1

Performance Tuning

SQL Server 2005 works great right out of the box. For most SQL Server users, performance is never an issue. They install SQL Server, load up their data, and the system works just great. Over the past few years, however, there has been an explosion in database size and in the hardware performance that is available to small and medium-sized businesses. This has allowed many more users to build larger and larger systems. Many of today's systems are now larger than yesterday's enterprise class systems that required mainframe class hardware. This explosion in size and performance is pushing more users against the limits of the out-of-the-box performance of SQL Server. This means that more and more SQL Server users are having to start learning about a side of SQL Server that they never needed to know about — performance tuning.

The good news is that as with the out-of-the-box experience, the SQL Server team has spent a lot of time working on making SQL Server an easy database system to performance tune. Unlike many of SQL Server's competitors, where you need many highly trained and very expensive consultants to diagnose and finetune performance problems, SQL Server has the tools and information available so that just about anyone can jump in and start tuning their databases.

Art or Science?

Performance tuning has always had a certain aura of mystique around it. The few masters of the art are regarded with awe and suspicion by those who haven't mastered the skills. For those outside the inner circle of those masters, the perception is that it takes many years to acquire the skills necessary to become a performance tuner.

The reality is that with a methodical approach, the right tools, and the right knowledge, anyone with a basic knowledge of T-SQL can start taking steps to improve the performance of their database system. As you start to master these skills, performance tuning begins to look less like an esoteric art and more like a focused science.

This book is about providing you with the knowledge you need to start taking a methodical approach to performance tuning, to show you which tools to use and when, and to give you the knowledge you need to be a successful SQL Server performance tuner.

The Science of Performance Tuning

Performance tuning SQL Server is a science. In this chapter I will show you how to get started, and I will walk you through the steps necessary to approach any performance problem — in fact, more than just any performance problem. The approach used here is one that works with just about any problem that needs troubleshooting.

This approach is nothing new or revolutionary. It is just a methodical scientific approach to problem solving using data collection and analysis to determine the root cause of any problem. This is the approach I learned during the three years I spent working in support at Microsoft. It is an approach that enabled me to solve problems for many Microsoft customers on a wide variety of different products and technologies:

1. You start with a problem statement that helps you describe what you are addressing, and what an acceptable outcome might be. This helps frame the rest of the process.
2. From the problem statement you can form some ideas as to what the problem might be. From these early ideas comes a plan of attack. The plan of attack includes details of the data you want to collect to support (or, unfortunately sometimes, refute) those ideas.
3. Once you have some data, you analyze it to confirm or reject the original idea, and use this to refine the problem statement as needed.
4. If the analysis produced a clear culprit as the root cause of the problem (in the case of performance tuning this would be that you have identified a bottleneck), then you proceed to find ways to remove the bottleneck.
5. When you think you have removed the bottleneck, you collect more data, re-analyze, and confirm that the issue is resolved.

Performance tuning is an iterative process. You may need to repeat the above steps many times before the system as a whole arrives at an acceptable level of performance. This doesn't change the basic process that needs to be followed. You simply repeat the steps until you have removed enough bottlenecks that the system has acceptable performance.

> *Unfortunately in some cases you may reach the limits of your budget in terms of time, money, or resources before you have acceptable performance. It is always important to understand what your time, financial, and resource constraints are as you start a project. If these matters haven't already been defined by your client or employer, make sure that you ask them to define these matters for you. Understanding the scope of these expectations is every bit as important to your success as your actual performance-tuning skills.*

The Problem Statement

The first step in this approach is to develop a problem statement. The problem statement doesn't need to be more than a simple sentence or two that briefly describes the issue you are dealing with. This is a very valuable first step as it starts to help you determine the true nature of the problem, and will help you determine a starting point for the following steps.

If you already have a problem query isolated, then the problem statement is pretty simple, and it will read something like, ''Stored procedure X takes too long to run. It is currently taking x seconds and needs to take less than y milliseconds.''

Many times a customer will call for assistance with a problem that they will describe simply as: ''Our SQL Server is running slow.'' Although that may well seem like a good description of the symptom to their end users, it's not really of much use in starting to performance tune a complex database system. Sometimes it may be all you have to go on. However, I would hope that after asking yourself a few questions you can come up with something a bit more descriptive.

Whenever you take a call like that, your immediate response should be to start asking questions. You should ask the customer to describe what it is that's slow. Are there multiple users on the system? Is it slow for all of them? Do they all do the same kind of work? What kind of workloads run on the system, and are there any batch jobs running at any time? These questions are all aimed at digging more information from the customer, partly for your benefit so you can better understand what the problem might be, but they are also aimed at getting the customer to think more about their system and to help them try and arrive at their own problem statement.

A much more focused problem statement would be something like

> *''User X has a slow running query. He uses the Y system, which calls A, B, and C stored procedures in the Foo database. This happens all the time.''*

The key elements to a good problem statement are:

- Who
- What
- When

And one final element that's not essential but can be useful would be to outline a successful resolution to the problem.

Who Is Having the Problem?

Is it just one user reporting the problem? What type of user is it? Does it affect multiple users? Are these users all of one type (for example, do they only access one set of features of the system), or are they spread out across many different features? Where are these users? Are they all in the same department, location, or region? Is their anything else unique about the users reporting the problem?

What Is the Problem?

How did the user report the problem? Is it a slowdown that's impacting them, or are they getting timeouts or deadlocks that are causing loss of functionality? What were they doing when the problem occurred? What else was going on when the problem occurred?

When Did the Problem Occur?

When did the problem occur? Did this just occur? Has it happened before? Is it consistent or intermittent? Is it happening now?

What Is the Resolution?

It is important to know from the beginning what a successful resolution might look like. In many cases with performance tuning, the goal is very poorly described simply as ''make it go faster''. How much faster do you want it to go? What is a reasonable amount of time for the query to take? What is an acceptable amount of time for the user to experience?

These are all vital bits of information that help you arrive at a problem statement that starts to describe the issue you need to investigate.

The Plan of Attack

Once you have a problem statement, you can formulate a plan of attack. This should simply state what you think the underlying problem is, what the data is that you want to collect, and what analysis of that data should show.

This step is short and easy, but it is also very important. It makes sure you start on the next step correctly and have thought about what you are going to do with any data you collect.

An example of a problem statement and plan of attack is shown below.

> *Problem statement:*
>
> *Several users of system X have reported poor performance. They are reporting this problem during peak hours, which for them are from 1 pm through 3:30 pm on Mondays, Wednesdays, and Fridays. They are connecting to database Y on Server Z. This application is the only application using this database and server. The server has dedicated local storage.*
>
> *Plan of attack:*
>
> *Capture Perfmon counters for server resources from Server Z. Review for system bottlenecks. Capture SQL Server wait types from database Y on Server Z. Review for long wait types. Data will be captured during three periods when the system exhibits the performance problem.*
>
> *Capture server level performance counters at 1-second intervals over three 5-minute sampling periods between 1 and 3:30 pm on days when the problem is reported. Capture counters at 5-second intervals between 1 and 3:30 pm on days when the problem is reported. If no baseline is available, capture server performance counters at 15-second intervals over a 24-hour period on days when the problem occurs, and also on at least one day when it doesn't occur.*
>
> *Capture SQL Server Wait stats. Sample at 30-second intervals over a 5-minute period. Repeat three times over the period when the user reports the problem. Review for long wait times.*

The plan of attack should detail what data you intend to collect, what you intend to do with it, and maybe even mention some follow-up actions depending on the analysis of the data collected. Another important aspect of the plan of attack regarding data collection is when you want to collect the data and for how long. This information may come from the problem statement (as in the above example), but in a case where the problem is either continuous or intermittent, you will need to be creative about when and for how long to sample.

Data Collection

The plan of attack has defined what data you want to collect and what you intend to do with it. Now you can go ahead and start collecting the data.

This step will involve setting up the data collection, then sitting back and letting the data roll in for some period of time. This could include collecting hundreds of GB of Performance Monitor logs, running a

SQL Trace for hours or days, or other data collection activity that may have a serious impact on the performance of the system from which you need to capture data.

Carefully consider what data to collect so that you can confirm any theory about the root cause of the problem while minimizing the impact on the system you need to tune.

Data Analysis

After you have collected the data you think you need, the next step is to analyze the data to confirm the original theory.

If after careful analysis of the data collected, it appears to confirm your theory, then you have just identified your bottleneck and can move onto removing it.

In some cases the data analysis may not be conclusive. In this case you may need to make a minor modification to the data collection policy and re-collect data.

The analysis may also provide results that point toward a different problem. When this occurs it is time to go back and revisit the problem statement again and to refine it with this new information.

In some cases the data analysis will show no problem at all. In fact, from the data everything looks just great. This is often the hardest option as you now have to go back to the problem statement with what at first seems like no additional data. You have, however, ruled out one potential cause, so you can start again with a few less places to go looking for the bottleneck.

Performance Tuning Applied

Now that we have introduced the scientific approach based on data collection and analysis, we will walk you through two different scenarios to help illustrate how this process might work in practice. The first scenario is one where your users start reporting apparently random performance problems. In the second scenario, you know exactly where the problem is, but you don't know what the exact bottleneck is.

Example 1 — Your Application Is Slow

This is the typical problem where the phone rings, and it's one of your users calling to report poor performance. This comes out of the blue and is soon followed by many more users all calling to report the same kind of problem with poor performance.

Problem Statement

Writing a problem statement for these general performance problems can be quite hard, but it's okay for it to be very general. As you start collecting data, you can refine the problem statement, making it more specific with each iteration. Here is a first draft of a problem statement for this problem.

Problem Statement:

Users of application X on Server Y are reporting performance problems when accessing features Z1, Z2, and Z3.

Plan of Attack

When you don't have a clear target for data collection, it is necessary to start at the top (the server resources) and work down until you find a bottleneck that is the root cause of the problem. In this case, on the first iteration you can collect multiple sets of data. A good place to start is with collecting performance counters of the server and SQL resources, maybe also collecting wait stats, and capturing a Profiler trace looking for long-running queries. This can be a lot of data to collect and analyze. Although it's a nice idea to cast a broad net like this, in practice it's better to start with just a few data sets and zoom in on the problem from there.

> *Plan of Attack:*
>
> *Collect Perfmon counters of the top level server and SQL resource counters at 5-second intervals for 20 minutes at some point between 09:00 and 11:00.*

There are a couple of things to note about this plan of attack. It's gone into quite a lot of detail about the frequency of counter sampling, how long the counters need to be collected for, and over what time the counters should be collected. This is all important information to have thought about and to define at this stage. In a larger organization or anywhere where you are dealing with a team of people who are responsible for the servers, this is very important as it may often be that this information has to be passed from the DBA team to the Ops team that will actually run the data collection.

Even if you don't have an environment with a separate team, it's still very useful to go into this amount of detail as it forces you to think about how much data you really need to collect.

Data Collection

With the plan of attack defined, you can proceed with data collection. In this case it's a pretty simple set of data to collect. This can be made even easier if you have a set of Logman scripts around that set up counter logs. An example of using Logman to set up a counter log is given in Chapter 12.

In the case where a separate Operations team manages the server, this is where you sit back and wait for the files to be dropped onto a shared server somewhere. If your environment is smaller and you are responsible for all activities on the server, then you will be busy setting up and running the relevant data collection tools.

Chapters 2 and 3 cover using Performance Monitor to capture Server and SQL resource counters.

Data Analysis

Once the data collection is complete, the task of analyzing the data starts. In this example there will be a single Performance Monitor log file to be analyzed. Chapters 2 and 3 cover interpreting the contents of the Log File.

Once the analysis is complete, it will hopefully point to a resource bottleneck with a server resource like CPU, Memory, or disk I/O. Alternatively, it might point to a SQL Server resource bottleneck.

The third option is that it doesn't indicate any resource bottleneck. In this case you will need to refine the problem statement and plan of attack and collect more data. The next iteration should focus on looking either at SQL Server waits or a Profiler Trace looking for long-running queries.

Example 2 — Stored Procedure X Is Slow

This example is a very different kind of problem. In this example, the problem is very well-defined. Rather than the whole application slowing down as in the previous example, in this case, a single function is reported with the problem. In this case you can trace the feature to a single stored procedure, or a small set of stored procedures. In this example we will deal with a single stored procedure.

Problem Statement

Unlike the previous example, in this example you know where the problem is, but you don't yet know what is causing it. There could be many reasons for a stored procedure slowing down. The goal of this performance tuning operation will be to determine why the stored procedure is slow. In essence, this means finding the bottleneck to be able to remove that bottleneck.

In this case, you can write a much more focused problem statement.

Problem Statement:

Stored procedure X is running slowly. In previous executions it takes on average 10 msec, with a min of 2 msec and a max of 15 msec. Since this morning it has been taking on average 600 msec. The goal is to identify the root cause of this slow-down and tune the stored procedure to improve performance back to the original execution speeds.

Plan of Attack

With a much more focused problem statement, the plan of attack will be correspondingly more focused. You shouldn't need to look at the server as a whole, and you don't need to find one particular long running query. A lot of the work you had to do in the previous example isn't needed here. A good plan of attack for this example is listed below.

Plan of Attack:

Review the execution of Stored Procedure X. Capture an execution plan of the stored procedure. Review the plan and tune the stored procedure to optimize execution time.

Data Collection

The data collection in this example is much easier, although there can be challenges with capturing a plan for the stored procedure. In some cases this might be as easy as running the query in SQL Server Management Studio with the relevant options enabled to show a graphical plan or to return a text plan. In other cases the data passed to the stored procedure is critical. In these cases you might need to capture real parameters using a Profiler Trace so that you can execute using the real data in Management Studio. In other cases you might need to use SQL Server profiler to capture the actual execution plan of a live running instance of the stored procedure being called by the application.

Data Analysis

The data analysis stage in this example is all about interpreting the execution plan and tuning it. This is covered in Chapter 9.

The iterations here are around the data collection and analysis pieces. As the problem has already been narrowed down to a particular stored procedure and the goal is to improve the performance of this

one stored procedure, there isn't usually a need to revisit the problem statement or the plan of attack. However, in some cases the reason for the stored procedure running slowly might be due to a server resource bottleneck. It might be that the data has grown to the point where processing it now takes more CPU than is available, that it needs more memory than is available, or that it now needs more I/O capacity than is available. In these cases then the problem statement remains the same, but the plan of attack might change to alter the data collection to review the server or SQL Server resources.

Tools

There is a wide variety of tools available to help you in your data collection and analysis, and to help with preventive measures such as monitoring activity, and capturing a baseline of your system's performance.

System Monitor

Also known as Perfmon or Performance Monitor on Windows Vista, this is the first tool you should think of when looking at performance tuning. There is a massive number of counters available to show you all aspects of performance from many different applications. Chapters 2 and 3 cover using System Monitor and discuss in detail which counters to look at.

SQL Server Profiler

SQL Profiler is the tool of choice when you need to find long-running queries in a highly variable workload. Profiler lets you capture a record of every query executed by SQL over a period of time. This is extremely useful when either there is a wide variety of queries run infrequently in the server or there are ad hoc user queries running as well. Under those conditions other tools don't help you find the long running query, and that's where Profiler comes in.

Using Profiler you can also capture a workload over a given period of time and then use this later to replay against a restore's database system. This is a great way of running a stress or performance test, and for repeatedly reproducing a database workload.

Chapters 5 and 10 discuss using SQL Server Profiler.

SQL Server Management Studio (Query Analyzer)

For many SQL Server users, SQL Server Management Studio will be the application they spend their work lives inside. It is now a Visual Studio compatible integrated development environment (IDE) and provides a single place to perform all your SQL-related work. Starting with a new Visual Studio solution/project-based approach, it includes:

- A server/database object explorer
- The template explorer, which is an invaluable aid for writing T-SQL scripts
- The Query Analyzer interface
- SQL Server profiler
- Database Tuning Advisor (DTA)
- A shell to launch third-party tools

SQL Server Performance Dashboard

SQL Server Management Studio comes with many performance-related reports already built. The SQL Server Performance Dashboard reports are a suite of reports that can be downloaded and installed.

These reports are a new feature that shipped after SQL Server 2005. The reports provide a wealth of performance information with no work required other than installing them. All you need to know is that they are there, where to find them, and how to read them. Chapter 13 covers the SQL Server Performance Dashboard.

Dynamic Management Views

The Dynamic Management Views (DMVs) in SQL Server 2005 are the source of a wealth of information about what is going on inside SQL Server. In earlier versions of SQL Server, some of this information was made available in system tables. In SQL Server 2005, the amount of information about what SQL Server is doing internally has increased dramatically.

Anyone looking at SQL Server performance should have a good understanding of the key DMVs.

Many of the chapters in this book discuss in detail the DMVs relevant for each chapter's topic. Another great source of information on using these DMVs is the SQL Server Best Practices website at:

```
http://technet.microsoft.com/en-gb/sqlserver/bb331794.aspx
```

This page includes a link to the SQL Server Best practices toolbox, which contains yet more extremely useful Scripts for querying the DMVs:

```
http://www.microsoft.com/technet/scriptcenter/scripts/sql/sql2005/default.mspx?mfr = true
```

Direct Administrators Connection — DAC

Sometimes a DBA trying to diagnose a busy server needs to find who is using all the resources. Other times a DBA needs to kill a long-running query. One of the challenges is that it is not always possible to get a new connection to SQL to start looking at what is going on. This is because the server no longer has any resources available to create a new connection.

SQL Server 2005 resolves this problem with the Direct Administrators Connection. This is a special connection that uses considerably fewer resources and is quite strongly restricted in what it can do, but it does allow a DBA to connect to a system that wouldn't be available otherwise. This is a tremendously useful feature and one that every SQL Server performance tuner should be aware of.

If your server is in a state where you need to use the DAC to connect, you want to make sure you use as few resources as possible. Connecting using SQL Server Management Studio can be very resource intensive as the simple task of clicking different nodes in the server explorer can cause resource intensive queries to run on the server.

The better option for using the DAC is to connect through SQLCMD, the command line interface to SQL Server. Using SQLCMD will use much fewer server resources than using SSMS, but it does challenge the user to know the many useful DMVs needed to identify any resource-hogging queries. Alternatively, you can keep a suite of useful SQL Scripts accessible and run these through SQLCMD across the DAC to find and kill the resource hog. The question is where to keep these valuable scripts so they are accessible. Some useful places include:

- A USB thumb drive
- A team website
- A well-known file share

Each has its advantages and disadvantages. You can try and keep your most useful scripts in all three locations. The challenge then is keeping them all synched with the latest version. Robocopy and a good source code control system can really help.

PSSDiag

PSSDiag is a general-purpose data collection tool used by CSS to collect data from a customer's server. If you are familiar with this, it is most likely because you called CSS with a server performance issue and they asked you to use it to collect data. PSSDiag can be found by searching the web for the latest download.

SQLDiag

SQLDiag is another CSS data collection tool, but this one is focused on collecting SQL Server statistics to help a support engineer diagnose your SQL Server performance problem. SQLDiag can also be found by searching the web for the latest download location.

Blocking Scripts

This CSS tool is a set of scripts to help identify blocking issues. Check out the PSS SQL Server Engineers Blog for the latest information on the new version called *Perf Stats Script*. A web search for SQL Blocking Script should reveal the latest download location.

Network Monitor and Netcap

Network Monitor (Netmon) and the network monitor capture utility (Netcap) are tools for capturing network traffic. Netmon includes both capture and analysis tools. Netcap is just a capture utility. Capturing a network trace can be helpful when the issue might be a connection problem or an authentication problem and it is not possible to see what is going on with the other SQL Server tools. The ability to capture a trace of every packet sent to and from the server over the network is an invaluable aid, although interpreting the results can require a lot of time and a deep knowledge of network protocols. In many cases there are protocol wizards built into Netmon that will help with interpreting network packets by breaking down the raw data into the relevant data structures.

Windbg, ntsd, and cdb

Windbg, ntsd, and cdb are the Windows debuggers. These are hardcore code development debuggers, and you wouldn't normally expect to hear anyone mention them in a discussion on SQL Server. However, they can be extremely useful for diagnosing client-side performance issues where there is no low-level tracing.

Visual Studio 2005 Team Edition for Database Professionals

Sometimes also referred to as Data Dude, this is a new application life cycle tool to empower team development of SQL Server. Anyone working on a production database should be using this, even if they are not on a team. The basic product has enough cool features to make it a great addition for any DBA. The team also recently released a cool suite of Power Tools that add some additional features. Keep an eye on Gert Draper's blog (listed in the next section) for the latest breaking news on this tool.

Knowledge

Another invaluable tool in performance tuning is knowledge, and there is no better source on internal knowledge on SQL Server than the blogs of the people who work on the product themselves. Some of the many useful blogs include:

- **SQL Server storage team blog:** `http://blogs.msdn.com/sqlserverstorageengine/default.aspx`
- **Gert Draper:** `http://blogs.msdn.com/gertd/`
- **Euan Garden:** `http://blogs.msdn.com/euanga/default.aspx`
- **Slava Ok:** `http://blogs.msdn.com/slavao/`
- **SQL Engine Tips:** `http://blogs.msdn.com/sqltips/default.aspx`
- **Ian Jose:** `http://blogs.msdn.com/ianjo/`
- **Wei Xaio:** `http://blogs.msdn.com/weix/`
- **SQL Manageability:** `http://blogs.msdn.com/sqlrem/default.aspx`
- **Query Opt team:** `http://blogs.msdn.com/queryoptteam/`
- **Craig Freedman:** `http://blogs.msdn.com/craigfr/`
- **SQL CAT Blog:** `http://blogs.msdn.com/sqlcat/`
- **SQL BP website:** `http://www.microsoft.com/technet/prodtechnol/sql/bestpractice/default.mspx`

Preventative Measures

Routine monitoring of SQL Server systems will help show performance problems as they start to arise, and not as they erupt into crisis. Addressing a small issue as it starts is often a lot easier than dealing with an erupting volcano of an issue when it occurs unexpectedly.

Capturing baseline performance numbers and understanding workload characteristics are another important part of monitoring system performance. A customer question might be about the value for a specific performance counter. This might take the form of a question something like this:

''Our system is getting 50 transactions per second. Is that OK?''

Without any idea what the hardware is, what the database looks like, and how much work is there in each transaction, the numbers are meaningless.

On the other hand, what if the customer had a performance baseline that showed that for a specific workload they could achieve 75 transactions per second with 100 users at 50 percent CPU load with an I/O throughput of 1.5 MB/Sec at a latency of 2 msec? From that baseline they can now see that only getting 50 transactions per second is only about two-thirds of what their system is capable of.

If they also noticed that those 50 transactions per second are consuming 95 percent CPU and the I/O throughput is now at 4 MB/Sec with a latency of 25 msec, that's a good indication that there are some

serious problems. First of all you are getting 60 percent of the transactions with almost double the CPU load. That's a great indication you either changed a query somewhere, you are getting a sub-optimal execution plan, your indexes are shot, you reached some size threshold on one of the tables, or some other bottleneck is occurring.

Without the baseline, 50 transactions per second is just a talking point with no point of reference to compare to. With the baseline, 50 transactions per second is the start of a meaningful investigation into a performance issue.

Part III of the book covers some of the basic techniques for monitoring and baselining system performance.

Summary

Performance tuning isn't an art, it's a science. In this chapter you learned about the science of performance tuning. You started with the steps in the methodical approach to performance tuning: starting from the problem statement, moving onto the plan of attack, data collection, followed by data analysis, and then repeating until you achieve your goal. You were introduced to some of the many tools available to assist with performance tuning. Some of these are covered in more detail throughout the remainder of this book. In the next chapter you will learn about using Performance Monitor to look at server resources.