solarwinds

Application Playbook:
SQL Server Monitoring for IT Pros

APPLICATION PLAYBOOK: SQL SERVER MONITORING FOR IT PROS

One of the main missions of the IT pro is to keep applications running successfully. Successful application performance from an end-user's viewpoint is different for different applications. Microsoft® Exchange™ is considered successful by users as long as email can be sent and received. Active Directory® is seen as running efficiently, as long as users can access the file shares they need. SQL Server®, on the other hand, functions like a collection of those and other services. Depending on your environment and how SQL Server is used, it's possible that SQL Server is responsible for moving, storing, querying, presenting, emailing, and controlling access to data. If any one piece of the puzzle is experiencing a performance issue, you will begin to lose the confidence of your user base. The key, as with other applications, is to make sure you stay out of the habit of putting out fires, and stay in the business of preventing fires.

The best way to do that is to implement a monitoring solution. But your work isn't done just because you are monitoring. Even the best monitoring solutions need to be configured with the alerting levels that match your organization and environment. More importantly, you need to understand the meaning of the metrics your are monitoring. Just because you aren't receiving alerts from the application doesn't mean that the metrics aren't showing problems with your SQL Server. You need to have a clear understanding of what these warnings and alerts mean. Just like other applications, the key resource groups to monitor, as part of SQL Server, are storage, CPU, and memory. Following that, you have query-level metrics that help identify the reason for database slowness. In this eBook, we will discuss the key metrics of these areas to be able to get a good grasp on SQL Server performance.

TABLE OF CONTENTS

MONITORING STORAGE METRICS

When we think about monitoring storage, it's easy to focus on available space. Obviously, SQL Server stops functioning if you run out of space, but storage issues can also be monitored for latency. Most monitoring applications display metrics that show the number of reads and writes on a database, along with the latency of read and write requests. These metrics can be pulled from most monitoring solutions, but are also available by reviewing the **Dynamic Management Views (DMV)** within SQL Server.

For storage issues, one of the more valuable DMVs is **sys.dm_io_virtual_file_stats**. This DMV keeps a running tally of the occurrences of waits and duration of waits, along with the number of read and write transactions. It allows you to see the total number, duration, and times of occurrences. By looking at those metrics, if you have a high number of waits, it can mean that you don't have enough drives to handle all of the IO requests, among other things. The key is to take that data set and create an average to understand what you are looking at.

Keep in mind that if one transaction does a large number of reads against one file, these average calculations will be skewed, so these numbers should not be the only things you monitor. If your waits are long in duration, this can highlight that your storage is too slow to handle SQL Server. There are countless reasons to explain why your storage would be slow. Suffice it to say, if you are showing long waits, it is recommended that you look at the medium you are using for server storage to make sure that the hardware is working properly. If it isn't, you may need to buy better drives, or talk to your SAN administrator about moving the storage to different LUNs. That said, if only one database is showing higher waits, this can point to issues with queries running on the database.

You should also look at counters in Windows® Performance Monitor.

- **I/O Database Writes/Sec and I/O Database Reads/Sec** lets you see the amount of calls to storage for reading and writing data.
- **Disk Queue Length** also signals that your storage medium can't keep up with your workload. When you have a consistently high disk queue length, it usually means you have transactions that are waiting for their turn to execute on the disk.

- **Page File Usage** isn't a sign of IO layer issues, but it can cause those issues. If a server has memory pressure and is paging memory out to disk, it can create higher than normal hard disk IO, which will take resources away from normal processes.

MONITORING CPU METRICS

As with monitoring storage, another key metric is CPU wait time. Wait stats show how long, and how many tasks, have had to wait, and categorizes them in useful and granular categories. One of the most common causes of CPU pressure occurs when SQL Server is installed on a machine and the default settings are not changed. The specific default settings that affect CPU the most are **Cost Threshold of Parallelism** and **Maximum Degree of Parallelism (MAXDOP).**

These metrics give you a high-level understanding of how a query is handled by SQL Server when you execute. Once a query is submitted, it is received by the SQL Server engine, which scores each step of the query with a cost. Take, for example, the join, where clauses are scored on how much of a resource cost they are.

If the cost is more than the set Cost Threshold of Parallelism, the engine will look to split out the work among multiple threads. The number of threads it will use depends on the cost of the work of each additional thread spun up for the work. The MAXDOP setting tells SQL Server the maximum number of threads it can spin up. It will keep spinning up threads until it hits the MAXDOP, or until the cost of the threads is below the Cost Threshold of Parallelism setting.

This is where default settings create issues. The default cost setting is only 5. This low cost makes it very difficult for SQL Server to be picky about which queries are parallelized.

There are a lot of different recommendations out there for how to configure this advanced option. You can refer to Jonathan Kehayias's article **here** for a recommendation on how to tune the setting. This uses the existing query workload to determine what the setting should be. For the MAXDOP setting, this is set to 0 by default. That setting in itself can sound slightly concerning. You would think that a setting of 0 would mean no threads, but it actually means the exact

opposite. This setting means there is no limit. It allows SQL Server to create as many threads of work as it determines are necessary. This can create problems if the cost is too low and MAXDOP is set to 0, because then SQL Server can spin up more threads than it should. When this happens, threads can get stuck in a situation of waiting for threads to complete. When the threads wait, this increments the wait statistic **CXPACKET.**

It's important to remember that query wait statistics are cumulative since the last time the SQL Server Service was started. You need to look at the average length of the wait per category to understand if you have, on average, long waits, which would indicate a problem, or short waits, which may not be that big of a deal now. If you look at wait stats and they show short CXPACKET waits, you should start monitoring that statistic more often, to see if it increases. If it increases over the next couple of weeks, this means you need to look at either tuning the Cost Threshold for Parallelism configuration, or backing down the MAXDOP setting.

MONITORING MEMORY (RAM) METRICS

Another set of metrics—compilations and recompilations of query plans—can be a sign of memory pressure.

- **Compilations** are defined as the actions that SQL Server takes when it receives a query for the first time. SQL Server creates a set of instructions, known as the query plan, for its worker threads to process the query.
- **Recompilations** are kicked off by SQL Server for queries that have an expired plan in the query cache.

***Note:** When these metrics are high, it does not always mean that you are experiencing CPU pressure, but that your CPU is getting a higher than normal number of requests.*

When SQL Server complies and creates the query plan, it is stored in the query cache, which lives in memory. If there is a shortage of memory, SQL Server will keep fewer query plans in memory than it should. That means that when there are a large number of compilations it can mean that there is a shortage of memory, and plans are being compiled because they aren't living in memory for very long.

However, this metric can also show high when users are running queries that don't exist in the query cache because those queries have never been run before. This metric needs to be used as a possible red flag, but you should also keep in mind that this can alert you to problems that aren't really there.

The same applies to CPU and storage. Wait stats are helpful metrics used to identify memory pressure as well.

PAGEIOLATCH

One wait stat that can show memory pressure is **PAGEIOLATCH**, which is the tag for the time that SQL Server waits for a page of data to be read in to memory. This is caused by storage issues or slow storage. That's the easiest thing to point at. There are other issues that can cause PAGEIOLATCH. A shortage of

memory can create this wait. Poorly written queries that read more data from disk than they should have to can also create this problem, but we are going to focus on memory. PAGEIOLATCH sometimes highlights a memory problem when SQL Server is waiting to load a page of data from disk to memory.

Page Life Expectancy

Another metric that will show when there is memory pressure is **Page Life Expectancy (PLE).** Page Life Expectancy is the measure of time that a page of data will stay in the buffer pool without references. The importance of this metric goes back to how memory is managed within SQL Server. When a query is submitted to SQL Server, the pages of data that the query is referencing are loaded into memory for easy access. The data is filtered according to the WHERE statement, and then data is returned. The data will continue to sit in memory, where it can be referenced again, until it needs to be flushed out to make room for other data that needs to be loaded into memory for another query. If pages are getting flushed from memory fairly regularly, that is usually a red flag for one of two issues.

- One issue could be that the query is written in a way that it is pulling too much data from disk. In this case, either the query needs to be reworked, or there are opportunities for index changes.
- The other possible issue, and the one that we are concerned with for this article, is that there isn't enough memory to store the data that needs to be processed by SQL Server.

Page Life Expectancy is a standard metric that most SQL Server monitoring software has on their dashboard. It can also be pulled using Performance Monitor on the SQL Server.

Buffer Cache Hit Ratio

Related to Page Life Expectancy, **Buffer Cache Hit Ratio** is also a good metric that can point at memory pressure. The Buffer Cache Hit Ratio shows the percentage of pages found in the SQL Server buffer pool of all of the data pages that were requested by queries. This metric can be pulled using Performance Monitor, just like Page Life Expectancy.

When looking at Buffer Cache Hit Ratio, you cannot look at that metric alone to determine if there are problems with your SQL Server. Buffer Cache Hit Ratio can have small drops due to users requesting data that has already been flushed from memory. The real value of it is to pair it with Page Life Expectancy. When you look at Buffer Cache Hit ratio and it shows a drop that PLE doesn't, there is no problem. A sign of memory pressure is when they both drop. In this case, the thinking is that if pages are not living long, and there are fewer pages in memory when requested, there isn't enough memory to hold all of the data required by SQL Server.

Paging

When you are experiencing memory pressure on your SQL Server, **memory paging** will show evidence of it. When a computer is out of available memory, but still has data that it has to put into memory, it will put this data on disk. This is called paging. So why is memory paging a problem? Because memory is faster for accessing data than disk, so, as you can imagine, when memory is paged to disk there is an enormous performance slowdown that you want to minimize.

MONITORING QUERY-LEVEL METRICS

Queries can cause a lot of performance problems on SQL server, especially if they are badly written. There may be redundant query lines, complex or looping syntaxes, query deadlocks, lack of proper indexing, improper partitioning of database tables, etc. Unlike hardware upgrades, query tuning can offer immediate and powerful results. However, effective tuning requires not only knowing the top SQL statements, but also the top wait types, SQL plans, the effect of missing indexes, blocked queries, and resource contention. So, it is imperative for DBAs, developers and system administrators to be able to look in the code and identify which query is causing more wait time and when. **SolarWinds Database Performance Analyzer** is a handy tool that gives you quick visibility into code-level metrics of the database to isolate queries with high wait times, look up the query codes, and identify problems with built-in performance tuning tips.

We can avoid some of these problems by writing good queries. Let's take a look at some best practices for query writing and reading an execution plan.

Writing Queries

A good rule of thumb for writing queries is to keep them tight and focused. This means that they should be written to only reference the data that they need.

- "SELECT *" queries end up putting all of the columns into memory in a table. This takes up additional space in memory that isn't necessarily needed.
- Also, joins should be limited to only the tables that need to be joined for the result set. Joins require additional data to be put into memory. If that data isn't needed for the query, it should be left off.

The key to writing a good query is only using the processing and memory you need, nothing more. Also, keep in mind that when writing a query, subqueries are fine to use as a join predicate, but not as columns. If you use a query for a column value, it will be run for each row of data. The same outcome can be achieved by creating a join, and the query optimizer will handle the query in a more efficient manner.

Query Execution Plan

If you are still having trouble getting a query to perform efficiently, look at the query's execution plan. You can look at the estimated or actual plan by clicking the buttons for estimated or actual in **SQL Server Management Studio.** The query plan shows you, in a graphical interface, how the query optimizer is processing the data for the query. It can detail for you how much data is read for each step, including how much is read from disk. It will also detail the exact steps taken.

For instance, when you see Table Scan steps, those are bad. These steps mean that the query optimizer has told SQL Server to go through each individual row in a table to find the values that it was looking for, because of a WHERE clause or a JOIN. As you can imagine, going through each individual row is a long, costly process. These steps tend to point out that the table that it is scanning is missing an index. The execution plan will also show when you are reading a lot of rows directly from disk. This can be a sign that you need of some indexing help, as well. There are well-documented DMVs within SQL Server that can show you exactly how large an index is and how often it is used.

HOW SOLARWINDS CAN HELP

SolarWinds® Server & Application Monitor (SAM) provides deep monitoring of SQLServer for application-centric troubleshooting. SAM also provides out-of-the-box monitoring for **over 200 applications,** including Exchange™, AD, SharePoint, Windows, Linux®, VMware®, and more.

SQL Server Monitoring with SAM

Leverage the built-in AppInsight™ for SQL dashboard to proactively monitor database connections, sessions, error logs, event logs, SQL agent job status, etc.

Track database metrics, such as database growth, disk I/O, file I/O, transactions, index fragmentation, log flushes, and more.

Isolate the top expensive queries by CPU time, and access the query code in a single click.

Monitor database storage metrics, such as database size, file size, shrink space, average read and write latency, and number of reads and writes/sec.

Get instant access to cache metrics, buffer manager performance counters, and latches and locks.

Identify performance problems by tracking page life expectancy, compilations and recompilations, disk queue length and memory statistics, and more.

Monitor other databases including Oracle®, MySQL®, MongoDB, IBM® DB2®, and PostgreSQL®.

Visualize contextual dependency and relationship of SQL Server with other applications, databases, and underlying physical server and virtual infrastructure.

Monitor server performance, response time, hardware health, and resource capacity metrics.

> *"We used the AppInsight for SQL in SAM to detect SQL connection's spike due to bad codes not closing their connections after process completion."*
>
> *IT Manager, Medium Enterprise Insurance Company*

> *"Database latency due to delayed rights and poor SQL statements identified by SAM allowed us to quickly identify and resolve the problem."*
>
> **Roger Blakely**
> *IT Director, Hulu, LLC*

DOWNLOAD FREE TRIAL | LEARN MORE | TRY ONLINE DEMO

Fully Functional for 30 Days

SolarWinds Database Performance Analyzer is a specialized database performance optimization tool for DBA and developers that uses wait-time analysis to solve even the most difficult performance issues across various database types. Leverage built-in tuning advisors to troubleshoot problems and improve database performance.

The integration of Server & Application Monitor and Database Performance Analyzer delivers contextual visibility of database infrastructure statistics and query-level response time analysis side by side, in a single web-interface.

LEARN MORE