CPU_Troubleshooting

Last updated by | Peter Hewitt | Nov 25, 2022 at 5:26 AM PST

Contents

CPU Troubleshooting

Issue

This article provides steps for analysing and troubleshooting issues related to high CPU utilization. It helps you identify the cause of sustained high CPU usage. Keep in mind that you can expect CPU usage to increase as a process or an application serves requests. However, if you consistently see CPU usage remain at a high level (80 percent or greater) for prolonged periods, the performance of the system or application will suffer. For that reason, it's important to understand the cause of sustained high CPU usage to be able to correct the problem.

Investigation/Analysis

Investigations to determine the cause and mitigation actions when there is a CPU issue should start by following the steps in the Workflow for High CPU troubleshooting article. Although this workflow is contained in the Managed Instance CSS Wiki, all of the steps can be applied to SQL Database.

This workflow describes the creation and analysis of the ASC report, including how to determine and confirm if the high CPU is a query compilation or query execution problem, the likely causes, recommended mitigation steps and how to investigate further.

Kusto: Real time resource stats

The Kusto table `MonDmRealTimeResourceStats` contains resource statistics for SQL database, including CPU usage, data and log IO usage, memory usage etc. The following Kusto query displays a graphical summary of the CPU

usage of the database for the specified period of time.

```
//MonRealTimeResourceStats - Summary of the CPU resource usage at the database level
MonDmRealTimeResourceStats
| where TIMESTAMP >= datetime(<start_datetime>)
| where TIMESTAMP <= datetime(<end_datetime>)
| where LogicalServerName =~ "<server_name>" and  database_name =~ "<database_name>"
| where replica_type == 0
| summarize avg(avg_cpu_percent) by bin(TIMESTAMP, 1min)
| render timechart
```

Deployment: Code regression

There is the possiblilty that the CPU issue is caused by a code regresion from a recent Azure SQL deployment. However, it must be stressed this isn't usually the case and the likely causes are those detailed in the high CPU troubleshooting workflow article previously mentioned.

Kusto keeps telemetry data for a limited period of time. This is usually 28 days, but varies from table to table. From the date and time the customer first reports the issue appeared, refer to this article Code Package Versions and Primary Node History to check the `code_package_version` version history of the server and database.

- If there is no code version change around the time of the start of the issue, then no deployments have take place and this can be ruled out as a cause of the CPU issue.
- If there is a code version change around the time of the start of the issue, it means the SQL database has either been upgraded or was hotpatched. It doesn't necessarily mean that the code change caused the CPU issue, but this is something to be aware of, and an IcM may be required for clarification. Before opening an IcM, check other IcMs to see if the same issue has been reported recently for the new code version.

Mitigation

The recommendations and mitigation actions to resolve high CPU issues are detailed in the High CPU troubleshooting article. Refer to this article for the necessary actions.

For example, high CPU related to query compilation can be caused by:

- Abusive usage of with RECOMPILE (if you see a lot of recompilations)
- Input variables data types or sizes change between executions
- High number of random ad-hoc queries
- Frequent auto update statistics that will trigger query recompile (a very edge case)
- Frequent schema changes (for example, create - drop index) that will also cause recompiles (also a very edge case) etc

High CPU related to query execution can be due to:

- Increased execution count due to workload
- A small amount of queries contributing to high CPU
- Missing indexes
- Outdated statistics

- Plan regressions
- Parallelism etc

Reference

The following is list of troubleshooting T-SQL and Kusto queries for reference.

1. If the CPU issue is ongoing, run these two T-SQL queries to list the current running top queries by CPU usage. The first query returns the session id, query status, query start time, CPU time in milliseconds, login name, host name, program name, query text, execution plan, and additional details. The second query returns the top 10 active CPU queries aggregated by query hash.

CurrentTopQueriesbyCPUusage

```
SELECT req.session_id, req.status, req.start_time, req.cpu_time AS 'cpu_time_ms',
req.logical_reads,req.dop,s.login_name,s.host_name,s.program_name,
object_name(st.objectid,st.dbid) 'ObjectName',
REPLACE (REPLACE (SUBSTRING (st.text,(req.statement_start_offset/2) + 1,
 ((CASE req.statement_end_offset        WHEN -1 THEN DATALENGTH(st.text)
   ELSE req.statement_end_offset END - req.statement_start_offset)/2) + 1),
   CHAR(10), ' '), CHAR(13), ' ') AS statement_text,
qp.query_plan,qsx.query_plan as query_plan_with_in_flight_statistics
FROM sys.dm_exec_requests as req
JOIN sys.dm_exec_sessions as s on req.session_id=s.session_id
CROSS APPLY sys.dm_exec_sql_text(req.sql_handle) as st
OUTER APPLY sys.dm_exec_query_plan(req.plan_handle) as qp
OUTER APPLY sys.dm_exec_query_statistics_xml(req.session_id) as qsx
ORDER BY req.cpu_time desc;
GO
```

Top10ActiveCPUqueriesaggregatedbyqueryhash(TSQL)

```
print '-- top 10 Active CPU Consuming Queries (aggregated)--'
select top 10 getdate() runtime,  * from
(SELECT query_stats.query_hash,
        SUM(query_stats.cpu_time) 'Total_Request_Cpu_Time_Ms',
        sum(logical_reads) 'Total_Request_Logical_Reads',
        min(start_time) 'Earliest_Request_start_Time',
        count(*) 'Number_Of_Requests',
        substring (REPLACE (REPLACE (MIN(query_stats.statement_text),  CHAR(10), ' '), CHAR(13), ' '), 1, 256)
FROM
        (SELECT req.*,
        SUBSTRING(ST.text, (req.statement_start_offset/2) + 1,
        ((CASE statement_end_offset
                WHEN -1 THEN DATALENGTH(ST.text)
                ELSE req.statement_end_offset END
                        - req.statement_start_offset)/2) + 1) AS statement_text
         FROM sys.dm_exec_requests AS req
         CROSS APPLY sys.dm_exec_sql_text(req.sql_handle) as ST) as query_stats
         group by query_hash) t
 order by Total_Request_Cpu_Time_Ms desc
```

2. If the high CPU usage occurred in the past, Query Store captures a history of queries, plans, and runtime statistics and retains them for review. Run this T-SQL query to list the top 15 queries by CPU usage from Query Store in the previous two hours.

- If required, modify the `DATEADD` parameters in line `rsi.start_time>=DATEADD(HOUR, -2, GETUTCDATE())` to return results to the adjusted time frame.

```
WITH AggregatedCPU AS
  (SELECT
    q.query_hash,
    SUM(count_executions * avg_cpu_time / 1000.0) AS total_cpu_ms,
    SUM(count_executions * avg_cpu_time / 1000.0)/ SUM(count_executions) AS avg_cpu_ms,
    MAX(rs.max_cpu_time / 1000.00) AS max_cpu_ms,
    MAX(max_logical_io_reads) max_logical_reads,
    COUNT(DISTINCT p.plan_id) AS number_of_distinct_plans,
    COUNT(DISTINCT p.query_id) AS number_of_distinct_query_ids,
    SUM(CASE WHEN rs.execution_type_desc='Aborted'
         THEN count_executions ELSE 0 END) AS aborted_execution_count,
    SUM(CASE WHEN rs.execution_type_desc='Regular'
         THEN count_executions ELSE 0 END) AS regular_execution_count,
    SUM(CASE WHEN rs.execution_type_desc='Exception'
         THEN count_executions ELSE 0 END) AS exception_execution_count,
    SUM(count_executions) AS total_executions,
    MIN(qt.query_sql_text) AS sampled_query_text
   FROM sys.query_store_query_text AS qt
   JOIN sys.query_store_query AS q ON qt.query_text_id=q.query_text_id
   JOIN sys.query_store_plan AS p ON q.query_id=p.query_id
   JOIN sys.query_store_runtime_stats AS rs ON rs.plan_id=p.plan_id
   JOIN sys.query_store_runtime_stats_interval AS rsi
     ON rsi.runtime_stats_interval_id=rs.runtime_stats_interval_id
   WHERE
    rs.execution_type_desc IN ('Regular', 'Aborted', 'Exception') AND
    rsi.start_time>=DATEADD(HOUR, -2, GETUTCDATE())
   GROUP BY q.query_hash),
OrderedCPU AS
   (SELECT *,
    ROW_NUMBER() OVER (ORDER BY total_cpu_ms DESC, query_hash ASC) AS RN
    FROM AggregatedCPU)
SELECT *
FROM OrderedCPU AS OD
WHERE OD.RN<=15
ORDER BY total_cpu_ms DESC;
GO
```

3. T-SQL query to return the top CPU consuming queries by query hash

TOPCPUconsumingqueries(TSQL)

```
-- top 15 CPU consuming queries by query hash
-- note that a query  hash can have many query id if not parameterized or not parameterized properly
-- it grabs a sample query text by min
WITH AggregatedCPU
AS
(
   SELECT  q.query_hash, SUM(count_executions * avg_cpu_time / 1000.0) AS total_cpu_millisec,
   SUM(count_executions * avg_cpu_time / 1000.0) /SUM(count_executions) as avg_cpu_millisec,
   max(rs.max_cpu_time/1000.00) as max_cpu_millisec,
   max(max_logical_io_reads) max_logical_reads,
   COUNT (distinct p.plan_id) AS number_of_distinct_plans,
   count (distinct p.query_id) as number_of_distinct_query_ids,
   sum (case when rs.execution_type_desc='Aborted' then count_executions else 0 end) as Aborted_Execution_Coun
   sum (case when rs.execution_type_desc='Regular' then count_executions else 0 end) as Regular_Execution_Coun
   sum (case when rs.execution_type_desc='Exception' then count_executions else 0 end) as Exception_Execution_
   sum (count_executions) as total_executions,
   min(qt.query_sql_text) as sampled_query_text
   FROM sys.query_store_query_text AS qt JOIN sys.query_store_query AS q
   ON qt.query_text_id = q.query_text_id
   JOIN sys.query_store_plan AS p ON q.query_id = p.query_id
   JOIN sys.query_store_runtime_stats AS rs ON rs.plan_id = p.plan_id
   JOIN sys.query_store_runtime_stats_interval AS rsi
   ON rsi.runtime_stats_interval_id = rs.runtime_stats_interval_id
   WHERE   rs.execution_type_desc in( 'Regular' , 'Aborted', 'Exception') and
    rsi.start_time >= DATEADD(hour, -2, GETUTCDATE())
   GROUP BY  q.query_hash
)
,OrderedCPU
AS
(
   SELECT  query_hash, total_cpu_millisec, avg_cpu_millisec,max_cpu_millisec,
   max_logical_reads, number_of_distinct_plans, number_of_distinct_query_ids,  total_executions,
   Aborted_Execution_Count,Regular_Execution_Count, Exception_Execution_Count, sampled_query_text,
   ROW_NUMBER () OVER (ORDER BY total_cpu_millisec DESC, query_hash asc) AS RN
   FROM AggregatedCPU
)
SELECT  * from OrderedCPU OD
WHERE OD.RN <=15 ORDER BY total_cpu_millisec DESC
```

4. T-SQL query to return the query text and execution plan by query hash

QueryTexAndExecutionPlanByQueryhash(TSQL)

```
--get query text and execution plan by query hash
SELECT  deqs.query_hash ,
        deqs.query_plan_hash ,
        deqp.query_plan ,
        dest.text
FROM    sys.dm_exec_query_stats AS deqs
        CROSS APPLY sys.dm_exec_query_plan(deqs.plan_handle) AS deqp
        CROSS APPLY sys.dm_exec_sql_text(deqs.sql_handle) AS dest
WHERE   deqs.query_hash IN (query_hash1, query_hash2, query_hash3)
```

5. T-SQL query to return the top 10 Active CPU queries by session

Top10ActiveCPUqueriesbysession

```
print '--top 10 Active CPU Consuming Queries by sessions--'
SELECT top 10 req.session_id, req.start_time, cpu_time 'cpu_time_ms', object_name(st.objectid,st.dbid) 'Object
        substring (REPLACE (REPLACE (SUBSTRING(ST.text, (req.statement_start_offset/2) + 1,
        ((CASE statement_end_offset
                WHEN -1 THEN DATALENGTH(ST.text)
                ELSE req.statement_end_offset END
                        - req.statement_start_offset)/2) + 1), CHAR(10), ' '), CHAR(13), ' '), 1, 512)  AS sta
         FROM sys.dm_exec_requests AS req
         CROSS APPLY sys.dm_exec_sql_text(req.sql_handle) as ST
         order by cpu_time desc
go
```

6. Kusto query to return top 10 CPU consuming query hashes

Top10CPUconsumingQueryHash

```
// top 10 QDS CPU Percent over time
// NOTE: for Managed Instance, remove the database_name filter to get the resource usage of the whole instance

let cpu_cap_in_sec=toscalar(
 MonDmRealTimeResourceStats
 | where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}"
 | where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
 | where replica_type == 0
 | top 1 by TIMESTAMP desc
 | project cpu_cap_in_sec );
MonWiQdsExecStats
   | where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}"
   | where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
   | where is_primary == 1
   | join kind= inner (
    MonWiQdsExecStats
   | where LogicalServerName =~ "{LogicalServerName}" and database_name =~ "{LogicalDatabaseName}"
   | where  TIMESTAMP >= datetime({StartTime}) and  TIMESTAMP <= datetime({EndTime})
   | where is_primary == 1
   | summarize sum(cpu_time) by query_hash
   | top  10 by  sum_cpu_time desc
) on query_hash
| summarize total_cpu_ms=(sum(cpu_time)*1.0/(1000)) by TIMESTAMP=bin(TIMESTAMP, 15min), query_hash
| order by query_hash asc, TIMESTAMP asc nulls first
| serialize
| extend PrevTimestamp=prev(TIMESTAMP, 1), Prev_query_hash=prev(query_hash, 1)
| where isnull(PrevTimestamp) == false
| where query_hash == Prev_query_hash
| extend elapsed_second = datetime_diff ('second',  TIMESTAMP, PrevTimestamp)
| project TIMESTAMP,  query_hash , cpu_percent_over_dtu=round((total_cpu_ms*100.0/1000)/(elapsed_second* cpu_c
| render timechart
```

Root Cause Classification

Cases resolved by this TSG should be coded to the following root cause: Performance\DTU Limit\CPU

How good have you found this content?