Higher Resource Consumption Due To Workload Increase

Last updated by | Francisco O | Nov 22, 2022 at 6:35 AM PST

Contents

Issue

Customers often inquire about resource usage increase, and while frequently this is related to specific performance issues / regressions, it is not uncommon that the pattern emerges as a consequence of a workload increase.

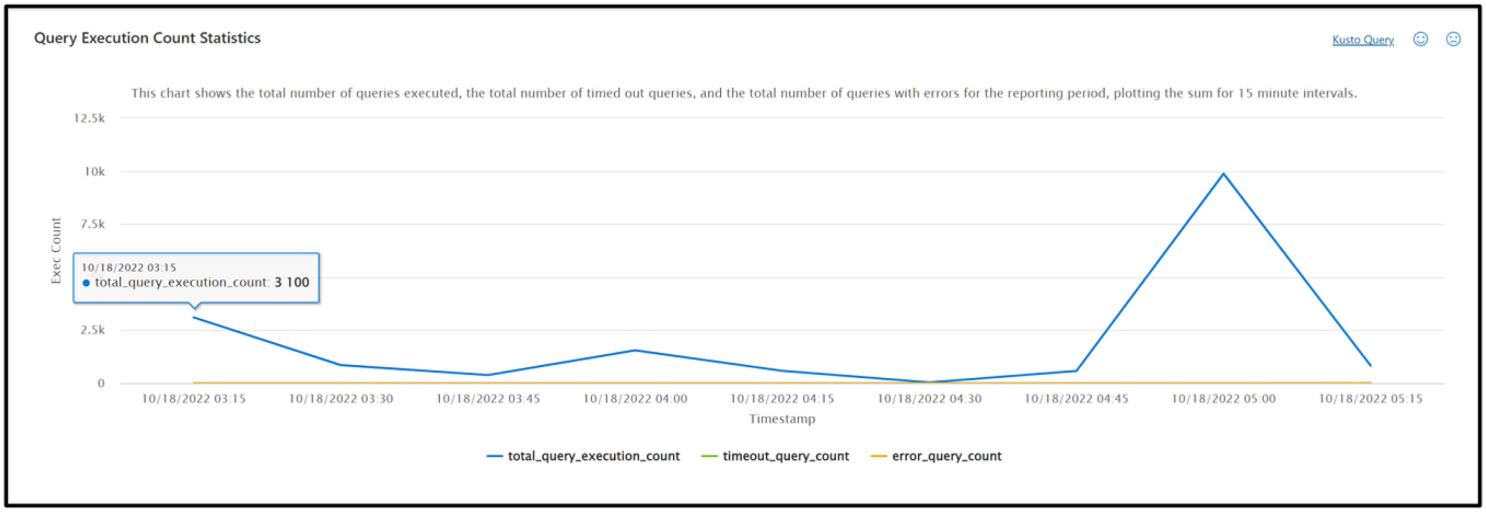
Investigation/Analysis

Below you can find relevant steps and queries that can be used to check for an eventual workload increase, by comparing the query count in distinct moments.

Checking Overall Query Count Distribution

Using ASC

On ASC, you can easily see the distribution of the query count accross the respective time interval(s) by running the Troubleshooter report, then go to **Performance** > **Overview** > **Query Execution Count Statistics**



Query Execution Count Statistics

This chart shows the total number of queries executed, the total number of timed out queries, and the total number of queries with errors for the reporting period, plotting the sum for 15 minute intervals.

Using Kusto

```
let serverName  = 'serverName';
let databaseName = 'databaseName';
let appName = 'AppName';
let nodeName = 'DB.';
let clusterName = 'tr.region.worker.database.windows.net';
let startDate = datetime('YYYY-MM-DD HH:MM:SS');
let endDate = datetime('YYYY-MM-DD HH:MM:SS');
MonWiQdsExecStats
| where
    AppName =~  appName and
    originalEventTimestamp between (startDate..endDate) and
    ClusterName =~ clusterName and
    NodeName =~ nodeName and
    LogicalServerName =~ serverName and
    database_name =~ databaseName
| summarize
    total_query_execution_count = sum(execution_count),
    timeout_query_count = sumif(execution_count, exec_type == 3),
    error_query_count = sumif(execution_count, exec_type == 4)
    by bin(originalEventTimestamp, 1h)
| render
    timechart
```

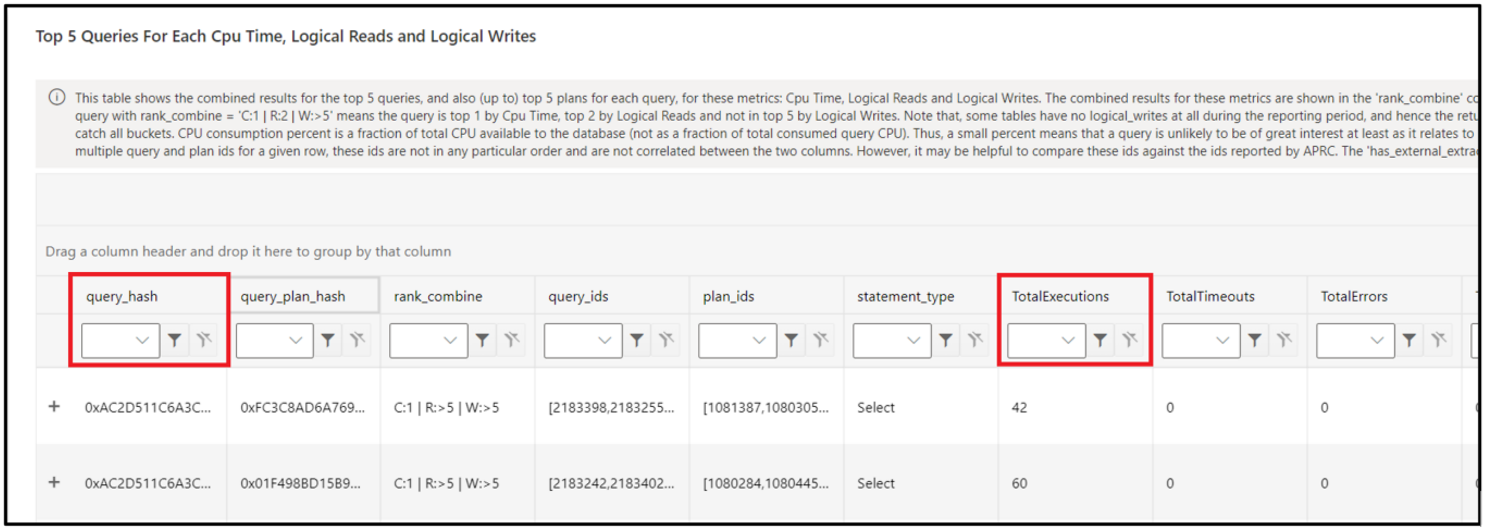
Using Query Store

```
SELECT
    rs.execution_type,
    SUM (count_executions) AS Execution_Count,
    ROUND(CONVERT(float, SUM(rs.avg_duration*rs.count_executions))/NULLIF(SUM(rs.count_executions), 0)*0.001,2
    ROUND(CONVERT(float, SUM(rs.avg_cpu_time*rs.count_executions))/NULLIF(SUM(rs.count_executions), 0)*0.001,2
    ROUND(CONVERT(float, SUM(rs.avg_logical_io_reads*rs.count_executions))/NULLIF(SUM(rs.count_executions), 0)
    ROUND(CONVERT(float, SUM(rs.avg_logical_io_writes*rs.count_executions))/NULLIF(SUM(rs.count_executions), 0
    ROUND(CONVERT(float, SUM(rs.avg_physical_io_reads*rs.count_executions))/NULLIF(SUM(rs.count_executions), 0
FROM
    sys.query_store_runtime_stats rs (NOLOCK)
INNER JOIN
    sys.query_store_runtime_stats_interval i ON rs.runtime_stats_interval_id = i.runtime_stats_interval_id
--WHERE
--    i.start_time BETWEEN '' AND ''
GROUP BY
  execution_type
```

Drilling down for specific Queries

Using ASC

On ASC you can see the specific queries and their respective execution count by going to > **Performance** > **Queries** > **Top 5 Queries For Each Cpu Time, Logical Reads and Logical Writes**



Top 5 Queries For Each Cpu Time, Logical Reads and Logical Writes

This table shows the combined results for the top 5 queries, and also (up to) top 5 plans for each query, for these metrics: Cpu Time, Logical Reads and Logical Writes. The combined results for these metrics are shown in the 'rank_combine' cc
query with rank_combine = 'C:1 | R:2 | W:>5' means the query is top 1 by Cpu Time, top 2 by Logical Reads and not in top 5 by Logical Writes. Note that, some tables have no logical_writes at all during the reporting period, and hence the retu
catch all buckets. CPU consumption percent is a fraction of total CPU available to the database (not as a fraction of total consumed query CPU). Thus, a small percent means that a query is unlikely to be of great interest at least as it relates to
multiple query and plan ids for a given row, these ids are not in any particular order and are not correlated between the two columns. However, it may be helpful to compare these ids against the ids reported by APRC. The 'has_external_extra

Drag a column header and drop it here to group by that column

	query_hash	query_plan_hash	rank_combine	query_ids	plan_ids	statement_type	TotalExecutions	TotalTimeouts	TotalErrors
+	0xAC2D511C6A3C...	0xFC3C8AD6A769...	C:1 | R:>5 | W:>5	[2183398,2183255...	[1081387,1080305...	Select	42	0	0
+	0xAC2D511C6A3C...	0x01F498BD15B9...	C:1 | R:>5 | W:>5	[2183242,2183402...	[1080284,1080445...	Select	60	0	0

If the same query hashes are seen in the top consuming queries over two distinct comparable periods, by simply checking the total number of executions, we can confirm if the workload increased, or not.

Otherwise, If the query hashes are different we might need to further analyze CPU / Logical Reads / Writes total values to compare workloads.

Using Kusto

Query Execution count by query hash

```
let serverName  = 'serverName';
let databaseName = 'databaseName';
let appName = 'AppName';
let nodeName = 'DB.';
let clusterName = 'tr.region.worker.database.windows.net';
let startDate = datetime('YYYY-MM-DD HH:MM:SS');
let endDate = datetime('YYYY-MM-DD HH:MM:SS');
MonWiQdsExecStats
| where
    AppName =~  appName and
    originalEventTimestamp between (startDate..endDate) and
    ClusterName =~ clusterName and
    NodeName =~ nodeName and
    LogicalServerName =~ serverName and
    database_name =~ databaseName
| top-nested of bin(originalEventTimestamp, 1h) by sum(execution_count), top-nested 5 of query_hash by exec_co
| sort by originalEventTimestamp asc nulls last
| project originalEventTimestamp, query_hash, exec_count
| render timechart
```

Query Execution count for a specific query hash, per second:

```
let serverName  = 'serverName';
let databaseName = 'databaseName';
let appName = 'AppName';
let nodeName = 'DB.';
let clusterName = 'tr.region.worker.database.windows.net';
let startDate = datetime('YYYY-MM-DD HH:MM:SS');
let endDate = datetime('YYYY-MM-DD HH:MM:SS');
let queryHash = "0x000000"; //query hash
let execCount =
MonWiQdsExecStats
| where
    AppName =~  appName and
    originalEventTimestamp between (startDate..endDate) and
    ClusterName =~ clusterName and
    NodeName =~ nodeName and
    LogicalServerName =~ serverName and
    database_name =~ databaseName
| where query_hash == queryHash
| summarize total_execution_count = sum(execution_count) by bin(originalEventTimestamp,15m), query_hash;
execCount
| project originalEventTimestamp, query_hash, total_execution_count, avg_exec_count_per_sec = (total_execution
```

Getting Query Details by query hash

The query below can be executed from the customer side, on respective user database, replacing the **q.query_hash** for the query hashes of the problematic queries previously identified.

This information includes the respective sql text and execution plan, to help the customer to identify the respective queries and potential improvement opportunities.

Using Query Store

```
with query_ids as (
    SELECT
        q.query_hash,
        q.query_id,
        p.query_plan_hash,
        SUM(qrs.count_executions) * AVG(qrs.avg_cpu_time)/1000. as total_cpu_time_ms,
        SUM(qrs.count_executions) AS sum_executions,
        AVG(qrs.avg_cpu_time)/1000. AS avg_cpu_time_ms,
             AVG(qrs.avg_logical_io_reads)/1000. AS avg_logical_io_reads_ms,
             AVG(qrs.avg_physical_io_reads)/1000. AS avg_physical_io_reads_ms
    FROM sys.query_store_query q
    JOIN sys.query_store_plan p on q.query_id=p.query_id
    JOIN sys.query_store_runtime_stats qrs on p.plan_id = qrs.plan_id
     WHERE q.query_hash in (0x0000000000000000,0x0000000000000000...) /*query hashes*/
    GROUP BY q.query_id, q.query_hash, p.query_plan_hash)
SELECT qid.*,
    qt.query_sql_text,
    p.count_compiles,
    TRY_CAST(p.query_plan as XML) as query_plan
FROM query_ids as qid
JOIN sys.query_store_query AS q ON qid.query_id=q.query_id
JOIN sys.query_store_query_text AS qt on q.query_text_id = qt.query_text_id
JOIN sys.query_store_plan AS p ON qid.query_id=p.query_id and qid.query_plan_hash=p.query_plan_hash
```

Mitigation

- Tune top consuming queries
- Increase SLO to manage the workload increase

Internal References

- IcM 335042550

How good have you found this content?

🙂 🙁