High numbers are largely dictated by the speed of your CPUs, so measure performance over time and compare this number to your baseline to understand when you may be deviating.

Batch Requests/Sec

Batch Requests/sec gives you a good indicator of just how much load is being placed on the system, which has a direct correlation to how much load is being placed on the processor. Since you could see a lot of low-cost queries on your system or a few high-cost queries, you can't look at this number by itself but must reference the other counters defined in this section; 10,000 requests in a second would be considered a busy system. Greater values may be cause for concern, completely depending on what is normal for your system. The best way to know which value has meaning within your own systems is to establish a baseline and then monitor from there. Just remember that a high number here is not necessarily cause for concern. If all your other resources are in hand and you're sustaining a high number of batch requests/sec, it just means your server is busy.

SQL Compilations/Sec

The SQL Compilations/sec counter shows both batch compiles and statement recompiles as part of its aggregation. This number can be extremely high when a server is first turned on (or after a failover or any other startup type event), but it will stabilize over time. Once stable, significant or sustained spikes in compilations different from a baseline measure is cause for concern and will certainly manifest as problems in the processor since query compilation is an expensive operation. If you are working with some type of object-relational mapping engine, such as nHibernate or Entity Framework, a high number of compilations might be normal, though no less costly. Chapter 14 covers SQL compilation in detail.

SQL Recompilations/Sec

SQL Recompilations/sec is a measure of the recompiles of both batches and statements. A high number of recompiles can lead to processor stress. Because statement recompiles are part of this count, it can be much higher than in versions of SQL Server prior to 2005. Chapter 17 covers query recompilation in detail.

Other Tools for Measuring CPU Performance

You can use the DMOs to capture information about your CPU as well. The information in these DMOs will have to be captured by running the query and then keeping the information as part of your baseline measurement.

Sys.dm_os_wait_stats

Wait statistics are a good way to understand whether there are bottlenecks on the system. You can't simply say something greater than *x* is a bad number, though. You need to gather metrics over time in order to understand what represents normal on your system. The deviations from that are interesting. Queries against this DMO that look for signal wait time can indicate CPU bottlenecks.

In the past, CXPACKET waits were considered a waste of time for measuring CPU performance. However, with SQL Server 2016 SP2 and SQL Server 2017 CU3, a new split on wait statistics has occurred. There is now a CXPACKET wait, which we care about, and a CXCONSUMER wait, which is not important. If you're running an Azure SQL Database, you'll see this same split. The core of the split is to cover consumers and producers of parallelism, a major cause of CPU bottlenecks on some systems. Consumers of parallelism, the operations receiving the data, generally have negligible but measurable waits, in other words, CXCONSUMER waits. Now, CXPACKET measures producers, or the operators that push the data. That changes things. CXPACKET is now a wait that indicates real load on the system that is affecting your CPU.

Sys.dm_os_workers and Sys.dm_os_schedulers

These DMOs display the worker and scheduler threads within the Windows operating system. Running queries against these regularly will allow you to get counts of the number of processes that are in a runnable state. This is an excellent indication of processor load.

Query Store

The Query Store isn't specifically a measure of CPU, but the information it captures does include both the CPU usage, aggregated, of the query in question and the wait statistics for those queries, including any related to CPU.

Measure CPU Behavior in Linux

You can still use `sys.dm_os_wait_stats` when running on Linux. This will give you wait statistics that can indicate a CPU load. Otherwise, you'll need to go the Linux system itself. The generally recommended method for looking at CPU is to use the `top` command. The output from that tool is documented here: https://bit.ly/2KbZmuZ.

Processor Bottleneck Resolutions

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

- Optimizing application workload
- Eliminating or reducing excessive compiles/recompiles
- Using more or faster processors
- Not running unnecessary software

Let's consider each of these resolutions in turn.

Optimizing Application Workload

To identify the processor-intensive queries, capture all the SQL queries using Extended Events sessions (which I will discuss in the next chapter) and then group the output on the CPU column. Another method is to take advantage of the Query Store (discussed in Chapter 11). You can retrieve information from `sys.query_store_runtime_stats` to see multiple, aggregated, CPU metrics on a per-query basis. The queries with the highest amount of CPU time contribute the most to the CPU stress. You should then analyze and optimize those queries to reduce stress on the CPU. Frequently, the cause for CPU stress is not extensive calculations within the queries but actually contention within logical I/O. Addressing I/O issues can often help you resolve CPU issues as well. You can also query directly against the `sys.dm_exec_query_stats` or `sys.dm_exec_procedure_stats` `dynamic management` view to see immediate issues in real time. Finally, using both a query hash and a query plan hash, you can identify and tune common queries or common execution plans (this is discussed in detail in Chapter 14). Most of the rest of the chapters in this book are concerned with optimizing application workload.

Eliminating Excessive Compiles/Recompiles

A certain number of query compiles and recompiles is simply to be expected, especially, as already noted, when working with ORM tools. It's when there is a large number of these over-sustained periods that a problem exists. It's also worth noting the ratio between them. Having a high number of compiles and a low number of recompiles means that few queries are being reused within the system (query reuse is covered in detail in Chapter 9). A high number of recompiles will cause high processor use. Methods for addressing recompiles are covered in Chapter 17.

Using More or Faster Processors

One of the easiest resolutions, and one that you will adopt most of the time, is to increase system processing power. However, because of the high cost involved in a processor upgrade, you should first optimize CPU-intensive operations as much as possible.

The system's processing power can be increased by increasing the power of individual processors or by adding more processors. When you have a high % Processor Time counter and a low Processor Queue Length counter, it makes sense to increase the power of individual processors. In the case of both a high % Processor Time counter and a high Processor Queue Length counter, you should consider adding more processors. Increasing the number of processors allows the system to execute more requests simultaneously.

Not Running Unnecessary Software

Corporate policy frequently requires virus checking software be installed on the server. You can also have other products running on the server. When possible, no unnecessary software should be running on the same server as SQL Server. Exterior applications that have nothing to do with maintaining the Windows Server or SQL Server are best placed on a different machine.

Network Bottleneck Analysis

In SQL Server OLTP production environments, you will find few performance issues that are because of problems with the network. Most of the network issues you face in an OLTP environment are in fact hardware or driver limitations or issues with switches or routers. Most of these issues can be best diagnosed with the Network Monitor tool. However, Performance Monitor also provides objects that collect data on network activity, as shown in Table 4-2.

Table 4-2. *Performance Monitor Counters to Analyze Network Pressure*

Object (Instance[,InstanceN])	Counter	Description	Value
Network Interface (Network card)	Bytes Total/sec	Rate at which bytes are transferred on the NIC	Average value < 50% of NIC capacity, but compare with baseline
Network Segment	% Net Utilization	Percentage of network bandwidth in use on a network segment	Average value < 80% of network bandwidth, but compare with baseline

Bytes Total/Sec

You can use the Bytes Total/sec counter to determine how the network interface card (NIC) or network adapter is performing. The Bytes Total/sec counter should report high values to indicate a large number of successful transmissions. Compare this value with that reported by the Network Interface\Current Bandwidth performance counter, which reflects each adapter's bandwidth.

To allow headroom for spikes in traffic, you should usually average no more than 50 percent of capacity. If this number is close to the capacity of the connection and if processor and memory use are moderate, then the connection may well be a problem.

% Net Utilization

The % Net Utilization counter represents the percentage of network bandwidth in use on a network segment. The threshold for this counter depends on the type of network. For Ethernet networks, for example, 30 percent is the recommended threshold when SQL Server is on a shared network hub. For SQL Server on a dedicated full-duplex network, even though near 100 percent usage of the network is acceptable, it is advantageous to keep the network utilization below an acceptable threshold to keep room for the spikes in the load.

Note You must install the Network Monitor Driver to collect performance data using the Network Segment object counters.

In Windows Server 2012 R2, you can install the Network Monitor Driver from the local area connection properties for the network adapter. The Network Monitor Driver is available in the network protocol list of network components for the network adapter.

You can also look at the wait statistics in `sys.dm_os_wait_stats` for network-related waits. But, one that frequently comes up is ASYNC_NETWORK_IO. While this can be an indication of network-related waits, it's much more common to reflect waits caused by poor programming code that is not consuming a result set efficiently.

Network Bottleneck Resolutions

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

- Optimizing application workload
- Adding network adapters
- Moderating and avoiding interruptions

Let's consider these resolutions in more detail.

Optimizing Application Workload

To optimize network traffic between a database application and a database server, make the following design changes in the application:

- Instead of sending a long SQL string, create a stored procedure for the SQL query. Then, you just need to send over the network the name of the stored procedure and its parameters.
- Group multiple database requests into one stored procedure. Then, only one database request is required across the network for the set of SQL queries implemented in the stored procedure. This becomes extremely important when talking about Azure SQL Database.
- Request a small data set. Do not request table columns that are not used in the application logic.
- Move data-intensive business logic into the database as stored procedures or database triggers to reduce network round-trips.
- If data doesn't change frequently, try caching the information on the application instead of frequently calling the database for information that is going to be exactly the same as the last call.
- Minimize network calls, such as returning multiple result sets that are not consumed. A common issue is caused by a result set returned by SQL Server that includes each statement's row count. You can disable this by using SET NOCOUNT ON at the top of your query.

SQL Server Overall Performance

To analyze the overall performance of a SQL Server instance, besides examining hardware resource utilization, you should examine some general aspects of SQL Server. You can use the performance counters presented in Table 4-3.

Table 4-3. *Performance Monitor Counters to Analyze Generic SQL Pressure*

Object(Instance[,InstanceN])	Counter
SQLServer:Access Methods	FreeSpace Scans/sec Full Scans/sec Table Lock Escalations/sec Worktables Created/sec
SQLServer:Latches	Total Latch Wait Time (ms)
SQLServer:Locks(_Total)	Lock Timeouts/sec Lock Wait Time (ms) Number of Deadlocks/sec
SQLServer:SQL Statistics	Batch Requests/sec SQL Re-Compilations/sec
SQLServer:General Statistics	Processes Blocked User ConnectionsTemp Tables Creation RateTemp Tables for Destruction

Let's break these down into different areas of concern to show the counters within the context where they would be more useful.

Missing Indexes

To analyze the possibility of missing indexes causing table scans or large data set retrievals, you can use the counter in Table 4-4.

Table 4-4. *Performance Monitor Counter to Analyze Excessive Data Scans*

Object(Instance[,InstanceN])	Counter
SQLServer:Access Methods	Full Scans/sec

Full Scans/Sec

This counter monitors the number of unrestricted full scans on base tables or indexes. Scans are not necessarily a bad thing. But they do represent a broader access of data, so they are likely to indicate a problem. A few of the main causes of a high Full Scans/sec value are as follows:

- Missing indexes
- Too many rows requested

- Not selective enough a predicate
- Improper T-SQL
- Data distribution or quantity doesn't support a seek

To further investigate queries producing these problems, use Extended Events to identify the queries (I will cover this tool in the next chapter). You can also retrieve this information from the Query Store (Chapter 11). Queries with missing indexes, too many rows requested, or badly formed T-SQL will have a large number of logical reads, caused by scanning the entire table or entire index and an increased CPU time.

Be aware that full scans may be performed for the temporary tables used in a stored procedure because most of the time you will not have indexes (or you will not need indexes) on temporary tables. Still, adding this counter to the baseline helps identify the possible increase in the use of temporary tables, which, when used inappropriately, can be bad for performance.

Dynamic Management Objects

Another way to check for missing indexes is to query the dynamic management view `sys.dm_db_missing_index_details`. This management view returns information that can suggest candidates for indexes based on the execution plans of the queries being run against the database. The view `sys.dm_db_missing_index_details` is part of a series of DMVs collectively referred to as the *missing indexes feature*. These DMVs are based on data generated from execution plans stored in the cache. You can query directly against this view to gather data to decide whether you want to build indexes based on the information available from within the view. Missing indexes will also be shown within the XML execution plan for a given query, but I'll cover that more in the next chapter. While these views are useful for suggesting possible indexes, since they can't be linked to a particular query, it can be unclear which of these indexes is most useful. You'll be better off using the techniques I show in the next chapter to associate a missing index with a particular query. For all the missing index suggestions, you must test them prior to implementing any suggestion on your systems.

The opposite problem to a missing index is one that is never used. The DMV `sys.dm_db_index_usage_stats` shows which indexes have been used, at least since the last restart of the SQL Server instance. Unfortunately, there are a number of ways that counters within this DMV get reset or removed, so you can't completely rely on it for a 100 percent accurate view of index use. You can also view the indexes in use with

a lower-level DMV, `sys.dm_db_index_operational_stats`. It will help to show where indexes are slowing down because of contention or I/O. I'll cover these both in more detail in Chapter 20. You may also find that the suggestions from the Database Tuning Advisor (covered in Chapter 10) may be able to help you with specific indexes for specific queries.

Database Concurrency

To analyze the impact of database blocking on the performance of SQL Server, you can use the counters shown in Table 4-5.

Table 4-5. *Performance Monitor Counters to Analyze SQL Server Locking*

Object(Instance[,InstanceN])	Counter
SQLServer:Latches	Total Latch Wait Time (ms)
SQLServer:Locks(_Total)	Lock Timeouts/sec
	Lock Wait Time (ms)
	Number of Deadlocks/sec

Total Latch Wait Time (Ms)

Latches are used internally by SQL Server to protect the integrity of internal structures, such as a table row, and are not directly controlled by users. This counter monitors total latch wait time (in milliseconds) for latch requests that had to wait in the last second. A high value for this counter can indicate that SQL Server is spending too much time waiting on its internal synchronization mechanism. For a detailed discussion, see the (older, but still relevant) white paper from Microsoft at `https://bit.ly/2wx4gAJ`.

Lock Timeouts/Sec and Lock Wait Time (Ms)

You should expect Lock Timeouts/sec to be 0 and Lock Wait Time (ms) to be very low. A nonzero value for Lock Timeouts/sec and a high value for Lock Wait Time (ms) indicate that excessive blocking is occurring in the database. Three approaches can be adopted in this case.

- You can identify the costly queries currently in cache using data from SQL Profiler or by querying `sys.dm_exec_query_stats`, and then you can optimize the queries appropriately.
- You can use blocking analysis to diagnose the cause of excessive blocking. It is usually advantageous to concentrate on optimizing the costly queries first because this, in turn, reduces blocking for others. In Chapter 20, you will learn how to analyze and resolve blocking.
- Extended Events supplies a blocking event called `blocked_process_report` that you can enable and set a threshold to capture blocking information. Extended Events will be covered in Chapter 6, and `blocked_process_report` will be addressed in Chapter 20.

Just remember that some degree of locks is a necessary part of the system. You'll want to establish a baseline to track thoroughly whether a given value is cause for concern.

Number of Deadlocks/Sec

You should expect to see a 0 value for this counter. If you find a nonzero value, then you should identify the victimized request and either resubmit the database request automatically or suggest that the user do so. More important, an attempt should be made to troubleshoot and resolve the deadlock. Chapter 21 shows how to do this.

Nonreusable Execution Plans

Since generating an execution plan for a stored procedure query requires CPU cycles, you can reduce the stress on the CPU by reusing the execution plan. To analyze the number of stored procedures that are recompiling, you can look at the counter in Table 4-6.

Table 4-6. *Performance Monitor Counter to Analyze Execution Plan Reusability*

Object(Instance[,InstanceN])	Counter
SQLServer:SOL Statistics	SOL Re-Compilations/sec

Recompilations of stored procedures add overhead on the processor. You want to see a value as close to 0 as possible for the SOL Re-Compilations/sec counter, but you won't ever see that. If you consistently see values that deviate from your baseline measures or that spike wildly, then you should use Extended Events to further investigate the stored procedures undergoing recompilations. Once you identify the relevant stored procedures, you should attempt to analyze and resolve the cause of recompilations. In Chapter 17, you will learn how to analyze and resolve various causes of recompilation.

General Behavior

SQL Server provides additional performance counters to track some general aspects of a SQL Server system. Table 4-7 lists a couple of the most commonly used counters.

Table 4-7. *Performance Monitor Counters to Analyze Volume of Incoming Requests*

Object(Instance[,InstanceN])	Counter
SQLServer:General Statistics	User Connections
SQLServer:SQL Statistics	Batch Requests/sec

User Connections

Multiple read-only SQL Server instances can work together in a load-balancing environment (where SQL Server is spread over several machines) to support a large number of database requests. In such cases, it is better to monitor the User Connections counter to evaluate the distribution of user connections across multiple SQL Server instances. User Connections can range all over the spectrum with normal application behavior. This is where a baseline is essential to determine the expected behavior. You will see how you can establish this baseline shortly.

Batch Requests/Sec

This counter is a good indicator of the load on SQL Server. Based on the level of system resource utilization and Batch Requests/sec, you can estimate the number of users SQL Server may be able to take without developing resource bottlenecks. This counter

value, at different load cycles, helps you understand its relationship with the number of database connections. This also helps you understand SQL Server's relationship with Web Request/sec, that is, Active Server Pages.Requests/sec for web applications using Microsoft Internet Information Services (IIS) and Active Server Pages (ASP). All this analysis helps you better understand and predict system behavior as the user load changes.

The value of this counter can range over a wide spectrum with normal application behavior. A normal baseline is essential to determine the expected behavior.

Summary

In this chapter, you learned how to gather metrics on the CPU, the network, and SQL Server in general. All this information feeds into your ability to understand what's happening on your system before you delve into attempting to tune queries. Remember that the CPU is affected by the other resources since it's the thing that has to manage those resources, so some situations that can look like a CPU problem are better explained as a disk or memory issue. Networks are seldom a major bottleneck for SQL Server. You have a number of methods of observing SQL Server internals behavior through Performance Monitor counters, just like the other parts of the system. This concludes the discussion of the various system metrics. Next, you'll learn how to put all that together to create a baseline.

CHAPTER 5

Creating a Baseline

In the previous three chapters, you learned a lot about various possible system bottlenecks caused by memory, the disk, and the CPU. I also introduced a number of Performance Monitor metrics for gathering data on these parts of the system. Within the descriptions of most of the counters, I referred to comparing your metric to a baseline. This chapter will cover how to gather your metrics so that you have that baseline for later comparison. I'll go over how to configure an automated method of gathering this information. A baseline is a fundamental part of understanding system behavior, so you should always have one available. This chapter covers the following topics:

- Considerations for monitoring virtual and hosted machines
- How to set up an automated collection of Performance Monitor metrics
- Considerations to avoid issues when using Performance Monitor
- Baselines for Azure SQL Database
- Creating a baseline

Considerations for Monitoring Virtual and Hosted Machines

Before you start creating the baseline, I will talk about virtual machines (VMs). More and more SQL Server instances are running on VMs. When you are working with VMs or you are hosting VMs in remote environments such as Amazon or Microsoft Azure, many of the standard performance counters will no longer display accurate information. If you monitor these counters within the VM, your numbers may not be helpful from a troubleshooting perspective. If you monitor these counters on the physical box, assuming you have access to it, which doubtless is shared by multiple different VMs, you

G. Fritchey, *SQL Server 2017 Query Performance Tuning*, https://doi.org/10.1007/978-1-4842-3888-2_5

will be unable to identify specific SQL Server instance resource bottlenecks. Because of this, additional information must be monitored when working with a VM. Most of the information that you can gather on disk and network performance is still applicable within a VM setting. All query metric information will be accurate for those queries. How long a query runs and how many reads it has are exactly that, the length of time and volume of reads. Primarily you'll find the memory and CPU metrics that are completely different and quite unreliable.

This is because CPU and memory are shared between machines within a virtualized server environment. You may start a process on one CPU and finish it on another one entirely. Some virtual environments can actually change the memory allocated to a machine as that machine's demands for memory go up and down. With these kinds of changes, traditional monitoring just isn't applicable. The good news is that the major VM vendors provide you with guidance on how to monitor their systems and how to use SQL Server within their systems. You can largely rely on these third-party documents for the specifics of monitoring a VM. Taking the two most common hypervisors, VMware and HyperV, here is a document from each:

- VMware Monitoring Virtual Machine Performance (`http://bit.ly/1f37tEh`)
- Measuring Performance on HyperV (`http://bit.ly/2y2U6Iw`)

The queues counters, such as processor queue length, are still applicable when monitoring within a VM. These indicate that the VM itself is starved for resources, starving your SQL Server instance so that it has to wait for access to the virtual CPU. The important thing to remember is that CPU and memory are going to be potentially slower on a VM because the management of the VM is getting in the way of the system resources. You may also see slower I/O on a hosted VM because of the shared nature of hosted resources.

There's also a built-in, automated, baseline mechanism within Azure SQL Database and any instance of SQL Server 2016 or greater, known as the Query Store. We'll cover the Query Store in detail in Chapter 11.

Another mechanism available for understanding how the system is behaving are the DMVs. It's hard to consider them the same thing as a baseline since they change so much depending the cache, reboots, failovers, and other mechanisms. However, they do provide a way to see an aggregated view of query performance. We'll cover them more in Chapter 6 and throughout the rest of the book.

Creating a Baseline

Now that you have looked at a few of the main performance counters, let's see how to bring these counters together to create a system baseline. These are the steps you need to follow:

1. Create a reusable list of performance counters.
2. Create a counter log using your list of performance counters.
3. Minimize Performance Monitor overhead.

Creating a Reusable List of Performance Counters

Run the Performance Monitor tool on a Windows Server 2016 machine connected to the same network as that of the SQL Server system. Add performance counters to the View Chart display of the Performance Monitor through the Properties ➤ Data ➤ Add Counters dialog box, as shown in Figure 5-1.

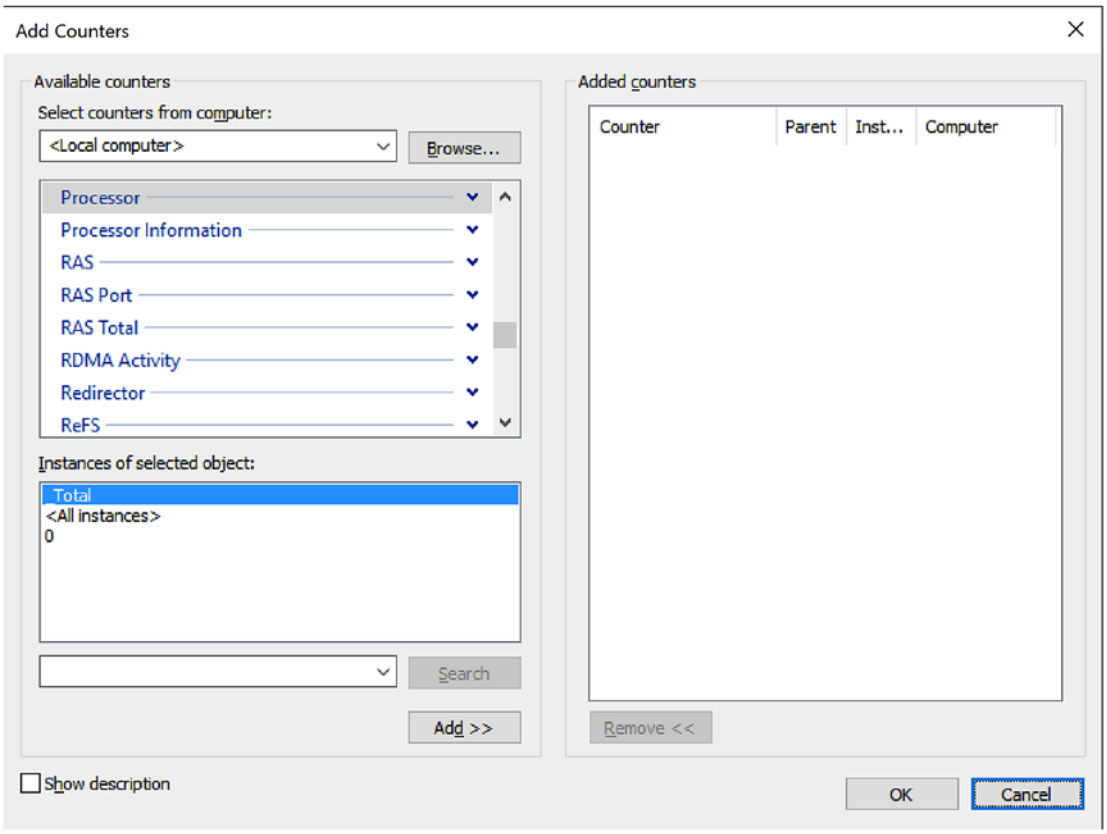

Figure 5-1. *Adding Performance Monitor counters*

For example, to add the performance counter SQLServer:Latches:Total Latch Wait Time(ms), follow these steps:

1. Select the option Select Counters from Computer and specify the computer name running SQL Server in the corresponding entry field, or, when running Performance Monitor locally, you'll see "<Local Computer>" like in Figure 5-1.
2. Click the arrow next to the performance object SQLServer:Latches.
3. Choose the Total Latch Wait Time(ms) counter from the list of performance counters.
4. Click the Add button to add this performance counter to the list of counters to be added.
5. Continue as needed with other counters. When finished, click the OK button.

When creating a reusable list for your baseline, you can repeat the preceding steps to add all the performance counters listed in Table 5-1.

Table 5-1. *Performance Monitor Counters to Analyze SQL Server Performance*

Object(Instance[,InstanceN])	Counter
Memory	Available MBytes Pages/sec
PhysicalDisk(Data-disk, Log-disk)	% Disk TimeCurrent Disk Queue Length Disk Transfers/sec Disk Bytes/sec
Processor(_Total)	% Processor Time % Privileged Time
System	Processor Queue Length Context Switches/sec
Network Interface(Network card)	Bytes Total/sec
Network Segment	% Net Utilization
SQLServer:Access Methods	FreeSpace Scans/sec Full Scans/sec
SQLServer:Buffer Manager	Buffer cache hit ratio
SQLServer:Latches	Total Latch Wait Time (ms)
SQLServer:Locks(_Total)	Lock Timeouts/sec Lock Wait Time (ms) Number of Deadlocks/sec
SQLServer:Memory Manager	Memory Grants Pending Target Server Memory (KB) Total Server Memory (KB)
SQLServer:SQL Statistics	Batch Requests/sec SQL Re-Compilations/sec
SQLServer:General Statistics	User Connections

Once you have added all the performance counters, close the Add Counters dialog box by clicking OK. To save the list of counters as an `.htm` file, right-click anywhere in the right frame of Performance Monitor and select the Save Settings As menu item.

The `.htm` file lists all the performance counters that can be used as a base set of counters to create a counter log or to view Performance Monitor graphs interactively for the same SQL Server machine. To use this list of counters for other SQL Server machines, open the `.htm` file in an editor such as Notepad and replace all instances of `\\SQLServerMachineName` with nothing (just a blank string) .

A shortcut to all this is outlined by Erin Stellato in the article "Customizing the Default Counters for Performance Monitor" (`http://bit.ly/1brQKeZ`). There's also an easier way to deal with some of this data using a tool supplied by Microsoft, Performance Analysis of Logs (PAL), available at `https://bit.ly/2KeJJmy`.

You can also use this counter list file to view Performance Monitor graphs interactively in an Internet browser, as shown in Figure 5-2.

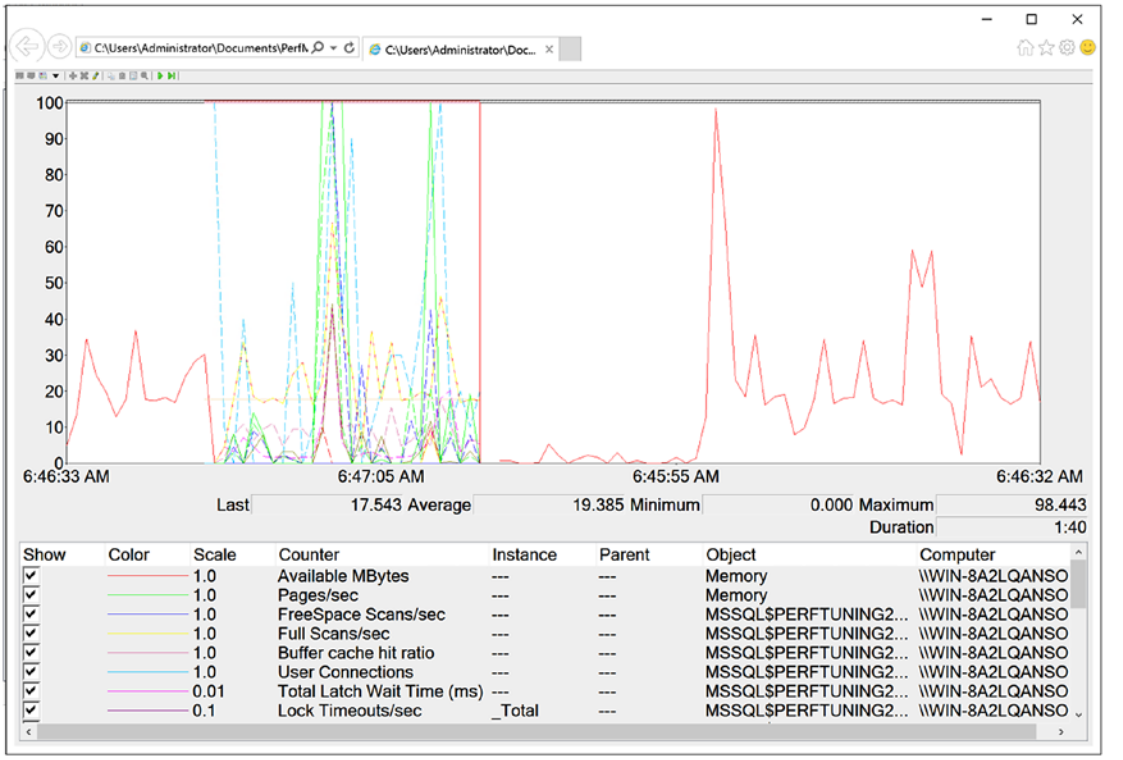

Figure 5-2. *Performance Monitor in Internet browser*

Creating a Counter Log Using the List of Performance Counters

Performance Monitor provides a counter log facility to save the performance data of multiple counters over a period of time. You can view the saved counter log using Performance Monitor to analyze the performance data. It is usually convenient to create a counter log from a defined list of performance counters. Simply collecting the data rather than viewing it through the GUI is the preferred method of automation to prepare for troubleshooting your server's performance or establishing a baseline.

Within Performance Monitor, expand Data Collector Sets ➤ User Defined. Right-click and select New ➤ Data Collector Set. Define the name of the set and make this a manual creation by clicking the appropriate radio button; then click Next just like I configured Figure 5-3.

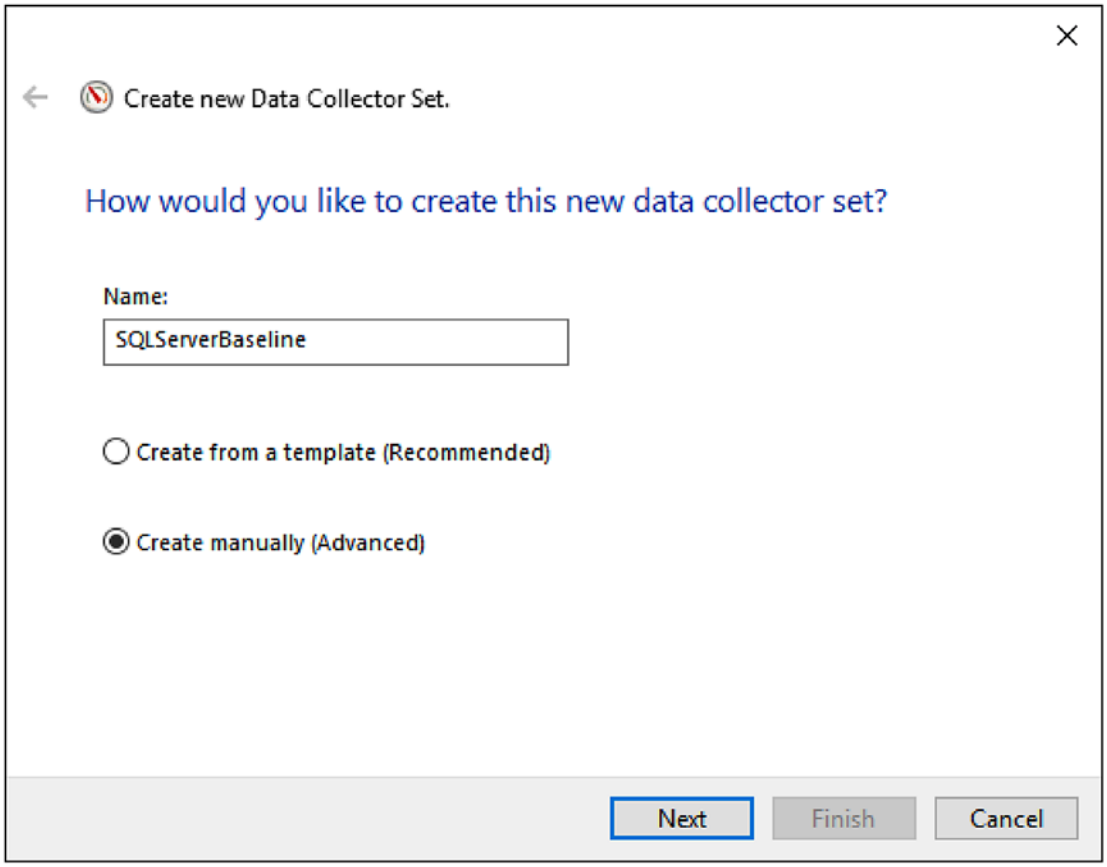

Figure 5-3. *Naming the data collector set*

You'll have to define what type of data you're collecting. In this case, select the check box Performance Counters under the Create Data Logs radio button and then click Next, as shown in Figure 5-4.