**Procedure Cache Tuning/Sizing - From 1GByte to 768KBytes - Increase the size of usable Data Cache**

Two of the biggest memory eaters of SQL Server cache are Data and Procedure Cache, Data cache offers us the advantage of hopefully not having to go to the disks to get our data because we already did that and its cached. One of the problems on real systems is that the procedure cache can grow really big and take data cache away resulting in you having to go to the disk more, one example is a client of mine I do support for, they are on SQL 2000 standard so that limits our memory to 1.7GB, the procedure cache alone was taking in the region of 700Mbytes. You can’t directly control how big the Procedure Cache grows but by good application design we can influence it.

How do we influence Procedure Cache growth? The answer is simple; we use Parameterisation and/or using Stored Procedures.

Let’s look at two examples; they need to be run on a SQL Server that nobody else is using so as not to influence the results.

dbcc freeproccache

go

declare @t datetime

set @t = getdate()

set nocount on

declare @i int

declare @f int

declare @sql nvarchar(max)

set @i = 0

while @i <= 100000

begin

set @sql = 'select @f = count(\*) from sys.objects where object\_id = ' + cast( @i as varchar(10) )

exec sp\_executesql @sql, N'@f int output', @f output

set @i = @i + 1

end

print datediff( second, @t, current\_timestamp )

Essentially every query I execute is different because I’m building my SQL string using constants rather than using parameters.

Let’s take our memory readings:

dbcc memorystatus

CACHESTORE\_SQLCP (Total) KB

---------------------------- --------------------

SinglePage Allocator 1178608

Procedure Cache Value

------------------------------ -----------

TotalProcs 18286

TotalPages 148794

InUsePages 41

Buffer Distribution Buffers

------------------------------ -----------

Stolen 2592  
Free 7191  
Cached 150151  
Database (clean) 101300  
Database (dirty) 910  
I/O 0  
Latched 0

In summary, the script took 371 seconds to run, the procedure cache is 1.1GBytes, there are 18,286 plans in cache, and there are 7191 pages (57Mbytes) free for the data cache.

Let’s do the same test (logically) but parameterise it.

dbcc freeproccache

go

declare @t datetime

set @t = getdate()

set nocount on

declare @i int

declare @f int

declare @sql nvarchar(max)

set @i = 0

set @sql = 'select @f = count(\*) from sys.objects where object\_id = @i'

while @i <= 100000

begin

exec sp\_executesql @sql, N'@f int output, @i int', @f output, @i

set @i = @i + 1

end

print datediff( second, @t, current\_timestamp )

go

Use DBCC MEMORYSTATUS to take the readings again...

CACHESTORE\_SQLCP (Total) KB

------------------------ --------------------

SinglePage Allocator 768

Buffer Distribution Buffers

------------------------------ -----------

Stolen 938

Free 157450

Cached 1546

Database (clean) 101300

Database (dirty) 910

I/O 0

Latched 0

Procedure Cache Value

------------------------------ -----------

TotalProcs 15

TotalPages 183

InUsePages 13

In summary, the script took 5 seconds to run, the procedure cache is 768Kbytes, there are 15 plans in cache, and there are 157450 pages (1.2GBytes) free for the data cache.

Surprised? It’s one of the reasons DBA’s keep bleating on about using stored procedures, it forces the mindset to use the procedure cache more effectively. By parameterising, but preferably using stored procedures we get plan reuse which means a) less compiles thereby reducing CPU load, b) more available pages for the data cache thereby reducing physical disk IO and c) DBA’s are always right ;).

Check out your own procedure cache size and see how big it is in proportion to the data cache.

How do you find these non-parameterised queries? You can look in master..syscacheobjects.

select \*

from master..syscacheobjects

For our last test, the parameterised one you will find a single Compiled Plan that has a usecounts of 100,001; whereas you check the first version you will see thousands of Compiled Adhoc Plans with a usecounts of 1.

Another method is to use profiler and see what queries are embedded or dynamically built and not parameterised.

We are stuck with this design decision at the moment, you can’t control the size of the procedure cache, you can only influence it with application design. In a real system it's not going to be clear cut like my example, in reality the size of the data cache and procedure cache will depend on what is going on; for instance, a lot of those one off plans would have been kicked out of cache and their pages reused for data, but, that all takes resource!