Query Performance and timeouts

Last updated by | Holger Linke | Oct 26, 2022 at 1:41 AM PDT

Contents

NOTE: If the SLO of the database is lower than S2 , it is considered a Sub-core SLO and there is not much that can be done to improve its performance. The best solution is to ask customer to scale up.

Issue summary -

Customers choose this supporting topic when they have identified that a particular query is running slower than expected, or timing out. You may also see comparative inputs such as query execution time changed from X to Y.

You may see a error message similar to :

System.Data.SqlClient.SqlException (0x80131904): Execution Timeout Expired. The timeout period elapsed prior to completion of the operation or the server is not responding. --- System.ComponentModel.Win32Exception (0x80004005)

You might even see connectivity issues in such cases.

It is important to establish if the query has ever executed completely, or has it always failed. This allows you you identify if this is something that has changed in the customer's environment or something that has never worked. In all cases it is good practice to get execution plans for analysis, and check with the customer what their index/statistics maintenance routine is, also establish if the timeout is set in the connection string.

Analysis -	Individual queries can have issues due to waiting on something (a waiting-related condition), bad plan, resource crunch etc.
Azure Support Center -	Perf Overview
Other Tools -	<Kusto / XTS / Powershell / CLI Reference>

Mitigation steps -

1. **Get the required details:**

From Customer:

- Specific query is responding slow / what customer is exactly doing using this SPROC.
- Server name / Database name.
- Query ID's if possible.
- Query Execution plan if possible.
- When the problem has started.
- Get a comparative time, to understand the expected query execution- the time taken by the query before the issue started.
- What are the changes in the database before the incident and during the incident.

2. **If this is a timeout issue ,from the callstack that the customer provided, ensure it is a query timeout issue instead of connection timeout.**

3. **Review the following details from ASC:**

- Check the following Insights on ASC:
    1. Insight on the main page.
    2. Performance insights on the troubleshooter. (ASC >> Tools >> SQL Troubleshooter >> Create Report >> Performance >> Performance Insights). Focus on Insights with status != All Clear.
- Work load details on the database during the time of incident.

ASC >> Tools >> SQL Troubleshooter >> Create Report >> Performance >> Overview.

- If customer is not sure about the exact query, review the same on ASC.

ASC >> Tools >> SQL Troubleshooter >> Create Report >> Performance >> Queries.

ASC >> Tools >> SQL Troubleshooter >> Create Report >> Performance >> Plans.

- Also review any Blocking and Deadlocking during the timeframe.

ASC >> Tools >> SQL Troubleshooter >> Create Report >> Performance >> Blocking and Deadlocking.

If its due to Deadlock, go to:
[Deadlocks]

- Configuration Changes:

ASC >> Tools >> SQL Troubleshooter >> Create Report >> Performance >> Config & Change History.

4. Make sure :

- Check for Statistics: [Statistics](/SQL-Database/Troubleshooting-Guides/Performance-and-Query-Exection/Reference/Statistics)
- Check for Indexes: [Indexes](/SQL-Database/Troubleshooting-Guides/Performance-and-Query-Exection/Reference/Indexes)
- Recommended daily maintenance solution for Indexes and Statistics.

Refer : https://techcommunity.microsoft.com/t5/Azure-Database-Support-Blog/How-to-maintain-Azure-SQL-Indexes-and-Statistics/ba-p/368787

(If the information from ASC was not sufficient, submit a feedback and use below) :

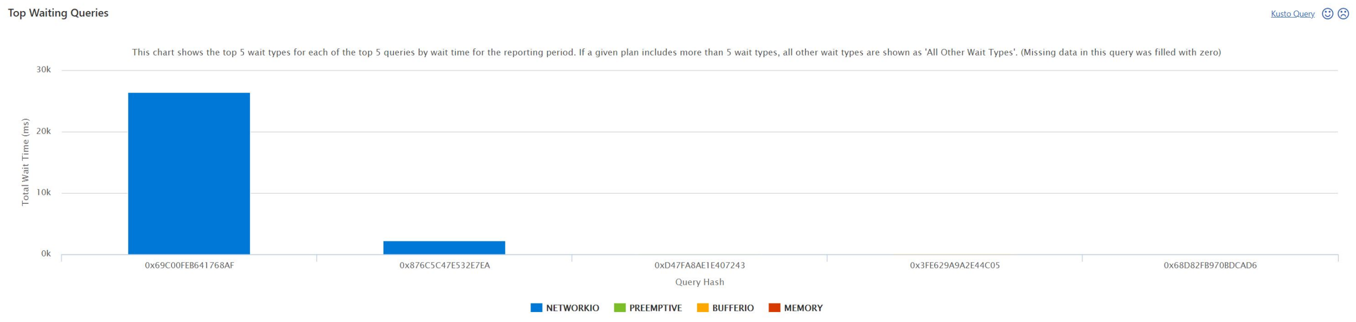
5. To identify expensive queries during a particular timeframe use:

```
MonWiQdsExecStats
|where LogicalServerName == "patorgau" and database_name ==  "patorganiserau.prod"
//| where originalEventTimestamp >  datetime(2017-11-01 01:30:17.5629893)
| where originalEventTimestamp > datetime(2017-11-01 01:43:23.0020000)
//| where originalEventTimestamp <  datetime(2017-11-01 02:30:17.5629893)
| where originalEventTimestamp < datetime(2017-11-01 02:28:23.7850000)
| extend interval_start_time_date = interval_start_time / 4294967296
| extend interval_start_time_time = interval_start_time - 4294967296 * interval_start_time_date
| extend interval_start = datetime(1900-1-1) + time(1d) * interval_start_time_date + time(1s) *
(interval_start_time_time / 300.0)
| extend interval_start_time_date = interval_start_time / 4294967296
| extend interval_start_time_time = interval_start_time - 4294967296 * interval_start_time_date
| extend interval_start = datetime(1900-1-1) + time(1d) * interval_start_time_date + time(1s) *
(interval_start_time_time / 300.0)
| extend interval_end_time_date = interval_end_time / 4294967296
| extend interval_end_time_time = interval_end_time - 4294967296 * interval_end_time_date
| extend interval_end = datetime(1900-1-1) + time(1d) * interval_end_time_date + time(1s) *
(interval_end_time_time / 300.0)
| extend Average_cpu_time = cpu_time / execution_count,
      Average_logical_reads = logical_reads / execution_count,
      Average_logical_writes = logical_writes / execution_count,
      Average_physical_reads = physical_reads / execution_count,
      Average_elapsed_time = elapsed_time / execution_count,
      Average_log_bytes_used = log_bytes_used / execution_count,
      Average_rowcount = rowcount / execution_count
|project interval_start,interval_end,AppName ,NodeName,originalEventTimestamp
,LogicalServerName,exec_type,Average_cpu_time,Average_logical_reads,Average_logical_writes,Average
_physical_reads,
Average_elapsed_time,Average_log_bytes_used,Average_rowcount
```

,logical_writes,logical_reads,min_logical_reads,max_logical_reads,physical_reads,min_physical_reads,max_physical_reads,execution_count,cpu_time,rowcount,elapsed_time, log_bytes_used, query_id,plan_id,query_plan_hash,database_name,tempdb_space_used,query_hash

6. If you have specific query ID and time range :

Go to SQL Troubleshooter >>> Performance >>> Queries >>> Top waiting queries



You can see the waiting queries and their wait types here. Also you can get the query hash for further investigation.

3. You can query particular query hash in kusto to fetch more details:

```
MonWiQdsWaitStats
| where TIMESTAMP > datetime() and TIMESTAMP  < datetime()
| where AppName == ''
| where database_name == ''
| where exec_type != 0
| project originalEventTimestamp, query_hash, wait_category, total_query_wait_time_ms, avg_query_wait_
| where query_hash == ''
```

The wait_category and the total_query_wait_time are good pointers.

Also you can get the query hash for further investigation.

You can review the waits on a particular query using:

```
MonWiQdsWaitStats
| where LogicalServerName == "mo-weu-prod" and database_name == "MintecOnline"
| where originalEventTimestamp > datetime(2018-07-12 17:00:00.0020000)
| where originalEventTimestamp < datetime(2018-07-12 18:00:00.7850000)
| where query_id in (19101, 12613)
```

7. To review further performance details :

For standalone databases: Performance Stand Alone DB's

For Elastic pool databases: Performance: Elastic Pool's

8. Tools to capture logs and analyze further:

PerfStats:
Collecting Perf Stats

Xevents:
Collecting XEvents

Public Doc Reference :

https://docs.microsoft.com/en-us/azure/sql-database/sql-database-monitor-tune-overview#monitor-database-performance

Internal MSFT Reference:

[Keywords]

Classification

Root cause path -

Could be mulitple

How good have you found this content?