Workflow for High CPU troubleshooting

Last updated by | Ricardo Marques | Mar 3, 2023 at 3:31 AM PST

Contents

This TSG is intended to present a workflow on a case where CPU is high. All solutions points to other TSGs that present solutions depending on each case.

Start at point 1

1.Take an ASC report

Start by identifying the period on when the customer had a high CPU event. If it`s happening now, maybe you can use the date from when the customer started to see the issue. This ASC report might be useful down the line.

2.Looking at the ASC report

Open the ASC report and start by going to **Performance** -> **CPU** Look at the graph **User CPU usage including resource stats and query store**. What is the CPU value for Query_Compile_CPU and Query_Exec_CPU? Wich one is contributing for the high CPU?

If its Query_Compile_CPU go to point 3.

If its Query_Exec_CPU go to point 4.

3.Query compilation

We have already narrowed down that the high CPU usage might be related with query compilation. Now, go to **Performance** -> **Queries** tab. Scroll down to **Query Parametrization**. From this table determine if the problem his with ad-hoc workloads or user_parametrized_queries.

Compilation metrics can be pulled from telemetry. Please take a look at Query compilation and to Query compilation 2

You can also check the sections **Top 5 Queries by Compile CPU Percentage** and **Top 5 Recompile Reasons by Compile CPU percentage**. They might give you more clarity on queries thar are being compiled and the reasons why queries are being recompiled.

Some possible problems:

- 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)
- excessive use of temp tables.

Possible solutions

Check the following TSGs:

- Forced parameterization
- Adhoc workloads
- High RecompileDNR and TemptableChanged recompilations

4.Query execution

By now we should know the big portion of the CPU is spent on query execution an not on compilation. Now, open the ASC report and go to **Performance** -> **Overview**. Scroll down to **Query Execution Count Statistics**.

Do you see a trend in rise in query execution count when the issue started? If yes, keep this information in mind, since it might be useful down the line.

Now, on the same tab, scroll to **Instance Wait Statistics**. What are the most noticable waits when the issue started? Take note of them. They will be useful in the future.

Now go to the **Performance** -> **Queries** tab. Scroll to **Top 5 Queries by CPU Consumption**.

Do you see just a few queries that stand out?

If yes, go to point 4.1

If not, go to point 4.2

4.1 Just a few queries are contributing to high CPU

If a small amount of queries are contributing to high CPU, maybe you can tackle the problem in a very straight forward manner by just troubleshooting each query. When any action is possible you can see the results almost immediatly.

Just scroll down to the section **CPU Utilization Over Time for the Top 5 Queries by CPU Consumption**.

Do you see any trend where the query suddenly started using more CPU?

If yes we might want to check if there is a plan regression. Go to **Performance** -> **Plans** and check **Queries with Plan Regressions**. If you find Regressed plans reported, go to 4.1.1.

If not proceed to 4.1.2

4.1.1 Plan regression

Possible solutions:

- check Plan regression TSG.

4.1.2 Query tunning

On **Performance** -> **Queries**, scroll to **Top Waiting Queries**. Check the following values:

- Top3_Wait_Categories
- AvgLogicalReads (and writes, depending on the type of query).
- Memory Grants
- Row Count (max, min and avg)

Also check **Performance** -> **Queries** -> **Top CPU Consuming Queries with Anti-Patterns**

- the query has an implicit conversion?
- the query contains a non-SARGable predicate?

Possible solutions:

- get the execution plan like described on this TSG
- if an anti-pattern was detected check this TSG. This can also be checked on the Execution plan
- if you saw high memory grants, check for outdated statistics, query design, indexing opportunities. This can be checked on the Execution plan.
- check of the query recompiled when the issue started - change the Kusto query and search by query hash. If the query recompiled a new plan due statistics change, update statistics. If the query recompiled due to schema change, it might be that an index was dropped. Check the query execution plan.
- Check also for parameter sniffing.
- You can use the wait statistics has a starting point to look at the plan and find issues.
- If the query contains a non-SARGable predicate, the query or table structure might have to be changed. Check this TSG

4.2 The issue seems to be a general worload problem

This is the most broader scenario, where different things can contribute to a high CPU consumption.

Some points you might want to explore:

- is there an increase on the workload? Correlate with the query execution count that you observed on point 4. If this is the case, suggest increasing the MI SLO

- the application was recently migrated? Just a few examples:

1. the database was moved from a different DBMS to SQL. The index structure and possibility data types still reflects the previous DBMS. If this is the case the issue needs to be tackled from execution plan standpoint. Use also the query referenced below to get the most expensive queries from a CPU perspective.
2. the database was moved from another SQL instance. On the previous instance the database was using a different Cardinality Estimation (CE). On this case you might want to tackle the problem from CE perspective.
3. The database was moved from a different server. It was using the same CE. You might want to compare settings (like maxdop and parallelism threshold values) and resources. Increase the MI SLO might be a good plan.

- get the most expensive CPU queries. You can use this or this query. The first is more what is happenning right now and the second one for a more overall review.

You can use also this query.

- If Parallelism waits were noticed on point 4, check if reducing Maxdop to 8 or 4 would improve. Cost treshold for parallelim can also be changed to a higher value (both points require testing). If the customer workload relies heavily on parallelism (for example, report extraction) check for indexing opportunities and for the increase of the MI SLO

- check if there isnt a more applicational design problem. For example, multiple queries with implicit convertions. Check this example.

- also check general statistics problems.

HighCPUQueries

```
;with high_cpu_queries as
(
select top 20
        query_hash,
        sum(total_worker_time) cpuTime
    from sys.dm_exec_query_stats
    where query_hash <> 0x0
    group by query_hash
    order by sum(total_worker_time) desc
)
select @@servername as server_name,
    coalesce(db_name(st.dbid), db_name(cast(pa.value AS INT)), 'Resource') AS [DatabaseName],
    coalesce(object_name(ST.objectid, ST.dbid), '<none>') as [object_name],
    qs.query_hash,
    qs.total_worker_time as cpu_time,
    qs.execution_count,
    cast(total_worker_time / (execution_count + 0.0) as money) as average_CPU_in_microseconds,
    cpuTime as total_cpu_for_query,
    SUBSTRING(ST.TEXT,(QS.statement_start_offset + 2) / 2,
        (CASE
            WHEN QS.statement_end_offset = -1  THEN LEN(CONVERT(NVARCHAR(MAX),ST.text)) * 2
            ELSE QS.statement_end_offset
            END - QS.statement_start_offset) / 2) as sql_text,
    qp.query_plan
from sys.dm_exec_query_stats qs
join high_cpu_queries hcq
    on hcq.query_hash = qs.query_hash
cross apply sys.dm_exec_sql_text(qs.sql_handle) st
cross apply sys.dm_exec_query_plan (qs.plan_handle) qp
outer apply sys.dm_exec_plan_attributes(qs.plan_handle) pa
where pa.attribute = 'dbid'
order by hcq.cpuTime desc,
    hcq.query_hash,
    qs.total_worker_time desc
option (recompile)
```

How good have you found this content?