You can also manage the configuration values for min server memory and max server memory by using the `sp_configure` system stored procedure. To see the configuration values for these parameters, execute the `sp_configure` stored procedure as follows:

```
EXEC sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
EXEC sp_configure  'min server memory';
EXEC sp_configure  'max server memory';
```

Figure 2-4 shows the result of running these commands.

	name	minimum	maximum	config_value	run_value
1	min server memory (MB)	0	2147483647	0	16

	name	minimum	maximum	config_value	run_value
1	max server memory (MB)	128	2147483647	2147483647	2147483647

Figure 2-4. *SQL Server memory configuration properties*

Note that the default value for the min server memory setting is 0MB and for the max server memory setting is 2147483647MB.

You can also modify these configuration values using the `sp_configure` stored procedure. For example, to set max server memory to 10GB and min server memory to 5GB, execute the following set of statements (`setmemory.sql` in the download):

```
USE master;
EXEC sp_configure  'show advanced option',   1;
RECONFIGURE;
exec sp_configure  'min server memory (MB)',  5120;
exec sp_configure  'max server memory (MB)',  10240;
RECONFIGURE WITH OVERRIDE;
```

The min server memory and max server memory configurations are classified as advanced options. By default, the `sp_configure` stored procedure does not affect/display the advanced options. Setting `show advanced option` to 1 as shown previously enables the `sp_configure` stored procedure to affect/display the advanced options.

The RECONFIGURE statement updates the memory configuration values set by `sp_configure`. Since ad hoc updates to the system catalog containing the memory configuration values are not recommended, the OVERRIDE flag is used with the RECONFIGURE statement to force the memory configuration. If you do the memory configuration through Management Studio, Management Studio automatically executes the RECONFIGURE WITH OVERRIDE statement after the configuration setting.

Another way to see the settings but not to manipulate them is to use the `sys.configurations` system view. You can select from `sys.configurations` using standard T-SQL rather than having to execute a command.

You may need to allow for SQL Server sharing a system's memory. To elaborate, consider a computer with SQL Server and SharePoint running on it. Both servers are heavy users of memory and thus keep pushing each other for memory. The dynamic memory behavior of SQL Server allows it to release memory to SharePoint at one instance and grab it back as SharePoint releases it. You can avoid this dynamic memory management overhead by configuring SQL Server for a fixed memory size. However, please keep in mind that since SQL Server is an extremely resource-intensive process, it is highly recommended that you have a dedicated SQL Server production machine.

Now that you understand SQL Server memory management at a very high level, let's consider the performance counters you can use to analyze stress on memory, as shown in Table 2-1.

Table 2-1. *Performance Monitor Counters to Analyze Memory Pressure*

Object(Instance [,InstanceN])	Counter	Description	Values
Memory	Available Bytes	Free physical memory	System dependent
	Pages/sec	Rate of hard page faults	Compare with baseline
	Page Faults/sec	Rate of total page faults	Compare with its baseline value for trend analysis
	Pages Input/sec	Rate of input page faults	
	Pages Output/sec	Rate of output page faults	
	Paging File %Usage Peak	Peak values in the memory paging file	
	Paging File: %Usage	Rate of usage of the memory paging file	
SQLServer: Buffer Manager	Buffer cache hit ratio	Percentage of requests served out of buffer cache	Compare with its baseline value for trend analysis
	Page Life Expectancy	Time page spends in buffer cache	Compare with its baseline value for trend analysis
	Checkpoint Pages/sec	Pages written to disk by checkpoint	Compare with baseline
	Lazy writes/sec	Dirty aged pages flushed from buffer	Compare with baseline
SQLServer: Memory Manager	Memory Grants Pending	Number of processes waiting for memory grant	Average value = 0
	Target Server Memory (KB)	Maximum physical memory SQL Server can have on the box	Close to size of physical memory
	Total Server Memory (KB)	Physical memory currently assigned to SQL	Close to target server memory (KB)
Process	Private Bytes	Size, in bytes, of memory that this process has allocated that can't be shared with other processes	

Memory and disk I/O are closely related. Even if you think you have a problem that is directly memory related, you should also gather I/O metrics to understand how the system is behaving between the two resources. I'll now walk you through these counters to give you a better idea of possible uses.

Available Bytes

The Available Bytes counter represents free physical memory in the system. You can also look at Available Kbytes and Available Mbytes for the same data but with less granularity. For good performance, this counter value should not be too low. If SQL Server is configured for dynamic memory usage, then this value will be controlled by calls to a Windows API that determines when and how much memory to release. Extended periods of time with this value very low and SQL Server memory not changing indicates that the server is under severe memory stress.

Pages/Sec and Page Faults/Sec

To understand the importance of the Pages/sec and Page Faults/sec counters, you first need to learn about page faults. A *page fault* occurs when a process requires code or data that is not in its *working set* (its space in physical memory). It may lead to a soft page fault or a hard page fault. If the faulted page is found elsewhere in physical memory, then it is called a *soft page fault*. A *hard page fault* occurs when a process requires code or data that is not in its working set or elsewhere in physical memory and must be retrieved from disk.

The speed of a disk access is in the order of milliseconds for mechanical drives or as low as .1 milliseconds for a solid-state drive (SSD), whereas a memory access is in the order of nanoseconds. This huge difference in the speed between a disk access and a memory access makes the effect of hard page faults significant compared to that of soft page faults.

The Pages/sec counter represents the number of pages read from or written to disk per second to resolve hard page faults. The Page Faults/sec performance counter indicates the total page faults per second—soft page faults plus hard page faults—handled by the system. These are primarily measures of load and are not direct indicators of performance issues.

Hard page faults, indicated by Pages/sec, should not be consistently higher than normal. There are no hard-and-fast numbers for what indicates a problem because these numbers will vary widely between systems based on the amount and type of memory as well as the speed of disk access on the system.

If the Pages/sec counter is high, you can break it up into Pages Input/sec and Pages Output/sec.

- *Pages Input/sec*: An application will wait only on an input page, not on an output page.
- *Pages Output/sec*: Page output will stress the system, but an application usually does not see this stress. Pages output are usually represented by the application's dirty pages that need to be backed out to the disk. Pages Output/sec is an issue only when disk load become an issue.

Also, check Process:Page Faults/sec to find out which process is causing excessive paging in case of high Pages/sec. The Process object is the system component that provides performance data for the processes running on the system, which are individually represented by their corresponding instance name.

For example, the SQL Server process is represented by the sqlservr instance of the Process object. High numbers for this counter usually do not mean much unless Pages/sec is high. Page Faults/sec can range all over the spectrum with normal application behavior, with values from 0 to 1,000 per second being acceptable. This entire data set means a baseline is essential to determine the expected normal behavior.

Paging File %Usage and Page File %Usage

All memory in the Windows system is not the physical memory of the physical machine. Windows will swap memory that isn't immediately active in and out of the physical memory space to a paging file. These counters are used to understand how often this is occurring on your system. As a general measure of system performance, these counters are applicable only to the Windows OS and not to SQL Server. However, the impact of not enough virtual memory will affect SQL Server. These counters are collected to understand whether the memory pressures on SQL Server are internal or external. If they are external memory pressures, you will need to go into the Windows OS to determine what the problems might be.

Buffer Cache Hit Ratio

The *buffer cache* is the pool of buffer pages into which data pages are read, and it is often the biggest part of the SQL Server memory pool. This counter value should be as high as possible, especially for OLTP systems that should have fairly regimented data access, unlike a warehouse or reporting system. It is extremely common to find this counter value as 99 percent or more for most production servers. A low Buffer cache hit ratio value indicates that few requests could be served out of the buffer cache, with the rest of the requests being served from disk.

When this happens, either SQL Server is still warming up or the memory requirement of the buffer cache is more than the maximum memory available for its growth. If the cache hit ratio is consistently low, you might consider getting more memory for the system or reducing memory requirements through the use of good indexes and other query tuning mechanisms, that is, unless you're dealing with reporting systems with lots of ad hoc queries. It's possible when working with reporting systems to consistently see the cache hit ratio become extremely low.

This makes the buffer cache hit ratio an interesting number for understanding aspects of system behavior, but it is not a value that would suggest, by itself, potential performance problems. While this number represents an interesting behavior within the system, it's not a great measure for precise problems but instead shows a type of behavior. For more details on this topic, please read the "Great SQL Server Debates: Buffer Cache Hit Ratio" article on Simple-Talk (`https://bit.ly/2rzWJv0`).

Page Life Expectancy

Page Life Expectancy indicates how long a page will stay in the buffer pool without being referenced. Generally, a low number for this counter means that pages are being removed from the buffer, lowering the efficiency of the cache and indicating the possibility of memory pressure. On reporting systems, as opposed to OLTP systems, this number may remain at a lower value since more data is accessed from reporting systems. It's also common to see Page Life Expectancy fall to very low levels during nightly loads. Since this is dependent on the amount of memory you have available and the types of queries running on your system, there are no hard-and-fast numbers that will satisfy a wide audience. Therefore, you will need to establish a baseline for your system and monitor it over time.

If you are on a machine with nonuniform memory access (NUMA) , you need to know that the standard Page Life Expectancy counter is an average. To see specific measures, you'll need to use the Buffer Node:Page Life Expectancy counter.

Checkpoint Pages/Sec

The Checkpoint Pages/sec counter represents the number of pages that are moved to disk by a checkpoint operation. These numbers should be relatively low, for example, less than 30 per second for most systems. A higher number means more pages are being marked as dirty in the cache. A *dirty page* is one that is modified while in the buffer. When it's modified, it's marked as dirty and will get written back to the disk during the next checkpoint. Higher values on this counter indicate a larger number of writes occurring within the system, possibly indicative of I/O problems. But, if you are taking advantage of indirect checkpoints, which allow you to control when checkpoints occur in order to reduce recovery intervals, you might see different numbers here. Take that into account when monitoring databases with the indirect checkpoint configured. For more information about checkpoints on SQL Server 2016 and better, I suggest you read the "Changes in SQL Server 2016 Checkpoint Behavior" article on MSDN (`https://bit.ly/2pdggk3`).

Lazy Writes/Sec

The Lazy writes/sec counter records the number of buffers written each second by the buffer manager's lazy write process. This process is where the dirty, aged buffers are removed from the buffer by a system process that frees up the memory for other uses. A dirty, aged buffer is one that has changes and needs to be written to the disk. Higher values on this counter possibly indicate I/O issues or even memory problems. The Lazy writes/sec values should consistently be less than 20 for the average system. However, as with all other counters, you must compare your values to a baseline measure.

Memory Grants Pending

The Memory Grants Pending counter represents the number of processes pending for a memory grant within SQL Server memory. If this counter value is high, then SQL Server is short of buffer memory, which can be caused not simply by a lack of memory but by issues such as oversized memory grants caused by incorrect row counts because your statistics are out-of-date. Under normal conditions, this counter value should consistently be 0 for most production servers.

Another way to retrieve this value, on the fly, is to run queries against the DMV `sys.dm_exec_query_memory_grants`. A `null` value in the column `grant_time` indicates that the process is still waiting for a memory grant. This is one method you can use to troubleshoot query timeouts by identifying that a query (or queries) is waiting on memory in order to execute.

Target Server Memory (KB) and Total Server Memory (KB)

Target Server Memory (KB) indicates the total amount of dynamic memory SQL Server is willing to consume. Total Server Memory (KB) indicates the amount of memory currently assigned to SQL Server. The Total Server Memory (KB) counter value can be very high if the system is dedicated to SQL Server. If Total Server Memory (KB) is much less than Target Server Memory (KB), then either the SQL Server memory requirement is low, the max server memory configuration parameter of SQL Server is set at too low a value, or the system is in warm-up phase. The *warm-up phase* is the period after SQL Server is started when the database server is in the process of expanding its memory allocation dynamically as more data sets are accessed, bringing more data pages into memory.

You can confirm a low memory requirement from SQL Server by the presence of a large number of free pages, usually 5,000 or more. Also, you can directly check the status of memory by querying the DMV `sys.dm_os_ring_buffers`, which returns information about memory allocation within SQL Server. I cover `sys.dm_os_ring_buffers` in more detail in the following section.

Additional Memory Monitoring Tools

While you can get the basis for the behavior of memory within SQL Server from the Performance Monitor counters, once you know that you need to spend time looking at your memory usage, you'll need to take advantage of other tools and tool sets. The following are some of the commonly used reference points for identifying memory issues on a SQL Server system. A few of these tools are only of use for in-memory OLTP management. Some of these tools, while actively used by large numbers of the SQL Server community, are not documented within SQL Server Books Online. This means they are absolutely subject to change or removal.

DBCC MEMORYSTATUS

This command goes into the SQL Server memory and reads out the current allocations. It's a moment-in-time measurement, a snapshot. It gives you a set of measures of where memory is currently allocated. The results from running the command come back as two basic result sets, as you can see in Figure 2-5.

	Process/System Counts	Value
1	Available Physical Memory	6080708608
2	Available Virtual Memory	8779246280704
3	Available Paging File	6996623360
4	Working Set	231723008
5	Percent of Committed Memory in WS	100
6	Page Faults	4199150
7	System physical memory high	1
8	System physical memory low	0
9	Process physical memory low	0
10	Process virtual memory low	0

	Memory Manager	KB
1	VM Reserved	16048564
2	VM Committed	181924
3	Locked Pages Allocated	0
4	Large Pages Allocated	0
5	Emergency Memory	1024
6	Emergency Memory In Use	16
7	Target Committed	5733864
8	Current Committed	181928
9	Pages Allocated	118376
10	Pages Reserved	0
11	Pages Free	2680
12	Pages In Use	146488
13	Page Alloc Potential	6455960
14	NUMA Growth Phase	0
15	Last OOM Factor	0
16	Last OS Error	0

Figure 2-5. *Output of DBCC MEMORYSTATUS*

The first data set shows basic allocations of memory and counts of occurrences. For example, Available Physical Memory is a measure of the memory available on the system, whereas Page Faults is just a count of the number of page faults that have occurred.

The second data set shows different memory managers within SQL Server and the amount of memory they have consumed at the moment that the `MEMORYSTATUS` command was called.

Each of these can be used to understand where memory allocation is occurring within the system. For example, in most systems, most of the time the primary consumer of memory is the buffer pool. You can compare the Target Committed value to the Current Committed value to understand if you're seeing pressure on the buffer pool. When Target Committed is higher than Current Committed, you might be seeing buffer cache problems and need to figure out which process within your currently executing SQL Server processes is using the most memory. This can be done using a dynamic management object, `sys.dm_os_performance_counters`.

The remaining data sets are various memory managers, memory clerks, and other memory stores from the full dump of memory that DBCC MEMORYSTATUS produces. They're only going to be interesting in narrow circumstances when dealing with particular aspects of SQL Server management, and they fall far outside the scope of this book to document them all. You can read more in the MSDN article "How to use the DBCC MEMORYSTATUS command" (`http://bit.ly/1eJ2M2f`).

Dynamic Management Views

There are a large number of memory-related DMVs within SQL Server. Several of them have been updated with SQL Server 2017, and some new ones have been added. Reviewing all of them is outside the scope of this book. There are three that are the most frequently used when determining whether you have memory bottlenecks within SQL Server. There are also another two that are useful when you need to monitor your in-memory OLTP memory usage.

Sys.dm_os_memory_brokers

While most of the memory within SQL Server is allocated to the buffer cache, there are a number of processes within SQL Server that also can, and will, consume memory. These processes expose their memory allocations through this DMV. You can use this to see what processes might be taking resources away from the buffer cache in the event you have other indications of a memory bottleneck.

Sys.dm_os_memory_clerks

A memory clerk is the process that allocates memory within SQL Server. Looking at what these processes are up to allows you to understand whether there are internal memory allocation issues going on within SQL Server that might rob the procedure cache of needed memory. If the Performance Monitor counter for Private Bytes is high, you can determine which parts of the system are being consumed through the DMV.

If you have a database using in-memory OLTP storage, you can use `sys.dm_db_xtp_table_memory_stats` to look at the individual database objects. But if you want to look at the allocations of these objects across the entire instance, you'll need to use `sys.dm_os_memory_clerks`.

Sys.dm_os_ring_buffers

This DMV is not documented within Books Online, so it is subject to change or removal. It changed between SQL Server 2008R2 and SQL Server 2012. The queries I normally run against it still seem to work for SQL Server 2017, but you can't count on that. This DMV outputs as XML. You can usually read the output by eye, but you may need to implement XQuery to get really sophisticated reads from the ring buffers.

A ring buffer is nothing more than a recorded response to a notification. Ring buffers are kept within this DMV, and accessing `sys.dm_os_ring_buffers` allows you to see things changing within your memory. Table 2-2 describes the main ring buffers associated with memory.

Table 2-2. *Main Ring Buffers Associated with Memory*

Ring Buffer	Ring_buffer_type	Use
Resource Monitor	RING_BUFFER_RESOURCE_MONITOR	As memory allocation changes, notifications of this change are recorded here. This information can be useful for identifying external memory pressure.
Out Of Memory	RING_BUFFER_OOM	When you get out-of-memory issues, they are recorded here so you can tell what kind of memory action failed.
Memory Broker	RING_BUFFER_MEMORY_BROKER	As the memory internal to SQL Server drops, a low memory notification will force processes to release memory for the buffer. These notifications are recorded here, making this a useful measure for when internal memory pressure occurs.
Buffer Pool	RING_BUFFER_BUFFER_POOL	Notifications of when the buffer pool itself is running out of memory are recorded here. This is just a general indication of memory pressure.

There are other ring buffers available, but they are not applicable to memory allocation issues.

Sys.dm_db_xtp_table_memory_stats

To see the memory in use by the tables and indexes that you created in-memory, you can query this DMV. The output measures the memory allocated and memory used for the tables and indexes. It outputs only the `object_id`, so you'll need to also query the system view `sys.objects` to get the names of tables or indexes. This DMV outputs for the database you are currently connected to when querying.

Sys.dm_xtp_system_memory_consumers

This DMV shows system structures that are used to manage the internals of the in-memory engine. It's not something you should normally have to deal with, but when troubleshooting memory issues, it's good to understand if you're dealing directly with something occurring within the system or just the amount of data that you've loaded into memory. The principal measures you'd be looking for here are the allocated and used bytes shown for each of the management structures.

Monitoring Memory in Linux

You won't have Perfmon within the Linux operating system. However, this doesn't mean you can't observe memory behavior on the server to understand how the system is behaving. You can query the DMVs `sys.dm_os_performance_counters` and `sys.dm_os_wait_stats` within a SQL Server 2017 instance running on Linux to observe memory behavior in that way.

Additional monitoring of the Linux OS can be done through native Linux tools. There are a large number of them, but a commonly used one is Grafana. It's open source with lots of documentation available online. The SQL Server Customer Advisory Team has a documented method for monitoring Linux that I can recommend: `http://bit.ly/2wi73bA`.

Memory Bottleneck Resolutions

When there is high stress on memory, indicated by a large number of hard page faults, you can resolve a memory bottleneck using the flowchart shown in Figure 2-6.

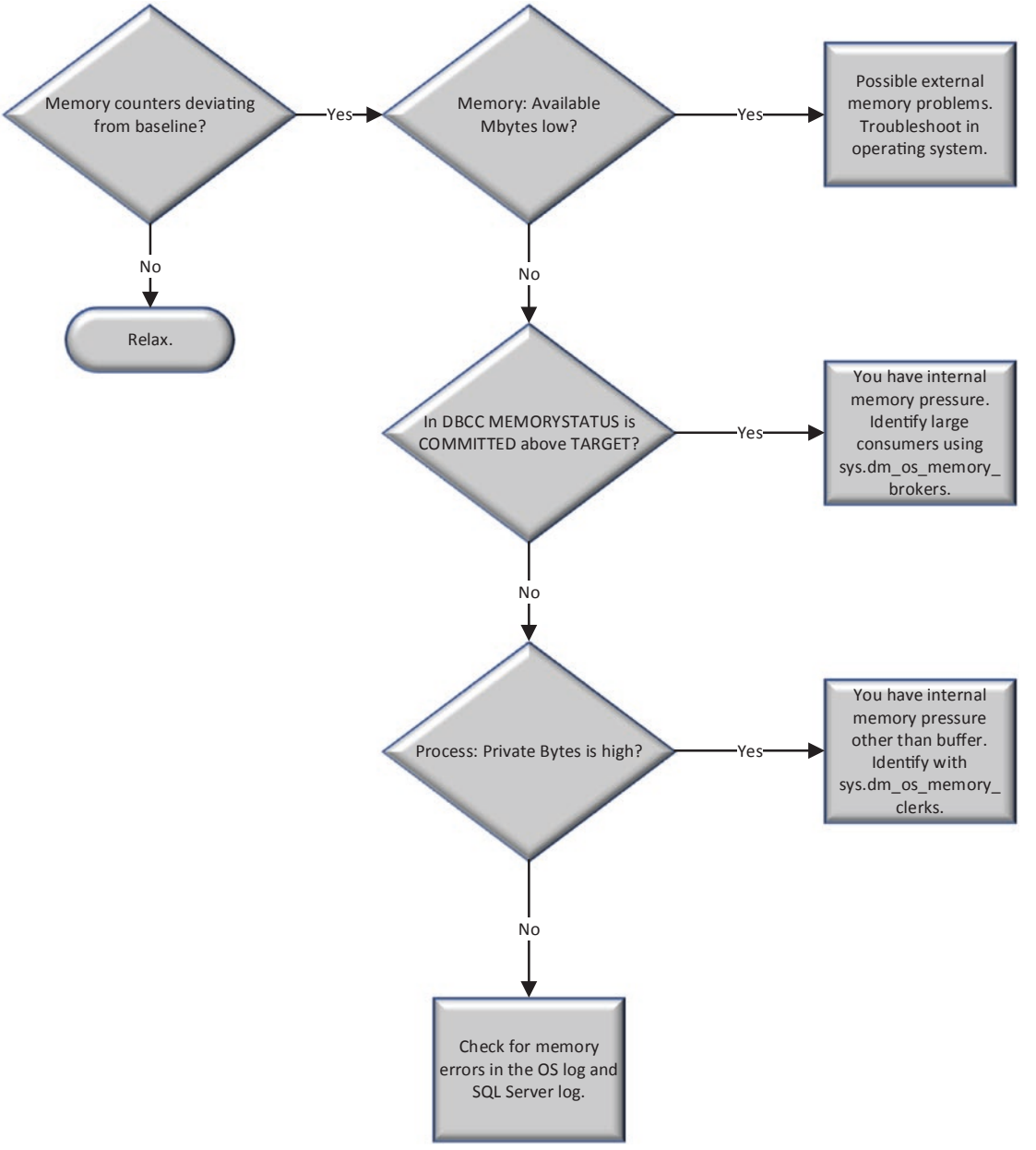

Figure 2-6. *Memory bottleneck resolution chart*

A few of the common resolutions for memory bottlenecks are as follows:

- Optimizing application workload
- Allocating more memory to SQL Server

- Moving in-memory tables back to standard storage
- Increasing system memory
- Changing from a 32-bit to a 64-bit processor
- Enabling 3GB of process space
- Compressing data
- Addressing fragmentation

Of course, fixing any of the query issues that can lead to excessive memory use is always an option. Let's take a look at each of these in turn.

Optimizing Application Workload

Optimizing application workload is the most effective resolution most of the time, but because of the complexity and challenges involved in this process, it is usually considered last. To identify the memory-intensive queries, capture all the SQL queries using Extended Events (which you will learn how to use in Chapter 6) or use Query Store (which we'll cover in Chapter 11) and then group the output on the `Reads` column. The queries with the highest number of logical reads contribute most often to memory stress, but there is not a linear correlation between the two. You can also use `sys.dm_exec_query_stats`, a DMV that collects query metrics for queries that are actively in cache to identify the same thing. But, since this DMV is based on cache, it may not be as accurate as capturing metrics using Extended Events, although it will be quicker and easier. You will see how to optimize those queries in more detail throughout this book.

Allocating More Memory to SQL Server

As you learned in the "SQL Server Memory Management" section, the max server memory configuration can limit the maximum size of the SQL Server buffer memory pool. If the memory requirement of SQL Server is more than the max server memory value, which you can tell through the number of hard page faults, then increasing the value will allow the memory pool to grow. To benefit from increasing the max server memory value, ensure that enough physical memory is available in the system.

If you are using in-memory OLTP storage, you may need to adjust the memory percentages allocated to the resource pools you have defined for your in-memory objects. But, that will take memory from other parts of your SQL Server instance.

Moving In-Memory Tables Back to Standard Storage

Introduced in SQL Server 2014, a new table type was introduced called the *in-memory* table. This moves the storage of tables from the disk to memory, radically improving the performance. But, not all tables or all workloads will benefit from this new functionality. You need to keep an eye on your general query performance metrics for in-memory tables and take advantage of the specific DMVs that let you monitor the in-memory tables. I'll be covering all this in detail in Chapter 24. If your workload doesn't work well with in-memory tables or you just don't have enough memory in the system, you may need to move those in-memory tables back to disk storage.

Increasing System Memory

The memory requirement of SQL Server depends on the total amount of data processed by SQL activities. It is not directly correlated to the size of the database or the number of incoming SQL queries. For example, if a memory-intensive query performs a cross join between two small tables without any filter criteria to narrow down the result set, it can cause high stress on the system memory.

One of the easiest and quickest resolutions is to simply increase system memory by purchasing and installing more. However, it is still important to find out what is consuming the physical memory because if the application workload is extremely memory intensive, you could soon be limited by the maximum amount of memory a system can access. To identify which queries are using more memory, query the `sys.dm_exec_query_memory_grants` DMV and collect metrics on queries and their I/O use. Just be careful when running queries against this DMV using a JOIN or ORDER BY statement; if your system is already under memory stress, these actions can lead to your query needing its own memory grant.

Changing from a 32-Bit to a 64-Bit Processor

Switching the physical server from a 32-bit processor to a 64-bit processor (and the attendant Windows Server software upgrade) radically changes the memory management capabilities of SQL Server. The limitations on SQL Server for memory go from 3GB to a limit of up to 24TB depending on the version of the operating system and the specific processor type.

Prior to SQL Server 2012, it was possible to add up to 64GB of data cache to a SQL Server instance through the use of Address Windowing Extensions. These were removed from SQL Server 2012, so a 32-bit instance of SQL Server is limited to accessing only 3GB of memory. Only small systems should be running 32-bit versions of SQL Server prior to 2017 because of this limitation.

SQL Server 2017 does not support the x86 chip set. You must move on to a 64-bit processor to use 2017.

Compressing Data

Data compression has a number of excellent benefits for storing and retrieving information. It has an added benefit that many people aren't aware of: while compressed information is stored in memory, it remains compressed. This means more information can be moved while using less system memory, increasing your overall memory throughput. All this does come at some cost to the CPU, so you'll need to keep an eye on that to be sure you're not just transferring stress. Sometimes you may not see much compression because of the nature of your data.

Enabling 3GB of Process Address Space

Standard 32-bit addresses can map a maximum of 4GB of memory. The standard address spaces of 32-bit Windows operating system processes are therefore limited to 4GB. Out of this 4GB process space, by default the upper 2GB is reserved for the operating system, and the lower 2GB is made available to the application. If you specify a /3GB switch in the `boot.ini` file of the 32-bit OS, the operating system reserves only 1GB of the address space, and the application can access up to 3GB. This is also called *4-gig tuning* (4GT). No new APIs are required for this purpose.

Therefore, on a machine with 4GB of physical memory and the default Windows configuration, you will find available memory of about 2GB or more. To let SQL Server use up to 3GB of the available memory, you can add the /3GB switch in the `boot.ini` file as follows:

```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
[operating systems]
```

```
multi(0)disk(0)rdisk(0)partition(1)\WINNT=
"Microsoft Windows Server 2016 Advanced Server"
/fastdetect /3GB
```

The /3GB switch should not be used for systems with more than 16GB of physical memory, as explained in the following section, or for systems that require a higher amount of kernel memory.

SQL Server 2017 on 64-bit systems can support up to 24TB on an x64 platform. It no longer makes much sense to put production systems, especially enterprise-level production systems, on 32-bit architecture, and you can't with SQL Server 2017.

Addressing Fragmentation

While fragmentation of storage may not sound like a performance issue because of how SQL Server retrieves information from disk and into memory, a page of information is accessed. If you have a high level of fragmentation, that will translate itself straight to your memory management since you have to store the pages retrieved from disk in memory as they are, empty space and all. So, while fragmentation may affect storage, it also can affect memory. I address fragmentation in Chapter 17.

Summary

In this chapter, you were introduced to the Performance Monitor and DMVs. You explored different methods of gathering metrics on memory and memory behavior within SQL Server. Understanding how memory behaves will help you understand how your system is performing. You also saw a number of possible resolutions to memory issues, other than simply buying more memory. SQL Server will make use of as much memory as you can supply it, so manage this resource well.

In the next chapter, you will be introduced to the next system bottleneck, the disk and the disk subsystems.

CHAPTER 3

Disk Performance Analysis

The disks and the disk subsystem, which includes the controllers and connectors and management software, are one of the single slowest parts of any computing system. Over the years, memory has become faster and faster. The same can be said of CPUs. But disks, except for some of the radical improvements we've seen recently with technologies such as solid-state disks (SSDs), have not changed that much; disks are still one of the slowest parts of most systems. This means you're going to want to be able to monitor your disks to understand their behavior. In this chapter, you'll explore areas such as the following:

- Using system counters to gather disk performance metrics
- Using other mechanisms of gathering disk behavior
- Resolving disk performance issues
- Differences when dealing with Linux OS and disk I/O

Disk Bottleneck Analysis

SQL Server can have demanding I/O requirements, and since disk speeds are comparatively much slower than memory and processor speeds, a contention in I/O resources can significantly degrade SQL Server performance. Analysis and resolution of any I/O path bottleneck can improve SQL Server performance significantly. As with any performance metrics, taking a single counter or a single measure and basing your determination of good or bad performance based on that measure will lead to problems. This is even more true when it comes to modern disk and I/O management systems between old-school RAID systems and modern disk virtualization because measuring

G. Fritchey, *SQL Server 2017 Query Performance Tuning*, https://doi.org/10.1007/978-1-4842-3888-2_3

I/O is a complex topic. Plan on using multiple metrics to understand how the I/O subsystem within your environment is behaving. With all the information here, this chapter covers only the basics.

There are other mechanisms in modern systems that are also going to make measuring I/O more difficult. A lot more systems are running virtually and sharing resources including disks. This will lead to a lot more random I/O, so you'll have to take that into account when looking at the measures throughout this chapter. Antivirus programs are a frequent problem when it comes to I/O, so be sure you validate if you're dealing with that prior to using the I/O metrics we're getting ready to talk about. You may also see issues with filter drivers acting as a bottleneck in your I/O paths, so this is another thing to look at.

One thing you need to know about before we talk about metrics and resolutions is how the checkpoint process works. When SQL Server writes data, it first writes it all to memory (and we'll talk about memory issues in Chapter 4). Any pages in memory that have changes in them are known as *dirty pages*. The checkpoint process occurs periodically based on internal measures and your recovery interval settings. The checkpoint process writes the dirty pages to disk and records all the changes to the transaction log. The checkpoint process is the primary driver of the write I/O activity you'll see within SQL Server.

Let's see how we can measure the behavior of the I/O subsystem.

Disk Counters

To analyze disk performance, you can use the counters shown in Table 3-1.