

# Perf\_Command\_Timeout

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

---

## Contents

- [Issue](#)
- [Investigation/Analysis](#)
- [Mitigation](#)
- [Root Cause Classification](#)

## Issue

If a customer is observing a performance issue due to command timeouts, you can use the TSG below.

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.

## Investigation/Analysis

In general command timeouts can be self-serve investigated using query data store

<https://docs.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver15> . exec\_type != 0 captures failed queries and 3 is timeout.

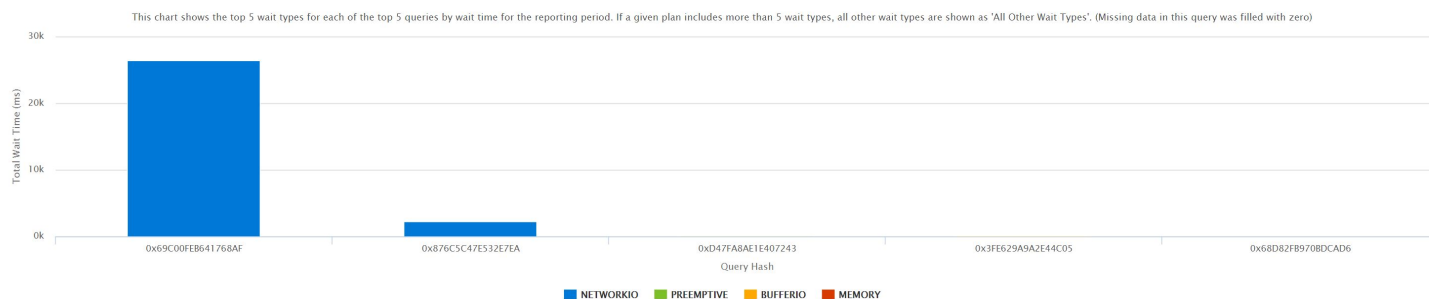
Please recommend the same to the customer.

That said, using wait stats for failed queries we can understand the bottlenecks; in many cases. If you already know the queries that have issues go to step 3. To identify the same follow the steps below:

1. Run ASC for the particular duration when the timeouts were observed.
2. Go to SQL Troubleshooter >>> Performance >>> Queries >>> Top waiting queries

Top Waiting Queries

Kusto Query



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_time_ms,
| where query_hash == ''
```

The wait\_category and the total\_query\_wait\_time are good pointers.

## Mitigation

Depending on the wait category you can recommend the following to the customer:

1. If the query bottleneck on IO :

- Recommendation is increase timeout.
- Further optimize query if possible.
- If query must perform lots of IOs and is latency sensitive than latencies per IO are better in BC offers.

2. If the query bottleneck is on CPU:

- Check other CPU intensive activities.
- Find of that is due to any other user load.
- Recommend to optimize\batch the query or scale the database.

Also follow the recommendations on : <https://docs.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver15> for various waits.

## Root Cause Classification

Cases resolved by this TSG should be coded to the following root cause: Workload Performance/User-issue/error

**How good have you found this content?**

