**SQL Server memory configuration**

One of the things that I frequently come across when reviewing SQL Server installations is just how many of them have not been set up with appropriate memory configuration settings, or, as in many cases, not set up in the way the administrators of the system had assumed they were; usually the dbas thought the system was set up to use all e.g. 8GB of RAM, but no changes had been made to the OS or SQL Server configuration, so their (32-bit) SQL Server would only be accessing 2 GB, and reporting that it was using 1.6 GB.

The problem is due in part to the fact that on 32-bit systems configuration changes usually have to be made both in SQL Server and at the OS level, and in part to the sprawl of documentation available on configuring SQL Server’s memory settings, as opposed to a single jumping off point which runs through all the settings and considerations that need to be made.  
Add to that the black art of establishing exactly how much memory SQL Server *is* using (most of the obvious options will only show how much memory the buffer pool is using) and it’s easy to see why it’s such a problem area.

In this post I’ll attempt to clear some of the smog and provide what I hope will be one document which answers most of the questions that arise about configuring SQL Server’s memory usage.  
This discussion will cover configuring memory for SQL Server 2000, SQL Server 2005 and SQL Server 2008 (with the exception of the Resource Governor). This blog assumes an [edition of SQL Server](http://msdn.microsoft.com/en-us/library/ms143685.aspx) that is not internally limited in its memory usage.

**32-bit or 64-bit?**

There’s a big difference between the memory configuration settings between 64-bit SQL Server and 32-bit SQL Server, so it’s not possible to start a discussion about SQL Server’s memory management without clarifying whether we are dealing with 32-bit versions or 64-bit versions of the product, as this is key to how much memory SQL Server can address, and (almost as importantly) how it addresses that memory.

**Configuring 32-bit SQL Server**

Until fairly recently 32-bit software was ubiquitous. The server Windows operating systems were 32-bit, your desktop, usually Windows XP was 32-bit. Therefore, I’ll be focusing a fair bit on 32-bit SQL Server as this is what requires the most configuration, and also where most of the confusion lies.

So, here goes.

The amount of memory a 32-bit process can access is 2^32 or 4294967296 bytes, or 4 GB.  
On a 32-bit Windows OS this 4 GB of memory is not all addressable by a single process. Instead, it’s partitioned by the OS into two address spaces. 2 GB is kept by the OS (commonly referred to as the kernel) and the remaining 2 GB is the user mode address space, or the area each application (process) will have access to. Whilst each user mode process gets 2 GB as its addressable memory range, the kernel mode area is shared between all processes. SQL Server runs in the user mode address space and is bound by default to the 2 GB of memory limit on a 32-bit OS.  
This directly addressable memory will hereon be referred to by what it is more commonly known as, the virtual address space or [VAS](http://msdn.microsoft.com/en-us/library/aa366912.aspx).

SQL Server’s default out-of-the-box memory limit is deliberately set to a very high value of 2147483647 , which basically means all available memory, but as you should now know, there’s no way it can actually use anywhere near that much memory, particularly on a 32-bit platform.

64-bit operating systems have a far far bigger address space open to them; 8 TB to be exact. Before you run off to your calculator to evaluate 2^64, the answer won’t be 8TB, but 8TB is what each user mode application gets due to current hardware and OS limitations. The kernel also gets 8TB and this kernel address space is shared by all processes, just as in 32-bit Windows.

Having said all that, I should point out that no current MS Windows OS can address more than 2 TB.

What this means for 64-bit SQL Server is that out of the box, it can address all the memory on a server without any special configuration changes either within SQL Server or at the OS level. The only thing you need to look at is a cap on its memory usage; capping memory usage is covered in the [‘max server memory’ and ‘min server memory’](http://www.eraofdata.com/blog/sql-server-memory-configuration/#maxservermemory) section which is further down.

**To /3GB or not to /3GB**

So, as 32-bit applications are natively restricted to a 2 GB VAS, OS configuration tweaks are required to allow access to more than 2 GB, and these are covered next.

The first modification is one that, rather ironically, should be used as a last resort. Ideally, it should be used on the advice of Microsoft Support (PSS).  
I’m choosing to get it out of the way now because the /3GB setting is probably the most well known and most misused.  
/3GB allows a 32-bit process to increase its VAS to 3 GB by taking away 1 GB of address space from the kernel, and this is why it’s a last resort; there’s no such thing as a free lunch as the removal of 1 GB of addressable memory from the OS can introduce instability. More on that shortly.

To allow a 32-bit process to gain a 3 GB VAS you have to add the [/3GB](http://msdn.microsoft.com/en-us/library/ms791558.aspx) switch to the Windows boot.ini file.

As I stated, this can introduce system instability by starving the OS of System Page Table Entries (PTEs). A discussion about PTEs is out of the scope of this blog, but its effects can be dramatic and cause blue-screens. The good news is that this mainly affected Windows 2000 so you should be fine if you’re on a later Windows version.

If you’re still looking after a legacy system, there is some scope for manoeuvre here, by adding the [/USERVA](http://support.microsoft.com/kb/316739) switch to the boot.ini it is possible to reduce the VAS increase from a straight 3 GB to a lower user-defined amount which will give the OS room to breathe, and thus resolve any instability issues.

The main reason you will be advised by PSS to use /3GB is if you are suffering VAS starvation issues, such as a bloated procedure cache as it can only reside in VAS memory (also see the next section on **MemToLeave**) because 32-bit version of SQL Server only allow database pages to reside in the part of the SQL Server memory cache (called the buffer pool) that is utilising awe enabled memory.

**MemToLeave**

(EDIT:[20110630] The correct terminology for this is [Virtual Address Space Reservation](http://sqlblog.com/blogs/jonathan_kehayias/archive/2009/07/07/understanding-the-vas-reservation-aka-memtoleave-in-sql-server.aspx).)  
Because of the inherent address space limitations of a 32-bit process, a certain amount of memory has to be set aside by SQL Server on startup that SQL Server uses for overheads. This memory is set aside in case it all gets used by the buffer pool.

COM objects, extended stored procs, third party backup solutions, some anti-virus apps, memory allocations exceeding 8K and the memory allocated to the threads SQL Server creates to service e.g. user connections come from a section of memory within the VAS but outside the buffer pool which is typically referred to as the MemToLeave area. This is 384 MB by default on an e.g. 2-proc 32-bit SQL. If you want to know more about how it is calculated, check [Jonathan Kehayias’s post](http://sqlblog.com/blogs/jonathan_kehayias/archive/2009/07/07/understanding-the-vas-reservation-aka-memtoleave-in-sql-server.aspx) covering this.

0.5 MB is the default thread stack size for a thread on 32-bit Windows. 64-bit Windows has a default stack size of 2 MB or 4 MB depending on which 64-bit flavour of Windows you are running (AMD x64 or IA64).  
SQL Server 2005 and beyond uses a [formula](http://msdn.microsoft.com/en-us/library/ms187024.aspx) to calculate the max worker threads setting which affects the size of the MemToLeave area.  
There is a SQL Server startup parameter (**-g**) which can be used to increase the MemToLeave area, but again, only do this if advised by PSS (it’s ignored on 64-bit as MemToLeave or VAS reservation won’t be an issue on that architecture) as this will reduce the maximum amount of memory the buffer pool can therefore use.

**4 GB of RAM and beyond**

So, we know 32-bit SQL Server can use 2 GB out of the box, and up to 3 GB (with an OS tweak that is best avoided, if at all possible).  
However, 32-bit SQL Server can benefit from much more memory than 3 GB with the help of OS and SQL Server configuration modifications which will be covered next.

**/PAE**

To address more than 4 GB of RAM on 32-bit Windows, the OS needs to have the [/PAE](http://msdn.microsoft.com/en-us/library/ms791485.aspx) switch added to the boot.ini file, although if your system supports hot-swappable memory you won’t need to add this as Windows should automatically be able to see the additional memory. If you’re not sure, take a look at how much memory the OS can see via System properties; if you have more than 4 GB installed and the OS is only showing 4 GB, review your boot.ini settings. I’m not mentioning specific Windows versions because the /PAE switch applies to all current 32-bit versions of Windows.  
(EDIT:[20110630] For Windows Server 2008 you have to run ‘BCDEDIT /SET PAE FORCEENABLE’ from a CMD prompt running under administrator privileges).

***Both* /PAE and /3GB**

Some systems have both /3GB and /PAE enabled.  
This is fine as long as the system does not have more than 16 GB of RAM. Add any more memory and [Windows will not recognise it](http://support.microsoft.com/kb/283037) because of the overhead required to manage the additional memory.

**Clusters**

No special configuration settings regarding memory settings are required for a cluster, but I thought I better mention clusters specifically because you won’t believe how many installations there are out there where there are different settings on different nodes within the same cluster.  
So, make sure any OS setting changes like /3GB or /PAE are consistently applied across *all* nodes.

**Enable AWE**

After configuring the OS, you’ll need to configure SQL Server by enabling AWE (Address Windowing Extensions). AWE is in essence a technique for ‘paging’ in sections of memory beyond the default addressable range.  
AWE can be enabled in SQL Server using Query Analyzer/SQL Server Management Studio (SSMS) via the following statements:  
  
sp\_configure 'show advanced options', 1  
RECONFIGURE  
GO  
sp\_configure 'awe enabled', 1  
RECONFIGURE  
GO  
  
AWE enablement is not a dynamic option and will require a SQL Server restart, so before you do that make sure the SQL Server service account has the ‘Lock Pages in Memory’ privilege assigned to it.  
Once AWE has been enabled within SQL Server and the ‘Lock Pages in Memory’ privilege has been assigned you should be good to go after a restart.

**‘max server memory’ and ‘min server memory’**

The more memory you give SQL Server, the greater the need to set an upper limit on how much it uses. When you start SQL Server it’ll ramp up its memory usage until it has used up all the memory it can access, which will either be an internal OS limit or a SQL Server configured limit.  
A 32-bit SQL Server instance will therefore grab up to 2 GB if the workload demands it and it is on default settings.  
An awe enabled SQL Server instance will go on using up all the memory on the system if the workload is there and an upper limit on its memory usage is not set.

Setting a limit has the double-benefit of not starving the OS of resources and avoiding ‘Out of memory’ errors which can occur on SQL Server systems that may have a lot of memory. The latter (rather contradictory) situation can arise because SQL Server will try and allocate more memory when it is already at the system limit (if no upper limit has been set via the ‘max server memory’ setting) instead of freeing up memory it is already using.

**Configuring memory for multiple instances**

A third reason to set an upper limit is if you have more than one SQL Server instance installed on a single host, as this will stop the instances competing for memory.  
Allocate a high enough ‘max server memory’ limit to each instance to allow it to do its job without running into memory starvation issues, whilst reserving the bulk of the memory for higher priority instances (if any) *and* the OS.  
This is where benchmarking comes in handy.

To set a max server memory limit of 12 GB via Query Analyzer/SSMS:  
  
sp\_configure 'max server memory', 12288  
RECONFIGURE  
GO

SQL Server ramps up its memory usage because by default it is set to use no memory on startup. This is controlled by the ‘min server memory’ setting. Specifying this to a higher value has the benefit of reserving a set amount of memory for SQL Server from the off which can provide a slight performance benefit, especially on busy systems. It’s actually not uncommon to see ‘min server memory’ and ‘max server memory’ set to the same value, to reserve all of SQL Server’s memory straight away. The downside is SQL Server will take slightly longer to start up than if ‘min server memory’ was set to a low value.

**SQL does not *really* release memory**

This will probably get me into a bit of trouble, as there are KBs that clearly state that SQL Server releases memory when the OS is under pressure.  
True, but only when it’s under a **lot** of pressure (although this is getting better with each version of SQL Server), and by then it’s often too late as the system is usually in such a degraded state by that stage that a restart is necessary.  
That’s why it’s vital to set an upper limit on its memory usage via the ‘max server memory’ setting.  
(Edit [20110627): Speaking from experience, this problem manifested itself most noticeably on SQL Server 2000 and earlier but the problem seems to have disappeared from SQL 2005 onwards. Let me know if you disagree (and send me the evidence!)).

**Memory corruption**

Slightly off-topic, but this is an appropriate place to bring this up.  
Certain builds of [Windows 2000](http://support.microsoft.com/?kbid=838647) and [Windows Server 2003](http://support.microsoft.com/Default.aspx?kbid=834628) contained a potentially serious memory corruption problem which affected SQL Server more than other applications, mainly because there are few applications that run on Windows that can utilise the amount of memory SQL Server does.  
It's difficult to overstate the problems this can cause, so make sure you're on the appropriate Windows Service Packs if you're running SQL Server on a PAE enabled system.  
Another issue that arose in [SQL Server 2000 SP4](http://support.microsoft.com/default.aspx?scid=kb;en-us;899761) was a bug that meant SQL Server only saw half the memory on awe enabled systems, although it was identified quickly and the hotfix for this was placed alongside the SP4 download.

**32-bit SQL Server on 64-bit Windows**

If you 32-bit SQL Server on 64-bit Windows the SQL Server process can access the entire 4 GB VAS.

**Checking SQL Server's memory usage**

This is another area where there is lot of confusion, so below is a run-through of the most common methods for confirming SQL Server's memory usage.

**Ignore Task Manager**

If you have an awe enabled SQL Server instance, do not rely on Task Manager to display memory usage as it does not show the AWE memory a process is using, so the memory usage figure it presents for the SQL Server process (sqlservr.exe) will be incorrect.

**DBCC MEMORYSTATUS**

Running the above command outputs the memory usage of SQL Server including how that memory is allocated, so unless you need to know how and where that memory is being used, the output it generates can be a bit bewildering. The important bits of this output pertaining to SQL Server's total memory usage are as follows:  
  
Buffer Counts Buffers  
------------------------------ --------------------  
Committed 3872  
Target 65536  
Hashed 2485  
Stolen Potential 60972  
External Reservation 0  
Min Free 64  
Visible 65536  
Available Paging File 702099  
  
The key figures in the above output are committed, target and hashed.  
Committed is the amount of memory in use by the buffer pool and includes AWE pages.  
Target is how big SQL Server wants the buffer to grow, so you can infer from this whether SQL Server wants more memory or is releasing memory.  
There's an excellent KB on interpreting all the output [INF: Using DBCC MEMORYSTATUS to Monitor SQL Server Memory Usage](http://support.microsoft.com/?id=271624) for SQL Server 2000 and [How to use the DBCC MEMORYSTATUS command to monitor memory usage on SQL Server 2005](http://support.microsoft.com/kb/907877/en-us).  
Edit (05/02/09): Remember the buffer count numbers refer to pages of memory which are 8K in SQL Server

**System Monitor (perfmon)**

Perfect way to get a quick reference on exactly how much memory SQL Server is using at that moment. Start System Monitor and add the SQL Server: Memory Manager: Total Server Memory (KB) counter.

Replace "SQL Server" with MSSQL$ and the name of the named instance if it's not a default instance, e.g. MSSQL$INSTANCE1.

**'Total' memory usage**

When trying to establish exactly how much memory SQL Server is using it's not just the buffer pool memory you have look at, but the MemToLeave area as well. The key point to bear in mind here is that it's not only SQL Server that can make allocations from this latter area of memory but third party processes as well, which can make it impossible to precisely account for SQL Server's absolute memory usage, contrary to some myths out there about calculating SQL Server's memory usage via e.g. DBCC MEMORYSTATUS, as such methods can only account for SQL Server's *own* memory allocations and *not* allocations by foreign processes.

Edit [20110627]: Soft NUMA section removed.

**64-bit**

I mentioned at the start of this post that all you have to worry about for 64-bit SQL Server is setting a max memory limit as SQL Server can access all the memory current Windows operating systems can support, and 8 TB in total. That’s mostly true, with the exception of a certain privilege that the SQL Server service account needs, and that’s the ‘Lock Pages in Memory’ privilege.  
This privilege is vital as it prevents the OS from paging out SQL Server memory to the swap file.  
With the introduction of SQL Server 2005, this right was restricted on 64-bit Windows to [only take effect on Enterprise Editions of SQL Server](http://support.microsoft.com/kb/918483/en-us), so if you’re wondering why your huge new multi-gigabyte multi-core 64-bit system is paging like crazy, this might be why. [Edit: This has finally been reversed for both SQL Server 2005 and SQL Server 2008 Standard Editions: <http://blogs.msdn.com/psssql/archive/2009/04/24/sql-server-locked-pages-and-standard-sku.aspx>  
Whilst we’re on the subject of paging on 64-bit SQL Server systems, take a look at the following KB:  
[How to reduce paging of buffer pool memory in the 64-bit version of SQL Server 2005](http://support.microsoft.com/default.aspx/kb/918483) which covers issues a number of issues that cause SQL Server’s (Standard or Enterprise editions) memory to be paged out.

**In summary…**

The table below describes how much memory SQL Server can use, and assumes an edition of SQL Server that has no internal limitations as to how much memory it can use, e.g. Express and Workgroup editions are limited to 1 GB and 3GB respectively.

|  |  |  |  |
| --- | --- | --- | --- |
| SQL Server type | Installed physical memory | | |
| Up to 4GB | | More than 4GB (/PAE enabled 1) |
| 32-bit SQL Server | Default memory usage | With /3GB 2 | All available RAM3 |
| 2 GB | 3 GB |
| 64-bit SQL Server | All available RAM 3 | | |
| 1 Not all 32-bit systems now need to have /PAE explicitly set in boot.ini for the OS to see more than 4 GB of RAM 2. Assuming /USERVA switch has not been used to tune memory usage to between 2 GB and 3 GB 3. Assuming 'max server memory' is left on defaults, otherwise SQL Server will use no more memory than that stipulated by the 'max server memory'setting. | | | |

When I started this post I wanted to keep it as short and succinct as possible, but there’s a lot more to configuring SQL Server’s memory usage than simply setting a ‘max server memory’ limit. Configuring SQL Server’s memory settings can be quite a complex undertaking, especially in a 32-bit environment. It’s not easy to cover all the pertinent points without branching off and describing the different areas of its memory architecture, although I’ve tried to provide the relevant information without going into too much detail.  
Hopefully, this blog has clarified how to configure SQL Server’s memory usage and provided enough information to answer most memory configuration related questions, although, as you might have guessed, there’s no black-and-white way of precisely determining SQL Server’s memory usage as there are so many external processes that can make allocations from within SQL Server’s address space…

Thanks also to Jonathan Keheyias for additional information and comments on this post. Please read [his post](http://sqlblog.com/blogs/jonathan_kehayias/archive/2009/07/07/understanding-the-vas-reservation-aka-memtoleave-in-sql-server.aspx) on this topic if you want to delve a bit deeper.

**Useful links**

[How It Works: DBCC MemoryStatus Locked Pages Allocated and SinglePageAllocator Values](http://blogs.msdn.com/psssql/archive/2009/05/15/how-it-works-dbcc-memorystatus-locked-pages-allocated-and-singlepageallocator-values.aspx)

[INF: Using DBCC MEMORYSTATUS to Monitor SQL Server Memory Usage](http://support.microsoft.com/kb/271624)

[MemToLeave and 64-bit SQL Server](http://blogs.msdn.com/psssql/archive/2009/08/26/come-on-64bit-so-we-can-leave-the-mem.aspx)