Table 3-1. *Performance Monitor Counters to Analyze I/O Pressure*

Object (Instance[,InstanceN])	Counter	Description	Value
PhysicalDisk (Data-disk, Log-disk)	Disk Transfers/sec	Rate of read/write operations on disk	Maximum value dependent on I/O subsystem
	Disk Bytes/sec	Amount of data transfer to/from per disk per second	Maximum value dependent on I/O subsystem
	Avg. Disk Sec/Read	Average time in ms to read from disk	Average value < 10 ms, but compare to baseline
	Avg. Disk Sec/Write	Average time in ms to write to disk	Average value < 10 ms, but compare to baseline
SQLServer: Buffer Manager	Page reads/sec	Number of pages being read into the buffer manager	Compare to baseline
	Page writes/sec	Number of pages being written out of the buffer manager	Compare to baseline

The PhysicalDisk counters represent the activities on a physical disk. LogicalDisk counters represent logical subunits (or partitions) created on a physical disk. If you create two partitions, say R: and S:, on a physical disk, then you can monitor the disk activities of the individual logical disks using logical disk counters. However, because a disk bottleneck ultimately occurs on the physical disk, not on the logical disk, it is usually preferable to use the PhysicalDisk counters.

Note that for a hardware redundant array of independent disks (RAID) subsystem (see the "Using a RAID Array" section for more on RAID), the counters treat the array as a single physical disk. For example, even if you have ten disks in a RAID configuration, they will all be represented as one physical disk to the operating system, and subsequently you will have only one set of PhysicalDisk counters for that RAID subsystem. The same point applies to storage area network (SAN) disks (see the "Using a SAN System" section for specifics). You'll also see this in many of the more modern disk systems and virtual disks. Because of this, some of the numbers represented in Table 3-1 may be radically lower (or higher) than what your system can support.

Take all these numbers as general guidelines for monitoring your disks and adjust the numbers to account for the fact that technology is constantly shifting, and you

may see different performance as the hardware improves. We're moving into more and more solid-state drives and even SSD arrays that make disk I/O operations orders of magnitude faster. Where we're not moving in SSD, we're taking advantage of iSCSI interfaces. As you work with these types of hardware, keep in mind that these numbers are more in line with platter-style disk drives and that those are fast becoming obsolete.

Disk Transfers/Sec

Disk Transfers/sec monitors the rate of read and write operations on the disk. A typical hard disk drive today can do about 180 disk transfers per second for sequential I/O (IOPS) and 100 disk transfers per second for random I/O. In the case of random I/O, Disk Transfers/sec is lower because more disk arm and head movements are involved. OLTP workloads, which are workloads for doing mainly singleton operations, small operations, and random access, are typically constrained by disk transfers per second. So, in the case of an OLTP workload, you are more constrained by the fact that a disk can do only 100 disk transfers per second than by its throughput specification of 1000MB per second.

Note An SSD can be anywhere from around 5,000 IOPS to as much as 500,000 IOPS for some high-end SSD systems. Your monitoring of Disk Transfers/sec will need to scale accordingly. See your vendor for details on this measure.

Because of the inherent slowness of a disk, it is recommended that you keep disk transfers per second as low as possible.

Disk Bytes/Sec

The Disk Bytes/sec counter monitors the rate at which bytes are transferred to or from the disk during read or write operations. A typical disk spinning at 7200RPM can transfer about 1000MB per second. Generally, OLTP applications are not constrained by the disk transfer capacity of the disk subsystem since the OLTP applications access small amounts of data in individual database requests. If the amount of data transfer exceeds the capacity of the disk subsystem, then a backlog starts developing on the disk subsystem, as reflected by the Disk Queue Length counters.

Again, these numbers may be much higher for SSD access since it's largely limited only by the latency caused by the drive to host interface.

Avg. Disk Sec/Read and Avg. Disk Sec/Write

Avg. Disk Sec/Read and Avg. Disk Sec/Write track the average amount of time it takes in milliseconds to read from or write to a disk. Having an understanding of just how well the disks are handling the writes and reads that they receive can be a strong indicator of where problems are. If it's taking more than about 10ms to move the data from or to your disk, you may need to take a look at the hardware and configuration to be sure everything is working correctly. You'll need to get even better response times for the transaction log to perform well.

In terms of measuring performance of your I/O system, these are the single best measure. Sec/Read or Write may not tell you which query or queries are causing problems. These measures will tell you absolutely how your I/O system is behaving, so I would include them along with any other set of metrics you gather.

Buffer Manager Page Reads/Writes

While measuring the I/O system is important, as mentioned earlier, you need to have more than one measure to show how the I/O system is behaving. Knowing the pages being moved into and out of your buffer manager gives you a great indication as to whether the I/O you are seeing is within SQL Server. It's one of the measures you'll want to add to any others when trying to demonstrate an I/O issue.

Additional I/O Monitoring Tools

Just like with all the other tools, you'll need to supplement the information you gather from Performance Monitor with data available in other sources. The really good information for I/O and disk issues are all in DMOs.

Sys.dm_io_virtual_file_stats

This is a function that returns information about the files that make up a database. You call it something like the following:

```
SELECT  *
FROM    sys.dm_io_virtual_file_stats(DB_ID('AdventureWorks2017'), 2) AS
divfs;
```

It returns several interesting columns of information about the file. The most interesting things are the stall data, which is the time that users are waiting on different I/O operations. First, `io_stall_read_ms` represents the amount of time in milliseconds that users are waiting for reads. Then there is `io_stall_write_ms`, which shows you the amount of time that write operations have had to wait on this file within the database. You can also look at the general number, `io_stall`, which represents all waits on I/O for the file. To make these numbers meaningful, you get one more value, `sample_ms`, which shows the amount of time measured. You can compare this value to the others to get a sense of the degree that I/O issues are holding up your system. Further, you can narrow this down to a particular file so you know what's slowing things down in the log or in a particular data file. This is an extremely useful measure for determining the existence of an I/O bottleneck. It doesn't help that much to identify the particular bottleneck. Combine this with wait statistics and the Perfmon metrics mentioned earlier.

Sys.dm_os_wait_stats

This is a useful DMO that shows aggregate information about waits on the system. To determine whether you have an I/O bottleneck, you can take advantage of this DMO by querying it like this:

```
SELECT  *
FROM    sys.dm_os_wait_stats AS dows
WHERE   wait_type LIKE 'PAGEIOLATCH%';
```

What you're looking at are the various I/O latch operations that are causing waits to occur. Like with `sys.dm_io_virtual_status`, you don't get a specific query from this DMO, but it does identify whether you have a bottleneck in I/O. Like many of the performance counters, you can't simply look for a value here. You need to compare the current values to a baseline value to arrive at your current situation.

The WHERE clause shown earlier uses PAGEIOLATCH%, but you should also look for waits related to other I/O processes such as WRITELOG, LOGBUFFER, and ASYNC_IO_COMPLETION.

When you run this query, you get a count of the waits that have occurred as well as an aggregation of the total wait time. You also get a max value for these waits so you know what the longest one was since it's possible that a single wait could have caused the majority of the wait time.

Don't forget that you can see wait statistics in the Query Store. We'll cover those in detail in Chapter 11.

Monitoring Linux I/O

For I/O monitoring, you'll be limited either to SQL Server internals or to taking advantage of Linux-specific monitoring tools such as were mentioned in Chapter 2. The fundamentals of input and output within the Linux system are not very different from those within the Windows OS. The principal difference is just in how you capture disk behavior at the OS level.

Disk Bottleneck Resolutions

A few of the common disk bottleneck resolutions are as follows:

- Optimizing application workload
- Using a faster I/O path
- Using a RAID array
- Using a SAN system
- Using solid-state drives
- Aligning disks properly
- Adding system memory
- Creating multiple files and filegroups
- Moving the log files to a separate physical drive
- Using partitioned tables

I'll now walk you through each of these resolutions in turn.

Optimizing Application Workload

I cannot stress enough how important it is to optimize an application's workload in resolving a performance issue. The queries with the highest number of reads or writes will be the ones that cause a great deal of disk I/O. I'll cover the strategies for optimizing those queries in more detail throughout the rest of this book.

Using a Faster I/O Path

One of the most efficient resolutions, and one that you will adopt any time you can, is to use drives, controllers, and other architecture with faster disk transfers per second. However, you should not just upgrade disk drives without further investigation; you need to find out what is causing the stress on the disk.

Using a RAID Array

One way of obtaining disk I/O parallelism is to create a single pool of drives to serve all SQL Server database files, excluding transaction log files. The pool can be a single RAID array, which is represented in Windows Server 2016 as a single physical disk drive. The effectiveness of a drive pool depends on the configuration of the RAID disks.

Out of all available RAID configurations, the most commonly used RAID configurations are the following (also shown in Figure 3-1):

- *RAID 0*: Striping with no fault tolerance
- *RAID 1:* Mirroring
- *RAID 5:* Striping with parity
- *RAID 1+0*: Striping with mirroring

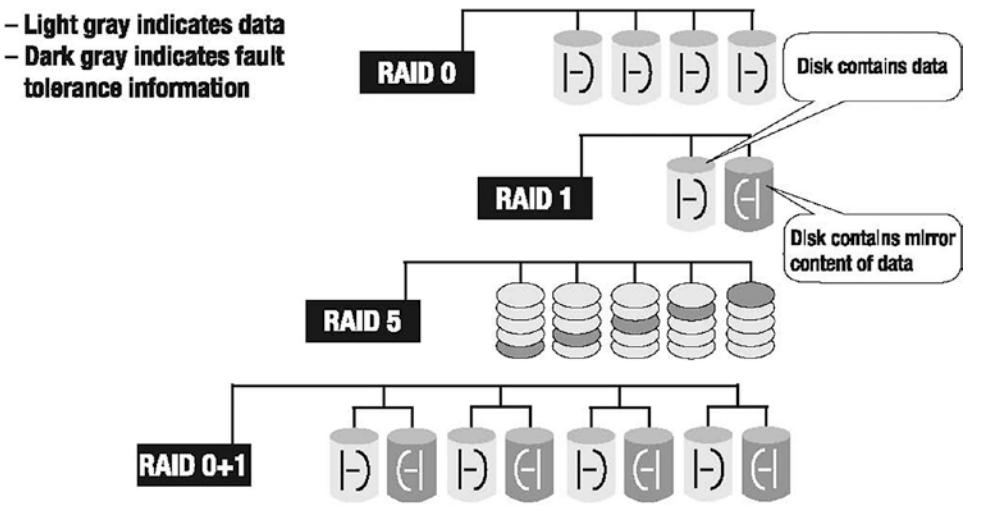

Figure 3-1. *RAID configurations*

RAID 0

Since this RAID configuration has no fault tolerance, you can use it only in situations where the reliability of data is not a concern. The failure of any disk in the array will cause complete data loss in the disk subsystem. Therefore, you shouldn't use it for any data file or transaction log file that constitutes a database, except, possibly, for the system temporary database called `tempdb`. The number of I/Os per disk in RAID 0 is represented by the following equation:

```
I/Os per disk = (Reads + Writes) / Number of disks in the array
```

In this equation, `Reads` is the number of read requests to the disk subsystem, and `Writes` is the number of write requests to the disk subsystem.

RAID 1

RAID 1 provides high fault tolerance for critical data by mirroring the data disk onto a separate disk. It can be used where the complete data can be accommodated in one disk only. Database transaction log files for user databases, operating system files, and SQL Server system databases (`master` and `msdb`) are usually small enough to use RAID 1.

The number of I/Os per disk in RAID 1 is represented by the following equation:

```
I/Os per disk =(Reads + 2 X Writes) / 2
```

RAID 5

RAID 5 is an acceptable option in many cases. It provides reasonable fault tolerance by effectively using only one extra disk to save the computed parity of the data in other disks, as shown in Figure 3-1. When there is a disk failure in RAID 5 configuration, I/O performance becomes terrible, although the system does remain usable while operating with the failed drive.

Any data where writes make up more than 10 percent of the total disk requests is not a good candidate for RAID 5. Thus, use RAID 5 on read-only volumes or volumes with a low percentage of disk writes.

The number of I/Os per disk in RAID 5 is represented by the following equation:

```
I/Os per disk = (Reads + 4 X Writes) / Number of disks in the array
```

As shown in this equation, the write operations on the RAID 5 disk subsystem are magnified four times. For each incoming write request, the following are the four corresponding I/O requests on the disk subsystem:

- One read I/O to read existing data from the data disk whose content is to be modified
- One read I/O to read existing parity information from the corresponding parity disk
- One write I/O to write the new data to the data disk whose content is to be modified
- One write I/O to write the new parity information to the corresponding parity disk

Therefore, the four I/Os for each write request consist of two read I/Os and two write I/Os.

In an OLTP database, all the data modifications are immediately written to the transaction log file as part of the database transaction, but the data in the data file itself is synchronized with the transaction log file content asynchronously in batch operations. This operation is managed by the internal process of SQL Server called the *checkpoint process*. The frequency of this operation can be controlled by using the recovery interval (min) configuration parameter of SQL Server. Just remember that the timing of checkpoints can be controlled through the use of indirect checkpoints introduced in SQL Server 2012.

Because of the continuous write operation in the transaction log file for a highly transactional OLTP database, placing transaction log files on a RAID 5 array will degrade the array's performance. Although, where possible, you should not place the transactional log files on a RAID 5 array, the data files may be placed on RAID 5 since the write operations to the data files are intermittent and batched together to improve the efficiency of the write operation.

RAID 6

RAID 6 is an added layer on top of RAID 5. An extra parity block is added to the storage of RAID 5. This doesn't negatively affect reads in any way. This means that, for reads, performance is the same as RAID 5. There is an added overhead for the additional write, but it's not that large. This extra parity block was added because RAID arrays are becoming so large these days that data loss is inevitable. The extra parity block acts as a check against this to better ensure that your data is safe.

RAID 1+0 (RAID 10)

RAID 1+0 (also referred to RAID 10) configuration offers a high degree of fault tolerance by mirroring every data disk in the array. It is a much more expensive solution than RAID 5, since double the number of data disks are required to provide fault tolerance. This RAID configuration should be used where a large volume is required to save data and more than 10 percent of disk requests are writes. Since RAID 1+0 supports *split seeks* (the ability to distribute the read operations onto the data disk and the mirror disk and then converge the two data streams), read performance is also very good. Thus, use RAID 1+0 wherever performance is critical.

The number of I/Os per disk in RAID 1+0 is represented by the following equation:

```
I/Os per disk = (Reads + 2 X Writes) / Number of disks in the array
```

Using a SAN System

SANs remain largely the domain of large-scale enterprise systems, although the cost has dropped. A SAN can be used to increase the performance of a storage subsystem by simply providing more spindles and disk drives to read from and write to. Because of their size, complexity, and cost, SANs are not necessarily a good solution in all cases. Also, depending on the amount of data, direct-attached storage (DAS) can be configured to run faster. The principal strength of SAN systems is not reflected in performance but rather in the areas of scalability, availability, and maintenance.

Another area where SANs are growing are SAN devices that use Internet Small Computing System Interface (iSCSI) to connect a device to the network. Because of how the iSCSI interface works, you can make a network device appear to be locally attached storage. In fact, it can work nearly as fast as locally attached storage, but you get to consolidate your storage systems.

Conversely, you may achieve performance gains by going to local disks and getting rid of the SAN. SAN systems are extremely redundant by design. But, that redundancy adds a lot of overhead to disk operations, especially the type typically performed by SQL Server: lots of small writes done rapidly. While moving from a single local disk to a SAN can be an improvement, depending on your systems and the disk subsystem you put together, you could achieve even better performance outside the SAN.

Using Solid-State Drives

Solid-state drives are taking the disk performance world by storm. These drives use memory instead of spinning disks to store information. They're quiet, lower power, and supremely fast. However, they're also quite expensive when compared to hard disk drives (HDDs). At this writing, it costs approximately $.03/GB for an HDD and $.90/GB for an SSD. But that cost is offset by an increase in speed from approximately 100 operations per second to 5,000 operations per second and up. You can also put SSDs into arrays through a SAN or RAID, further increasing the performance benefits. There are a limited number of write operations possible on an SSD drive, but the failure rate is no higher than that from HDDs so far. There are also hybrid solutions with varying price points and performance metrics. For a hardware-only solution, implementing SSDs is probably the best operation you can do for a system that is I/O bound.

Aligning Disks Properly

Windows Server 2016 aligns disks as part of the install process, so modern servers should not be running into this issue. However, if you have an older server, this can still be a concern. You'll also need to worry about this if you're moving volumes from a pre-Windows Server 2008 system. You will need to reformat these in order to get the alignment set appropriately. The way data is stored on a disk is in a series of *sectors* (also referred to as *blocks*) that are stored on tracks. A disk is out of alignment when the size of the track, determined by the vendor, consists of a number of sectors different from the default size you're writing to. This means that one sector will be written correctly, but the next one will have to cross two tracks. This can more than double the amount of I/O required to write or read from the disk. The key is to align the partition so that you're storing the correct number of sectors for the track.

Adding System Memory

When physical memory is scarce, the system starts writing the contents of memory back to disk and reading smaller blocks of data more frequently, or reading large blocks, both of which cause a lot of paging. The less memory the system has, the more the disk subsystem is used. This can be resolved using the memory bottleneck resolutions enumerated in the previous section.

Creating Multiple Files and Filegroups

In SQL Server, each user database consists of one or more data files and usually one transaction log file. The data files belonging to a database can be grouped together in one or more filegroups for administrative and data allocation/placement purposes. For example, if a data file is placed in a separate filegroup, then write access to all the tables in the filegroup can be controlled collectively by making the filegroup read-only (transaction log files do not belong to any filegroup).

You can create a filegroup for a database from SQL Server Management Studio, as shown in Figure 3-2. The filegroups of a database are presented in the Filegroups pane of the Database Properties dialog box.

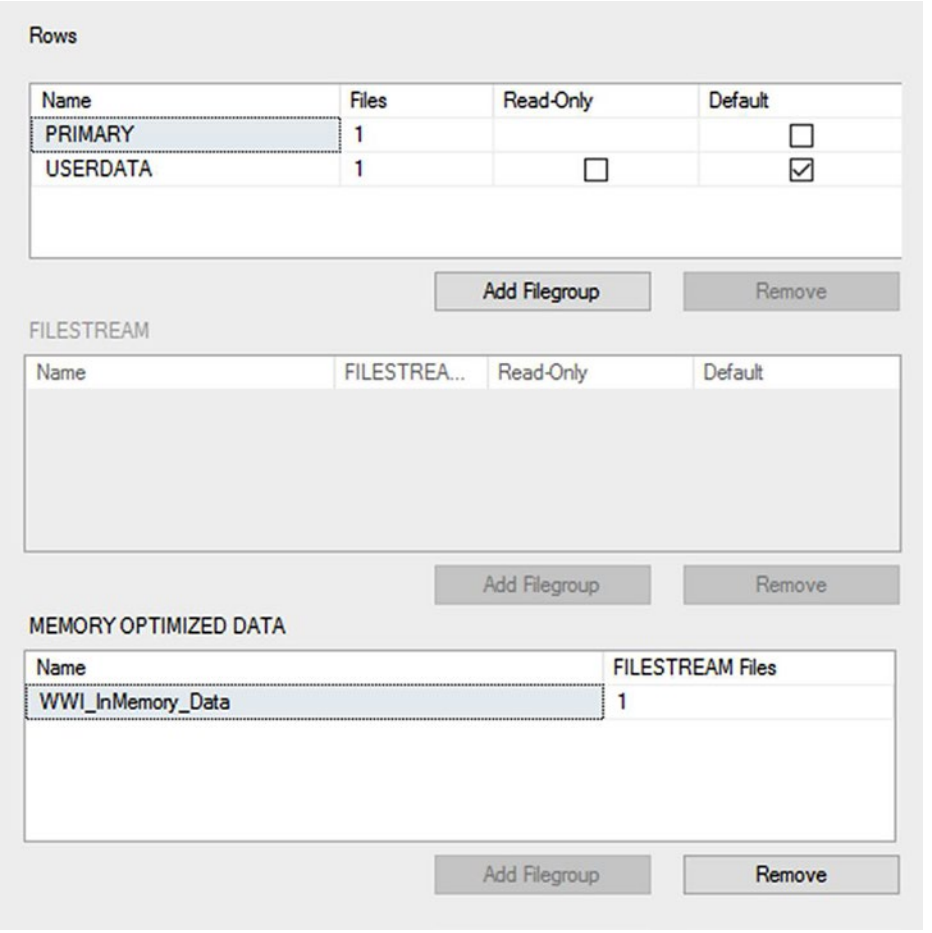

Figure 3-2. *Filegroups configuration*

In Figure 3-2, you can see that there are three filegroups defined for the WideWorldImporters database. You can add multiple files to multiple filegroups distributed across multiple I/O paths so that work can be done in parallel across the groups and distributed storage after you also move your database objects into those different groups, literally putting multiple spindles and multiple I/O paths to work. But, simply throwing lots of files, even on different disks, through a single disk controller may result in worse performance, not better.

You can add a data file to a filegroup in the Database Properties dialog box in the Files window by selecting from the drop-down list, as shown in Figure 3-3.

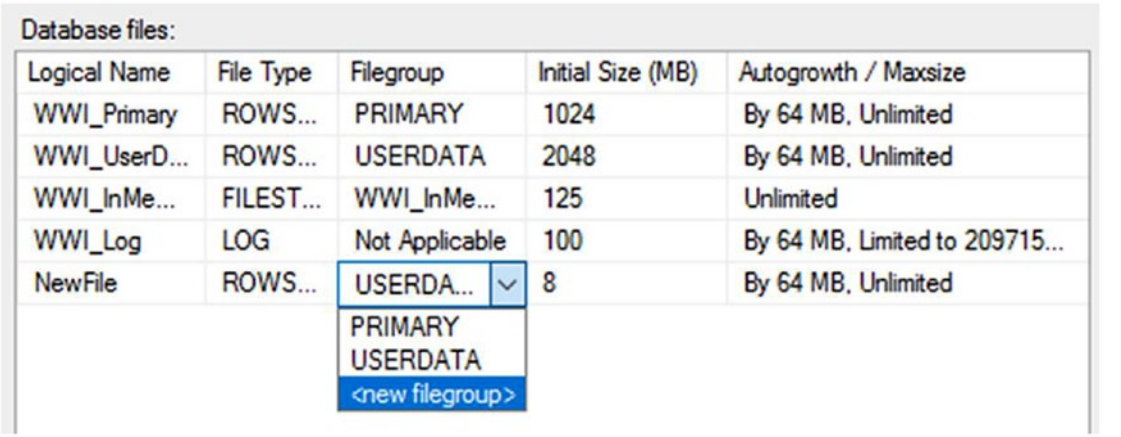

Figure 3-3. *Data files configuration*

You can also do this programmatically, as follows:

```
ALTER DATABASE WideWorldImporters
ADD FILEGROUP Indexes;
ALTER DATABASE WideWorldImporters
ADD FILE
    (
        NAME = AdventureWorks2017_Data2,
        FILENAME = 'c:\DATA\WWI_Index.ndf',
        SIZE = 20GB,
        FILEGROWTH = 10%
    )
TO FILEGROUP Indexes;
```

By separating tables that are frequently joined into separate filegroups and then putting files within the filegroups on separate disks or LUNS, the separated I/O paths can result in improved performance, assuming of course the paths to those disks are properly configured and not overloaded (do not mistake more disks for automatically more I/O; it just doesn't work that way). For example, consider the following query:

```
SELECT si.StockItemName,
       s.SupplierName
FROM Warehouse.StockItems AS si
JOIN Purchasing.Suppliers AS s
    ON si.SupplierID = s.SupplierID;
```

If the tables `Warehouse.StockItems` and `Purchasing.Suppliers` are placed in separate filegroups containing one file each, the disks can be read from multiple I/O paths, increasing performance.

It is recommended for performance and recovery purposes that, if multiple filegroups are to be used, the primary filegroup should be used only for system objects, and secondary filegroups should be used only for user objects. This approach improves the ability to recover from corruption. The recoverability of a database is higher if the primary data file and the log files are intact. Use the primary filegroup for system objects only, and store all user-related objects on one or more secondary filegroups.

Spreading a database into multiple files, even on the same drive, makes it easy to move the database files onto separate drives, making future disk upgrades easier. For example, to move a user database file (`WWI_Index.ndf`) to a new disk subsystem (F:), you can follow these steps:

1. Detach the user database as follows:

   ```
   USE master;
   GO
   EXEC sp_detach_db 'WideWorldImporters';
   GO
   ```

2. Copy the data file `WWI_Index.ndf` to a folder `F:\Data\` on the new disk subsystem.

3. Reattach the user database by referring files at appropriate locations, as shown here:

```
USE master;
GO
sp_attach_db 'WideWorldImporters',
 'R:\DATA\WWI_Primary.mdf',
’R:\DATA\WWI_UserData.ndf’,
 'F:\DATA\WWI_Indexes.ndf',
’R:\DATA\WWI_InMemory.ndf’,
 'S:\LOG\WWI_Log.1df ';
GO
```

4. To verify the files belonging to a database, execute the following commands:

```
USE WideWorldImporters;
GO
SELECT * FROM sys.database_files;
GO
```

Moving the Log Files to a Separate Physical Disk

SQL Server transaction log files should always, when possible, be located on a separate hard disk drive from all other SQL Server database files. Transaction log activity primarily consists of sequential write I/O, unlike the nonsequential (or random) I/O required for the data files. Separating transaction log activity from other nonsequential disk I/O activity can result in I/O performance improvements because it allows the hard disk drives containing log files to concentrate on sequential I/O. But, remember, there are random transaction log reads, and the data reads and writes can be sequential as much as the transaction log. There is just a strong tendency of transaction log writes to be sequential.

However, creating a single disk for all your log files just brings you back to random I/O again. If this particular log file is mission critical, it may need its own storage and path to maximize performance.

The major portion of time required to access data from a hard disk is spent on the physical movement of the disk spindle head to locate the data. Once the data is located, the data is read electronically, which is much faster than the physical movement of the head. With only sequential I/O operations on the log disk, the spindle head of the log disk can write to the log disk with a minimum of physical movement. If the same disk is used for data files, however, the spindle head has to move to the correct location before writing to the log file. This increases the time required to write to the log file and thereby hurts performance.

Even with an SSD disk, isolating the data from the transaction log means the work will be distributed to multiple locations, improving the performance.

Furthermore, for SQL Server with multiple OLTP databases, the transaction log files should be physically separated from each other on different physical drives to improve performance. An exception to this requirement is a read-only database or a database with few database changes. Since no online changes are made to the read-only database, no write operations are performed on the log file. Therefore, having the log file on a separate disk is not required for read-only databases.

As a general rule of thumb, you should try, where possible, to isolate files with the highest I/O from other files with high I/O. This will reduce contention on the disks and possibly improve performance. To identify those files using the most I/O, reference `sys.dm_io_virtual_file_stats`.

Using Partitioned Tables

In addition to simply adding files to filegroups and letting SQL Server distribute the data between them, it's possible to define a horizontal segmentation of data called a *partition* so that data is divided between multiple files by the partition. A filtered set of data is a segment*;* for example, if the partition is by month, the segment of data is any given month. Creating a partition moves the segment of data to a particular filegroup and only that filegroup. While partitioning is primarily a tool for making data management easier, you can see an increase in speed in some situations because when querying against well-defined partitions, only the files with the partitions of data you're interested in will be accessed during a given query through a process called *partition elimination*. If you assume for a moment that data is partitioned by month, then each month's data file can be set to read-only as each month ends. That read-only status means you'll recover the system faster, and you can compress the storage resulting in some performance

improvements. Just remember that partitions are primarily a manageability feature. While you can see some performance benefits from them in certain situations, it shouldn't be counted on as part of partitioning the data. SQL Server 2017 supports up to 15,000 partitions (just remember, that's a limit, not a goal). Let me repeat, partitioning is absolutely not a performance enhancement tool.

Summary

This chapter focused on gathering and interpreting metrics about the behavior of your disks. Just remember that every set of hardware can be fundamentally different, so applying any hard-and-fast set of metrics around behavior can be problematic. You now have the tools to gather disk performance metrics using Performance Monitor and some T-SQL commands. The resolutions for disk bottlenecks are varied but must be explored if you are dealing with bottlenecks related to disk behavior.

The next chapter completes the examination of system bottlenecks with a discussion of the CPU.

CHAPTER 4

CPU Performance Analysis

This chapter concludes the book's exploration of the system, with a discussion about CPU, network, and general SQL Server metrics. The CPU is the work engine of a system and keeps everything running. All the different calculations required for gathering and delivering data, maintaining the system, and ordering access are performed by the CPU. Getting bottlenecked on the CPU can be a difficult process to work out of. Unlike memory, which you can sometimes easily install more of, or disks, which you can sometimes easily add more of or upgrade, CPUs are an integral part of the system you're running on and can frequently be upgraded only by buying newer machines. So, you'll want to keep an eye on CPU usage. Networks are seldom a major bottleneck for SQL Server, except of course when dealing with Azure SQL Database, but it's good to keep an eye on them too. Finally, there are some SQL Server internal processes that you'll need to gather metrics on. This chapter covers the following topics:

- How to gather metrics on the processor
- Additional metrics available through T-SQL queries
- Methods for resolving processor bottlenecks

Processor Bottleneck Analysis

SQL Server makes heavy use of any processor resource available. You're more likely to be bottlenecked on I/O or memory, but you can hit issues with your CPUs as well. The measures we're covering here are focused on the operating systems and SQL Server. However, in a virtualized environment, the measures we're looking at for CPU are much less likely to reflect reality. You'll need to deal with whatever hypervisor or system you're

G. Fritchey, *SQL Server 2017 Query Performance Tuning*, https://doi.org/10.1007/978-1-4842-3888-2_4

using for virtualization to understand exactly how some of the OS-level CPU measures are actually reflecting reality. You may be experiencing external pressure or even external throttling, none of which will be visible with the counters outlined here.

You can use the Performance Monitor counters in Table 4-1 to analyze pressure on the processor resource.

Table 4-1. *Performance Monitor Counters to Analyze CPU Pressure*

Object (Instance[,InstanceN])	Counter	Description	Value
Processor(_Total)%	Processor Time	Percentage of time processor was busy	Average value < 80%, but compare to baseline
	% Privileged	Percentage of processor time spent in privileged mode	Average value < 10%, but compare to baseline
System	Processor Queue Length	Number of requests outstanding on the processor	Average value < 2, but compare to baseline
	Context Switches/sec	Rate at which processor is switched per processor from one thread to another	Average value < 5,000, but compare to baseline
SQL Server:SQL Statistics	Batch Requests/sec	SQL command batches received per second	Based on your standard workload
	SQL Compilations/sec	Number of times SQL is compiled	Based on your standard workload
	SQL Recompilations/sec	Number of recompiles	

Let's discuss these counters in more detail.

% Processor Time

% Processor Time should not be consistently high (greater than 80 percent). The effect of any sustained processor time greater than 90 percent is effectively the same as that of 100 percent. If % Processor Time is consistently high and disk and network counter values are low, your first priority must be to reduce the stress on the processor. Just remember that the numbers here are simply suggestions; people can disagree with these numbers for valid reasons. Use them as a starting point for evaluating your system, not as a specific recommendation.

For example, if % Processor Time is 85 percent and you are seeing excessive disk use by monitoring I/O counters, it is quite likely that a major part of the processor time is spent on managing the disk activities. This will be reflected in the % Privileged Time counter of the processor, as explained in the next section. In that case, it will be advantageous to optimize the disk bottleneck first. Further, remember that the disk bottleneck in turn can be because of a memory bottleneck, as explained earlier in the chapter.

You can track processor time as an aggregate of all the processors on the machine, or you can track the percentage utilization individually to particular processors. This allows you to segregate the data collection in the event that SQL Server runs on three processors of a four-processor machine. Remember, you might be seeing one processor maxed out while another processor has little load. The average value wouldn't reflect reality in that case. Use the average value as just an indicator and the individual values as more of a measure of actual load and processing on the system.

In a virtualized environment, the CPUs may be virtualized so that what you're seeing isn't accurate. So, for example, in a VMware system, if you install the VMware Tools, you'll be able to look at a VM Processor counter to see the processor usage of the host machine. Using this measure you can tell whether the CPU usage you're seeing in your OS is reflected in the hosting machine or whether you're actually just maxing out your virtual CPUs. On the other hand, running HyperV, you'd need to look to \Hyper-V Hypervisor Logical Processor(_Total)\% Total Run Time for the same measure. You'll have other measures depending on the hypervisor you're using.

% Privileged Time

Processing on a Windows server is done in two modes: *user mode* and *privileged* (or *kernel*) mode. All system-level activities, including disk access, are done in privileged mode. If you find that % Privileged Time on a dedicated SQL Server system is 20 to 25 percent or more, then the system is probably doing a lot of external processing. It could be I/O, a filter driver such as encryption services, defective I/O components, or even out-of-date drivers. The % Privileged Time counter on a dedicated SQL Server system should be at most 5 to 10 percent, but use your baseline to establish what looks like normal behavior on your systems.

Processor Queue Length

Processor Queue Length is the number of threads in the processor queue. (There is a single processor queue, even on computers with multiple processors.) Unlike the disk counters, the Processor Queue Length counter does not read threads that are already running. On systems with lower CPU utilization, the Processor Queue Length counter is typically 0 or 1.

A sustained Processor Queue Length counter of greater than 2 generally indicates processor congestion. Because of multiple processors, you may need to take into account the number of schedulers dealing with the processor queue length. A processor queue length more than two times the number of schedulers (usually 1:1 with processors) can also indicate a processor bottleneck. Although a high % Processor Time counter indicates a busy processor, a sustained high Processor Queue Length counter is a more certain indicator. If the recommended value is exceeded, this generally indicates that there are more threads ready to run than the current number of processors can service in an optimal way.

Context Switches/Sec

The Context Switches/sec counter monitors the combined rate at which all processors on the computer are switched from one thread to another. A context switch occurs when a running thread voluntarily relinquishes the processor, is preempted by a higher-priority ready thread, or switches between user mode and privileged mode to use an executive or subsystem service. It is the sum of Thread:Context Switches/sec for all threads running on all processors in the computer, and it is measured in numbers of switches.