# SQL Server: Optimizing Ad Hoc Statement Performance

Module 5: Plan Cache Pollution

Kimberly L. Tripp
Kimberly@SQLskills.com
http://www.SQLskills.com/blogs/Kimberly

pluralsight
hardcore developer training

# Course Overview

- **Statement execution methods**
- **Estimates and selectivity**
- **Statement caching**
- **Plan cache pollution**
  - Ad hoc plan cache pollution defined
  - Plan cache usage and limits
  - Plan cache stores
  - Verifying the state of plan cache
  - Balancing plan cache pollution, CPU, and parameter-sniffing problems
- **Statement execution summary**

# Ad Hoc Plan Cache Pollution Defined

- **BECAUSE every statement goes into the ad hoc plan cache for exact textual matching**
  - Plans start to fill up cache
- **Plans are more expensive to put into cache than data, therefore SQL Server values them a bit more highly (sometimes stealing pages from the buffer pool to give to the plan cache)**
- **Plans – even if only used once – can sit in the cache for quite some time**

# Plan Cache Usage and Limits

- **The "plan cache" is also known as the "procedure cache"**
- **Uses "stolen" pages from the buffer pool (data pages)**
- **View cached plans: *sys.dm_exec_cached_plans***
- **Plan cache memory limits:**
  - SQL Server 2005 SP2 and higher
    - 75% of visible target memory from 0-4GB
      + 10% of visible target memory from 4GB-64GB
      + 5% of visible target memory > 64GB
  - SQL Server 2005 RTM/SP1
    - 75% of visible target memory from 0-8GB
      + 50% of visible target memory from 8GB-64GB
      + 25% of visible target memory > 64GB
  - SQL Server 2000
    - 4GB upper cap on the plan cache

| Current Versions | |
|---|---|
| Memory | Plan Cache |
| 4GB | 3.0GB |
| 8GB | 3.5GB |
| 16GB | 4.2GB |
| 32GB | 5.8GB |
| 64GB | 9.0GB |
| 128GB | 12.2GB |
| 256GB | 30.0GB |

# Plan Cache Stores

- *CACHESTORE_OBJCP* = "Object Plans"
  - Stored procedures, functions, triggers…
  - Generally, it's desirable to have a higher value especially when reused
- *CACHESTORE_SQLCP* = "SQL Plans"
  - Ad hoc SQL statements (including parameterized ones)
  - Prepared statements
  - OK when highly reused, but often not reused
- **Lots of other cachestores but "SQL Plans" is what we're focused on and it can be the most problematic**

```
SELECT [mc].*
FROM [sys].[dm_os_memory_clerks] AS [mc]
ORDER BY [mc].[pages_kb] DESC
```

# Verifying State of Plan Cache

```
-- How much of your cache is allocated to single-use plans?
SELECT [Cache Type] = [cp].[objtype]
 , [Total Plans] = COUNT_BIG (*)
 , [Total MBs] = SUM (CAST ([cp].[size_in_bytes]
       AS DECIMAL (18, 2))) / 1024 / 1024
 , [Avg Use Count] = AVG ([cp].[usecounts])
 , [Total MBs - USE Count 1] = SUM (CAST ((CASE
       WHEN [cp].[usecounts] = 1 THEN [cp].[size_in_bytes]
       ELSE 0 END) AS DECIMAL (18, 2))) / 1024 / 1024
 , [Total Plans - USE Count 1]
     = SUM (CASE WHEN [cp].[usecounts] = 1 THEN 1 ELSE 0 END)
 , [Percent Wasted] = [Total MBs - USE Count 1]/[Total MBs]*100
FROM [sys].[dm_exec_cached_plans] AS [cp]
GROUP BY [cp].[objtype]
ORDER BY [Total MBs - USE Count 1] DESC;
```

# Balancing Plan Cache Pollution, CPU, and PSP (1)

- **Server setting: *optimize for ad hoc workloads***
  - On first execution, only the *query_hash* will go into cache
  - On second execution (if), the plan will be placed in cache
- **Create a single and more consistent plan with covering indexes – might make the plans more stable!**
  - SQL Server will pick up SOME stable statements IF and ONLY IF they're SAFE
    - (Back to the rules from the whitepaper)
    - Note: this only reduces compilation costs (e.g. CPU) but it does not reduce plan cache pollution because every ad hoc statement still goes into the ad hoc cache
  - If you create a bunch of stable plans that SQL doesn't see as SAFE but they essentially are (one *query_plan_hash* for each *query_hash*) then you can consider the database setting: forced parameterization
    - If you're finding A LOT of single-use statements that have the same query_hash and are executed frequently but with only one *query_plan_hash* then this is ideal!
  - But, remember, ad hoc statements are always placed in the ad hoc plan cache so you still need *optimize for ad hoc workloads*…set that FIRST!

# Balancing Plan Cache Pollution, CPU, and PSP (2)

- Analyze the plan cache for the number of query plans per *query_hash* (as well as the number of executions)

```
SELECT [query_hash]
    , [# of Plans] = COUNT (DISTINCT [query_plan_hash])
    , [Execution Total]= SUM ([execution_count])
...
```

- Two primary scenarios to consider

Scenario 1

| query_hash | # of Plans | # of Executions |
|---|---|---|
| 0x04BB791B589774AD | 1 | 6456456 |
| 0x1706E9EC3049A95B | 6 | 276543 |
| 0x5BD9FF487079B335 | 1 | 124345 |
| 0x6604520C5200ABC0 | 1 | 78905 |
| 0x77BA5A89C7EBE605 | 1 | 14342 |
| 0xA078B4BC8768A9A6 | 1 | 4567 |
| 0xB81E270A58A79D16 | 1 | 6 |

Mostly stable plans
(only 1 plan per *query_hash*)

Scenario 2

| query_hash | # of Plans | # of Executions |
|---|---|---|
| 0x04BB791B589774AD | 34 | 6456456 |
| 0x1706E9EC3049A95B | 1 | 276543 |
| 0x5BD9FF487079B335 | 8 | 124345 |
| 0x6604520C5200ABC0 | 3 | 78905 |
| 0x77BA5A89C7EBE605 | 2 | 14342 |
| 0xA078B4BC8768A9A6 | 9 | 4567 |
| 0xB81E270A58A79D16 | 24 | 6 |

Mostly UNstable plans
(multiple plans per *query_hash*)

# Verifying State of Plan Cache (2)

```
-- How much is each query_hash using and how many plans?
SELECT [qs].[query_hash]
    , [Distinct Plan Count]
        = COUNT (DISTINCT [qs].[query_plan_hash])
    , [Execution Total]
        = SUM ([qs].[execution_count])
    , [Total MB]
        = SUM (cp.size_in_bytes) / 1024.0 / 1024.0
FROM [sys].[dm_exec_query_stats] AS [qs]
   INNER JOIN [sys].[dm_exec_cached_plans] AS [cp]
      ON [qs].[plan_handle] = [cp].[plan_handle]
GROUP BY [qs].[query_hash]
ORDER BY [Execution Total] DESC;
GO
```

# Balancing Plan Cache Pollution, CPU, and PSP (3)

- **Scenario 1: Consider changing parameterization to *FORCED***
- **Then, for all of the statements that have more than one plan, you'll need to make sure that they are not forced**
  - Can you change the code?
    - Change the ad hoc statements to be executed with a recompilation option (see my next course: *Optimizing Stored Procedure Performance*)
  - What if you can't change the code?
    - Use "templatized" plan guides to take the few statements that are NOT stable and make *SIMPLE* (recompiled)
    - Use *sp_get_query_template* to get the templatized query and parameters

```
EXEC sp_create_plan_guide N'PlanGuideName'
     , @StatementWithUnStableQueryPlan
     , N'TEMPLATE'
     , NULL
     , @Parameters
     , N'OPTION(PARAMETERIZATION SIMPLE)';
```

# Balancing Plan Cache Pollution, CPU, and PSP (4)

- **Scenario 2: Keep the default parameterization mode: *SIMPLE***
- **Then, for all of the statements that have only one plan, force them individually**
  - Can you change the code?
    - Change the ad hoc statements to be executed with *sp_executesql* or stored procedures (see my next course: *Optimizing Stored Procedure Performance*)
  - What if you can't change the code?
    - Use "templatized" plan guides to take the few statements that are stable and make them *FORCED* (reduced compilation/CPU)
    - Use *sp_get_query_template* to get the templatized query and parameters

```
EXEC sp_create_plan_guide N'PlanGuideName'
     , @StatementWithStableQueryPlan
     , N'TEMPLATE'
     , NULL
     , @Parameters
     , N'OPTION(PARAMETERIZATION FORCED)';
```

# Balancing Plan Cache Pollution, CPU, and PSP (5)

- **Start by configuring '*optimize for ad hoc workloads*'**
  - This helps, but it's not always enough
- **Additionally, consider a SQL Agent Job to periodically wake up and analyze the plan cache for single-use plans**
  - If more than a certain percentage of memory is being wasted…
  - If more than a certain amount of memory (GB) is being wasted…
- **Clearing the plan cache**
  - `DBCC FREEPROCCACHE [ ( { plan_handle | sql_handle | pool_name } ) ] [WITH NO_INFOMSGS]`
    - All plans: *DBCC FREEPROCCACHE*
    - A single plan: *DBCC FREEPROCCACHE (plan_handle)*
    - All plans for a single database: *DBCC FLUSHPROCINDB (DBID)*
  - `DBCC FREESYSTEMCACHE ( 'ALL' [, pool_name ] ) [WITH {[MARK_IN_USE_FOR_REMOVAL], [NO_INFOMSGS]}]`
  - All ad hoc and prepared plans: `DBCC FREESYSTEMCACHE (N'SQL Plans')`

# Alternatives to Ad Hoc Statements

- **When you can change the code and when the plan is stable:**
  - Use forced statement caching
    - *sp_executesql*
    - Stored procedures (more options/control)
  - Only one plan goes into the cache
    - Saves cache (an ad hoc plan is NOT saved, only the prepared statement)
    - Saves compilation time
  - This SOUNDS perfect…
  - When used with a statement whose plans are inconsistent – problems from parameter sniffing can be worse
- **When you can't change the code then you need to determine:**
  - Can you tune an ad hoc statement to make it safe (or, at least stable)?
  - Can you benefits from changing to forced parameterization? (test!)
    - Are there any statements that require a plan guide template to force them to the opposite of the database option chosen?

# Summary: Plan Cache Pollution

- **My general recommendation:**
  - Reduce plan cache pollution
  - Better to use the cache allocated by reusing plans for stable plans
- **How can you better reuse plans?**
  - Better tuning to create safe or more stable plans
  - Use *sp_executesql* to force those that are stable
  - Use stored procedures
- **Reusing plans isn't always a good thing:**
  - Recompilation is not always bad
  - Compiled plans aren't always perfect for all executions
- **Ad hoc statements are going to happen in every environment**
  - Must do some form of maintenance
- **Plan cache pollution is also very likely – incredibly rare that we aren't setting the configuration option AND setting up regular clearing of the 'SQL Plans' cache**