IDERA® | Database Tools Whitepaper

ADVANCED SQL SERVER PERFORMANCE TUNING

CONTENTS

INTRODUCTION

The roles of database professionals are always evolving. However, their tuning skills should remain sharp. Tuning proficiency is still one of the critical requirements for both database administrators and developers, whether the database is on-premises or delivered via a platform such as infrastructure-as-a-service or platform-as-a-service. This whitepaper discusses advanced database performance tuning for both on-premises SQL Server and Azure SQL Database. It also covers tips and techniques for troubleshooting bottlenecks with complex causes and how to remediate them for hardware, operating systems, and the database itself.

OVERVIEW

Fixing a slow query is often a simple matter of adding an index. However, solving this problem may require other methods. A database administrator may also need other tools such as indexed views, plan guides, and the query store to solve unusual performance issues. For example, a large number of tenants with wildly different amounts of data impacts performance in a way that needs a specialized approach to solve. Database administrators cannot always fix these problems. However, they can often find temporary solutions that will last until a long-term fix can be implemented. You may not use these tips every day. However, you should find opportunities to improve SQL Server performance with them.

SQL SERVER

Disk latency results in blocking and is the most common cause of bottlenecked performance in SQL Server. Bottlenecking can occur in both the disk and fabric of the database. The disk is the physical media where the data is stored. At the same time, the fabric of a database is the communication path between the application and disk, including the host bus adapter (HBA), fiber adapter ports, and network switch. Database administrators can observe bottlenecking with a variety of tools such as dynamic management views of SQL Server and Windows tools like Perfmon, Resmon, and Storport.

AZURE SQL DATABASE

System monitoring tools in Azure SQL include sys.resource_stats for the master database, which collects performance data every five minutes. The dynamic management views for the Azure PaaS include sys.dm_db_resource_stats, which shows resource utilization for each database. It collects data every 15 seconds and stores it for one hour. Database administrators can also use storage area networking tools and tools from the Azure portal to determine when the performance of a SQL Server database is bottlenecked.

Storage Limitations

The General Purpose service tier in Azure provides every database file with dedicated IOPS and throughput based on file size, such that bigger files get more resources. Small databases may require more storage, depending on their input and output operations per second and throughput requirements. You may also be able to improve performance by increasing the file size if the input and output latency on database files is high, or its input and output operations per second per throughput is reaching the limit. Azure also has an instance-level limit on the maximum log write throughput, which is 22 MB/s. That means that you might not be able to reach the maximum file throughout on the log file because you have reached the instance throughput limit.

RAM Considerations

The available RAM in Azure SQL depends on the VCORE count of the database since adding more VCORES increases RAM. Four VCORES may provide sufficient processing capability. However, 20 GB RAM might not be enough for a 500 GB database.

SQL SERVER AND AZURE SQL DATABASE

The following techniques apply to both SQL Server and Azure databases.

Viewing Blocking with DMVs

Several dynamic management views provide information on blocking. These include sys.dm_tran_locks, which replaces syslockinfo and sp_lock. Each row has information on the requests for a resource. For example, a request_status of WAIT means that resource is being blocked. sys.dm_exec_requests indicates blocking with a suspended status and a blocking_session_id greater than 0, while sys.dm_os_waiting_tasks also does so with a blocking_session_id greater than zero. sys.dm_exec_sessions joins with the above dynamic management views on session_id to provide additional details on blocking.

Queries

Dynamic management views that provide query statistics include sys.dm_exec_query_stats, sys.dm_exec_trigger_stats and sys.dm_exec_procedure_stats, which were introduced in SQL Server 2005. sys.dm_exec_query_profiles was also added to SQL Server 2014, which requires using SET STATISTICS PROFILE ON. This tool monitors the progress of queries in real-time, typically to identify the part of the query that is running the slowest. Beginning in SQL Server 2016, sys.dm_exec_function_stats can generate query statistics.

The SQL Server query optimizer validates queries and parses them into a tree representation. It continues looking for faster queries until it finds one that is good enough or times out. The optimizer will occasionally make a poor choice. The dynamic management view of the query optimizer is sys.dm_exec_query_optimizer_info, which helps the database administrators understand the workloads of their databases. They can also use this dynamic management view to view statistics by phase, statement types, and cursor information.

Query plans are essential for improving query performance. However, it is essential to understand their limitations. For example, they do not provide all of the necessary facts. Furthermore, their cost estimates are not always accurate, although they can be measured in fractions of a second. Also, the estimated query cost can change depending on storage speed and whether the data is already cached. Users should never fully trust the estimated cost of a query plan for this reason, so they should verify it with the SET STATISTICS IO and SET STATISTICS TIME commands.

Query operators provide an excellent opportunity for developing a query plan since they describe how SQL Server executes the query. Operators like Sort, Hash, Spool, and Filter incur a high-performance cost because they require SQL Server to retrieve all of the data before generating the output for the query, thus reducing concurrency and causing blocking. Also, improve performance by comparing seeks with scans, the estimated row count with the actual row count, and the required memory with the desired memory. Additional areas for improvement include key and RID lookups, and missing indexes. Use caution when generating statistics with trace flag 2371, and conditions like @param = is null and column = @param.

Search arguments (SARGs), also known as predicates, can provide good performance by using indexes. Non-SARG expressions hurt performance because they force scans. Other factors that can indicate poor query performance include warnings and predicate pushes that involve probes, residuals, and filters. Data type conversion can cause severe bottlenecks, which occurs when a query joins columns that do not have matching data types. Trusted constraints can help eliminate these joins by using a UNIQUE constraint to ignore a DISTINCT clause.

Microsoft has invested much effort in making query plans more adaptive with tools like intelligent query processing, which can help with memory grant problems. Intelligent query processing includes many features that can improve the performance of existing workloads with minimal effort. Make workloads eligible for intelligent query processing by using Transact-SQL to enable the appropriate database compatibility level for the database. For example, do this with the following command:

ALTER DATABASE [WideWorldImportersDW] SET COMPATIBILITY_LEVEL = 150.

SQL Server Wait Types

The wait types that the SYS.DM_OS_WAIT_STATS DMV typically shows include ASYNC_IO_COMPLETION, WRITELOG, and PAGEIOLATCH_xx.

ASYNC_IO_COMPLETION shows the inputs and outputs waiting to complete. It is often associated with non-read/write operations like file growth rather than data pages.

WRITELOG shows the log flush operations waiting to complete. This wait type is most often caused by disk latency, although high activity may also be a contributing factor.

PAGEIOLATCH_xx shows the data pages that are waiting for a buffer. Disk latency usually causes that. However, many input and output queries and a lack of buffer space can also cause this wait type.

SUMMARY

The most effective methods of tuning a SQL Server database depend to some degree on whether it is located on-premises or an Azure platform. Disk latency is often a significant cause of bottlenecked performance on an on-premises database. In contrast, network latency is typically more critical for databases on cloud platforms. Other sources of bottlenecking include hardware, operating system, and the database, which are less dependent on the physical location of the server. Dynamic management views are essential tools for obtaining the information needed to resolve these issues through query planning and other techniques.

IDERA'S SOLUTION

SQL Diagnostic Manager

Monitor, alert, diagnose, and report on SQL Server availability and performance.

SQL Diagnostic Manager for SQL Server helps database administrators to find and fix SQL Server performance problems in physical, virtual, and cloud environments. Unlike its competition, it provides effective scalability, advanced SQL query analysis and optimization, prescriptive analysis with corrective SQL scripts, powerful automated alert responses, broad PowerShell integration, complete customization, and extensive support for current and legacy SQL Server and Windows.

Start a free, fully-functional, 14-day trial today!

Start for FREE