Database configuration

Enterprise applications

NetApp
March 13, 2024

Table of Contents

Database configuration 1
Microsoft SQL Server shared instance versus dedicated instance 1
CPU configuration 1
Microsoft SQL Server memory configuration 5

Database configuration

Microsoft SQL Server shared instance versus dedicated instance

If an application has many schemas and stored procedures, it could potentially affect other apps that share a SQL Server instance.

Instance resources could potentially become divided or locked, which in turn causes performance issues for other apps that have databases hosted on the shared SQL Server instance.

Troubleshooting performance issues can be complicated because you must figure out which instance is the root cause. This question is weighed against the costs of operating system licenses and SQL Server licenses. If application performance is paramount, then a dedicated instance is highly recommended.

Microsoft licenses SQL Server per core at the server level and not per instance. For this reason, database administrators are tempted to install as many SQL Server instances as the server can handle to save on licensing costs, which can lead to major performance issues later.

NetApp recommends choosing dedicated SQL Server instances whenever possible to obtain optimal performance.

CPU configuration

Modifying server and database settings can help achieve better and more efficient database performance.

Hyperthreading

Hyperthreading is Intel's proprietary simultaneous multithreading (SMT) implementation, which improves parallelization of computations (multitasking) performed on x86 microprocessors.

Hardware that uses hyperthreading allows the logical hyperthread CPUs to appear as physical CPUs to the operating system. SQL Server then sees the physical CPUs, which the operating system presents, and so can use the hyperthreaded processors.

The caveat here is that each SQL Server version has its own limitations on the compute power it can use. For more information, see Compute Capacity Limits by Edition of SQL Server.

There are two main schools of thought when licensing SQL Server. The first is known as a server + client access license (CAL) model; the second is the per processor core model. Although you can access all the product features available in SQL Server with the server + CAL strategy, there is a hardware limit of 20 CPU cores per socket. Even if you have SQL Server Enterprise Edition + CAL for a server with more than 20 CPU cores per socket, the application cannot use all those cores at a time on that instance. Figure shows the SQL Server log message after startup indicating the enforcement of the core limit.

Log entries indicate number of cores being used after SQL Server startup.

```
2017-01-11 07:16:30.71 Server      Microsoft SQL Server 2016
(RTM) - 13.0.1601.5 (X64)
     Apr 29 2016 23:23:58
     Copyright (c) Microsoft Corporation
     Enterprise Edition (64-bit) on Windows Server 2016
Datacenter 6.3 <X64> (Build 14393: )

2017-01-11 07:16:30.71 Server      UTC adjustment: -8:00
2017-01-11 07:16:30.71 Server      (c) Microsoft Corporation.
2017-01-11 07:16:30.71 Server      All rights reserved.
2017-01-11 07:16:30.71 Server      Server process ID is 10176.
2017-01-11 07:16:30.71 Server      System Manufacturer:
'FUJITSU', System Model: 'PRIMERGY RX2540 M1'.
2017-01-11 07:16:30.71 Server      Authentication mode is MIXED.
2017-01-11 07:16:30.71 Server      Logging SQL Server messages
in file 'C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\Log\ERRORLOG'.
2017-01-11 07:16:30.71 Server      The service account is 'SEA-
TM\FUJIA2R30$'. This is an informational message; no user action
is required.
2017-01-11 07:16:30.71 Server      Registry startup parameters:
     -d C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\DATA\master.mdf
     -e C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\Log\ERRORLOG
     -l C:\Program Files\Microsoft SQL Server
\MSSQL13.MSSQLSERVER\MSSQL\DATA\mastlog.ldf
     -T 3502
     -T 834
2017-01-11 07:16:30.71 Server      Command Line Startup
Parameters:
     -s "MSSQLSERVER"
2017-01-11 07:16:30.72 Server      SQL Server detected 2 sockets
with 18 cores per socket and 36 logical processors per socket,
72 total logical processors; using 40 logical processors based
on SQL Server licensing. This is an informational message; no
user action is required.
2017-01-11 07:16:30.72 Server      SQL Server is starting at
```

Therefore, to use all CPUs, you should use the per-processor core license. For detailed information about SQL Server licensing, see SQL Server 2022: Your modern data platform.

CPU affinity

You are unlikely ever to need to alter the processor affinity defaults unless you encounter performance problems, but it is still worth understanding what they are and how they work.

SQL Server supports processor affinity by two options:

- CPU affinity mask
- Affinity I/O mask

SQL Server uses all CPUs available from the operating system (if the per-processor core license is chosen). It creates schedulers on all the CPUs to make best use of the resources for any given workload. When multitasking, the operating system or other applications on the server can switch process threads from one processor to another. SQL Server is a resource-intensive application, and so performance can be affected when this occurs. To minimize the effect, you can configure the processors such that all the SQL Server load is directed to a preselected group of processors. This is achieved by using the CPU affinity mask.

The affinity I/O mask option binds SQL Server disk I/O to a subset of CPUs. In SQL Server OLTP environments, this extension can enhance the performance of SQL Server threads issuing I/O operations.

Max Degree of Parallelism (MAXDOP)

By default, SQL Server uses all available CPUs during query execution (if the per-processor core license chosen).

Although this is great for large queries, it can cause performance problems and limit concurrency. A better approach is to limit parallelism to the number of physical cores in a single CPU socket. For example, on a server with two physical CPU sockets with 12 cores per socket, regardless of hyperthreading, MAXDOP should be set to 12. MAXDOP cannot restrict or dictate which CPU is to be used. Instead, it restricts the number of CPUs that can be used by a single batch query.

NetApp recommends for DSS such as data warehouses, start with this setting at 50 or so and tuning up or down as appropriate. Make sure you measure for the critical queries in your application and adjust if necessary.

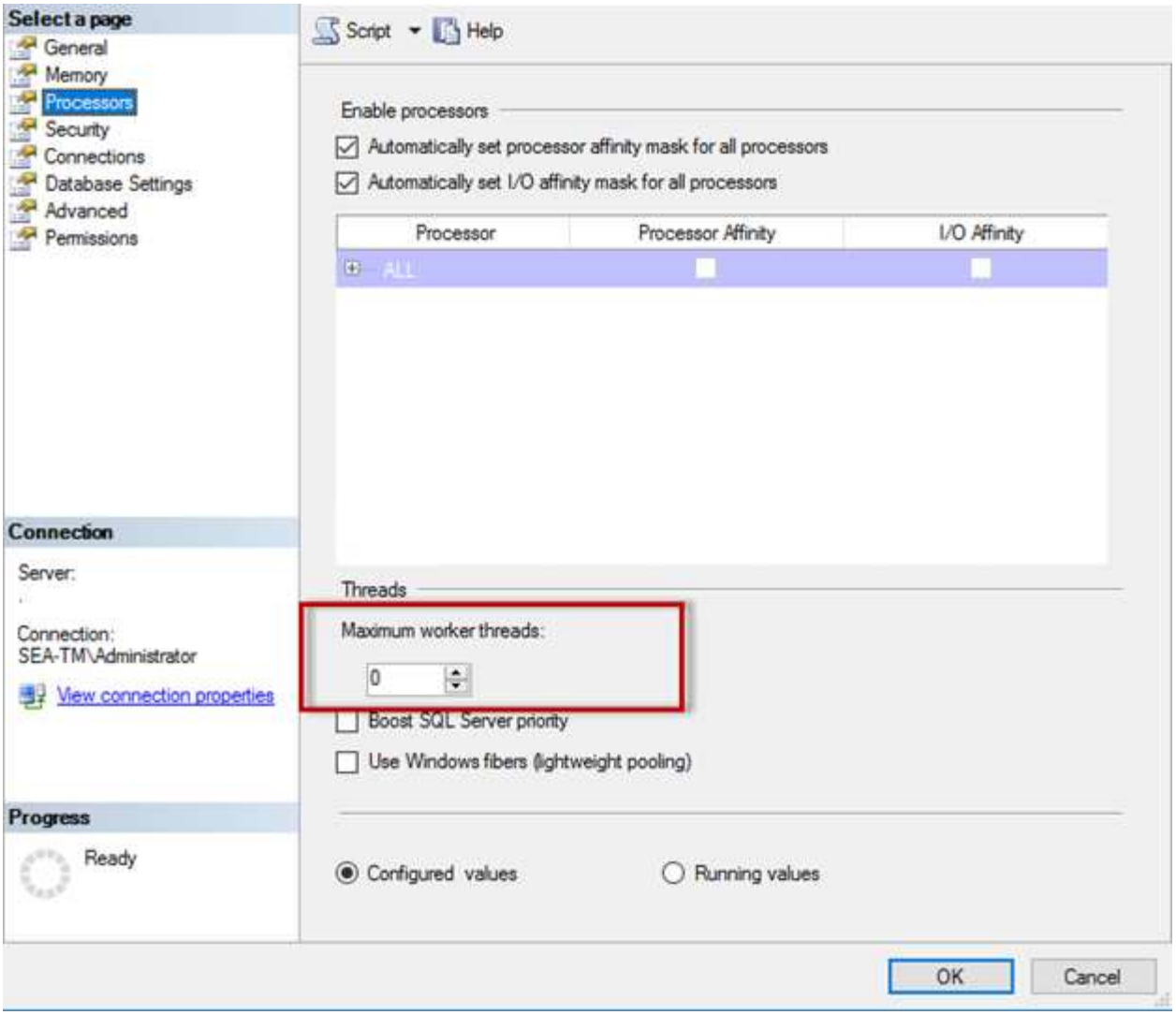
Max worker threads

The max worker threads option helps to optimize performance when large numbers of clients are connected to SQL Server.

Normally, a separate operating system thread is created for each query request. If hundreds of simultaneous connections are made to SQL Server, then one thread per query request consumes large amounts of system resources. The max worker threads option helps improve performance by enabling SQL Server to create a pool of worker threads to service a larger number of query requests.

The default value is 0, which allows SQL Server to automatically configure the number of worker threads at startup. This works for most systems. Max worker threads is an advanced option and should not be altered without assistance from an experienced database administrator (DBA).

When should you configure SQL Server to use more worker threads? If the average work queue length for each scheduler is above 1, you might benefit from adding more threads to the system, but only if the load is not CPU-bound or experiencing any other heavy waits. If either of those is happening, adding more threads does not help because they end up waiting for other system bottlenecks. For more information about max worker threads, see Configure the max worker threads Server Configuration Option.



Configuring max worker threads using SQL Server Management Studio.

```
The following example shows how to configure the max work threads option
using T-SQL.

EXEC sp_configure 'show advanced options', 1;

GO

RECONFIGURE ;

GO

EXEC sp_configure 'max worker threads', 900 ;

GO

RECONFIGURE;

GO
```

Microsoft SQL Server memory configuration

The following sections explain some of the critical memory configuration settings.

Max server memory

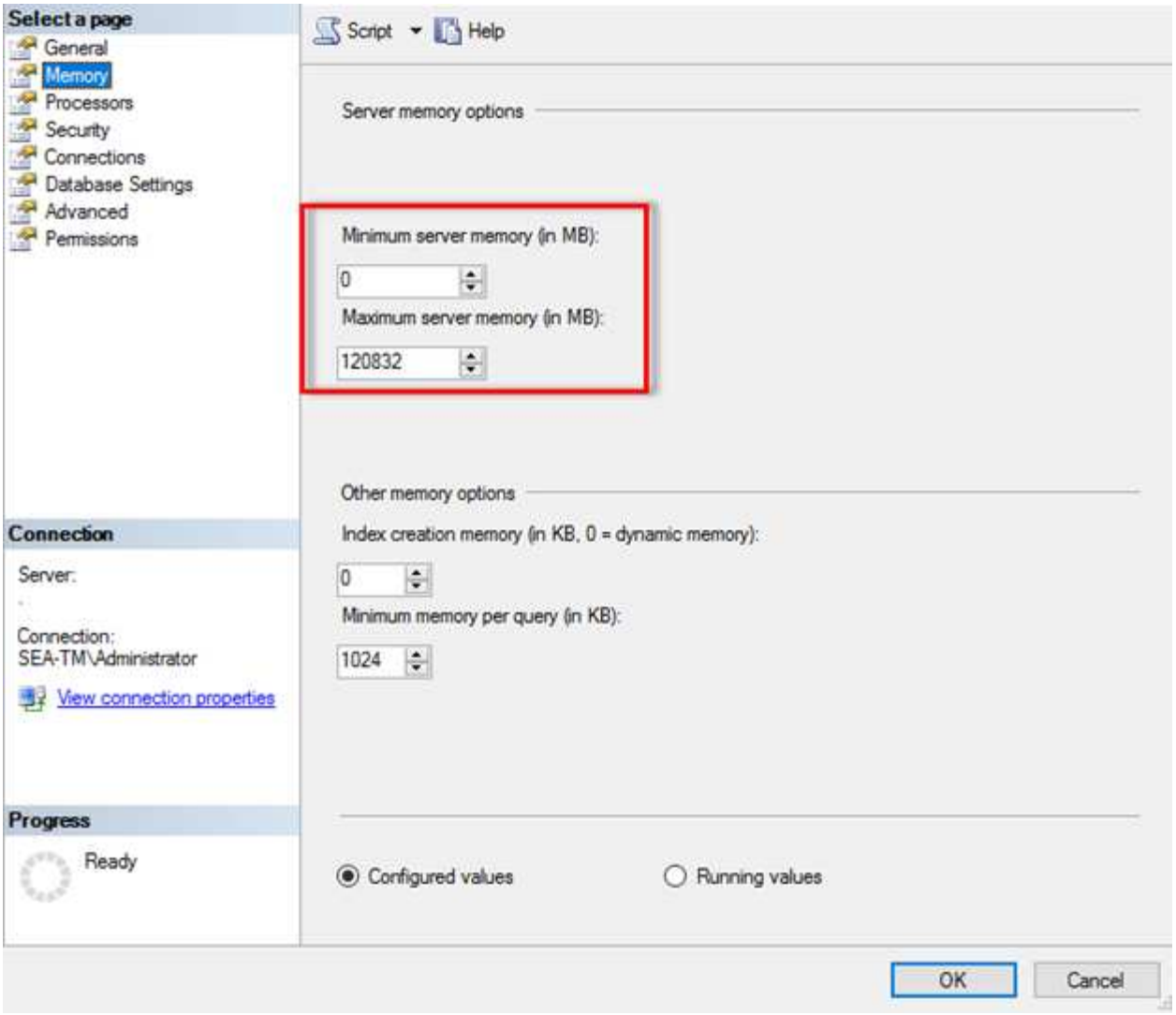
The max server memory option sets the maximum amount of memory that the SQL Server instance can use.

It is generally used if multiple applications are running on the same server where SQL Server is running and you want to guarantee that these applications have sufficient memory to function properly.

Some applications only use whatever memory is available when they start and do not request more even if needed. That is where the max server memory setting comes into play.

On a SQL Server cluster with several SQL Server instances, each instance could be competing for resources. Setting a memory limit for each SQL Server instance can help guarantee best performance for each instance.

NetApp recommends leaving at least 4GB to 6GB of RAM for the operating system to avoid performance issues.



Adjusting minimum and maximum server memory using SQL Server Management Studio.

Using SQL Server Management Studio to adjust minimum or maximum server memory requires a restart of the SQL Server service. You can adjust server memory using transact SQL (T-SQL) using this code:

```
EXECUTE sp_configure 'show advanced options', 1

GO

EXECUTE sp_configure 'min server memory (MB)', 2048

GO

EXEC sp_configure 'max server memory (MB)', 120832

GO

RECONFIGURE WITH OVERRIDE
```

Nonuniform memory access

Nonuniform memory access (NUMA) is a memory-access optimization method that helps increase processor speed without increasing the load on the processor bus.

If NUMA is configured on the server where SQL Server is installed, no additional configuration is required because SQL Server is NUMA aware and performs well on NUMA hardware.

Index create memory

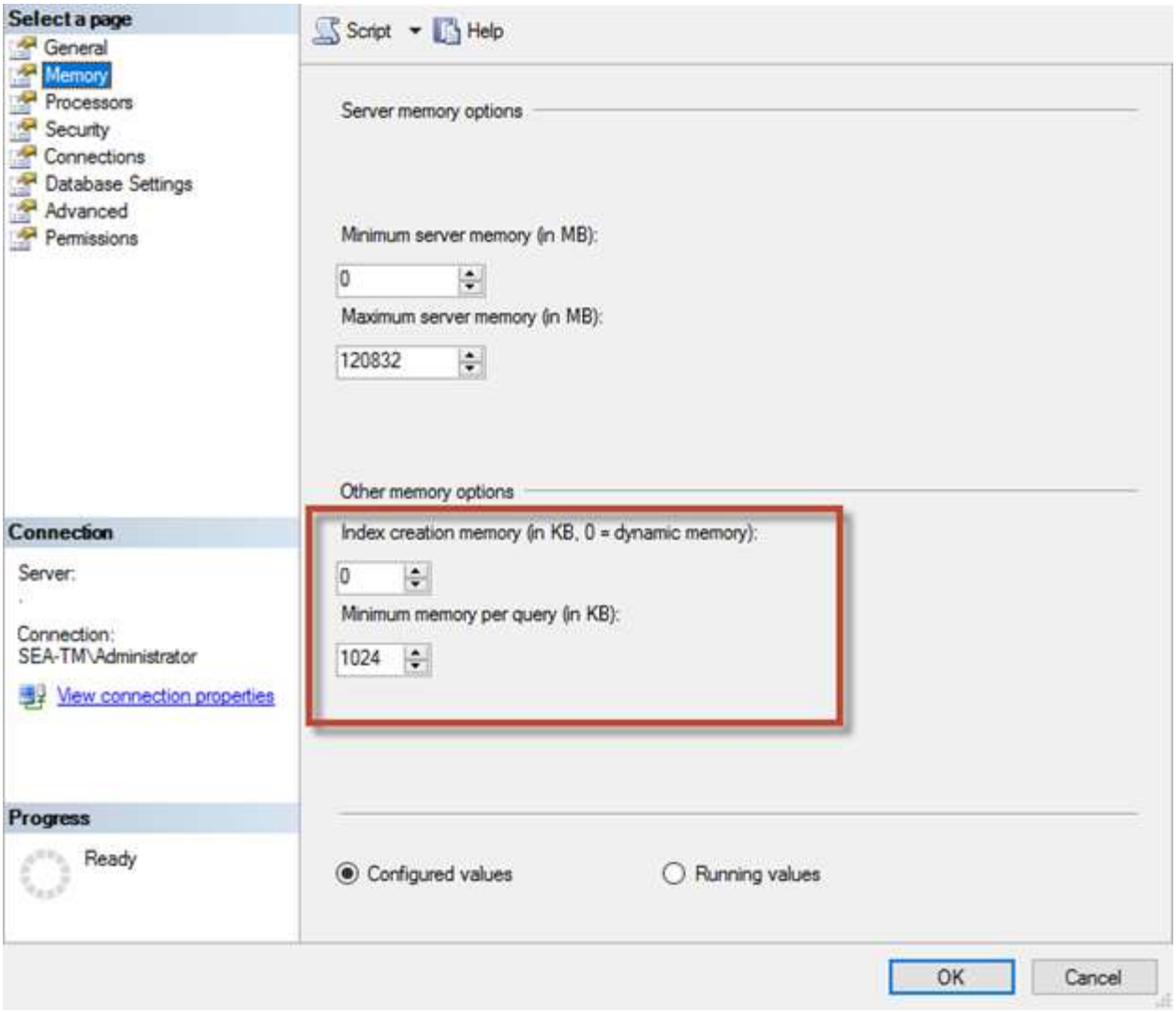
The index create memory option is another advanced option that you should not usually change.

It controls the maximum amount of RAM initially allocated for creating indexes. The default value for this option is 0, which means that it is managed by SQL Server automatically. However, if you encounter difficulties creating indexes, consider increasing the value of this option.

Min memory per query

When a query is run, SQL Server tries to allocate the optimum amount of memory for it to run efficiently.

By default, the min memory per query setting allocates > or = to 1024KB for each query to run. It is a best practice is to leave this setting at the default value of 0 to allow SQL Server to dynamically manage the amount of memory allocated for index creation operations. However, if SQL Server has more RAM than it needs to run efficiently, the performance of some queries can be boosted if you increase this setting. Therefore, as long as memory is available on the server that is not being used by SQL Server, any other applications, or the operating system, then boosting this setting can help overall SQL Server performance. If no free memory is available, increasing this setting might hurt overall performance.



Buffer pool extensions

The buffer pool extension provides seamless integration of an NVRAM extension with the database engine buffer pool to significantly improve I/O throughput.

The buffer pool extension is not available in every SQL Server edition. It is available only with the 64-bit SQL Server Standard, Business Intelligence, and Enterprise editions.

The buffer pool extension feature extends the buffer pool cache with nonvolatile storage (usually SSDs). The extension allows the buffer pool to accommodate a larger database working set, forcing the paging of I/O between the RAM and the SSDs and effectively offloading small random I/Os from mechanical disks to SSDs. Because of the lower latency and better random I/O performance of SSDs, the buffer pool extension significantly improves I/O throughput.

The buffer pool extension feature offers the following benefits:

- Increased random I/O throughput
- Reduced I/O latency
- Increased transaction throughput
- Improved read performance with a larger hybrid buffer pool

- A caching architecture that can take advantage of existing and future low-cost memory

NetApp recommends configuring the buffer pool extensions to:

- Make sure that an SSD-backed LUN (such as NetApp AFF) is presented to the SQL Server host so that it can be used as a buffer pool extension target disk.
- The extension file must be the same size as or larger than the buffer pool.

The following example shows a T-SQL command to set up a buffer pool extension of 32GB.

```
USE master

GO

ALTER SERVER CONFIGURATION

SET BUFFER POOL EXTENSION ON

  (FILENAME = 'P:\BUFFER POOL EXTENSION\SQLServerCache.BUFFER POOL
EXTENSION', SIZE = 32 GB);

GO
```

Copyright information

Trademark information