

SQLskills Immersion Event IE1: Internals and Performance

Module 2: Data File Internals and Maintenance

Paul S. Randal
Paul@SQLskills.com



Overview

- Physical layout considerations
- Allocation algorithms
- Instant initialization
- Auto-grow
- To shrink or not to shrink?
- Tempdb

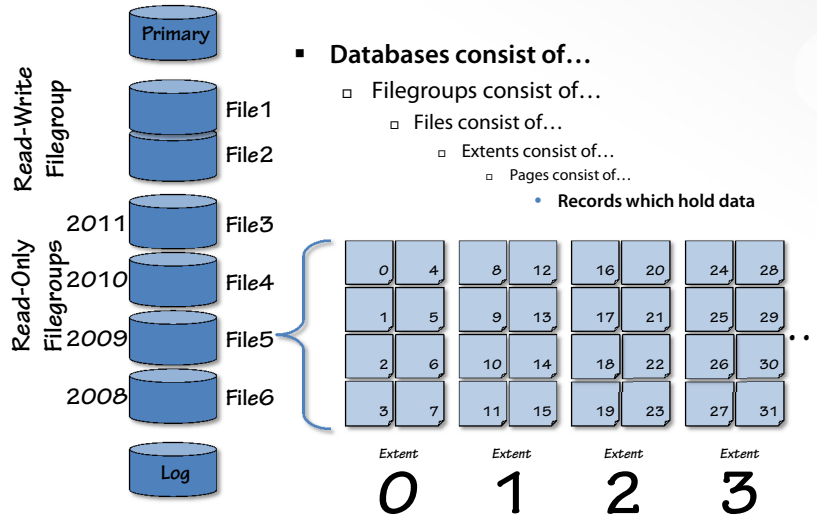


2

© SQLskills, All rights reserved.
<http://www.SQLskills.com>



Database Components



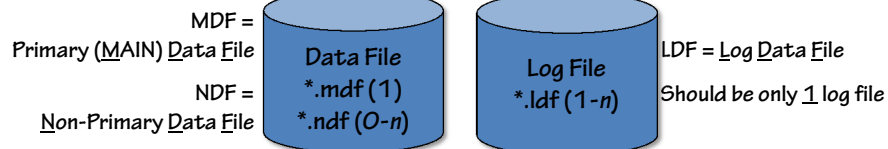
Database Structure

Up to 32,767 databases per instance



Up to 32,767 files per database
(Total is for both data and log files)

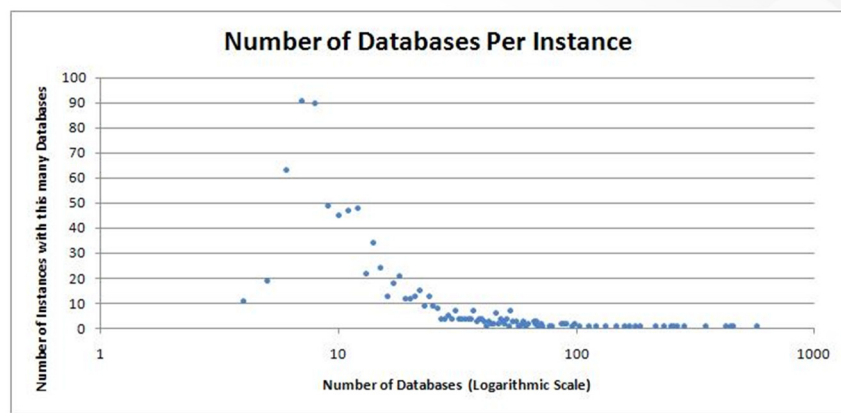
Minimum of 2 files – one for data, one for log



Consolidation Issues

- **Too many databases per instance can lead to:**
 - Performance problems (lack of resources)
 - Constantly fighting for buffer pool resources
 - I/O subsystem placement difficulties
 - Maintenance problems (lack of resources)
 - Constant background I/O and CPU load from maintenance, DBCC CHECKDB, backups
- **Too many database files can lead to long startup time**
 - Every file has to be opened and read to get the file header page

Databases Per Instance

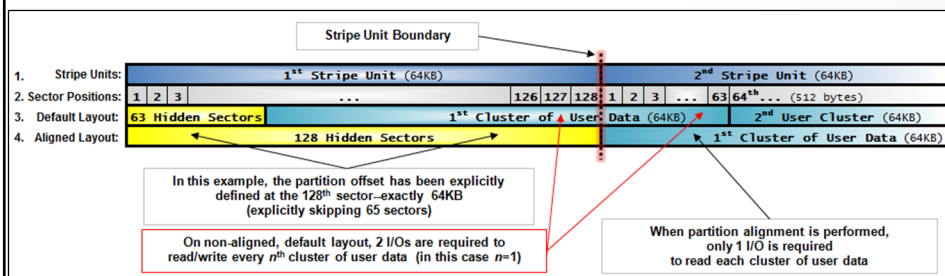


Files and Filegroups

- **A filegroup contains one or more files**
 - A table or index is wholly contained in a single filegroup
- **Every database has at least one filegroup – the PRIMARY filegroup**
 - Contains at least one file – the MDF
 - Multiple data files can be added to the filegroup
- **Multiple secondary filegroups can be defined**
 - Each secondary filegroup can have multiple data files
- **Many reasons for doing this**
 - Manageability, performance, availability...

Partition Alignment

- **With the recommended 64KB RAID stripe size, and the recommended 64KB NTFS cluster size, but with bad alignment, two I/Os are required to read/write every 64KB of data**



- **With correct alignment, only a single I/O is required**

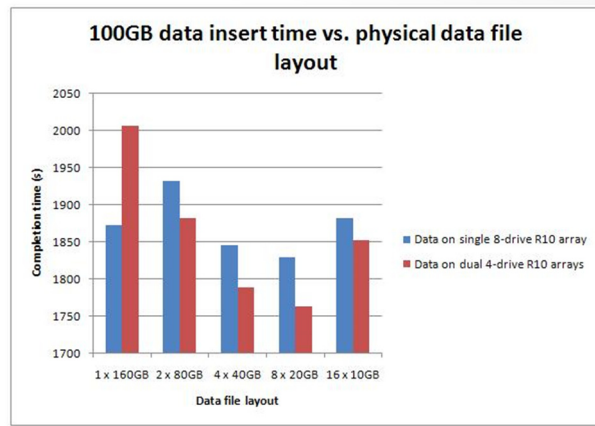
Volume Configuration

- **Disk partition offsets (when not on Windows Server 2008)**
 - Before WS2008, default offset is 31.5KB, WS2008 uses 1MB
 - This is not RAID stripe or NTFS allocation unit aligned
 - Leads to extra I/Os to read a portion of data
 - This can cause up to 30-40% performance drop
 - As documented by MS with many customers
 - On WS2008, still must check after formatting to ensure I/O subsystem did not intercept and use incorrect alignment
 - Upgrading to WS2008 does not fix alignment of existing partitions
- **RAID stripe size should be 64KB or higher**
- **NTFS cluster (allocation unit) size**
 - Should usually be 64KB
- **See WP: Disk Partition Alignment Best Practices For SQL Server**
 - <http://msdn.microsoft.com/en-us/library/dd758814.aspx>

Physical Layout Considerations (1)

- **RAID array configuration**
 - Know your workload and choose an appropriate RAID level (e.g. not RAID-5 for a high volume OLTP system)
 - Different kinds of data onto different storage
 - E.g. rarely used LOB data on RAID 5, OLTP data on RAID 1/10
- **Defrag the volumes**
 - NTFS-level fragmentation DOES affect performance of large scans
 - Not needed if the volumes are not being shared
 - Do not do volume defragging while SQL Server is running
- **Pre-allocate the files to prevent them having to grow**
- **Capacity planning considerations**
 - <http://www.sqlskills.com/BLOGS/PAUL/post/Tool-for-estimating-the-size-of-a-database.aspx> (<http://bit.ly/TTUjJ>)

Multiple Data Files?



- Testing on 15k 300GB SCSI in Dell MD3000i iSCSI, 8-core server
- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/Benchmarking-do-multiple-data-files-make-a-difference.aspx> (<http://bit.ly/as3ZMV>)

Physical Layout Considerations (2)

- **Separation of files**
 - Degree of separation depends on I/O workload and I/O subsystem
 - E.g. SAN can do a better job of spreading I/O costs than a single drive
 - Consider separating:
 - Log files from data files, even separate databases on different LUNs
 - SQL Server files separate from other uses (e.g. OS files)
- **Decide on file/filegroup layout**
 - tempdb (see later in this module)
 - Multiple data files for increased performance?
 - Multiple filegroups for partitioned maintenance?
 - Multiple filegroups to allow partial database availability?
- **Note: SQL Server 2012 allows files to be on SMB shares!**
- **WP: Physical Database Storage Design**
 - <http://www.microsoft.com/technet/prodtechnol/sql/2005/physdbstor.mspx>
(<http://bit.ly/T28kef>)

Data File Layout Survey: <10GB

What's the physical layout of your database, and why? (Databases under 10GB)

Single filegroup, single file		71%	259
Single filegroup, multiple files on a single drive		5%	18
Single filegroup, multiple files spread over multiple drives/arrays/LUNs		13%	47
Multiple filegroups, one per drive/array/LUN, for performance		3%	11
Multiple filegroups, one per drive/array/LUN, for recoverability		0%	1
Multiple filegroups, one per drive/array/LUN, for manageability		1%	3
Multiple filegroups, one per drive/array/LUN, for multiple reasons		3%	11
Multiple filegroups, all on same drive/array/LUN, for any reason		4%	14
Total: 364 responses			

- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/physical-database-layout-vs-database-size.aspx> (<http://bit.ly/o09EA>)



13

© SQLskills. All rights reserved.
<http://www.SQLskills.com>

Data File Layout Survey: 10-100GB

What's the physical layout of your database, and why? (Databases between 10GB and 100GB)

Single filegroup, single file		43%	135
Single filegroup, multiple files on a single drive		9%	27
Single filegroup, multiple files spread over multiple drives/arrays/LUNs		19%	58
Multiple filegroups, one per drive/array/LUN, for performance		11%	34
Multiple filegroups, one per drive/array/LUN, for recoverability		0%	1
Multiple filegroups, one per drive/array/LUN, for manageability		2%	5
Multiple filegroups, one per drive/array/LUN, for multiple reasons		11%	34
Multiple filegroups, all on same drive/array/LUN, for any reason		5%	17
Total: 311 responses			

- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/physical-database-layout-vs-database-size.aspx> (<http://bit.ly/o09EA>)



14

© SQLskills. All rights reserved.
<http://www.SQLskills.com>

Data File Layout Survey: 100GB-1TB

What's the physical layout of your database, and why? (Databases between 100GB and 1TB)

Single filegroup, single file		20%	45
Single filegroup, multiple files on a single drive		5%	11
Single filegroup, multiple files spread over multiple drives/arrays/LUNs		17%	39
Multiple filegroups, one per drive/array/LUN, for performance		21%	48
Multiple filegroups, one per drive/array/LUN, for recoverability		1%	3
Multiple filegroups, one per drive/array/LUN, for manageability		1%	2
Multiple filegroups, one per drive/array/LUN, for multiple reasons		25%	57
Multiple filegroups, all on same drive/array/LUN, for any reason		11%	25
Total: 230 responses			

- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/physical-database-layout-vs-database-size.aspx> (<http://bit.ly/o09EA>)



15

© SQLskills. All rights reserved.
<http://www.SQLskills.com>

Data File Layout Survey: 1TB+

What's the physical layout of your database, and why? (Databases over 1TB)

Single filegroup, single file		11%	16
Single filegroup, multiple files on a single drive		2%	3
Single filegroup, multiple files spread over multiple drives/arrays/LUNs		8%	12
Multiple filegroups, one per drive/array/LUN, for performance		11%	17
Multiple filegroups, one per drive/array/LUN, for recoverability		2%	3
Multiple filegroups, one per drive/array/LUN, for manageability		3%	4
Multiple filegroups, one per drive/array/LUN, for multiple reasons		52%	78
Multiple filegroups, all on same drive/array/LUN, for any reason		11%	16
Total: 149 responses			

- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/physical-database-layout-vs-database-size.aspx> (<http://bit.ly/o09EA>)



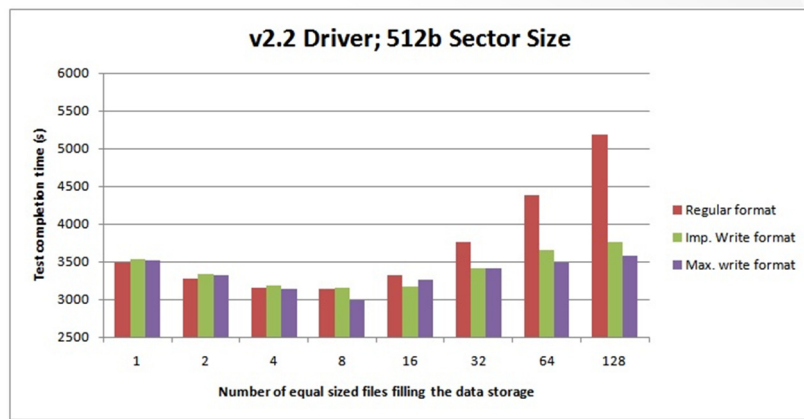
16

© SQLskills. All rights reserved.
<http://www.SQLskills.com>

SSDs

- SSD = Solid State Drive
- Extremely low I/O latency and high throughput
- Expensive, but quickly coming down in price
- Yes, enterprise-ready
 - We have many enterprise clients running on them
- Excellent for random I/Os, but also good anywhere there is an I/O bottleneck
- Don't just put tempdb or transaction logs on them!
- Don't ignore index fragmentation when using them!
- See Paul's SSD blog series:
 - <http://www.sqlskills.com/BLOGS/PAUL/category/SSDs.aspx>

Multiple Data Files With SSDs?



- Testing on Fusion-io 640GB ioDrive Duo, 8-core server
- Source: [http://www.sqlskills.com/BLOGS/PAUL/post/Benchmarking-Multiple-data-files-on-SSDs-\(plus-Fusion-ios-latest-driver\).aspx](http://www.sqlskills.com/BLOGS/PAUL/post/Benchmarking-Multiple-data-files-on-SSDs-(plus-Fusion-ios-latest-driver).aspx) (<http://bit.ly/isXIkW>)

How Does Allocation Work?

- **With a single-file database, very simple**
- **When a table is stored in a filegroup with multiple files, two allocation algorithms kick-in**
 - Round-robin allocation
 - Allocations are made from each data file in the filegroup in turn, essentially striping the data across multiple files
 - Proportional fill
 - Allocations are made from each data file during round-robin proportional to the amount of free space in each file
- **Misconception: you cannot add another data file to a filegroup and rebalance the allocations across all files**
 - You have to create a new filegroup to do that

How Does the Buffer Pool Work?

- **Sometimes called the 'buffer cache'**
- **Memory block used to hold in-memory copies of pages from data files**
- **Pages are read into the buffer pool when requested by other parts of the Storage Engine and not already in memory**
 - When a page is already in memory, this is a logical I/O
 - When a page has to be read from disk, this is a physical I/O
 - All I/Os start as logical and may become physical
- **Page lifetime in memory depends on many things**
 - Two last access times are tracked and provide an LRU indication
 - Tracked as smallint, not a full datetime
 - When memory pressure is felt, the least-recently-used pages are 'tossed' out of the buffer pool to make way for needed pages
 - Also done proactively by the lazywriter background process
- **Hash tables exist to quickly find page images in memory**

More on the Buffer Pool

- **What's in the buffer pool?**
 - `SELECT * FROM sys.dm_os_buffer_descriptors`
 - Be aware that large amounts of index fragmentation can lead to lots of wasted buffer pool space
 - Blog posts with scripts:
<http://www.sqlskills.com/BLOGS/PAUL/category/Buffer-Pool.aspx>
- **Memory management**
 - Buffer pool does full-extent reads when ramping up in Enterprise Edition
 - And with TF840 in other editions: <http://support.microsoft.com/kb/912322/en-us>
 - As long as the I/O subsystem can keep up, no downsides to enabling

Creating or Growing Data Files

- **Whenever a data file is created or grown, its 'high-water mark' must be set in NTFS**
 - This is the point in the file up to which NTFS knows the data is trustworthy and will allow it to be read
- **In SQL Server 2000, and by default in later versions, this is done by zeroing out the new/additional space**
 - This is done by SQL Server, single-threaded, by issuing successive writes of blocks of zeroes to the file
 - The new /additional space cannot be used until the process completes and the allocation that triggered it will pause until it is done
- **In SQL Server 2005+ this process can be skipped by enabling instant file initialization**
 - Only applies to data files

Instant File Initialization in Action

- **Hardware configuration – Dell Precision 670**
Dual Proc (x64) with Dual Core, 4 GB Memory RAID 1+0 array w/4-142 GB, 15000rpm disks
- **Software configuration: SQL 2005**
 - With zero initialization
 - CREATE DATABASE with 20 GB Data file = 14:02 minutes
 - ALTER DATABASE BY 10 GB = 7:01 minutes
 - RESTORE 30 GB DATABASE (EMPTY Backup) = 21:07 minutes
 - RESTORE 30 GB DATABASE (11GB Backup) = 38:28 minutes
 - With instant file initialization
 - CREATE DATABASE with 20 GB Data file = 1.3 seconds
 - ALTER DATABASE BY 10 GB = 0.4 seconds
 - RESTORE 30 GB DATABASE (EMPTY Backup) = 5 seconds
 - RESTORE 30 GB DATABASE (11 GB Backup) = 19:42 minutes



Instant File Initialization

- Allows SQL Server to use Windows XP + Windows Server 2003+ technology to skip the zeroing process
- Instead of doing the zeroing, SQL Server calls the Windows API SetFileValidData, which sets the high-water mark
- Disabled by default because there is a potential security risk of enabling it:
 - If a volume is being shared by a database and a file server storing 'secure files' and if the file server deletes some files, and then the database grows, it may pick up some of the space used by the deleted files
 - As instant file initialization skips zeroing the new space, the old raw bytes of the deleted files will be accessible to a DBA using DBCC PAGE
 - If this is not the case then no security risk, or if the DBA is a Windows Administrator then the risk is already there
- If you can, turn it on.

Enabling Instant File Initialization

- **Cannot be enabled from within SQL Server**
- **Requires:**
 - NTFS volumes with Windows XP SP2 or Windows 2003+ Server
 - SQL 2005+, any edition
 - "Perform Volume Maintenance Tasks" security permission granted to SQL Server service account or group
- **Use Local Security Policy Editor to grant the permission**
 - Administrative Tools -> Local Security Policy and then Local Policies -> User Rights Assignment
- **Defaults to Local Administrators Group**
 - Can grant this to lower privileged account
- **Instant initialization happens automatically once SQL Server is restarted after granting the permission**
- **Use TF 3605 + TF 3004 to watch initialization process in the error log**

Auto-Grow

- **When a file is full, it will grow to provide more space**
 - By default, new space is zero-initialized, which takes time
 - Allocation pauses while the grow takes place
 - Commonly the auto-growth default is not changed leading to massive amounts of auto-growth
 - 1MB in SQL 2005+, 10% in previous versions (10% still for log files)
- **Best practice to manually manage file sizes**
 - Manually growing files allows YOU to decide which files to grow, by how much, and when
 - Allows file fragmentation to be minimized
 - Monitor file usage and alert when threshold reached
- **Auto-grow should always be left enabled 'just in case'**
 - If it's off, and no-one monitors space, the workload could stop
 - Monitor auto-growth using SQL Trace or Extended Events
 - Always have file growth to be a fixed amount, not a percentage

Auto-Shrink

- **When to use? NEVER!**
 - About the only thing there is NOT an 'it depends' answer for
 - This should be one of the first things that gets checked when you take over a new database
- **Why not?**
 - Data file shrink (same code as auto-shrink) is almost guaranteed to introduce index fragmentation within the data files
 - The database will most likely just auto-grow again, and then auto-shrink, auto-grow in a cycle that wastes resources
 - You can't control when it kicks in and affects performance
 - Blog post: <http://www.sqlskills.com/blogs/paul/post/Auto-shrink-e28093-turn-it-OFF!.aspx> (<http://bit.ly/fylBzd>)
- **Again, always make sure auto-shrink is turned off**

Demo

Impact of shrink on index fragmentation

When to Use Data File Shrink?

- **Should be a very rare operation**
 - Because it causes index fragmentation
- **Three scenarios:**
 - Emptying a file before removing it
 - When a large amount of data has been deleted AND the space won't be reused
 - Moving a file/filegroup/database to read-only
- **Even then, shrink may not be the best method**
 - Remember – it causes fragmentation
- **So, how to perform a data file shrink?**

How to Shrink a Data File?

- **Create a new filegroup and move all indexes into it**
 - Use CREATE INDEX... WITH DROP_EXISTING and specify the location to be the new filegroup
 - Doesn't move LOB data, but see <http://bit.ly/H75AWL>
 - Can be performed online
- **If any table are heaps, use shrink to move them**
 - Heaps are unordered so shrink does not fragment them
 - Beware shrink is very slow in SQL 2005+ for heaps and LOB data
- **Drop old filegroup**
- **If you have to shrink a data file, use ALTER INDEX ... REORGANIZE to remove index fragmentation**
 - Rebuilding the indexes will grow the files again

Common Shrink Misuse

- Using shrink in a regular maintenance plan (or auto-shrink and auto-grow enabled)
 - Leads to shrink-grow-shrink-grow cycle
 - Causes massive fragmentation
 - MS Dynamics CRM does this – be aware!
 - <http://bit.ly/x9a8Pa>
- Using shrink after an index rebuild to reclaim space
 - As we'll see in the index fragmentation module, an index rebuild needs space to build a new copy of the index, maybe growing the file
 - Shrinking after a rebuild will reclaim that space BUT will undo the effects of the index rebuild, wasting a lot of time and resources

Data File Management Survey

Leave auto-grow set to the default		13%	16
Set auto-grow to a percentage		8%	10
Set auto-grow to a fixed size		40%	51
Auto-grow is turned off (because of SCOM)		2%	2
Auto-grow is turned off (some other reason)		3%	4
Any method plus monitoring of file sizes/usage		19%	24
Any method plus manual file/database shrink		1%	1
Any method plus shrink in a maintenance plan		2%	3
Any method plus auto-shrink enabled		0%	0
Don't know		3%	4
Other? Enter here...		9%	11
Total: 126 responses			

- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/Importance-of-data-file-size-management.aspx> (<http://bit.ly/ZtvSD>)

Traditional Tempdb Uses

- **User objects:**
 - Table variables (@)
 - If they get large enough to spill to disk
 - #temp tables/indexes
 - Global ##temp tables/indexes
 - Regular tables
- **Internal objects**
 - Worktables (e.g. sort, intermediate results, ...) from memory spills to disk
 - Workfiles (only for hash joins and hash aggregates) from spills to disk
 - Intermediate sort results from index create/rebuilds
 - Only if SORT_IN_TEMPDB is specified
 - Intermediate DBCC CHECKDB results (in a worktable)
 - Version store (main one + online index operations one)
- **Full list and explanations in BOL: Capacity Planning for Tempdb**

Tempdb Uses in SQL 2005+

- **Version store**
 - The common version store, which underpins:
 - Snapshot isolation/read-committed snapshot isolation
 - DML triggers (for the inserted and deleted tables)
 - MARS (multiple active result sets)
 - The online index operations version store
- **Internal objects**
 - Temporary LOB storage
 - LOB data used as parameters, variables

Version Store

- Stores all version records (as described in Module 1) created in all databases
- One new allocation unit created every minute
 - Variable size depending on how many versions created in that minute
- Cleanup task runs every minute or so
 - If all version records in allocation unit are no longer required, it can be deleted
- Non-logged
- Long-running transactions using snapshot isolation can interrupt cleanup and lead to tempdb growth
- See Books Online: Row Versioning Resource Usage
 - <http://msdn.microsoft.com/en-us/library/ms175492.aspx>

Tempdb Sizing

- No easy way to figure out initial size as so many operations can use tempdb space
- General practice is to create tempdb, run production workload and see how big tempdb gets
 - General query workload
 - Index maintenance operations
- Set tempdb size to resulting size
- Enable appropriate auto-growth, equal on all files
- Enable instant file initialization for instance
- We'll talk about the number of files to create in a few slides...
- Books Online:
 - Capacity Planning for Tempdb
 - <http://msdn.microsoft.com/en-us/library/ms345368.aspx>

Tempdb Sizing and Restart

- Tempdb reverts to the last specifically set size after a server restart
- If tempdb grows during operations, that is not the same as specifically setting the size
- Avoid excessive auto-growth after a server restart by manually setting the size

Tempdb and I/O Subsystems

- Best practice is to separate tempdb from other databases due to I/O contention and put on high-performance I/O subsystem
 - Entirely depends on I/O subsystem and workload involved
- Favorite use of SSDs today
 - SSDs are best suited to random I/O, but generally will give a performance boost to an I/O subsystem that's overwhelmed
- Page checksums on tempdb weren't available until SQL 2008
 - On by default for new instances of SQL 2008 onwards
 - Must enable for tempdb for upgraded instances
- Can be put on RAM drive (no durability requirements)
 - See KB 917047 (<http://support.microsoft.com/kb/917047>)
- In 2012, tempdb in a cluster can be on non-shared storage
- Books Online – Optimizing Tempdb Performance
 - <http://msdn.microsoft.com/en-us/library/ms175527.aspx>

Contention in Tempdb

- Tempdb is very susceptible to contention issues because there's only one tempdb per instance to support all user database operations
- Contention is very often:
 - PAGEIOLATCH
 - Thread waiting for a page to be read from disk
 - PAGELATCH
 - Multiple threads competing for access to an in-memory page
- PAGEIOLATCH contention can be alleviated by:
 - Creating multiple data files, and/or
 - Putting tempdb on a high-performing I/O subsystem
 - Avoiding excessive use of tempdb

In-Memory Tempdb Contention

- Some query workloads cause multiple concurrent threads to repeatedly create/drop small temp tables and/or worktables
- Causes PAGELATCH contention on allocation bitmaps and system catalogs
 - Use sys.dm_os_waiting_tasks to see waits on PAGELATCH_EX
 - SGAM page to manipulate mixed extents (resource 2:1:3)
 - PFS page to allocate/deallocate pages (resource 2:1:1)
 - System catalogs (usually resource 2:1:103 – sys.sysmultiobjrefs)
 - Don't use unique or primary key constraints (e.g. in TVPs)
 - Fixed somewhat by 2008 R2 CU9
- Contention can occur in all versions
- This can sometimes (rarely) happen in user databases with VERY high-end allocation workloads

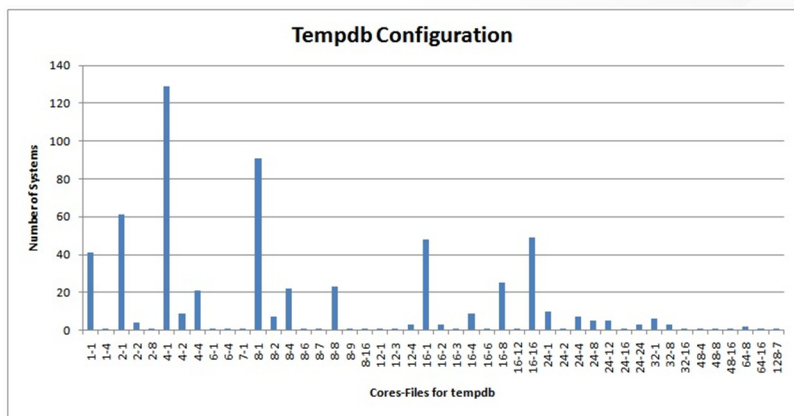
Tempdb Allocation in 2005+

- Tempdb allocation algorithms were tweaked in 2005
 - No changes after 2005
- There is now a cache of worktables and temp tables
 - When a temp table/worktable is dropped, a single IAM page, a data page/extent, and its metadata entry remain
 - Worktable cache is very small
 - Temp table cache is (relatively) unbounded
 - As long as temp table is less than 8MB when dropped
 - Only for temp tables NOT created by ad hoc statements
 - The next temp table/worktable creation pulls one out of the cache
 - All kinds of reasons a temp table may not be cached
 - See Paul White's detailed blog post:
 - http://sqlblog.com/blogs/paul_white/archive/2012/08/17/temporary-object-caching-explained.aspx (<http://bit.ly/QrCYcG>)
- Tempdb contention can (and does) still happen in 2005 onwards

Alleviating In-Memory Contention

- Trace flag 1118 (discussed in KB 328551) removes use of mixed extents (single-page allocations) in all databases
 - Removes SGAM contention
 - Still works and may be needed in 2005+, but not as much as on 2000
- Creating multiple data files to spread and reduce contention
 - For 2000, # data files should be equal to # processor cores
 - For 2005+, you can still do 1:1 but you may not need to
 - Start with # data files = ¼ to ½ # processor cores and add more if needed
 - Bob Ward (CSS) at PASS 2011:
 - < 8 cores, use #files = #cores
 - > 8 cores, use 8 files, increasing by chunks of 4
 - Be careful of having too many data files
- Note: this does not apply to log files

Tempdb Configuration Survey



- Source: <http://www.sqlskills.com/BLOGS/PAUL/post/Tempdb-configuration-survey-results.aspx> (<http://bit.ly/dRZzKb>)

More Info on Tempdb

- Moving tempdb – see KB 224071
- WP: Working with tempdb in SQL Server 2005
 - <http://www.microsoft.com/technet/prodtechnol/sql/2005/workingwithtempdb.mspx> (<http://bit.ly/oq0VH>)
- Blog post: Misconceptions Around TF 1118
 - <http://www.sqlskills.com/BLOGS/PAUL/post/Misconceptions-around-TF-1118.aspx> (<http://bit.ly/vNMLv>)
- Blog post series on tempdb
 - <http://www.sqlskills.com/BLOGS/PAUL/post/Comprehensive-tempdb-blog-post-series.aspx> (<http://bit.ly/T29Cpr>)
- Books Online:
 - Troubleshooting Insufficient Disk Space in Tempdb
 - <http://msdn.microsoft.com/en-us/library/ms176029.aspx>
 - Capacity Planning for Tempdb
 - <http://msdn.microsoft.com/en-us/library/ms345368.aspx>

Review

- Physical layout considerations
- Allocation algorithms
- Instant initialization
- Auto-grow
- To shrink or not to shrink?
- Tempdb

Questions!