IDERA® | SQL Server Whitepaper

6 SQL SERVER PERFORMANCE BOOSTERS WITH YOUR EXISTING HARDWARE

TABLE OF CONTENTS

INTRODUCTION

Nearly all successful applications can benefit from the performance-driven strategies outlined in this whitepaper. Many of the categories and items discussed contain easy wins that can boost performance and scalability with far less effort than expected. The techniques outlined here can also serve as a checklist for determining whether it is time to consider throwing more hardware or other resources (such as consultants, third-party solutions, and so on) at a problem.

Topics addressed in this whitepaper are the focus of countless books, articles, white papers, and blog posts. As a result, this whitepaper does not provide an exhaustive overview, but helps raise awareness of potential performance boosting opportunities, provides background information and context for evaluating the effectiveness of these techniques, and outlines some key approaches and performance benefits that stem from implementing these approaches where applicable.

> Scientists dream about doing great things.
> **Engineers do them.**
>
> – James A. Michener

VM HOST OVERCOMMIT

VMs have come a very, very long way over the last 5 years or so. In fact, virtualization has become more commonplace than dedicated, physical hardware in my experience. Most of the time it works great, but a busy production SQL Server wants resources now and it expects/needs them to be available as if the machine were truly physical. It is possible, and tempting, to overcommit a host. What I mean by this – is you can have a host with 128 GB of RAM, 8 CPUs with hyperthreading turned on (so 16 CPUs hyperthreaded), and from that host you can create 4 virtual machines each with 8 vCPUS and 64 GB RAM. You can even do this with the disk (look up “Thin Provisioning”). In plain English – you can create VMs that all-together have more resources than actually exist on the host, hedging a bet that not all of the VM machines will need all their allocated resources at once, so there should always be enough to go around. The VM Host software will move resources back and forth and manage the flow “to and from” the machines you’ve created.

If you need to have consistent performance from your production SQL Server, this type of over-allocation can create surprises. Trying to figure out where the slowness is coming when SQL is the victim and not at all responsible itself can take a lot of time and effort to spot. Whenever possible, do not overcommit resources on a host if your SQL Server’s performance is mission critical. When resources on a host are shared and over-commitment is present, use reservations to guarantee resource availability to your production SQL Server.

CPU POWER OPTION

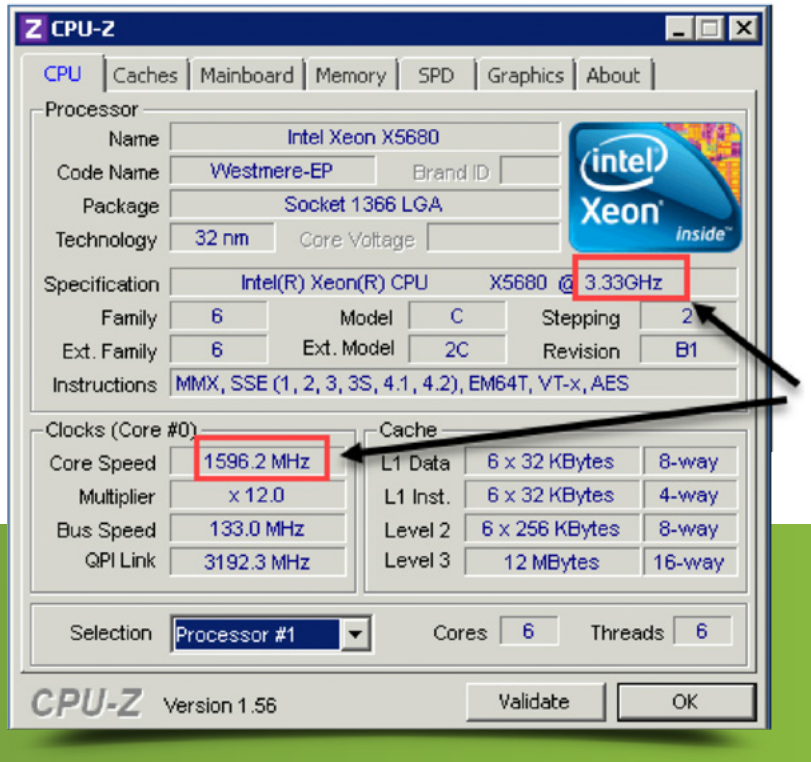
Along the way, somewhere around 2010 by my best estimation, hardware vendors (and Microsoft too) began "going green". By this, I mean shipping hardware with a feature enabled in the BIOS (turned on by default) that forced less power consumption by throttling back the GHz of the CPUs to use less energy. Microsoft, in turn, added to their Operating Systems a "Power Settings" feature that by default installed with a Server set on "Balanced" power mode, which also took the action of throttling back the GHz. A server with these settings is capable of ramping up to its full potential, but only after a set time of running "hot enough". In other words – you have to be pushing the envelope of what is essentially a slower engine before you get the rest of the horsepower you paid for allocated to you! If your application / database has a lot of iterative / single threaded processes, or is servicing a chatty application that approaches the database with many, many small queries, your architecture dictates that you can only go as fast as the slowest single CPU. If you're not able to take advantage of parallelism, leaving these settings at their default could have a significant performance impact.

Bigger servers, like those often used to host many VMs, often have many CPUs but all at a slower GHz speed than the smaller machines. (One word explains this phenomenon: Heat). The effect of code not capable of parallelization in combination with power savings options being enabled on very large hosts is even more severe than on smaller servers with faster CPUs.

In order to check and see what speed your CPUs "are capable" of performing at versus where they "are performing", download and take a peek using a free tool provided by CPU-Z.

Here, you can also read about Brent Ozar's personal discovery about this phenomenon.

Additionally, you can utilize PowerShell to identify performance issues with power management by querying performance counters. You can use this PowerShell script from Microsoft's TechNet Gallery:



Yikes!
Where's all that power you were expecting?

MAINTENANCE

When you create or restore a database, maintenance plans are not automatically created. A DBA must explicitly create maintenance plans, either by using the Maintenance Plan Wizard, by creating or downloading custom scripts, or by using a 3rd party solution like those found in Idera's Admin Toolset.

Fragmentation

Data that is inserted or updated that doesn't physically fit onto the page where it logically belongs pointers to redirect any future IO activity to the location where the information was stored instead. This following of a pointer to another location, instead of reading information in-line, causes additional IOs. Things being out of order is called fragmentation. SQL Server performs best if maintenance is done to keep things tidy and straight. You can read more about the types of fragmentation here.

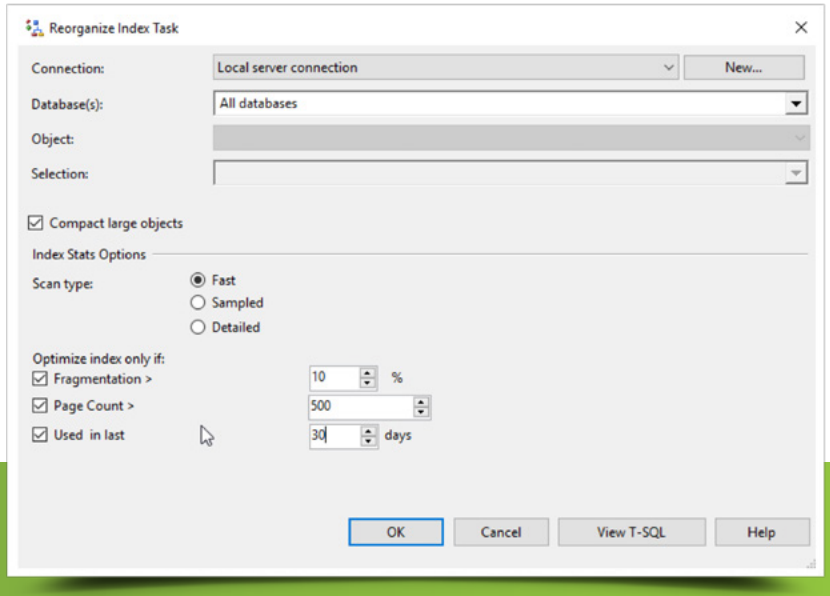
Only Clean What Needs to be Cleaned

SQL Server's default maintenance plans make it easy to point and click, using a GUI to get some maintenance up and running. Prior to SQL Server 2016, they were extremely limited in what they could do. If running SQL Server 2014 or earlier, these SSIS plans should not be used for index maintenance. Instead use a stored procedure based solution.

The issue with the pre-2016 maintenance plans is that they are not intelligent. They update EVERYTHING. Let me ask you a question – do you open up the hutch and night after night load the Christmas Dinner or Thanksgiving Feast dishes into the dishwasher, day after day? No. You use those dishes once a year (if that), and you wash them only when they are dirty. The older maintenance plan wizard created plans that cleaned every dish in the house every day regardless of whether or not it was used.

The new maintenance plan wizard allows the user to "clean" only that which is "dirty". If using the SSIS solution make sure to have 2 steps. There should be a reorganize step that runs against each database and only optimizes if the index is 10% or 15% fragmented and 500 or 1000 pages in size. The third option may make sense to include but is quite optional. If your SQL Server restarts monthly or more often then this checkbox should be left unchecked. If your server restarts less often then consider including it with a time frame around 30 days.

The second step should be a rebuild step. It should use all the same settings except the fragmentation percent which should be 25% or 30%.



Auto-Shrink, Just Say No.

I know. The plan wizard. It has an option to auto shrink files. It's just a bad idea. Don't do it.

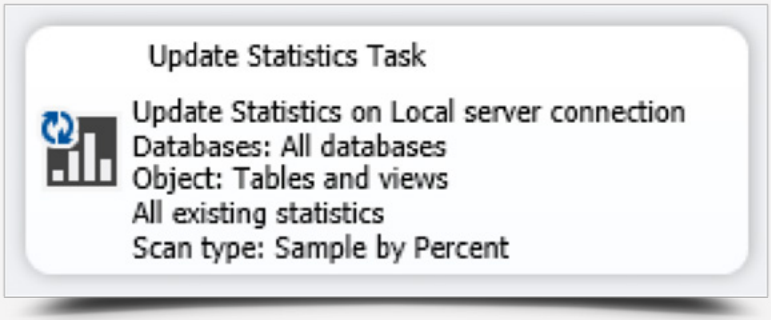
If you have a file that has become bloated with empty space because of a one-off scenario like temporarily failing log backups or a large housekeeping operation then manually shrinking the file is not out of order. But that should be done as a one-off manual operation no more often than the scenario that caused it to bloat.

Statistics

One thing that makes SQL Server so successful is its phenomenal internal query execution engine, which relies on complex statistics about the nature, cardinality, and density of data to speed the execution of queries. However, without regularly updated statistics, SQL Server's ability to properly determine the most efficient approaches to satisfying a query starts to wane. SQL Server is by default set to update statistics automatically, but in my experience, it does a poor job of staying on top of updating statistics on its own.

Moreover, in systems where large volumes of changes happen regularly (or even occasionally), it is easy for statistics to get out of whack, and for performance to suffer. Even worse, if statistics are allowed to degrade over time, it is not uncommon to encounter situations where the slow buildup of statistical 'sludge' will lead to performance problems that slowly degrade until reaching a 'breaking point,' where performance can take a catastrophic turn for the worse with some key queries or operations.

While an in-depth overview of updating statistics is outside the scope of this document, an easy way to achieve roughly 70% of the benefit of a full-blown statistics updating routine with about 2% of the effort is to simply add a step to the index maintenance plan to update statistics. Be sure to choose all databases, all indexes, and sample by 10 or more percent. If performance doesn't improve as much as desired, or even gets worse, consider a higher percentage. A full scan is a 100% sample rate.



While not perfect, this step will go through each database on your server and do a bare-minimum update of all statistics.

Additional resources and insights into routinely updating statistics can be found in Books Online, and through several articles and blog posts online. The key to remember is that without regularly updating statistics, performance will suffer, even if SQL Server is configured to automatically update statistics.

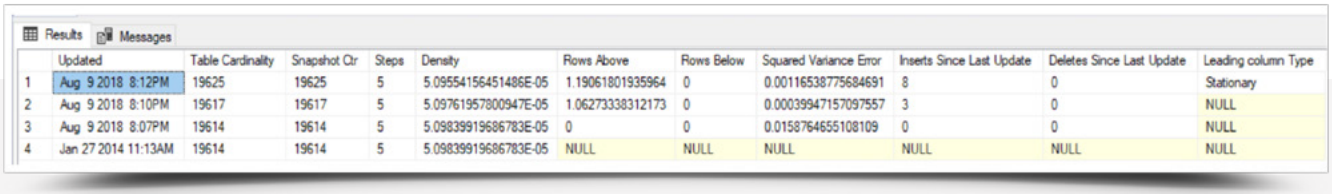
Viewing additional statistics information

Using trace flags 2388 and 2389 allows you to view details for the last 4 times that statistics were run. These details include a lot of helpful information that can help you diagnose issues that may be related to the statistics. Not sure how often an object was inserted or deleted from since the last statistics were run? This information can be seen using in the “Inserts Since Last Update” and “Deletes Since Last Update” columns in the results. These metrics can help you understand how often the object is modified and the appropriate time to run statistics again.

Other metrics from these query trace flags can be useful as well, such as the “Leading column Type” which helps you understand how the leading column in the statistics is used. This column also helps the optimizer identify if the statistics are ascending or not, so the appropriate plan can be chosen.

To use the query trace flags, use the following example.

```
DBCC TRACEON (2388)
DBCC TRACEON (2389)
DBCC SHOW_STATISTICS (‘Person.Address’, ‘PK_Address_AddressID’)
DBCC TRACEOFF (2388)
DBCC TRACEOFF (2389)
```



Results | Messages

	Updated	Table Cardinality	Snapshot Ctr	Steps	Density	Rows Above	Rows Below	Squared Variance Error	Inserts Since Last Update	Deletes Since Last Update	Leading column Type
1	Aug 9 2018 8:12PM	19625	19625	5	5.09554156451486E-05	1.19061801935964	0	0.00116538775684691	8	0	Stationary
2	Aug 9 2018 8:10PM	19617	19617	5	5.09761957800947E-05	1.06273338312173	0	0.00039947157097557	3	0	NULL
3	Aug 9 2018 8:07PM	19614	19614	5	5.09839919686783E-05	0	0	0.0158764655108109	0	0	NULL
4	Jan 27 2014 11:13AM	19614	19614	5	5.09839919686783E-05	NULL	NULL	NULL	NULL	NULL	NULL

FILES

Quantity

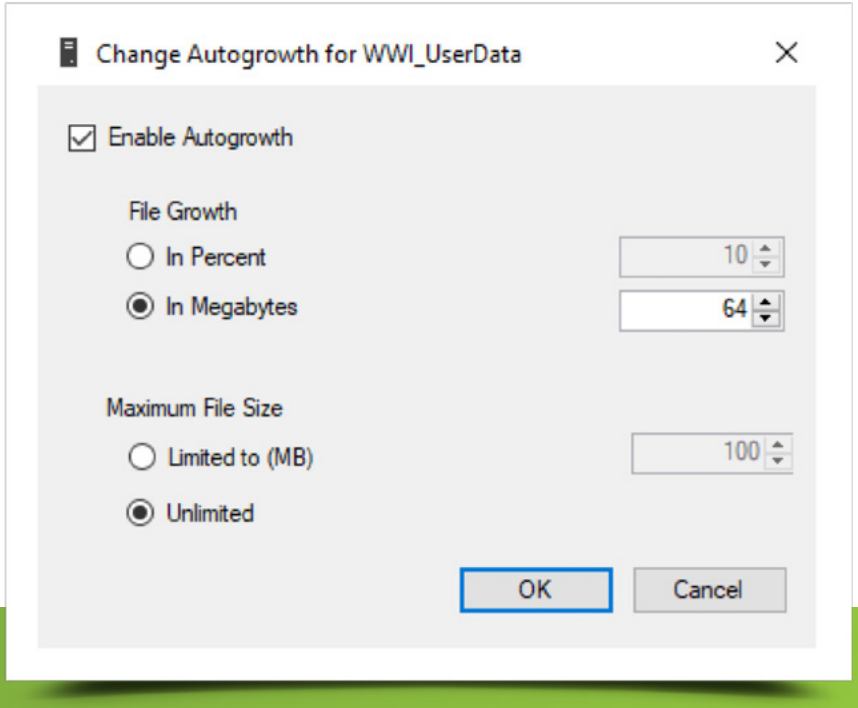
In the past databases were broken into multiple files so that the files could be put onto different physical volumes and involve more spindles in the disk operations. Now that most database servers run against SAN or NAS storage with many, this particular reasoning for having multiple files in a filegroup is not as pressing.

Still, creating multiple files does make sense for a few reasons:

1. If doing partitioning and putting older, staler data into a separate file and filegroup and perhaps moving those files to a lower cost, slower tier of storage.
2. To take advantage of parallelism and seeks or scans in parallel, allowing multiple processors to each work independently on their own file synonymously.
3. When dealing with highly transactional systems multiple files may make sense. The STORPORT driver is used by Windows for communicating with the storage and is limited to 255 outstanding IRPs per LUN per Path. An IRP is an I/O request packet. Think of this like a TCP packet for networking. These IRPs transmit the commands to the storage. You may experience a bottleneck in the event that there are many frequently updated files on the LUN holding your data or log files. This can be seen using the PHYSICALDISK\CURRENT DISK QUEUE LENGTH counter, which will cap at 255. Additionally, you may use a STORPORT ETL trace to identify latency. Should this occur, the best resolution is to add an additional LUN that can hold another data file or be used to move files from other databases to the new LUN, which would effectively provide an additional 255 IRPs.

Growth

When SQL Server data or log files become full they must grow. Autogrowth should be enabled and growth should always be set "In Megabytes", not "In Percent". Choose an appropriate amount of growth between 64MB for small databases and 1024MB for larger databases.



Location

The answer to the question “where should I put the database files?” varies greatly based on the architecture of the system. But one thing remains the same – get as many IOPS as possible. SQL Server loves moving data around!

Consider putting data and logs on separate logical volumes, even if the underlying disks are the same. So too, should the operating system be on its own volume. This will allow for more disk controllers and more disk queues to further reduce latency at the operating system level.

If using mechanical disks, this can be achieved by including as many spindles as possible in the disk array that supports the SQL Server. Separating the logs completely so that they are on even just a mirrored set can work wonders – logs are sequential in nature and are negatively impacted by the spastic nature of random IO. Keeping them isolated and away from the distraction of disk mechanism arm movement is best – think of it like separating your freeway driving from your around the town errand driving. The first gets great gas mileage because it is consistent and uninterrupted. The latter suffers from all the starting and stopping. Intersperse two and you get the minimum MPG for both, not the average MPG.

In an SSD solution this is less important as the disks tend to natively offer much higher IOPS numbers.

This is all well and good if there is only 1 tier of storage, but what if the environment offers a portion of SSD storage and another portion of mechanical storage? How can a DBA decide which files to put where?

SQL Server offers a DMO to measure which files are the busiest. This DMO will show the number of times from which each file was read and to which each file was written to along with the total number of bytes involved. The io_stall columns show how much time was spent waiting on the disk subsystem to respond to requests. These numbers reset to zero upon every service restart so don’t expect to get a decent reading unless the server has been up for a while:

SQL Database Files, Bytes, IOs and Experienced Stall Time

Don’t be surprised to see TempDB at the top of this list. It may seem strange to dedicate the fastest available disks to temporary space rather than mission-critical data, but often it is the best case for performance.

PUT BUSY FILES ON SSD STORAGE

Speaking of files, for many SQL Server implementations, fast response times for the TempDB space is crucial if you want good performance. TempDB is not only used for "overflow" when available memory space is not enough, but it is used for all temporary objects, including tables, table variables, and triggers. It is a good idea to get a look at your SQL Server file use to understand which database and files are the most intensive.

The script below is a variation of the script presented earlier. It collects information about Files, Bytes, IO and Stall Time, stores it as a snapshot and then allows the DBA to take a second reading (and more) to compare back to the original collection. This is useful because the DMOs are like odometers that collect the information from when the SQL Server Service was started and can be polluted by nightly maintenance and long running code that happened in the past.

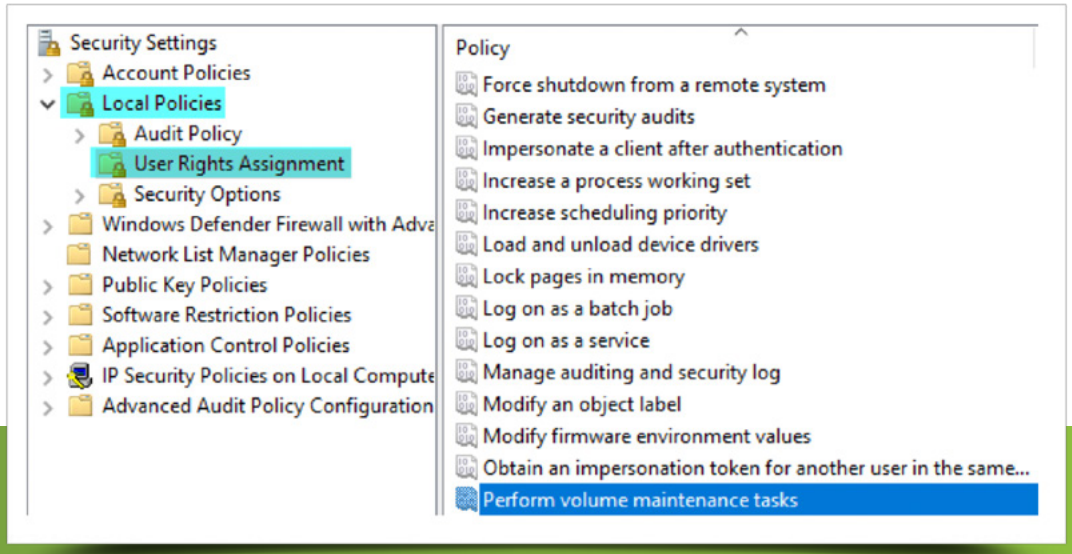
SQL Database File Stall Snapshot and Compare Script

CONFIGURATION TWEAKS

Instant File Initialization

This feature is only useful when running SQL Server on Windows. Instant file initialization (aka Perform volume maintenance tasks) is a very useful permission for the SQL Server service to have. It will help performance whenever performing large file operations such as backups, restores, or data file growth events. Without permission to this feature the SQL Server service will request space from the operating system and the operating system will only deliver the space to the service after it has been zeroed out. If the file growth event or the restore event involves a significant amount of disk space then this can be very time-consuming. With this permission granted then the operating system can deliver the requested space without first zeroing it out. This can be nearly instantly.

Since SQL Server 2016 the SQL Server installation wizard has offered the opportunity to set this permission during the install. If you skipped that checkbox during installation or are running a prior version then it can be set by going to the SQL Server and running "secpol.msc" from a run command window. Browse as shown in the screenshot below and make sure the SQL Server service account is listed in the group of accounts allowed to access this feature.



Server Configuration Settings

SQL Server comes pre-set with default settings that are general. Enterprise-level performance requires many of these settings be reviewed and appropriately tuned. This section covers those that are most commonly adjusted and ripe areas for performance improvements just waiting to be had.

A strategy to improve performance is to run large, single queries in multiple, parallel threads. Parallel query execution is not new to SQL Server, but the system default settings controlling parallelism have not kept up with the newer and more powerful hardware available now. These first 2 settings combine to fine tune this parallel execution for modern hardware.

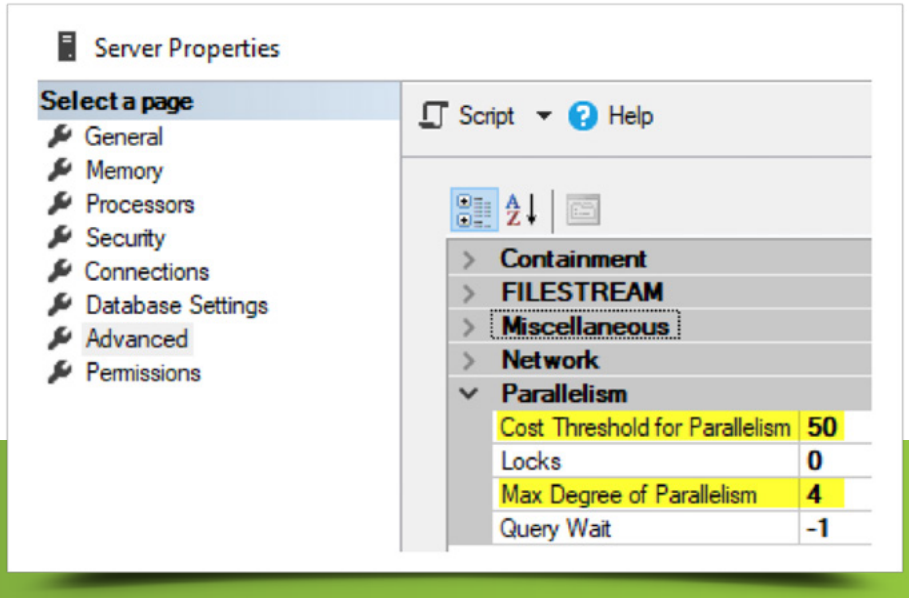
Cost Threshold for Parallelism

The first setting is “cost threshold for parallelism”. There is a cost associated with breaking a query into multiple pieces and then putting those pieces back together again so there isn’t value in doing this to very small queries. This setting answers the question “how big should a query be to be considered for parallel query execution?”. The system default is 5 which correlates to an estimate of 5 clock seconds to execute the query. 5 seconds sounds reasonable until you realize that this estimate is based on a server performance from back when dual Pentium III processors, 4GB of RAM, and a 5 disk Ultra-SCSI array was the norm. With a more modern machine a much larger number tends to make sense. Consider starting with something between 25 and 50.

Max Degree of Parallelism

The second parallelism setting is “Max Degree of Parallelism”. If a query estimate is high enough that the previous setting allows it to be executed in parallel then this setting controls the number of threads into which it can be broken. The system default is 0 which means all available processors. Back when machines commonly had between 1 and 4 processors that was reasonable. It still is if your server has 5 or fewer cores. However, if you have 6 or more cores, choose a number that matches about half the number of threads the machine supports, with a maximum value of 8.

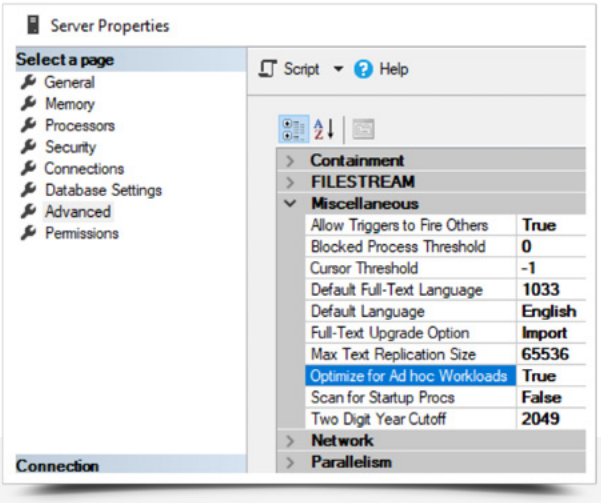
Both of these settings can be found in the server properties in SSMS under the advanced pane and in the Parallelism section or they can be set using sp_configure.



Optimize for Ad-Hoc Workloads

The next setting to be covered is “Optimize for Ad hoc Workloads”. This setting changes how query plans are stored in the plan cache. With the setting set to false, the system default, every non-trivial plan that is generated by SQL Server is saved in the cache. If the server workload has a lot of single-use queries then the cache will quickly bloat with plans that won’t ever be used again. This is a waste of valuable system memory. With this setting enabled then any non-trivial plan that is generated is not saved in its entirety upon the first execution. Instead, a plan stub is saved with a fraction of the memory. Then, only if the exact query plan is generated a second time, will it be saved in its entirety for future reuse. It is almost universally considered best practice to enable this setting.

This setting can be found in the server properties in SSMS under the advanced pane and in the Miscellaneous section or it can be set using sp_configure.

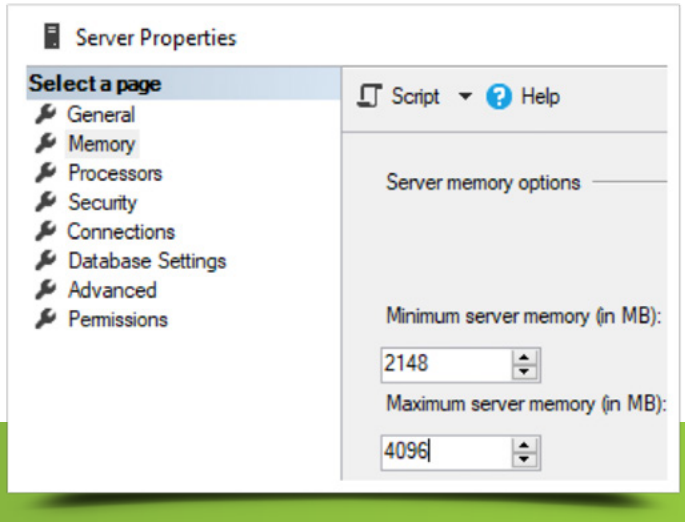


Min and Max Server Memory

The final 2 settings to be covered also work in a pair. They are “min server memory (MB)” and “max server memory (MB)”. These settings control how memory is allocated from the operating system.

SQL Server will allocate memory from the operating system as long as there is memory available. It will continue to do so until it has left the operating system with so little memory that it can’t continue to operate properly. Certainly, this is not ideal. By setting the max server memory setting to a value below the total amount of memory on the box you can be sure that SQL Server will stop allocating memory before it starves the OS of memory. As a general rule, try to leave 4GB of memory for the operating system, another 1GB for SSRS if it is running, and another 2GB if users will be connecting to the server using RDP. Subtract those numbers from the OS amount and set the maximum memory to the difference. After the server has been running for a while review the memory statistics of the box and reevaluate that number.

The minimum server memory number isn’t nearly as important as the maximum. Its main job is to limit the amount of memory a virtual host can deallocate from a SQL Server VM in an overallocation event. If the server has a full memory reservation, which is preferred, then this setting never comes into play. If not, set the minimum to a value 25% less than the maximum. These settings can be found in the server properties in SSMS under the memory pane or they can be set using sp_configure.



Trace Flags

Starting with SQL Server 2016 we are less beholden to trace flags than in previous versions of SQL Server. Some of the most popular trace flags, T1117 (TempDB only), T1118, and T2371 are now automatically enabled. If running a version of SQL Server prior to 2016 then T1118 and T2371 should definitely be enabled and T1117 should probably also be enabled but could be skipped if there are many databases with multiple files.

There is a new trace flag, T3427, which should be enabled for SQL Server 2016 starting with SP1 CU2. It does not need to be turned on for anything earlier or from SQL Server 2017 forward—assuming recent service packs and cumulative updates have been installed. It resolves a TempDB CPU usage bug.

Trace flags T4199 and T3226 both continue to provide value and should be used on new instances of SQL Server. T4199 enables optimizer fixes that were delivered in a cumulative update or service pack in an off-by-default state. T3226 tells SQL Server not to create log entries for successful log backups. If your server does log backup hourly – or even more often – then your error log can end up being filled with these messages and hide an actual error that you really need to see.

ABOUT THE AUTHORS

Mindy Curnutt

Mindy Curnutt is a business owner and 5X Microsoft Data Platform MVP and has been involved in the SQL Community for 20+ years. She has been an SME on multiple MS SQL Server Cert Exams and is the co-author of 4 books, most recently MS Press SQL Server 2017 Administration Inside-Out. She is the President of the North Texas SQL Server User's Group, a national public speaker and active Girls and Stem volunteer and mentor. You can often find Mindy in her hometown of Dallas, TX at the Sons of Hermann Hall Acoustic Picker's Jam on Thursday nights.

Eric Blinn

Eric Blinn is the Senior Database Architect at Squire Patton Boggs, an international law firm with 47 offices in 20 countries, based out of Cleveland, Ohio. He is a content creator and facilitator for the data track at DriveIT, an Akron, Ohio based IT training firm. Eric is an active member of the SQL Server community and currently serves as the Vice President of the Ohio North PASS Local Group. He has spoken on SQL Server topics at a number of technology conferences and training events. When not working on SQL Server Eric enjoys smoked meats and going fishing with his family.

Daniel Janik

Daniel Janik is an independent consultant from Austin, TX and a 2X Microsoft Data Platform MVP. He has been supporting SQL Server solutions in many roles from DBA to developer over his 20-year career. Daniel is also former Microsoft and was a field engineer (PFE) for 6 years. He speaks regularly at SQL Saturday events and user groups. You can find him on his blog SQLTechBlog.com.

ACHIEVE 24/7 SQL MONITORING

WITH SQL DIAGNOSTIC MANAGER

- Monitor performance for physical, virtual, and cloud environments.
- Monitor queries and query plans to see the causes of blocks and deadlocks.
- Monitor application transactions with SQL Workload Analysis add-on.
- View expert recommendations from SQL Doctor to optimize performance.
- Alert predictively with settings to avoid false alerts.
- View summary of top issues and alerts with the web console add-on.

Start for FREE

IDERA

IDERA.com