instead**
  Ranged sharding tends to cause imbalance and hotspot data. In addition, as presharding is not

supported, balancing is inevitable as more data is written. Therefore, we recommend you not use ranged sharding, unless you have special requirements for queries based on the shard key range.

> Note :
> During the `shardCollection` operation, 1 in `sh.shardCollection("records.people", { zipcode: 1 } )` indicates ranged sharding, and `hashed` in `sh.shardCollection("records.people", { zipcode: "hashed" } )` indicates hashed sharding. Be careful not to use them incorrectly.

- **Restrain from using non-sharded tables in a sharded cluster**
  If the `shardCollection` command is not executed on a MongoDB sharded cluster, the data will be stored only in the primary shard by default. A large number of non-sharded tables will cause different data volumes in different shards. If the cluster runs for a long time, some shards may have a high data volume, or the disk may become full. In this case, you have to use `movePrimary` to manually migrate the data, which makes OPS more complicated.

Command Support in Sharding Cluster v3.2

Last updated : 2021-06-07 16:33:46

Sharding Strategy

- Ranged-based sharding is supported.
- The shardkey is indexed compound fields.
- Sharding is required for all data sets in a shard instance. It is recommended to store non-sharded data in a separate replica set instance.

Authentication Mechanism

MongoDB is fully compatible with SCRAM-SHA-1 and MONGODB-CR.

Supported Sharded Cluster Commands

Category	Command	Subcommand	Supported
Basic CRUD commands	find	filter	Yes
		sort	Yes
		projection	Yes
		hint	Yes
		skip	Yes
		limit	Yes
		batchSize	Yes
		singleBatch	Yes
		comment	Yes
		maxScan	Yes

		maxTimeMS	No
		readConcern	Yes
		max	Yes
		min	Yes
		returnKey	Yes
		showRecordId	Yes
		snapshot	No
		tailable	No
		oplogReplay	No
		noCursorTimeout	Yes
		awaitData	No
		allowPartialResults	No
	insert	The `shardkey` field is required and must be the same for batch INSERT operations	Yes
	update	The updated field cannot be `shardkey`	Yes
	delete	-	Yes
	findandmodify	-	Yes
	count	-	Yes
	distinct	The `shardkey` field is required	Yes
	aggregate	-	Yes
	group	-	No
	mapReduce	-	No
	getmore	-	Yes
	getLastError	-	No
	getPrevError	-	No

	resetError	-	No
	eval	-	No
	geoNear	-	No
	geoSearch	-	No
	parallelCollectionScan	-	No
Diagnostic commands	collStats	-	Yes
	dbstats	-	Yes
	explain	-	Yes
	listDatabases	-	Yes
	serverStatus	-	No
	top	-	No
Sharding commands	enableSharding	-	Yes
	shardCollection	-	Yes
Management commands	listCollections	-	Yes
	dropDatabase	-	Yes
	drop	-	Yes
	creareIndexes	-	Yes
	listIndexes	-	Yes
	dropIndexes	-	Yes
	logout	-	Yes
	renameCollection	-	No
	copydb	-	No
	create	-	No
	clone	-	No
	cloneCollection	-	No

	cloneCollectionAsCapped	-	No
	convetToCapped	-	No
	filemd5	-	No
	fsync	-	No
	clean	-	No
	connPoolSync	-	No
	connectionStatus	-	No
	compact	-	No
	collMod	-	No
	reIndex	-	No
	setParameter	-	No
	getParameter	-	No
	repairDatabase	-	No
	repairCursor	-	No
	touch	-	No
	shutdown	-	No
	logrotate	-	No
	killop	-	No
User management commands	-	-	No
Role management commands	-	-	No
Replica set commands	-	-	No

Command Support in v3.6

Last updated : 2021-06-07 16:33:46

For a list of MongoDB's commands, please see MongoDB’s command documentation.

TencentDB for MongoDB v3.6 does not support the following commands:

Category	Unsupported Command
Sharding Commands	addShard
	removeShard
Query and Write Operation Commands	getPrevError
Role Management Commands	dropAllRolesFromDatabase
Replication Commands	replSetAbortPrimaryCatchUp
	replSetFreeze
	replSetGetConfig
	replSetGetStatus
	replSetInitiate
	replSetMaintenance
	replSetReconfig
	replSetResizeOplog
	replSetStepDown
	replSetSyncFrom
	resync
Administration Commands	cloneCollection
	cloneCollection
	cloneCollectionAsCapped
	compact
	connPoolSync

	logRotate
	setParameter
	shutdown
Diagnostic Commands	availableQueryOptions
	dbHash
	getCmdLineOpts
	getLog
	shardConnPoolStats
System Events Auditing Commands	logApplicationMessage

Development OPS
Database Problems in MongoDB 3.6

Last updated : 2020-08-24 17:30:42

Problem Description

If you drop a database and then create a new database with the same name in a TencentDB for MongoDB 3.6 instance repeatedly, the error "database does not exist" may occur when you read from, write to, or drop this database as shown below:

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

Solutions

This is a common problem, which may be caused by mongos not refreshing its metadata cache as shown below. For more information, please see db.dropDatabase().

> **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.

Select one of the following two solutions for troubleshooting:

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

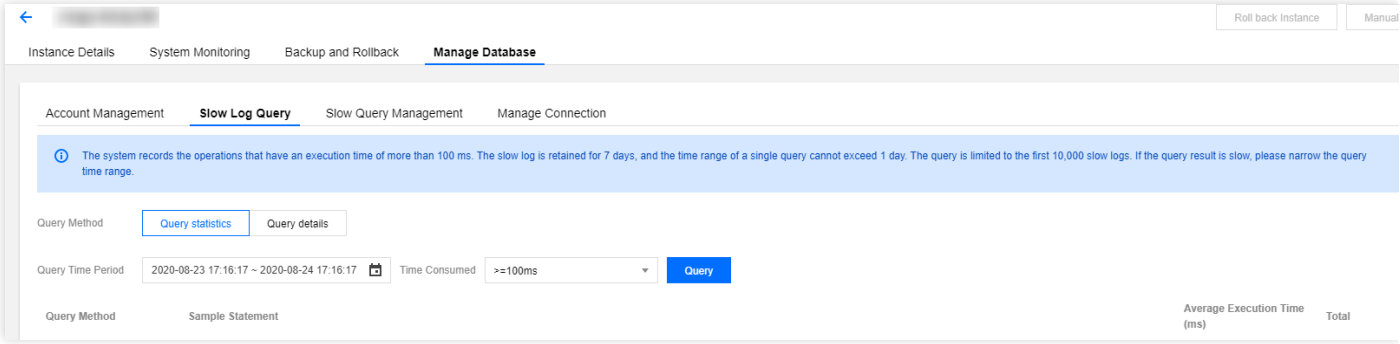
High CPU Utilization

Last updated : 2020-08-24 17:30:42

If you find the CPU utilization is high when using MongoDB, you can troubleshoot the problem as follows:

1. Check whether your database operation is too frequent.
   Log in to the TencentDB for MongoDB Console and view the QPS on the system monitoring page. If the QPS is high, you may evaluate whether the instance needs to be upgraded. If the QPS is not high, check whether there are any slow queries.

2. Check whether there are slow logs on mongod.
   Log in to the TencentDB for MongoDB Console and view the slow logs of the instance by using "Query statistics".



   Pay attention to keywords such as `command`, `COLLSCAN`, `IXSCAN`, `keysExamined`, and `docsExamined`. For more log descriptions, please see Log Messages.

   - `command` indicates an operation recorded in a slow log.
   - `COLLSCAN` indicates that a full-table scan is performed. `IXSCAN` indicates that an index scan is performed. For descriptions of other fields, please see Explain Results.
   - `keysExamined` refers to the number of index entries scanned. `docsExamined` refers to the number of documents scanned. Larger `keysExamined` and `docsExamined` values indicate that no index is created or the created index is less distinctive. Please check the fields for which an index is created.

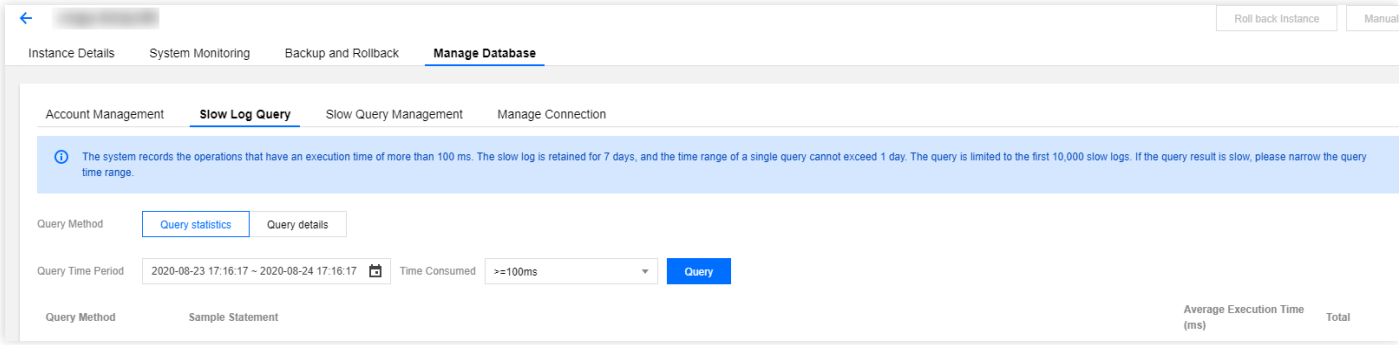
Slow Query Problems

Last updated : 2020-08-24 17:30:43

If you find the request latency becomes obviously longer when using TencentDB for MongoDB, you can troubleshoot the problem in the following steps:

1. Check whether the "Latency" monitoring metric for instances is exceptional.
   Log in to the TencentDB for MongoDB Console and check the instance monitoring data on the system monitoring page. The "Latency" metric mainly reflects the time from a request arriving at the access layer to it returning to the client after being processed. If the request latency is high, check whether there are slow logs on mongod.

2. Check whether there are slow logs on mongod.
   Log in to the TencentDB for MongoDB Console and view the slow logs of the instance by using "Query statistics".



   Pay attention to keywords such as `command`, `COLLSCAN`, `IXSCAN`, `keysExamined`, and `docsExamined`. For more log descriptions, please see Log Messages.
   The keywords that require attention include the following:

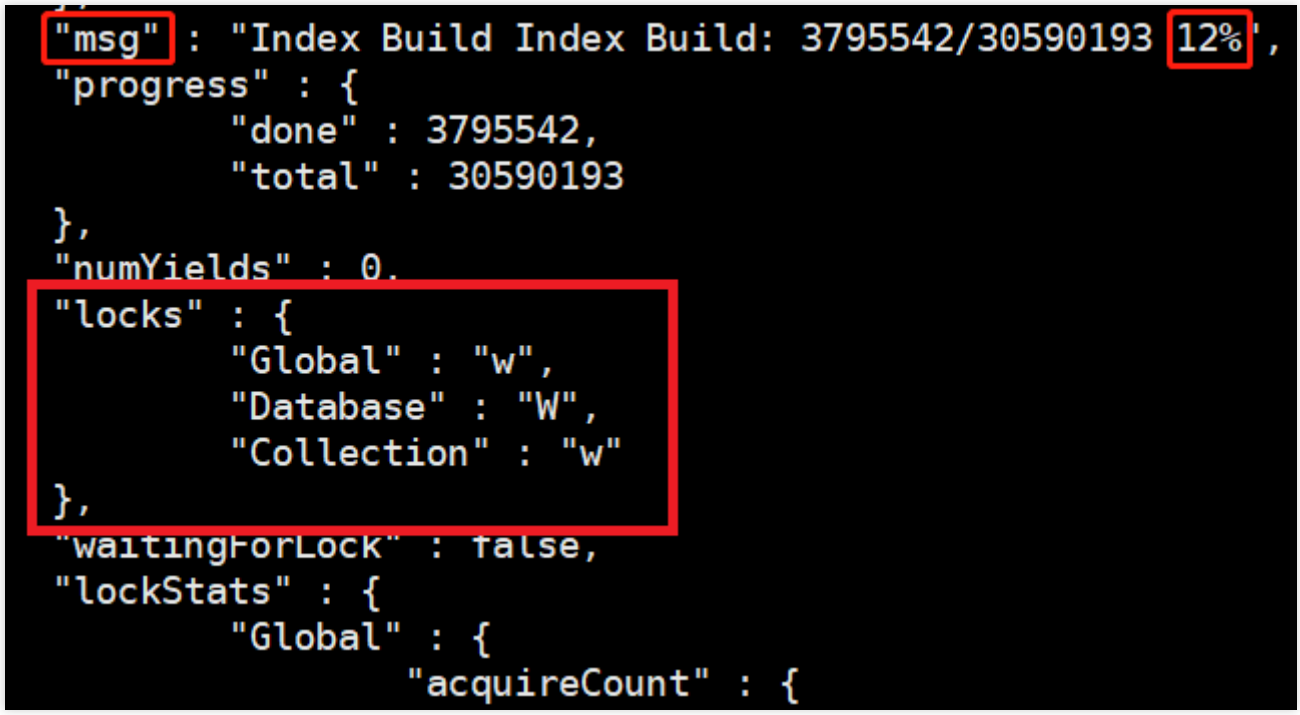
   - `command` indicates an operation recorded in a slow log.
   - `COLLSCAN` indicates that a full-table scan is performed. `IXSCAN` indicates that an index scan is performed. For descriptions of other fields, please see Explain Results.
   - `keysExamined` refers to the number of index entries scanned. `docsExamined` refers to the number of documents scanned. Larger `keysExamined` and `docsExamined` values indicate that no index is created or the created index is less distinctive. Please check the fields for which an index is created.

3. Check whether the request is locked due to an index created in the foreground.
   If there is no problem with the index used for business queries, check whether an index is created

in the foreground during peak business hours. An index is created in the foreground by default for a collection (the `background` option is `false`), which will block all other operations until the index is created in the foreground. If you choose to create an index in the background, MongoDB can still provide read/write services during the creation of the index. However, it takes more time to create the index in this way. For options to create an index, please see db.collection.createIndex().
You can view the progress of index creation by running the `currentOp` command as shown below:

```
db.currentOp(
{
$or: [
{ op: "command", "query.createIndexes": { $exists: true } },
{ op: "insert", ns: /¥.system¥.indexes¥b/ }
]
}
)
```

The returned result is shown as follows. The `msg` field indicates the progress of index creation. The `locks` field indicates the lock type of the operation. For more information on locks, please see Database Profiler Output.



Lock Mode	Description
R	Represents Shared (S) lock.
W	Represents Exclusive (X) lock.
r	Represents Intent Shared (IS) lock.
w	Represents Intent Exclusive (IX) lock.

4. Check whether the mongos load is too high.
   The "Latency" metric mainly reflects the time from a request arriving at the access layer to it returning to the client after being processed. If there are no slow logs on mongod, but the request latency is high, it may result from high mongos load. There are many reasons 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 in the console.

> ⚠ **Note :**

All instance connections will be interrupted at the moment of restarting mongos, but the business can be directly reconnected. Therefore, restarting mongos will not continuously affect the business.

Connection Problems

Last updated : 2020-08-24 17:27:14

You may encounter two types of connection problems when using TencentDB for MongoDB. You can troubleshoot them as follows.

High Connection Utilization

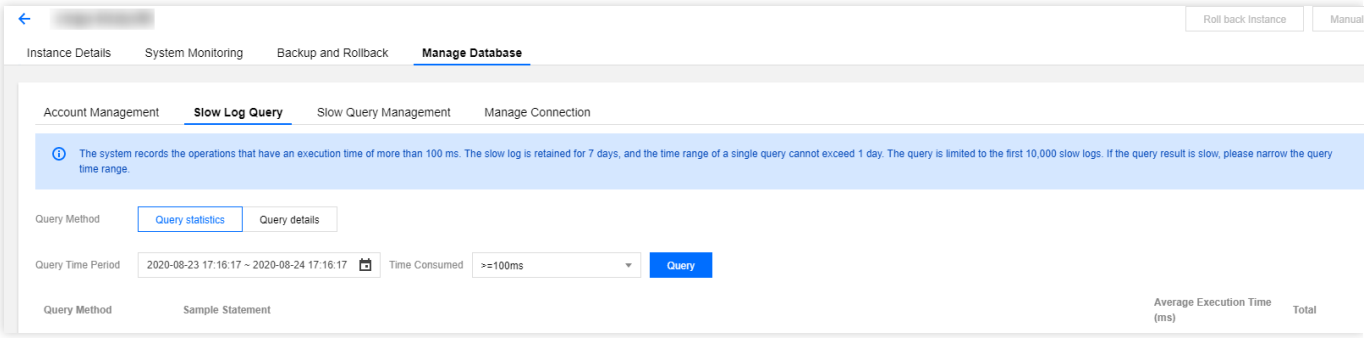
If you find that the connection utilization of the instance is too high, you can refer to the following procedure for troubleshooting.

1. Check whether a connection pool is used in the business.
   The MongoDB service is provided in a mode where each network connection is processed by a single thread (one-thread-per-connection). Too many network connections generate too many threads, which will increase context switch and memory overheads. Establishing connections and performing authentication for each request greatly affect performance. Therefore, you are recommended to use a connection pool for your business to limit the number of connections of the instance and release the connections no longer in use. All programming language versions of MongoDB drivers generally encapsulate an object, so you need to construct a global object and use it in subsequent requests to send requests to the instance when using MongoDB. You can use the default connection pool size for the object in the Driver or configure the size by specifying the `maxPoolSize` option when constructing an object.

2. If you have already set a connection pool, check your business for any unexpected exceptions.
   - Check whether there are any business release changes or code logic defects that lead to a large number of connections.
   - Check whether there are any connection leaks by checking whether the number of connections on relevant CVM instances is exceptional.
   - Check whether the business has a large number of sudden real requests.

3. Check whether there are any slow queries that cause connections to be occupied.
   - If your business is confirmed to be normal, check whether there are any index exceptions; for example, a previously established index is deleted by mistake.
   - If the index is normal, check whether there are a large number of slow queries, which occupy the connections and lead to establishment of more connections.
     Log in to the TencentDB for MongoDB Console and view the slow logs of the instance by using

"Query statistics".



Pay attention to keywords such as `command`, `COLLSCAN`, `IXSCAN`, `keysExamined`, and `docsExamined`. For more log descriptions, please see Log Messages.

The keywords that require attention include the following:

i. `command` indicates an operation recorded in a slow log.

ii. `COLLSCAN` indicates that a full-table scan is performed. `IXSCAN` indicates that an index scan is performed. For descriptions of other fields, please see Explain Results.

iii. `keysExamined` refers to the number of index entries scanned. `docsExamined` refers to the number of documents scanned. Larger `keysExamined` and `docsExamined` values indicate that no index is created or the created index is less distinctive. Please check the fields for which an index is created.

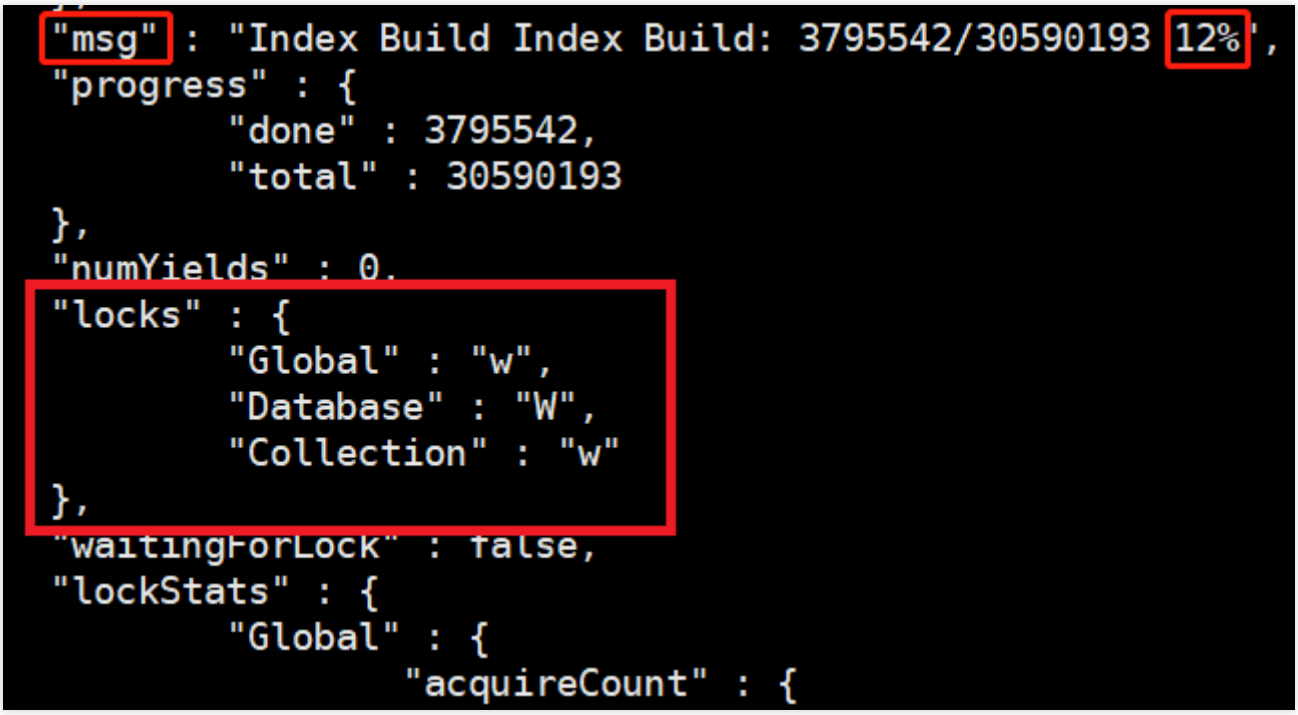
4. Check whether the request is locked due to an index created in the foreground.

If there is no problem with the index used for business queries, check whether an index is created in the foreground during peak business hours. An index is created in the foreground by default for a collection (the `background` option is `false`), which will block all other operations until the index is created in the foreground. If you choose to create an index in the background, MongoDB can still provide read/write services during the creation of the index. However, it takes more time to create the index in this way. For options to create an index, please see db.collection.createIndex().

You can view the progress of index creation by running the `currentOp` command as shown below:

```
db.currentOp(
{
$or: [
{ op: "command", "query.createIndexes": { $exists: true } },
{ op: "insert", ns: /¥.system¥.indexes¥b/ }
]
}
)
```

The returned result is shown as follows. The `msg` field indicates the progress of index creation. The `locks` field indicates the lock type of the operation. For more information on locks, please see Database Profiler Output.



Lock Mode	Description
R	Represents Shared (S) lock.
W	Represents Exclusive (X) lock.
r	Represents Intent Shared (IS) lock.
w	Represents Intent Exclusive (IX) lock.

5. Evaluate whether the instance configuration meets your business requirements.
   If the business connection utilization displayed in the console is still high after a connection pool is set and a highly distinctive index is created, it may indicate that the instance configuration cannot meet the actual business needs. In such case, you need to evaluate the instance specification

actually needed based on your business model, peak traffic, QPS, and TPS and then upgrade your instance accordingly.

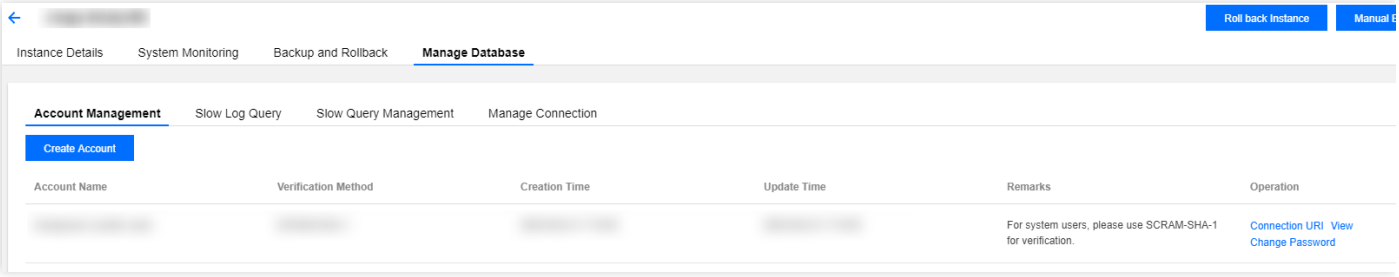
Connection Rejection

If a connection is rejected, you can troubleshoot the problem as follows.

1. Check whether the connection utilization of the instance is 100%.
   Log in to the TencentDB for MongoDB Console and view the monitoring metrics "Connection Count" and "Connection Utilization" on the system monitoring page.
   If the connection utilization of the instance is 100%, check whether your business is exceptional. If necessary, you can quickly release the connections by restarting mongos in the console.

   > **⚠ Note :**
   >
   > All instance connections will be interrupted at the moment of restarting mongos, but the business can be directly reconnected. Therefore, restarting mongos will not continuously affect the business. If the number of business connections increases rapidly and the connection utilization reaches 100% again after the restart, it indicates that the business does have a large number of valid connections and there are no connection leaks. In such case, you need to find why there is such a large number of connections in your business by referring to the troubleshooting procedure for high connection utilization.

2. Check whether the username and password are correct.
   Make sure that the username and password are correct. If they are incorrect, log in to the console and modify them.



3. Check whether the mongo shell version is correct.
   To ensure successful authentication, install mongo shell 3.0 or above. For detailed directions, please see Install MongoDB.

4. Check whether the authentication database is correct.

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