**You can run the following query to get all DMV and DMF names**:

**SELECT \* FROM sys.system\_objects**

**WHERE name LIKE 'dm\_%'**

**ORDER BY name**

89 for 2005, 136 for 2008, R2

^^^^^^^^^^^^^^^^^^^^^^^^^^^

**Monitoring CPU bottlenecks**

A CPU bottleneck is often caused by a non-optimal query plan, a poor configuration, poor design factors, or insufficient hardware resources. The following are some commonly used queries to help you identify what causes a CPU bottleneck.

The following query gives you a high-level view of which currently cached batches or procedures are using the most CPU.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**SELECT TOP 50**

 **SUM(qs.total\_worker\_time) AS total\_cpu\_time,**

 **SUM(qs.execution\_count) AS total\_execution\_count,**

 **COUNT(\*) AS number\_of\_statements,**

 **qs.sql\_handle**

**FROM sys.dm\_exec\_query\_stats AS qs**

**GROUP BY qs.sql\_handle**

**ORDER BY SUM(qs.total\_worker\_time) DESC**

The following query shows the aggregate CPU usage by cached plans with SQL text.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**SELECT**

 **total\_cpu\_time,**

 **total\_execution\_count,**

 **number\_of\_statements,**

 **s2.text**

 **--(SELECT SUBSTRING(s2.text, statement\_start\_offset / 2, ((CASE WHEN statement\_end\_offset = -1 THEN (LEN(CONVERT(NVARCHAR(MAX), s2.text)) \* 2) ELSE statement\_end\_offset END) - statement\_start\_offset) / 2) ) AS query\_text**

**FROM**

 **(SELECT TOP 50**

 **SUM(qs.total\_worker\_time) AS total\_cpu\_time,**

 **SUM(qs.execution\_count) AS total\_execution\_count,**

 **COUNT(\*) AS number\_of\_statements,**

 **qs.sql\_handle --,**

 **--MIN(statement\_start\_offset) AS statement\_start\_offset,**

 **--MAX(statement\_end\_offset) AS statement\_end\_offset**

 **FROM**

 **sys.dm\_exec\_query\_stats AS qs**

 **GROUP BY qs.sql\_handle**

 **ORDER BY SUM(qs.total\_worker\_time) DESC) AS stats**

 **CROSS APPLY sys.dm\_exec\_sql\_text(stats.sql\_handle) AS s2**

The following query shows the top 50 SQL statements with high average CPU consumption.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**SELECT TOP 50**

**total\_worker\_time/execution\_count AS [Avg CPU Time],**

**(SELECT SUBSTRING(text,statement\_start\_offset/2,(CASE WHEN statement\_end\_offset = -1 then LEN(CONVERT(nvarchar(max), text)) \* 2 ELSE statement\_end\_offset end -statement\_start\_offset)/2) FROM sys.dm\_exec\_sql\_text(sql\_handle)) AS query\_text, \***

**FROM sys.dm\_exec\_query\_stats**

**ORDER BY [Avg CPU Time] DESC**

The following shows DMV queries to find out excessive compiles/recompiles.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select \* from sys.dm\_exec\_query\_optimizer\_info**

**where**

 **counter = 'optimizations'**

 **or counter = 'elapsed time'**

The following sample query gives you the top 25 stored procedures that have been recompiled. The plan\_generation\_num indicates the number of times the query has recompiled.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select top 25**

 **sql\_text.text,**

 **sql\_handle,**

 **plan\_generation\_num,**

 **execution\_count,**

 **dbid,**

 **objectid**

**from sys.dm\_exec\_query\_stats a**

 **cross apply sys.dm\_exec\_sql\_text(sql\_handle) as sql\_text**

**where plan\_generation\_num > 1**

**order by plan\_generation\_num desc**

An inefficient query plan may cause increased CPU consumption.

The following query shows which query is using the most cumulative CPU.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**SELECT**

 **highest\_cpu\_queries.plan\_handle,**

 **highest\_cpu\_queries.total\_worker\_time,**

 **q.dbid,**

 **q.objectid,**

 **q.number,**

 **q.encrypted,**

 **q.[text]**

**from**

 **(select top 50**

 **qs.plan\_handle,**

 **qs.total\_worker\_time**

 **from**

 **sys.dm\_exec\_query\_stats qs**

 **order by qs.total\_worker\_time desc) as highest\_cpu\_queries**

 **cross apply sys.dm\_exec\_sql\_text(plan\_handle) as q**

**order by highest\_cpu\_queries.total\_worker\_time desc**

The following query shows some operators that may be CPU intensive, such as ‘%Hash Match%’, ‘%Sort%’ to look for suspects.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select \***

**from**

 **sys.dm\_exec\_cached\_plans**

 **cross apply sys.dm\_exec\_query\_plan(plan\_handle)**

**where**

 **cast(query\_plan as nvarchar(max)) like '%Sort%'**

 **or cast(query\_plan as nvarchar(max)) like '%Hash Match%'**

If you have detected inefficient query plans and that cause high CPU consumption, run UPDATE STATISTICS on the tables involved in the query and check to see if the problem persists. Then, gather the data and report the problem to PerformancePoint Planning support.

If your system has excessive compiles and recompiles, it could result in a CPU-bound performance problem on the system.

You can run the following DMV queries to find out excessive compiles/recompiles.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select \* from sys.dm\_exec\_query\_optimizer\_info**

**where**

**counter = 'optimizations'**

**or counter = 'elapsed time'**

The following sample query gives you the top 25 stored procedures that have been recompiled. The plan\_generation\_num indicates the number of times the query has recompiled.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select top 25**

**sql\_text.text,**

**sql\_handle,**

**plan\_generation\_num,**

**execution\_count,**

**dbid,**

**objectid**

**from sys.dm\_exec\_query\_stats a**

**cross apply sys.dm\_exec\_sql\_text(sql\_handle) as sql\_text**

**where plan\_generation\_num > 1**

**order by plan\_generation\_num desc**

If you have detected excessive compilation or recompilation, gather as much data as you can and report it to Planning support.

**Memory bottlenecks**

Before you start memory pressure detection and investigation, make sure you have enabled the advanced options in SQL Server. Run the following query on the master database to turn on this option first.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

sp\_configure 'show advanced options'

go

sp\_configure 'show advanced options', 1

go

reconfigure

go

Run the following query to check memory-related configuration options first.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

sp\_configure 'awe\_enabled'

go

sp\_configure 'min server memory'

go

sp\_configure 'max server memory'

go

sp\_configure 'min memory per query'

go

sp\_configure 'query wait'

go

Run the following DMV query to see the CPU, scheduler memory, and buffer pool information.

 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select**

**cpu\_count,**

**hyperthread\_ratio,**

**scheduler\_count,**

**physical\_memory\_in\_bytes / 1024 / 1024 as physical\_memory\_mb,**

**virtual\_memory\_in\_bytes / 1024 / 1024 as virtual\_memory\_mb,**

**bpool\_committed \* 8 / 1024 as bpool\_committed\_mb,**

**bpool\_commit\_target \* 8 / 1024 as bpool\_target\_mb,**

**bpool\_visible \* 8 / 1024 as bpool\_visible\_mb**

**from sys.dm\_os\_sys\_info**

**I/O bottlenecks**

Identify I/O bottlenecks by examining the latch waits. Run the following DMV query to find I/O latch wait statistics.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select wait\_type, waiting\_tasks\_count, wait\_time\_ms, signal\_wait\_time\_ms, wait\_time\_ms / waiting\_tasks\_count**

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

**where wait\_type like 'PAGEIOLATCH%' and waiting\_tasks\_count > 0**

**order by wait\_type**

Identify an I/O problem if your waiting\_task\_counts and wait\_time\_ms change significantly from what you see normally. It is important to get a baseline of performance counters and key DMV query outputs when SQL Server is running smoothly.

These wait\_types can indicate whether your I/O subsystem is experiencing a bottleneck.

Use the following DMV query to find currently pending I/O requests. Execute this query periodically to check the health of I/O subsystem and to isolate physical disk(s) that are involved in the I/O bottlenecks.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

**select**

 **database\_id,**

 **file\_id,**

 **io\_stall,**

 **io\_pending\_ms\_ticks,**

 **scheduler\_address**

**from sys.dm\_io\_virtual\_file\_stats(NULL, NULL)t1,**

 **sys.dm\_io\_pending\_io\_requests as t2**

**where t1.file\_handle = t2.io\_handle**

The query usually returns nothing in the normal situation. You need to investigate further if this query returns some rows.

You can also find I/O bound queries by executing the following DMV query.

**select top 5 (total\_logical\_reads/execution\_count) as avg\_logical\_reads,**

 **(total\_logical\_writes/execution\_count) as avg\_logical\_writes,**

 **(total\_physical\_reads/execution\_count) as avg\_physical\_reads,**

 **Execution\_count, statement\_start\_offset, p.query\_plan, q.text**

**from sys.dm\_exec\_query\_stats**

 **cross apply sys.dm\_exec\_query\_plan(plan\_handle) p**

 **cross apply sys.dm\_exec\_sql\_text(plan\_handle) as q**

**order by (total\_logical\_reads + total\_logical\_writes)/execution\_count Desc**

The following DMV query can be used to find which batches/requests are generating the most I/O. A DMV query like the following can be used to find the top five requests that generate the most I/Os. Tuning those queries will improve the system performance.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

**select top 5**

 **(total\_logical\_reads/execution\_count) as avg\_logical\_reads,**

 **(total\_logical\_writes/execution\_count) as avg\_logical\_writes,**

 **(total\_physical\_reads/execution\_count) as avg\_phys\_reads,**

 **Execution\_count,**

 **statement\_start\_offset as stmt\_start\_offset,**

 **sql\_handle,**

 **plan\_handle**

**from sys.dm\_exec\_query\_stats**

**order by (total\_logical\_reads + total\_logical\_writes) Desc**

**Blocking**

**Run the following query to determine the blocking sessions**.

**select blocking\_session\_id, wait\_duration\_ms, session\_id from**

**sys.dm\_os\_waiting\_tasks**

**where blocking\_session\_id is not null**

Use this call to find out which SQL is returned by the blocking\_session\_id. For example, if the blocking\_session\_id is 87, run this query to get the SQL.

dbcc INPUTBUFFER(87)

The following query shows SQL waits analysis and top 10 resources waited on.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

 **select top 10 \***

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

**--where wait\_type not in ('CLR\_SEMAPHORE','LAZYWRITER\_SLEEP','RESOURCE\_QUEUE','SLEEP\_TASK','SLEEP\_SYSTEMTASK','WAITFOR')**

**order by wait\_time\_ms desc**

To find out which spid is blocking another spid, create the following stored procedure in your database and then execute the stored procedure. This stored procedure reports the blocking situation. Type sp\_who to find out @spid; @spid is optional parameter.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

create proc dbo.sp\_block (@spid bigint=NULL)

as

select

 t1.resource\_type,

 'database'=db\_name(resource\_database\_id),

 'blk object' = t1.resource\_associated\_entity\_id,

 t1.request\_mode,

 t1.request\_session\_id,

 t2.blocking\_session\_id

from

 sys.dm\_tran\_locks as t1,

 sys.dm\_os\_waiting\_tasks as t2

where

 t1.lock\_owner\_address = t2.resource\_address and

 t1.request\_session\_id = isnull(@spid,t1.request\_session\_id)

The following are examples of using this stored procedure.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&

exec sp\_block

exec sp\_block @spid = 7