IDERA | SQL Server Whitepaper

BEGINNING PERFORMANCE TUNING

BY PINAL DAVE
DEVELOPER EVANGELIST

SOLUTION OVERVIEW

To get more out of everything we buy is intrinsic to human nature. We all have this in our DNA. When we do investments of any kind, we look for various parameters before we take the calculated risks. Assume we are at a vehicle showroom, out of the thousand questions we ask from speed, new accessories and lastly the mileage. We enquire so much to get the best and want to know what each and every button can do as functionality in the vehicle we buy. These behaviors don't change in the IT too. Ask a developer or DBA, their most dreaded time is when the server is unresponsive or the application requests are not being served. All of us have a need to do some sort of performance tuning and often are overwhelmed with the situation in hand. This only creates confusion and we resort to the common resolution – using a search engine.

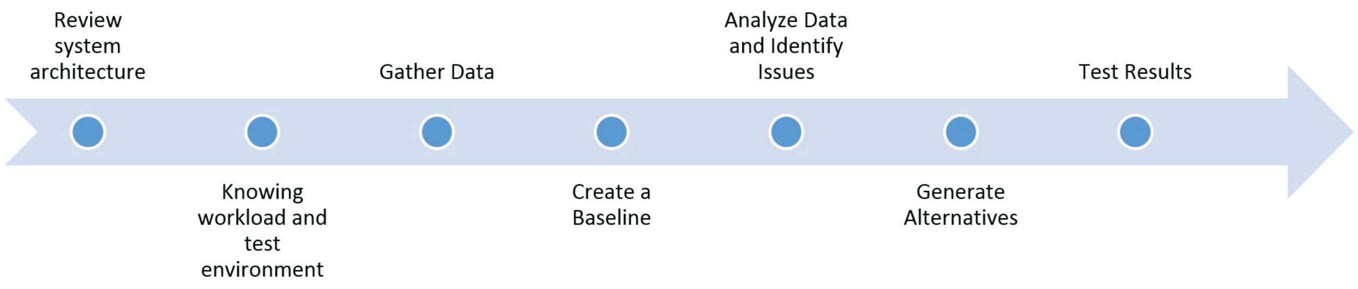
We want to get back to basics when working with Performance Tuning SQL Server.

TABLE OF CONTENTS

STARTING PERFORMANCE TUNING

Performance tuning is the methodology of systematically identifying bottlenecks in applications and improving performance. There are a number of places to start performance testing. Performance for an application can be a bad on various dimensions:

- Operating System
- Deployment – Virtualization or not
- Platform - Web Server, Frameworks used etc
- Database
- Configuration Settings


Review system architecture
Knowing workload and test environment
Gather Data
Create a Baseline
Analyze Data and Identify Issues
Generate Alternatives
Test Results

Though each of the above can be tuned in different ways and configurations, in essence performance tuning is fundamentally about proper planning. Let us walk through this process first.

Review System Architecture

Knowing your system's architecture is the first step. Always keep your application's process flow diagram handy to understand how data will flow inside the system. This diagram will also aid in brainstorming future enhancements. The purpose of this step is to make sure everyone working on the application is familiar with the process that is being used.

Knowing Workload and Environment

End users often give us a generic statement like – "Application is not working", "Application is unusable", "Application is erroring" etc. They are telling us something is wrong in the system but it doesn't help us in solving the situation.

When such statements pop-up it is important to know that your system is:

- Running error-free
- The environment is stable
- The machine resources are not saturated

Audit your environment:

- Are there are any periodic / regular issues (failures, errors, deadlocks, timeouts etc.)?
- Check for common issues like: disk space utilization, memory utilization, CPU peaks, Network interface errors or load.
- Are there any anomalies that point you to possible cause(s) of slowness in the system?

Gather Data

The most common tools to start performance testing are tools available out-of-box with the Operating System. Observe the system under PerfMon with a collection of counters (once every 10 seconds) which do not adversely affect performance. Look for trends and anomalies in the graphs collected.

Create a Baseline

When you have a system running in production, you start by collecting an initial set of data. The first time the data is collected, it is called as baseline measurement. This denotes the normal behavior of the system – CPU, Memory, Disk, Network, Query performance etc.

If a baseline is available and when the system is stalling or giving errors, we can always collect the data again to see where the deviation in numbers are.

Analyze Data and Identify Issues

Though PerfMon is a great starting tool, find out is there are any counters indicating a performance problem: excessive exceptions, excessive recompilations, page splits, paging, disk queuing or other queuing?

We recommend starting with PerfMon because we will be able to get an idea of where a possible problem is. Once we have found the high-level reason for performance bottleneck, we can always use the other tools to find the root cause.

If you identified a possible high-level issue – try to understand from workload what changed recently. If there is sudden depletion of memory and you see a pattern of too many reporting queries in the workload – we have a starting point.

Generate Alternatives

If you find an issue, the next step is to validate your theory in a test environment. It is critical to be 100% sure of the next steps before taking bold actions. It is highly recommended that you make one change at a time to check how the system responds to that change. If you know the performance was because of a recent upgrade / deployment then -

- Implement a quick fix or rollback the recent changes to see the effect.
- Disable a portion of code, to determine the impact on execution time if queries are running slow.
- Check if reads can be scaled out to another server for reporting needs.

Test Results

Once you find a possible cause in the test environment and a possible fix for the same, verify the theory with results on the production box. Check if it made any difference to the system.

- Did the last change make an improvement? If not, examine why.
- Determine if it is necessary to back-out the last change. Even if it was successful, does it disable something that is needed for continued tuning?
- After fixing any problem, evaluate the results and create a new baseline. This should be the new baseline when the system is in a normal state after new code changes.

Document Everything

Record every activity that has been done on the production box from time to time. It is always advisable to have a secure, reliable backup handy at any time so that you can always rollback to the last reliable state. Though you started the above using PerfMon, let us look at some of the fundamental tools one can use with SQL Server.

SQL SERVER TOOLS – TOP 6

A lot of times the basic tools are the ones that help us a lot. We are a big fan of the statistics T-SQL commands, which can give us a wealth of information in a simple way. We will start our journey with query tuning which is one of the most requested item.

SET STATISTICS ON

The below table will summarize what is available in SQL Server for query tuning requirements for more than 15 years.

STATISTICS TIME ON	Gives us information about how much time was used for parsing and compilation. Post the execution of a given query, it also shows the CPU and elapsed time for executing a particular query.
STATISTICS IO ON	As the name suggests – it shows the table, scan counts with logical and physical reads for a given query. This has been one of the handy tools if you want to identify which table in a complex query is causing all the reads.
STATISTICS PROFILE ON	This shows a profile of the execution plan along with the result set from query. This can be used for advanced tuning techniques. To execute this command, one needs to have SHOWPLAN permissions on the objects part of the query.
STATISTICS XML ON	Shows the XML view of the execution plans

As mentioned before, these SET commands have been with SQL Server for a while and are really powerful in getting a firsthand view of how a query runs / performs.

SQL SERVER PROFILER

SQL Server Profiler is yet another tools that can be useful when trying to diagnose what is going wrong in our SQL Server environment. This tool has been enhanced in every release. This tool is basically a tracing tool for our SQL Server. It listens to all the incoming traffic and collects information on what is getting executed in the system.

Profiler has rick interface to create, manage, analyse and reply trace data. It also allows us to use templates to save them in trace files for later analysis and reply.

Though SQL Server Profiler is a deprecated feature from SQL Server 2012 version, it is good to know the various building blocks that are part of Profiler:

Event	These are actions generated by the SQL engine like Logins, Insert statements, Update statements, start/end of SP, errors, locks acquired, security checks, cache additions etc.
Event Class	We define the type of event to be traced. Typical example is Audit Login, SQL: BatchCompleted etc.
Event Category	A category defines a logical grouping of events inside SQL Server Profiler.
Data Column	These are attributes of an event class. For example: BinaryData, TextData, ApplicationName, LoginName, CPU, Duration, SPID etc.
Template	This defines the configuration for a trace with event class to monitor. Saved as a file (.tdf) this gives us a starting point for quick trace initialization.
Filter	When a trace is initialized we can define criteria to filter the data collected by this trace. The idea here is to collect relevant data while keeping the trace file under check.

SQL SERVER MANAGEMENT STUDIO

From SQL Server 2005, SQL Server Management Studio (SSMS) is the one stop tool for configuring, managing, accessing, administering and developing with SQL Server. The current version of SSMS can work with SSAS, SSRS, SSIS, Azure databases and more. The SSMS is built on top of Visual Studio shell and can be of great help in troubleshooting. Two things capabilities which is worth mentioning here:

Activity Monitor – This can be invoked using the toolbar or CTRL+ALT+A. This is a details page allows us to have a quick glance on Running Processes, Resources Waits, Data File IO access and Recently Expensive Queries. Apart from it, is also shows a graphical view of Processor Time, Waiting Tasks, Database IO and Batch Requests/sec.

InBuilt Reports – There are a number of reports inbuilt with SSMS. Right click at the database node to view DB specific reports and right click on the Server node to get access to server specific reports. Some of the reports like Schema Change History can be handy in understanding the DDL commands executed recently inside the database / server. The schema change history report depends on the default trace to be running.

SQL SERVER ERROR LOGS

Error logs perse are not a performance tool, but it contains a wealth of information which can come handy when there are performance problems. When SQL Server starts it logs a number of information like: # of processors available and how many SQL Server is using, total available memory, current version of SQL Server and Windows, Authentication mode selected, startup parameters, default collation used etc. Additionally, if configured, error logs can also contain information about deadlocks or stack dumps.

SQLDIAG

SQLDIAG is a configurable data collection tool. “Out-of-the-box” it can collect event logs, Performance monitor logs, Profiler trace for both SQL and AS, blocking script, SQLDiag reports, machine configuration information to target performance and stress testing. SQLDiag can be configured to collect virtually any type of custom diagnostic, including T-SQL scripts and commands, batch files and executable, VBScripts and other ActiveX scripts, file copies, file lists, registry queries and saves etc. Tool also leverages its execution at pre-determined time. For example if we have configured a stress test script at 10 pm given date and need logs to collected starting from this time and you are not going to be available at that time, just execute this tool and pass above mentioned time as parameter and done.

SQLDiag is installed as part of SQL 2005 onwards tools; you don’t need to install extra add-ins. The executable and its associated xml file is installed at

- <SQL Server install location> \110\Tools\Binn\sqldiag.exe (for SQL 2012 installation)
- <SQL Server install location> \100\Tools\Binn\sqldiag.exe (for SQL 2008 installation)
- <SQL Server install location> \90\Tools\Binn\sqldiag.exe (for SQL 2005 installation)

GRAPHICAL EXECUTION PLANS

An execution plan is the steps that SQL Server tools behind the scenes to execute a query. To decode an execution plan is an art. Most programmers / DBA's use this powerful tool to see where their queries are spending most of the time.

The execution plans give a blueprint of – what are the logical / physical operators used, estimated rows vs actual rows, if parallelism was used, number of rows from a given operator, how filter conditions were applied, which indexes were used and much mode.

Some of the other tools that we can use include:

- Windows Error Logs
- Task Manager
- Distributed Replay Utility
- Extended Events
- Database Tuning Advisor
- Database Console Commands (DBCC)
- Built-in DMV and DMF functions

5 GENERIC DATABASE TIPS

Maintain Your Index

Well maintained indexes can prevent performance degradation caused by low page density level and/or logical and extent fragmentation. Poorly planned maintenance can cause severe performance problems. Rebuild your indexes when the fragmentation levels are above 30% and create a maintenance plan which will run at specific time of the week / day.

Watch Out for Blockings and Deadlocks

This is a simple tip. Look out for deadlocks in the systems and try to find the root cause for the same. If there are blocking threads, check why the system is taking longer than expected time to execute. We can use "Blocked Process Threshold" option to monitor blocking behavior in the system which are abnormal.

Large/Long Transactions can be Dangerous

Where possible, don't update or read a large table all at once. Full table reads or updates can overwhelm servers and/or networks. In addition, they may trigger lock escalation and result in unintended blocking. Furthermore, when data modification is long running due to transaction batch size, it may take a long time to roll back when you decide to roll back the transaction. Try to do modifications in batches and in smaller chunks of transaction.

Consider aging Out Old Data

Some tables are full of records that will no longer be used and are filtered out by date range or other conditions. Large tables make indexes less efficient and potentially the data is less contiguous on the disk. “Cleaning as you go” won’t work in all situations, so you might need to create a process to scrub tables periodically. In some cases, a combination of both strategies is necessary.

Recompilation can be Fixed

When heavily used stored procedures recompile due to various reasons (such as set option change or temp table reference in a cursor), this will use a schema stability lock and block other spids that execute the same stored procedure. Use SQL profiler to capture the SP: Recompile event with objected and eventsubclass columns to identify the stored procedure that is getting recompiled and the reason for the recompilation. Take corrective actions based on the eventsubclass column values

CONCLUSION

Performance tuning is an art. Most often we are pushed to explore new ways to do performance tuning and testing. In this whitepaper, we looked at a possible approach to performance troubleshooting. Start from basic tools to find a high-level problem area and then dig deep into solving the problem. Just like when we are sick, the doctor asks a number of questions, does a number of tests before concluding what is wrong. In a similar way, we need to do the basic initial diagnosis, do proper tests and finally find what would be a proper solution to performance problems in our SQL Server database.

ABOUT THE AUTHOR

PINAL DAVE

Pinal Dave is a Developer Evangelist and Technology Enthusiast. He has authored 11 SQL Server database books, 11 Pluralsight courses and have written over 2900 articles on the database technology on his blog at a http://blog.sqlauthority.com. Along with 9+ years of hands on experience he holds a Masters of Science degree and a number of certifications, including MCTS, MCDBA and MCAD (.NET).,