* Created by Kulwant Singh, last modified on [25 Jul 2018](https://confluence.markit.com/pages/diffpagesbyversion.action?pageId=200269649&selectedPageVersions=4&selectedPageVersions=5)

Table of Contents

**Introduction1**

**Questions2**

First Steps3

Appendix: Scripts that can be of use4

1. Introduction

This document forms part of the "Consistent approach to investigating performance issues" which is designed to standardize performance investigations and their output.  This document specifically addresses the first stages of a SQL server and database performance investigation, what questions to ask, what to initially look at and what information should be attached to the incident ticket.

2. Questions

The first part of a performance investigation is to understand what the performance problem actually is.  Answers to the following questions must be sought before any actual investigation is undertaken:

* Which SQL server/ database have the problem?
* What is slow?
  + - SQL Server
    - Specific database
    - A user’s session
    - A single business process/ SQL agent job
    - A single SQL statement
    - Some executions of a single SQL statement
* SQL server replication/ DB mirroring/ Always ON
* How can what is slow be identified, e.g.  SPID(s), SQL Text, Stored Proc, Job, session ID etc.
* When did the problem start?
* When was the perform last acceptable and how long is acceptable?
* Have there been any changes since the last time the performance was acceptable?

The answers to these questions should be included in any incident created for the performance problem.

3. First steps

The answers to the above questions should guide the first stage of the performance analysis.  This should be:

* In all cases, generate IDERA reports for (CPU, Memory, Disk and Database Statistics) of the problem period, and if there is one, the acceptable period.
* If it is a session or business process, then if these can be uniquely identified, Check the session details through SP\_WhoIsActive command
* If it is a single SQL statement, check the query execution plan by running SP\_WhoIsActive @get\_plans = 1
* The error/event log should be reviewed for errors during the problem period. For memory related issues(I/O *requests taking longer than 15 seconds to complete*, System.OutOfMemoryException,

After generating and analyzing the above reports, the following three questions should be considered:

* Is the performance problem solely with the server level?
* Is the performance problem solely with the specific database level?
* Is the performance problem obvious and can it be fixed at the database level?
* Unless you can answer "yes" to the first question, then the other GTS teams should be involved - process to be defined.
* Unless you can answer "yes" to the second and third question, all the generated reports and your analysis should be uploaded to the incident ticket and, if required, help sought from other team members.

4. Appendix: Scripts that can be of use

1. **Check if the instances are running on their preferred node on cluster by running the below command in powershell.**

$groups = (Get-ClusterGroup -cluster NJ4PDDARCLS03.PRD.MKAPP.NET | where {$\_.name.contains("NJ4-CG")}); $groups |% {$owner = (($\_ | get- clusterownernode).OwnerNodes[0]); write-host $\_.name"-"$\_.OwnerNode"-->"$owner.name;}

**2. Read the Error Logs** - sp\_readerrorlog 0,1,’error’

**3. Check any blocking or deadlocks using the below TSQL command.**

select \* from sys.sysprocesses where blocked <>0

**4. Check Long Running Queries using the TSQL command** sp\_whoisactive

**5. If there is slowness in execution of queries or processing the queries, check the wait types using below command.**

select \* from sys.dm\_os\_wait\_stats

The wait type reasons can be checked on the Microsoft blog.

https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-os-wait-stats-transact-sql?view=sql-server-2017

**6. Check the last successful run of index optimize job and check the fragmentation level on the database using below query.**

SELECT dbschemas.[name] as 'Schema', dbtables.[name] as 'Table', dbindexes.[name] as 'Index', indexstats.avg\_fragmentation\_in\_percent, indexstats.page\_count FROM sys.dm\_db\_index\_physical\_stats (DB\_ID(), NULL, NULL, NULL, NULL) AS indexstats INNER JOIN sys.tables dbtables on dbtables.[object\_id] = indexstats.[object\_id] INNER JOIN sys.schemas dbschemas on dbtables.[schema\_id] = dbschemas.[schema\_id]INNER JOIN sys.indexes AS dbindexes ON dbindexes.[object\_id] = indexstats.[object\_id] AND indexstats.index\_id = dbindexes.index\_id WHERE indexstats.database\_id = DB\_ID() ORDER BY indexstats.avg\_fragmentation\_in\_percent desc

This query can be modified to focus on specific tables by append the table name to the 'where' clause:  
  
 WHERE indexstats.database\_id = DB\_ID() AND dbtables.[name] like '%%'  
  
In order to reduce fragmentation we will have to reorganize or rebuild the indexes. Choosing between reorganizing and rebuilding depends on the query results. For heavily fragmented indexes a rebuild process is needed, otherwise index reorganization should be sufficient.

The following table summarizes when to use each one:

|  |  |  |
| --- | --- | --- |
| **Reference Values (in %)** | **Action** | **SQL statement** |
| avg\_fragmentation\_in\_percent > 5 AND < 30 | Reorganize Index | ALTER INDEX REORGANIZE |
| avg\_fragmentation\_in\_percent > 30 | Rebuild Index | ALTER INDEX REBUILD |

Note : Please consult with the team before doing this operation as this require high resources to run and also needs the database to be offline in case of rebuild offline.

**7. Check last successful run of the stats updated job. Also check the last stats updated on a database.**

select a.id as 'ObjectID', isnull(a.name,'Heap') as 'IndexName', b.name as 'TableName',stats\_date (id,indid) as stats\_last\_updated\_time from sys.sysindexes as a inner join sys.objects as b on a.id = b.object\_id where b.type = 'U'

**8. To troubleshoot slowness due to memory crunch use the below TSQL commands.**

* **SQL Server Database wise CPU Utilization**

WITH DB\_CPU AS (SELECT DatabaseID, DB\_Name(DatabaseID)AS [DatabaseName], SUM(total\_worker\_time)AS [CPU\_Time(Ms)] FROM sys.dm\_exec\_query\_stats AS qs CROSS APPLY(SELECT CONVERT(int, value)AS [DatabaseID] FROM sys.dm\_exec\_plan\_attributes(qs.plan\_handle) WHERE attribute =N'dbid')AS epa GROUP BY DatabaseID) SELECT ROW\_NUMBER()OVER(ORDER BY [CPU\_Time(Ms)] DESC)AS [SNO], DatabaseName AS [DBName], [CPU\_Time(Ms)], CAST([CPU\_Time(Ms)] \* 1.0 /SUM([CPU\_Time(Ms)]) OVER()\* 100.0 AS DECIMAL(5, 2))AS [CPUPercent] FROM DB\_CPU WHERE DatabaseID > 4 -- system databases AND DatabaseID <> 32767 -- ResourceDB ORDER BY SNO OPTION(RECOMPILE);

* **Script to find Top 20 Costliest Stored Procedures – High CPU:**

SELECT TOP (20)

p.name AS [SP Name], qs.total\_worker\_time AS [TotalWorkerTime],

qs.total\_worker\_time/qs.execution\_count AS [AvgWorkerTime],

qs.execution\_count, ISNULL(qs.execution\_count/DATEDIFF(Second, qs.cached\_time, GETDATE()), 0) AS [Calls/Second], qs.total\_elapsed\_time,

qs.total\_elapsed\_time/qs.execution\_count AS [avg\_elapsed\_time],

qs.cached\_time FROM sys.procedures AS p WITH (NOLOCK)

INNER JOIN sys.dm\_exec\_procedure\_stats AS qs WITH (NOLOCK) ON p.[object\_id] = qs.[object\_id] WHERE qs.database\_id = DB\_ID() ORDER BY qs.total\_worker\_time DESC OPTION (RECOMPILE);

* **Script to find Top 20 Costliest Queries – High CPU**

SELECT TOP (20)

    st.text AS Query, qs.execution\_count, qs.total\_worker\_time AS Total\_CPU,

    total\_CPU\_inSeconds = --Converted from microseconds

    qs.total\_worker\_time/1000000, average\_CPU\_inSeconds = --Converted from microseconds (qs.total\_worker\_time/1000000) / qs.execution\_count,

    qs.total\_elapsed\_time, total\_elapsed\_time\_inSeconds = --Converted from microseconds  qs.total\_elapsed\_time/1000000,

 qp.query\_plan FROM sys.dm\_exec\_query\_stats AS qs

CROSS APPLY sys.dm\_exec\_sql\_text(qs.sql\_handle) AS st

CROSS apply sys.dm\_exec\_query\_plan (qs.plan\_handle) AS qp

ORDER BY qs.total\_worker\_time DESC OPTION (RECOMPILE);