

16

Shortcuts to Efficient Data Collection and Quick Analysis

WHAT'S IN THIS CHAPTER

- How to quickly check SQL Server for obvious bottlenecks
- Tools to keep handy and configured, ready to launch

The aim of this chapter is to bring together many of the concepts and tools introduced throughout the book into a concise “this is what you really need to know” guide that cuts through the mustard and enables you to begin troubleshooting with small steps in the right direction.

All the topics are presented in greater depth in other parts of the book, and you’ll find references to the other chapters throughout.

THE FIVE-MINUTE SQL SERVER HEALTH CHECK

How confident are you that your SQL Server is configured optimally? Do you know where your bottlenecks are or which hardware resources will be exhausted first? Do you know if you need more memory or if your disks are fast enough?

This section explains how to give your SQL Server a basic health check, detailing half a dozen quick checks you can make, including the thresholds you need to care about and your options for remediation.

Where You Should Start

The first area to look at on a system running SQL Server is the utilization of hardware resources, the core of which are memory, storage, and CPU.

If you can imagine those three resources in a stack starting with memory, you want to start at the top of the stack, ruling out issues as you work your way down because resource problems further up the stack can cause issues further down the stack.

For example, under low-memory conditions you will likely see an increase in storage activity as the system page file starts to be used as a temporary store for committed memory. This action also drives up CPU utilization, as Windows needs to manage the process of paging.

What you might notice first is the high CPU utilization or that the storage system is running slower than normal. Don't be tempted to add more CPUs or faster disks on this evidence alone, however, because in this case you'll be missing the underlying problem, which is a lack of physical memory.

This is why when you troubleshoot performance problems or review an existing system, you look at the resources in the order of memory, storage, and then CPU. As you work your way down the stack you should be trying to rule out resource problems as much as find the underlying cause.

Examining Memory for Bottlenecks

Chapter 2 goes into a lot of depth on the topic of memory, and you should understand at least the core concepts introduced in that chapter if you're going to delve deeply into reviewing memory utilization.

At this point, however, what you need for the five-minute health review is a set of quick and easy checks to highlight any *obvious* problems so you can determine whether you need to move down the stack or if you've found the issue and need to fix it. Performance Monitor (PerfMon) is an ideal tool to do this (see Chapter 9), and what follows are the counters and thresholds that I use to get a quick indication of any likely memory pressures.

Memory/Available MBytes

This counter indicates the amount of free physical memory available for Windows, and you should aim to keep this at a few hundred MBs on a busy system to ensure that there's always enough memory for any unexpected requirements.

If you really want to squeeze every bit of performance out of the server, however, then keeping this counter consistently above 100MB will suffice as long as the server is stable and consistent.

If this counter is consistently low, then you should lower the SQL Server setting for max server memory (see Chapter 2) or review other potential memory consumers on the server.

SQLServer:Memory Manager/Target Server Memory (KB)

This counter represents the amount of memory that SQL Server *wants* to have. It directly relates to the max server memory setting if one has been configured or the equivalent dynamic value if you've left SQL Server to manage memory.

This counter doesn't tell you much on its own but when you compare it to the next counter you'll understand why it's useful.

SQLServer:Memory Manager/Total Server Memory (KB)

This counter represents the amount of memory that SQL Server *actually* has. It is closely related to the preceding counter and you should compare the two to see whether there is a significant difference between what SQL Server *wants* and what SQL Server *has*.

As long SQL Server has been running for a while or has been busy enough to grow the memory usage, then you should expect these values to be very close if not identical.

If SQL Server doesn't have all the memory that it wants, then make a note that there could be external memory pressure or the max server memory setting might be too high.

Buffer Manager:Page Life Expectancy

The Page Life Expectancy counter (PLE) was introduced in one of the later service packs for SQL Server 2000 and is a great indicator of memory pressure within SQL Server. It shows you the amount of time, in seconds, that SQL Server expects to be able to keep unreferenced pages in cache.

Chapter 2 discusses caching in detail and Chapter 1 introduces the lazywriter, which will drop pages out of cache to ensure that free pages are always available for use. Therefore, if SQL Server doesn't have enough memory to process its workload, then pages will be dropped from cache much quicker than when SQL Server has a lot of memory.

On an OLTP system, Microsoft recommends that PLE is at least 300 seconds, which means that SQL Server expects to be able to keep unreferenced pages cached for five minutes.

In a data warehousing (DW) environment, it's a bit harder to be so prescriptive because you expect many large queries, which will frequently be causing cache flushes, and that's normal. However, PLE can still be useful as an indicator of how well SQL Server is coping with its current memory allocation, and even in a data warehouse I would aim to be averaging a PLE of at least 100 seconds.

PAGE LIFE EXPECTANCY VS. BUFFER CACHE HIT RATIO

Page Life Expectancy is a much better indicator of memory pressure than the counter that was used before PLE was introduced.

Buffer Cache Hit Ratio, which has been a commonly used counter for years, shows the percentage of page requests that were found in cache, rather than having to be read from disk.

It sounds like a perfect counter for measuring the effectiveness of your cache, but in reality the values are mostly very high (high is good) and the granularity is too large to be of any great use. For example, while 99% often indicates things are good, 98.5% can be very bad; even when other counters indicate impending memory pressures, you'll find that Buffer Cache Hit Ratio often changes very little.

Now that you have an idea of how well SQL Server and Windows are coping with the memory situation, you can move down the stack and review the underlying storage.

Checking Storage Performance

Chapter 4 describes in detail how to test and tune your storage, but what you're looking for here (just like for memory) are a couple of checks that you can make to see if the storage is performing adequately. Exactly what to check and what results indicate potential problems, are detailed in the following sections.

LogicalDisk:Avg. Disk sec/Transfer

This counter is a measure of the amount of time, in seconds, that it takes for Windows to make a transfer to disk; and it provides a useful high-level indicator of storage performance.

For disks that contain SQL Server database files you want your disk transfers to be consistently under 20 milliseconds (ms) (0.020 seconds), and ideally under 10ms.

If you're not getting these performance levels, then you can break down this measurement further to see the split between read and writes by checking these two counters:

- LogicalDisk:Avg. Disk sec/Read
- LogicalDisk:Avg. Disk sec/Write

If there's a significant difference between read and write performance, then check the controller cache and the RAID level to see if there's anything you can do to rebalance the cache or recommend a faster RAID type.

For example, suppose you observe the average disk transfer times for your data files consistently running at 75ms, so you check the read and write performance and see average reads at 16ms and average writes at 87ms.

Clearly, the problem is with write performance, so you check the controller cache. It's battery-backed and set at the default of 50% read/50% write, so you could rebalance the cache in favor of writes to quickly gain improved performance.

You also check the RAID level and look at disk sector alignment using DiskPart. It's RAID5, which is notoriously bad for writes, and hasn't been disk sector aligned, so you're missing out on a free performance gain.

You make a recommendation to implement disk sector alignment as a short-term fix and to move to RAID10 to improve both read and write performance.

If anything in this example is unfamiliar to you, refer to Chapter 4 for more details.

sys.dm_io_virtual_file_stats

Checking the Windows view of storage performance using PerfMon is useful because PerfMon is a great centralized tool for an overall health check, but you can also drill down into SQL Server to look at storage performance more accurately.

The `sys.dm_io_virtual_file_stats` dynamic management view (DMV) enables you to view the I/O details for all your database files; and with a couple of simple calculations, you can view the read and write latency that SQL Server encountered on the files themselves.



sys.dm_io_virtual_file_stats is actually a dynamic management function (DMF), but the DMV moniker is generally used for all views and functions starting with sys.dm_.

Listing 16-1 shows the sample code to calculate the read and write latency of all the database files on a SQL Server instance.



Available for
download on
Wrox.com

LISTING 16-1: Script to show I/O latency for all database files

```
SELECT DB_NAME(database_id) AS 'Database Name',
       file_id,
       io_stall_read_ms / num_of_reads AS 'Avg Read Transfer/ms',
       io_stall_write_ms / num_of_writes AS 'Avg Write Transfer/ms'
FROM   sys.dm_io_virtual_file_stats(-1, -1)
WHERE  num_of_reads > 0
       AND num_of_writes > 0
```

Table 16-1 shows sample output from a production system. Files with a `file_id` of 2 are transaction log files, which I have marked with an asterisk.

TABLE 16-1: Sample Output from Query against the `sys.dm_io_virtual_file_stats` DMV Showing Database File Latency

DATABASE NAME	FILE_ID	AVG READ TRANSFER/MS	AVG WRITE TRANSFER/MS
Tempdb	1	3	7
Tempdb	2*	1	3
Tempdb	3	3	7
Tempdb	4	3	7
Tempdb	5	3	7
Distribution	1	30	4
Distribution	2*	3	2
Sales	1	30	11
Sales	2*	4	2
Sales_Web	1	20	10
Sales_Web	2*	2	4

To give you a bit of background on these databases, the server is running a custom retail management system. The Sales database is 60GB and is replicated using transactional replication, and the Sales_Web database is 20GB and also replicated.

All the database files are on a SAN; the data files are all on an 8-disk array configured as RAID10, and all transaction log files are on a 4-disk array configured as RAID10.

From the output you can see that all the transaction log files have a read and write latency of less than 5ms, so there's nothing to worry about there.

You can also see that four data files have been configured for tempdb (see Chapter 7 to understand why you would do this); and that they have an average read and write latency of 3ms and 7ms, respectively, which you would be happy with, as the goal is to be less than 10ms.

The data files for the Distribution, Sales, and Sales_Web databases have the largest latency and are the biggest cause for concern at up to 30ms.

Before you recommend new disks, however, bear in mind that these averages have been calculated since SQL Server started and will include I/O for not just peak times, but also maintenance windows that might skew the results (index rebuilds, for example, will hit the disks hard but won't necessarily impact the end user).

In this situation you can turn to SQL Server Waits (see Chapter 3) to look for waits on `PAGEIOLATCH_*` during busy periods to get an idea of whether or not I/O on the data files is causing a noticeable problem for users.

Now that you have an idea of whether or not I/O might be an issue, you can move down the stack again and review CPU usage.

Looking into CPU Usage

Your main concern when looking at CPU usage during a quick health check is the split between user and kernel mode CPU utilization and the amount of CPU time being consumed by the SQL Server service.

The PerfMon counters you want to look at and how they relate to each other are described in the following sections.

Processor: % Processor Time

In short, this is just a measure of how busy your CPUs are. Whether or not this value is acceptable depends on a lot of factors. Generally, a consistent value >90% is considered to be bad, as the server is working very hard and there is little room for additional workload.

However, if you had just deployed new hardware and saw a consistent value of 70%, you might also consider that bad because it might not provide enough headroom for growth.

All you're concerned about at this stage is determining how busy the CPUs are, and then you'll check what they are working on.

Processor: % Privileged Time

This counter will tell you how much of the % Processor Time is spent handling kernel mode operations. This is useful to know because it's a measure of the time Windows is spending managing its resources, rather than running applications.

Microsoft guidelines indicate a threshold of 30% for this counter, so anything over that could be a problem. A classic example of a problem causing this counter to be high was introduced at the start of this chapter: a low-memory condition.

When Windows is low on memory, data will start to be paged out to disk and the CPU will be working in kernel/privileged mode to handle this. The Processor: % Privileged Time counter will show an increase in value; so whenever you see this you should also check for memory pressure (which is why you should review memory first).

Processor: % User Time

This counter, along with Processor: % Privileged Time, makes up the total % Processor Time. Whereas Privileged Time represents time working on kernel mode operations, User Time represents time spent working on applications (like SQL Server), which is what you want the CPU to be spending most of its time on.

Microsoft guidelines indicate that 70% or greater is a good value for this counter.

Process: % Processor Time:sqlservr

When you measure CPU utilization, observe high % Processor Time, and see that most of that time is spent running user mode applications, it's a good idea to check whether SQL Server is using the bulk of processor time.

You don't want to dive straight into the guts of SQL Server before confirming that's where the problems lies, and this counter enables you to check the CPU utilization for the SQL Server process itself.

There is no specified threshold for this counter. What you're looking for when troubleshooting high CPU utilization is whether SQL Server is using more than any other process. If it isn't, then you should troubleshoot the process that *is* using the most CPU time.

PerfMon Tips

If you have a long data collection from PerfMon, then I encourage you to use the Performance and Analysis of Logs (PAL) tool described in Chapter 9.

However, when looking at a live server, I like to set up the report view in PerfMon with all the counters mentioned so far so I can keep an eye on how the system is handling any changes in workload. Figure 16-1 shows a screenshot from a live system with all the counters added.

\\PRODSQL			
LogicalDisk		R:	S:
Avg. Disk sec/Transfer		0.000	0.000
			T:
			0.001
Memory			
Available MBytes		442.000	
MSSQL\$B2:Buffer Manager			
Page life expectancy		564.000	
MSSQL\$B2:Memory Manager			
Target Server Memory (KB)		31,457,280.000	
Total Server Memory (KB)		31,457,280.000	
Processor			
		_Total	
% Privileged Time		12.856	
% Processor Time		48.966	
% User Time		36.230	

FIGURE 16-1

What Are You Waiting For?

Now that you have a good idea of what's going on at the server level using PerfMon, you can start to look at what's going on within SQL Server; and the first area to look at is SQL Server waits (see Chapter 3).

Wait time is effectively dead time, so if you can reduce the amount of time that SQL Server spends waiting, you'll be able to achieve better overall performance.

Whenever a task in SQL Server has to wait for something before it can continue, information on the reason for the wait is tracked by SQL Server and can be viewed through DMVs. Aggregating this data across all connections gives you a performance profile for SQL Server, and tracking it for a particular connection enables you to determine the bottleneck for a specific workload.

The two DMVs used to troubleshoot SQL Server waits are detailed in the following sections, with sample queries that will help you to isolate issues.

sys.dm_os_waiting_tasks

This DMV, which lists all tasks that are currently waiting on something, is the most accurate for viewing current waits. It contains information to identify a task, an associated session, details of the wait, and blocking tasks.

However, a task only has an entry for as long as it's waiting, so this DMV is most effectively used as a series of snapshots, or to troubleshoot a live issue when a long-running task or a large volume of tasks are waiting.

Listing 16-2 is a basic query for this DMV that shows any task from a user session that is currently waiting for something.

Listing 16-3 is a query that lists any tasks that are running a parallel operation, and shows the T-SQL and execution plans used. Note the use of the `CROSS APPLY` operator to pass in the `sql_handle` and `plan_handle` from `sys.dm_exec_requests` to the `sys.dm_exec_sql_text` and `sys.dm_exec_query_plan` DMVs for every row returned.

**LISTING 16-2: Script to show waiting tasks from user sessions**Available for
download on
Wrox.com

```
SELECT *
FROM sys.dm_os_waiting_tasks
WHERE session_id > 50
```

**LISTING 16-3: Script to show tasks waiting on CXPACKET with the T-SQL and plan**Available for
download on
Wrox.com

```
SELECT wt.*,
       st.text,
       qp.query_plan
FROM sys.dm_os_waiting_tasks wt
LEFT JOIN sys.dm_exec_requests er
ON wt.waiting_task_address = er.task_address
CROSS APPLY sys.dm_exec_sql_text(er.sql_handle) st
CROSS APPLY sys.dm_exec_query_plan(er.plan_handle) qp
WHERE wt.wait_type = 'CXPACKET'
ORDER BY wt.session_id
```

In a high-CPU utilization scenario, this query enables you to instantly identify the sessions running in parallel (which are notoriously CPU heavy), along with the code being executed and the plan that SQL Server decided to use to execute it.

sys.dm_os_wait_stats

This DMV is an aggregation of all wait times from all queries since SQL Server started, and it is ideal for monitoring and serverwide tuning.

Just running a simple query like that shown in Listing 16-4 can be very telling, indicating which wait types SQL Server has waited on the most and enabling you to pinpoint resource bottlenecks.

You can also try ordering the results by `max_wait_time_ms DESC` to view the wait types that have caused the biggest wait time for any single task.

LISTING 16-4: Script to show waits ordered by total wait time

```
SELECT *
FROM sys.dm_os_wait_stats
ORDER BY wait_time_ms DESC
```

You will find details on what the wait types mean and which ones you can discount in Chapter 3.

The wait times are cumulative since the last service restart but you can also reset them by issuing this DBCC command:

```
DBCC SQLPERF ('sys.dm_os_wait_stats',clear)
```

Resetting the statistics can be useful when you want to build the aggregations over a fixed time period or even on a development box before you run a piece of code to see what the aggregated waits are just for that code.

BEST PRACTICES

The best tip I can give you to enable you to respond quickly to live issues is to have all your investigative queries close to hand and ready to run at a moment's notice. Unless you have detailed logging in place through a third-party tool, you'll often find that by the time you get your queries written or loaded, the problem has passed.

I like to add the queries in Listings 16-2, 16-3, and 16-4 as keyboard shortcuts in SQL Server Management Studio so I can respond promptly with a quick keystroke. You can add shortcut keys by selecting Options from the Tools menu and clicking the Keyboard option. Figure 16-2 shows the Options window on my laptop, including the shortcuts I created.

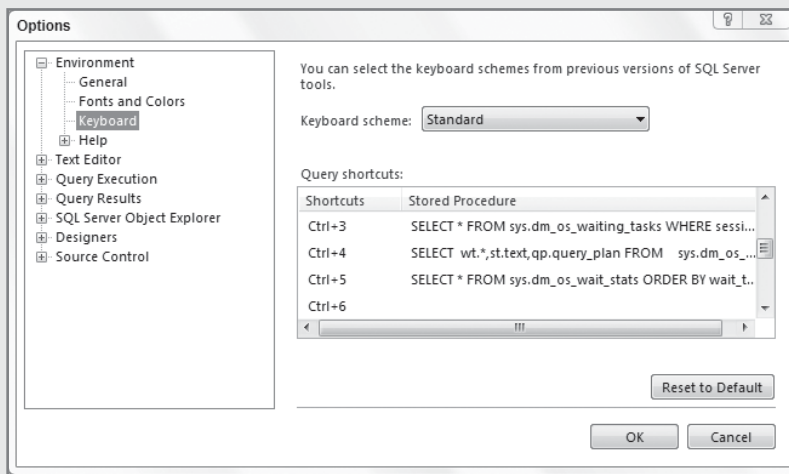


FIGURE 16-2

Now that you've carried out a basic health check of the environment to pick up any telltale signs of potential resource issues, it's time to plan your data collection and subsequent analysis.

TIPS FOR DATA COLLECTION AND QUICK ANALYSIS

In a moment reminiscent of someone spilling the secrets of great magicians, I'm going to tell you how the top SQL Server professionals quickly and efficiently gather data and analyze the results.

You have already seen the tricks of the trade if you've read the rest of the book, and you may well have come to the same conclusion already: All you need for data collection and analysis is the PerfStats script from Chapter 11 and SQL Nexus from Chapter 13.

These two tools are by far the most important resources I have found for troubleshooting customer environments; there are no dependencies on third-party software or even nonstandard SQL Server features, so you can download and run these tools on any SQL Server environment.

The really great thing about the PerfStats script is that it's so well configured out of the box that you don't need to do much to customize it for any scenario that you're facing.

The most common changes that I routinely make are adding a few more trace events to troubleshoot code problems or disabling the SQL Trace and Event Log collectors to just focus on waits and locking.

You can save yourself a lot of trouble by preconfiguring a few different configuration files for different scenarios. I've added a few samples that I find useful to this book's companion website on www.wrox.com, including configuration and batch files for the following:

- Collecting everything needed for full SQL Nexus functionality
- Collecting everything except a SQL Trace
- Collecting only waits and locking information
- Collecting only a PerfMon log
- Collecting only a SQL Trace

A customer recently told me that their former DBA told management that it would take him three months to analyze recent SQL Trace output from their production system (I have no idea what he was going to do with it).

I ran the trace files through SQL Nexus and gave them meaningful results in 15 minutes. Even better, this valuable tool is free!

SUMMARY

This chapter has introduced you to the concept of the five-minute health check, providing tips about how to check for system-wide issues and what to look for. It finished by providing guidance on the tools to use for data collection and analysis.

The key points from this chapter are as follows:

- Always review resources in the order of memory, storage, and then CPU.
- Use the handful of PerfMon counters described to get an indication of where to drill down further.
- View SQL Server waits to determine bottlenecks:
 - `sys.dm_os_waiting_tasks` for current waiting tasks
 - `sys.dm_os_wait_stats` for aggregated server waits
- Use shortcut keys in SQL Server Management Studio to run your best troubleshooting queries with a quick keystroke.
- Learn to love the following two tools that will make your job easier:
 - PerfStats, for consolidated data collections
 - SQL Nexus, to analyze your PerfStats results