TencentDB for DBbrain

Best Practices

Product Documentation

Contents

Best Practices

Fixing High CPU Utilization on MySQL Instance

Last updated：2022-07-31 17:16:07

Problem Description

Generally, high CPU utilization of a TencentDB for MySQL instance will cause system exceptions, such as slow response, failure to get connections, and timeout. Performance drop is usually caused by a high number of retries upon timeout, and high CPU utilization is usually caused by exceptional SQL statements. High numbers of lock conflicts, lock waits, or uncommitted transactions may also cause high CPU utilization of TencentDB for MySQL instances.

When a database executes business query or statement modification tasks, the CPU will request data blocks (8 KB by default) from the memory.

If the memory has the target data, the CPU will execute the computation task and return the result to you, which may involve actions requiring high CPU usage such as sorting.

If the memory does not have the target data, the database will get the data from the disk.

The two data acquisition processes above are called logical read and physical read, respectively. Therefore, poorly performing SQL statements can easily cause the database to generate a lot of logical reads during the execution, resulting in high CPU utilization. They may also make the database generate a lot of physical reads, resulting in high IOPS and I/O latency.

Solutions

DBbrain provides three key features you can use to troubleshoot and optimize exceptional SQL statements that cause high CPU utilization.

Exception diagnosis: detects and diagnoses exceptions 24/7 and provides optimization suggestions in real time.

Slow SQL analysis: analyzes slow SQL statements of the current instance and provides corresponding optimization suggestions.

Audit log analysis: performs in-depth analysis on SQL statements and provides optimization suggestions based on TencentDB audit data (full SQL).

Method 1 (recommended). Use the exception diagnosis feature to troubleshoot database exceptions

The exception diagnosis feature can proactively locate and perform optimization for failures, and does not require OPS experience. It can diagnose not only exceptions of high CPU utilization but also all common exceptions and failures of TencentDB for MySQL instances.

Directions:

1. Log in to the DBbrain Console, select **Diagnosis and Optimization** on the left sidebar, and select **Exception Diagnosis** at the top.
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" or "Historical" tab or the target diagnosis item in "Diagnosis Prompt" to access the diagnosis details page.

Event Overview: includes the diagnosis items, start and end times, risk level, duration, and overview.

Symptom: 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.

Expert Suggestion: provides optimization suggestions, including but not limited to SQL optimization (index and rewrite), resource configuration optimization, and parameter fine-tuning.

5. Select the **Expert Advice** tab to view the optimization suggestion provided by DBbrain for this failure. In this example, the optimization suggestion is provided for the SQL statement, which lacks the corresponding index during execution. In this case, full-table scan needs to be performed and a single execution is costly. Therefore, the CPU utilization tends to be high or even 100% in high-concurrence scenarios.

Method 2. Use the "slow SQL analysis" feature to troubleshoot SQL statements that lead to high CPU utilization

1. Log in to the DBbrain Console, select **Diagnosis and Optimization** on the left sidebar, and select **Slow SQL Analysis** at the top.
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 the time period to be queried. If slow SQL statements exist in this period, the time points of occurrence and the number of statements will be displayed in a bar chart in "SQL Statistics".

Click the bar chart, and the information of all corresponding slow SQL statements (those aggregated by template) will be displayed in the list below, and the duration distribution of SQL statements in the specified time period will be displayed on the right.

4. You can identify and filter SQL statement execution data in the SQL statement list through the following method:

1. Sort the SQL statements by average duration (or maximum duration). Examine the top SQL statements in terms of duration. We do not recommend you sort the statements by total duration, as the data may be affected by a high number of executions.
2. Then, examine 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.

3. Click an SQL statement to view its details, resource consumption, and optimization suggestions.

On the analysis page, you can view the complete SQL template, SQL statement samples, optimization suggestions, and description. You can optimize your SQL statements based on the expert suggestions provided by DBbrain to improve SQL performance and reduce SQL execution duration.

On the statistics page, you can perform cross-sectional analysis of the root cause of a slow SQL statement based on the percentages of total lock wait time, total scanned rows, and total returned rows in the statistics report, and then optimize the statement accordingly.

On the duration distribution page, you can view the execution duration distribution intervals of the specified type of aggregated SQL statements and the access percentage of source IPs.

Method 3. Use the "audit log analysis" feature to troubleshoot SQL statements that cause high CPU utilization

Prerequisites: the instance needs to have the database audit feature enabled.

1. Log in to the DBbrain Console, select **Diagnosis and Optimization** on the left sidebar, and select **Audit Log Analysis** at the top.

2. Select (enter or search for) an instance ID in the top-left corner to switch to the target instance.

3. You can select to display the insight view by QPS or the number of slow queries. Click **Create Audit Task** in the top-right corner of the view. and select the task start time and duration. Then, click **Confirm**.

After the task is created, it will be displayed in the task list. After the task is completed, find the target record and click **View SQL Analysis** to access the SQL analysis details page.

4. On the SQL analysis page, you can display the view by SQL Type, Host, User, or SQL Code. You can specify a time period to expand the view and view data at specific time points.

The aggregated details and execution information of SQL statements in the specified time period are displayed in the table below. If you select a time period and stretch it in the table, the SQL data will change accordingly, and only the SQL analysis result of the selected time period will be displayed.

5. The aggregated SQL statement execution information (including the number of executions, total, maximum, and minimum delay, and total, maximum, and minimum numbers of affected rows) is displayed in the table in the bottom of the SQL analysis details page. You can sort SQL statements by multiple metrics to identify the ones that require optimization.

Example:

Sort the SQL statements by the number of executions to identify the ones whose number of executions is high or has an exception change. Then, analyze the rationality of the number of executions and optimize the statements based on DBbrain's suggestions.

By viewing the number of executions, total delay, and maximum delay, you can find that the average execution delay of the first and second SQL statements with the highest numbers of executions is very short, which indicates that the performance of the two SQL statements is normal.

You can see that the single execution time of the third statement is about 100 seconds. Since its number of executions is relatively low, its total delay ranks third. However, this SQL type must be optimized. You can click the SQL to view information such as expert suggestions, the resource consumption analysis curve, and the source IP analysis provided by DBbrain.

There is another type of SQL statement that requires your attention. You can see that the difference between the maximum and minimum delays of the fourth statement is large (more than 200 seconds). This indicates that the entire system is fluctuating, and you need to analyze whether the fluctuation is caused by network problems or changes in the execution plan due to data volume change.

If the number of executions and the average duration of SQL statements are relatively rational and the SQL statements with a large number of executions are optimal, it means that the performance has reached a bottleneck. We recommend you upgrade the instance specification configuration or use read/write separation to disperse SQL statements with a large number of executions, or use the preset cache database for optimization.

Fixing Lock Conflict on MySQL Instance

Last updated：2022-07-31 17:11:17

Problem Description

Lock conflict is a problem frequently faced by MySQL database businesses, and its symptoms vary by scenario. In some scenarios, it may directly result in business crash, while in other scenarios, it may be hard to perceive and cause problems such as incorrect and messy data.
The variety, complexity, and imperceptibility of lock conflict scenarios create issues that require a skilled DBA to resolve. Troubleshooting such issues is also time-consuming.
The exception diagnosis feature of DBbrain includes dozens of lock diagnoses such as deadlocks, row lock waits, table locks, read-only locks, DDL/SELECT/DML lock waits, and SQL MDL lock waits, helping you easily solve lock conflicts in an efficient and professional manner.
This document uses the common row lock wait and deadlock as examples to describe how you can use DBbrain to solve a lock conflict.

Solution

Scenario 1. Row lock wait

Step 1. View the row lock wait event

Method 1:
1.1 Log in to the DBbrain Console and select **Monitoring & Alarm** > **Exception Alarm** on the left sidebar.
1.2 Select the time period to be queried and select "Row lock wait" in the "Item" column for filtering.
1.3 The list will display the row lock wait events of the instance. Click **Details** in the "Operation" column to access the event details page.
Method 2:
1.1 Log in to the DBbrain Console, select **Diagnosis and Optimization** on the left sidebar, and select **Exception Diagnosis**.
1.2 Select the target instance above and select "Real-Time" (the data is updated dynamically by default) or "Historical" (you can customize the time period).
1.3 Check whether there are row lock wait events in the "Diagnosis Prompt" section. You can click an event to access its details page.

Step 2. Solve the row lock wait event

1. On the "Symptom Description" tab on the event details page, you can view the database statement running conditions when the row lock wait event occurs.
2. Switch to the "Intelligent Analysis" tab and view the cause of the row lock wait event.
In the row lock wait transaction, you can clearly view the status of blocked transaction statements. For example, the DELETE statement in the figure below is in LOCK WAIT state.
Based on the holdlock transaction result, quickly locate the current status and ID of the transaction with a row lock.
The performance monitoring curve displays the trends of the number of row lock waits.
3. Switch to the "Expert Suggestion" tab and solve the lock conflict problem as prompted. For example, you can run the `kill` command to kill the session `3965158` and release the lock.

Scenario 2. Deadlock

Note:
Since MySQL has deadlock monitoring and automatic transaction rollback capabilities, most deadlock scenarios can heal themselves and are imperceptible to the business. However, problems such as system crash or data inconsistency may occur due to fragile business logic or extreme conditions. Therefore, you should pay attention to deadlocks.

Step 1. View the deadlock event

Method 1:
1.1 Log in to the DBbrain Console and select **Monitoring & Alarm** > **Exception Alarm** on the left sidebar.
1.2 Select the time period to be queried and select "Deadlock" in the "Diagnosis Item" column for filtering.
1.3 The list will display the deadlock events of the instance. Click **Details** in the "Operation" column to access the event details page.
Method 2:
1.1 Log in to the DBbrain Console, select **Diagnosis and Optimization** on the left sidebar, and select **Exception Diagnosis**.
1.2 Select the target instance above and select "Real-Time" (the data is updated dynamically by default) or "Historical" (you can customize the time period).
1.3 Check whether there are deadlock events in the "Diagnosis Prompt" section. You can click an event to access its details page.

Step 2. Solve the deadlock event

In the "Symptom Description" tab on the event details page, you can analyze the following information and optimize the corresponding statements to solve the deadlock event:
Occurrence time and source IP for future traceability.
Two SQL statements with deadlocks. For example, the two INSERT INTO statements below.
Which statement is rolled back (Rollback) and which statement is properly executed (Normal) based on the `Status` field value.

(Optional) Specific lock hold and type based on `LockRequest` and `LockHold` .

Fixing High CPU Utilization in MongoDB Instance

Last updated：2022-08-13 20:18:46

Problem Description

In daily Ops, if the CPU utilization of a MongoDB database is too high, it will be easy to cause system exceptions; for example, reads/writes slow down, connections are used up, and more connection timeouts occur. A large number of access timeouts will also trigger repeated reconnection and authentication on the client, which may eventually lead to a database crash.
It is very common for MongoDB databases in the production environment to experience a high CPU utilization. This problem is generally caused by SQL exceptions, high traffic, in-memory sort operations, 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, the database will get the data from the disk.
The two data acquisition processes above are called logical read and physical read, respectively. Therefore, poorly performing SQL statements can easily cause the database to generate a lot of logical reads during the execution, resulting in a high CPU utilization. They may also make the database generate a lot of physical reads, resulting in a high IOPS and I/O latency.

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: It detects and diagnoses exceptions 24/7 and provides optimization suggestions in real time.
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 **Performance Optimization** on the left sidebar. On the displayed page, select the **Exception Diagnosis** tab.
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 select **Historical** and set the time range to be queried. If any failure occurred in this period, you can view its overview information in **Diagnosis Prompt** on the right.
4. Click **View Details** in the **Real-Time Diagnosis** or **Diagnosis Records** section or click an item in the **Diagnosis Prompt** section to enter 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.

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 **Performance 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 this page, select the time period to be queried. If slow SQL statements exist in this period, the time points of occurrence and the number of statements will be displayed in a bar chart in "SQL Statistics".
Click the bar chart, and the information of all corresponding slow SQL statements (those aggregated by template) will be displayed in the list below, and the consumed time distribution of SQL statements in the specified time period will be displayed on the right.
4. You can identify and filter SQL statement execution data in the SQL statement list in the following way:
1. Sort the SQL statements by average time consumed (or maximum time consumed). Check slow SQL statements. Do not sort the statements by total time consumed, as the data may be affected by a high number of executions.
2. Then, check the numbers of returned rows and scanned rows.
If there is a 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 currentOp information. DBbrain's real-time session feature enables you to view all the operations being executed in the database and kill specified time-consuming SQL statements. This releases resources such as CPU and disk I/O. In addition, the **Kill Sessions during a Period** option can continuously kill sessions based on specified conditions. If the database is blocked, you can use this option to fix the exception swiftly.

Kill session:

Click **Performance Optimization** > **Real-Time Session** > **Active Session**, select the session to be killed, and kill it.

Kill sessions during a period:

The **Kill Sessions during a Period** option can be configured in dimensions such as database, host, type, and time. There are two trigger mechanisms: **Scheduled stop** and **Manual stop**.

To stop killing sessions during a period, click **Stop** to close upcoming scheduled tasks or terminate manually triggered tasks.

Fixing Short Node Oplog Retention Period in MongoDB Instance

Last updated：2022-08-13 20:18:46

Problem Description

In MongoDB's replica set structure, a replica node will save the data logs synced from the primary node. This method is similar to the binlog source-replica replication method of MySQL. MongoDB uses the oplog to sync data from the primary node to the replica node. However, the configured oplog size is not infinite, and if it is exceeded, older oplog entries will be overwritten.
In daily Ops scenarios, when a replica database fails, the primary node still runs normally, and the failure will not affect the running status of the entire replica set. However, if the replica node is restarted, it will need to restore data from the primary node. At this time, if older oplog entries have been overwritten, the restored data will be incomplete since the overwritten data is lost, and the replica node cannot return to normal status. In other cases, database restarts and long primary-replica delays may also cause the problem of insufficient oplog entries.

Solution

DBbrain can help you observe the risks with oplog storage on all production nodes 24/7 in real time.

Using the exception diagnosis feature to troubleshoot database exceptions (recommended)

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 **Performance Optimization** on the left sidebar. On the displayed page, select the **Exception Diagnosis** tab.
2. In the overview section, a yellow mark indicates that a risk item is found at this time point.
3. In the **Diagnosis Details** list on the right, the risk item that the oplog storage period is too short is displayed at the same time.
4. Click the risk item to view more detailed exception information. The optimization suggestion will give reasonable measures based on the current situation.