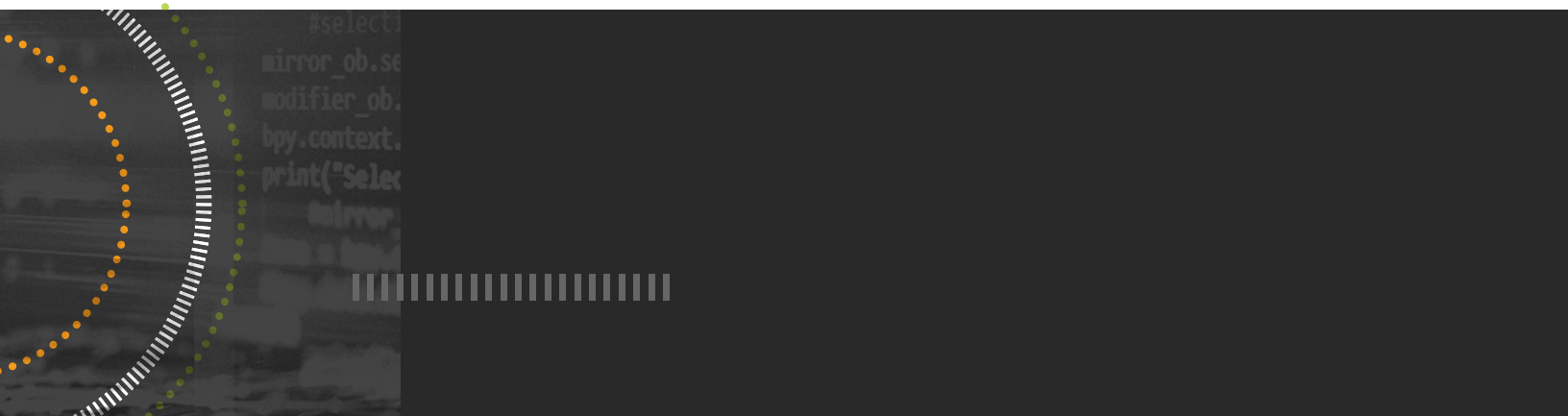
solarwinds

WHITEPAPER

Optimizing SQL Server Analysis Services (SSAS)

by Steven Wright



Optimizing SQL Server Analysis Services (SSAS)

Troubleshooting performance problems with SQL Server Analysis Services (SSAS) can be a frustrating exercise. This insightful guide by Steven Wright, database sales engineering manager at SolarWinds, helps uncover the myriad forces that can conspire to degrade SSAS performance. In this five-part guide, Wright explains MOLAP SSAS architecture, examines MDX query performance, and offers time-saving tips for tracking trace events and troubleshooting SSAS bottlenecks. While the information in this whitepaper focuses on SSAS in Multidimensional mode (MOLAP), future editions will expand on differences found in SSAS Tabular mode.

Part 1: Understanding SSAS Basic MOLAP Architecture

To better understand SSAS performance optimization and troubleshooting, it's first important to understand the basic architecture of multidimensional SSAS, all the moving parts, and how they interact. Part 1 covers these topics along with an introduction to the types of activity that occur in SSAS and an introduction to the deeper topics covered in the rest of the series.

Part 2: Identifying Impacts on MDX Query Performance

We start by discussing what can impact MDX query performance. This installment focuses on processing activity, how to identify if processing is impacting your MDX performance, and some suggestions for how you might minimize the impact with a more efficient processing strategy.

Part 3: Improving MDX Query Performance

Part 3 goes more in depth specifically around identifying potential issues where the MDX code itself may not be optimized. Even without knowledge in writing MDX queries yourself, SolarWinds® SQL Sentry provides visibility to the user if this is the problem and can help you make suggestions to the developers to get them started making improvements to MDX query performance.

Part 4: Tracking Trace Events

Beyond strategies for the queries themselves, it also helps to understand where SSAS is spending most of its time working to provide the results to your request. This section covers several useful trace events you can track or view within SQL Sentry to identify how your requests are being handled by SSAS, so you know where to focus your performance improvement efforts.

Part 5: Troubleshooting Bottlenecks

The final section covers bottlenecks with your server's physical resources: how to identify whether there are resource bottlenecks, where they lie, and what you can do about them to keep SSAS running at peak efficiency.

Part 1

BASIC ARCHITECTURE OF SQL SERVER ANALYSIS SERVICES

I've heard many people say they see SSAS as a "black box" on their servers with no real visibility into troubleshooting performance issues. SQL Sentry lifts the shroud from SSAS, but for some it may not be obvious how to interpret all the information gained from this new visibility.

Many resources cover OLAP architecture and MDX, but much less information is available to explain the various metrics available and how they correlate with each other. In response, I developed a presentation to "bridge the gap" for DBAs who may be new to Multidimensional SSAS or developers who lack a dedicated administrator and need to be able to tell where the bottleneck might be on their SSAS server. This whitepaper is a compilation of presentations and the series of blog posts I've written about how to troubleshoot SSAS performance. Anyone working with SSAS should find this whitepaper useful, but users of SQL Sentry should find themselves even more proficient in troubleshooting SSAS performance issues. We'll start with a very basic introduction to the service itself and how a query is handled and move into specific metrics and identifying bottlenecks.

With a few basic pieces of information, you should be able to tell whether there's room for MDX query optimization or whether the bottleneck lies elsewhere.

Basic Architecture

SSAS is its own engine. The executable is msmdsrv.exe. It's licensed with SQL Server, but in large production environments, you aren't likely to be running it on the same machine as a SQL Server. Two primary types of activity occur here: querying and processing.

Types of Activity

Querying can be done by various methods, but the most common type of queries is Multidimensional Expressions (MDX). This is an analytical query language optimized for calculations and aggregations of data. I won't go into the how's and why's of MDX itself as there are many resources already available on its mastery. While this series won't tell you how to write MDX, by the time we're done, you should be able to tell, with a few basic pieces of information described later, whether there's room for query optimization or if the bottleneck lies elsewhere.

Processing is essentially the maintenance activity of your cubes. It's how cubes are created and updated with new data and/or how existing data is recalculated and reorganized. Just as with SQL Server, what level of processing you perform and when is essential to ensuring proper performance without interfering with peak querying times (and we'll cover those later).

The Formula Engine

Under the covers, there are two primary "engines" within SSAS: the Formula Engine (FE) and the Storage Engine (SE). The FE accepts MDX requests and parses the query to process. It will send requests to the SE when data is needed. It then performs calculations on the retrieved data to provide a final result set. The FE is single threaded. If you're watching the CPUs of your multi-core server with SSAS while a request is being handled and you only see one core spike, it's probably because the FE was doing the work at that time.

The Multidimensional Storage Engine

The SE, on the other hand, is multi-threaded. The primary architectural difference between Multidimensional and Tabular modes of SSAS is in the Storage Engine. Note in this whitepaper we focus on Multidimensional mode. In MOLAP SSAS, the SE reads and writes data to and from the file system for the server. This can include working against the Windows file cache. This is an important distinction between SSAS and the relational engine in SQL Server as well. The SQL Server relational engine has an internal cache, and then goes to disk for any other needed IO. It doesn't use the Windows file cache for database files. SSAS, on the other hand, does. This means when SSAS is interacting with the file system on your server, the process won't necessarily result in physical disk IO. SSAS uses its own caches—as well as the file system cache—to get what it needs.

If you're watching the CPUs of your multicore server with SSAS while a request is being handled and you only see one core spike, it's probably because the Formula Engine was doing the work at that time.

SSAS Caches

So how does SSAS cache data? Each engine within SSAS has its own set of caches. The FE caches store flat values (the Flat Cache) as well as calculated data (the Calculation Cache). It's important to understand these caches are often scoped, which means the data might be available only for a limited time or only for a particular user, even in cache. The flat cache in particular is, in fact, restricted to a max of 10% of the TotalMemoryLimit property in SSAS, which we will discuss later.

When we talk about SSAS cache optimization, warming, etc., we're more often dealing with the SE caches. That's a bit of an oversimplification, but this is an introduction, after all. The SE consists of a Dimension Cache and a Measure Group Cache to store the data retrieved from the corresponding sources in the file system itself.

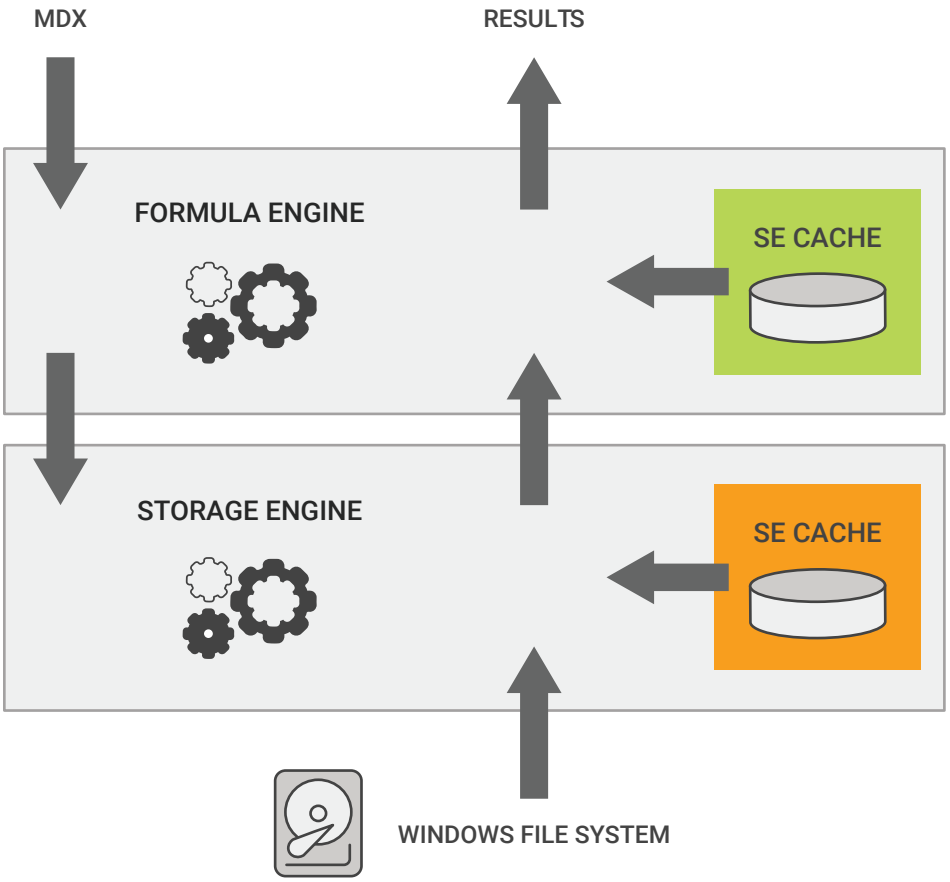
Impacts on MDX Performance

So now that we understand the basic components under the covers in SSAS, how are they used to handle a request? First, the query is accepted and the FE parses it. If the appropriate data is already available in the proper scope in the FE cache, some or all may come from there.

The FE then requests any other needed data from the SE. The SE then retrieves whatever it can from its caches. If still more data is needed, the SE queries the file system on the server.

The data from the file system may, at least in part, come from the Windows file cache. Anything else results in physical disk IO. These steps may be iterated as needed while data is moved from SE caches to FE caches, etc.

So, now that we have a good understanding of how SSAS handles MDX requests, the question we need to answer is, "What can impact MDX performance and how can we tell?"



Anatomy of an MDX Query

Part 2

IDENTIFYING IMPACTS ON MDX QUERY PERFORMANCE

In the last section, I mentioned SQL Sentry "lifts the veil" from the black box known as SSAS. But there is little information available to help someone new to the technology easily interpret all this new information to optimize the performance of their servers. In this section, I want to dig deeper into the common sources of bottlenecks and what information is available to troubleshoot these issues.

In the section, I'll break into six categories the factors that can impact MDX performance on your SSAS server:

- Processing
- Unoptimized code
- CPU
- Memory
- Disk
- Network

I'll tackle each one by giving an explanation and refer to specific metrics, primarily performance counters, you can use to analyze the given issue.

Processing

As I mentioned before, processing involves the creation and updating of cubes with new data and/or recalculating and reorganizing existing data. I related this activity to maintenance on SQL Server and many of the same factors you consider when you implement maintenance in SQL Server apply to processing in SSAS.

Processing primarily involves the Storage Engine (SE), which you'll remember is multi-threaded, and can be a resource-intensive operation. Because processing might involve new data or recalculation of existing data, cached data will become outdated and will be flushed.

This then means the next time a query runs that requires that data, the SE will have to query the file system to retrieve needed data from the Windows cache or the disk system to replace the flushed data. This will result in a performance hit to the query on its first run after processing.

Querying while processing is occurring on the same cube can lead to blocks. Querying any cube while processing is occurring can lead to general resource contention on the server.

So, how can we tell when processing is occurring on the server? While many performance metrics have varying levels of usefulness, I'll mention a handful for each topic to give you better insight. Don't think if I don't mention it, it's not useful, or vice

Querying any cube
while processing is occurring can lead to general resource contention on the server.

versa. The point of this whitepaper is to give you a better understanding of what's going on in SSAS and how to make better use of the information tools such as SQL Sentry provide.

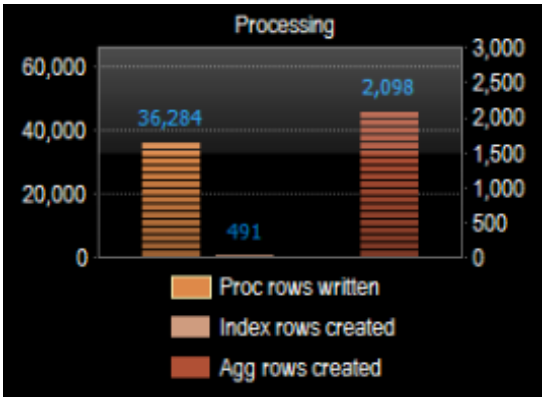
- **Processing:** Rows written/sec tells you the rate of rows written during processing
- **Proc Aggregations:** Rows created/sec more specifically tells you the rate aggregation rows are created during processing.
- **Proc Indexes:** Rows/sec tells the rate of rows from MOLAP files used to create indexes.

So, a non-zero value for these counters confirms processing is taking place. These metrics can also help determine the effectiveness of processing tuning efforts. If your tuning efforts are effective, you should see an increase in these values accordingly.

Improving Processing-Related Performance

How do you improve performance related to processing? I mentioned earlier that you can relate SSAS processing with maintenance in SQL Server. Many of the same techniques applied in SQL Server maintenance apply here as well.

- **Scheduling:** If possible, try to schedule processing activities during time of least querying activity on the server to avoid blocks and resource contention.
- **Processing Type:** Just as you would with SQL Server, use the most appropriate processing commands with SSAS. Just as there are different types of backup strategies in SQL Server, such as Full, Log, Differential, etc., the same principle applies in SSAS processing. As an example, use ProcessData and ProcessIndex, where appropriate, instead of ProcessFull.
- **Partitioning:** If you're working with SQL Server Enterprise Edition, take advantage of the ability to break Measure Group data into partitions. Not every partition may need to be processed every time. Partitioning can serve to reduce the overall amount of work. Remember the SE is multi-threaded, so architect your data to take advantage of parallel processing capabilities.



Monitoring Processing

Part 3

IMPROVING MDX QUERY PERFORMANCE

In the last section, I showed you some counters that can help identify when SSAS processing is taking place and at what rate. I then gave some suggestions as to how you might be able to optimize your processing strategies to improve overall performance on your server. In this section, I'll talk about unoptimized MDX code as the source of your performance issues.

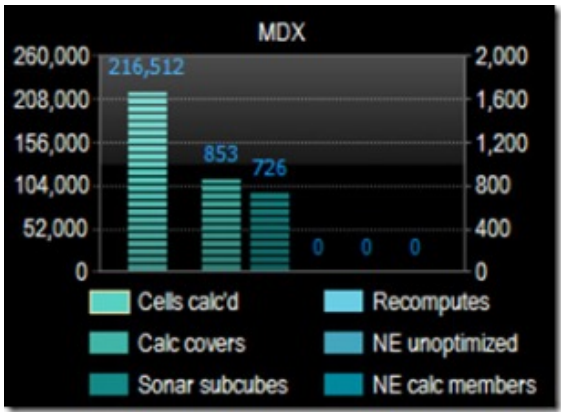
UNOPTIMIZED CODE

There are several resources available on how to write MDX, so I won't be going into that here. The point I want to make is there are ways you can identify queries running on your SSAS instance that might benefit from optimization, even without your actually knowing MDX. In this section, I'll mention a handful of key performance counters that will allow you to identify issues related to query optimization. You can then take this knowledge to the developers of the queries with useful feedback as to how to improve performance.

Specific to MDX, a few performance counters can provide a lot of insight. I'll group them into three general issues. The first group are:

- MDX: Total cells calculated
- MDX: Number of calculation covers
- MDX: Total Sonar subcubes

A high number for any of these metrics while a query is executing suggests the query is using cell-by-cell calculations instead of a block-oriented, also known as "subspace," evaluation. So, what does this mean? To stick with our SQL example, you can relate cell-by-cell calculations to using cursors in SQL. This is often an inefficient and resource-intensive way of processing data. You've likely heard of the term "RBAR" in SQL development, which stands for "Row by Agonizing Row." Well, you could call this CBAC, or "Cell by Agonizing Cell."



Monitoring MDX

Another counter that can indicate serious performance issues with MDX:

- MDX: Total recomputes

This indicates errors in calculations during query execution. A non-zero value here can indicate issues where unnecessary recalculations are taking place and can even lead to a fallback to cell-by-cell mode in an effort to eliminate the error.

One more set of counters I think gives us a good look into MDX performance:

- MDX: Total NON EMPTY unoptimized
- MDX: Total NON EMPTY for calculated members

In a nutshell, SSAS data tends to be sparsely populated data. Consider data on sales orders, for example. A typical sales order is not likely to include at least one of every item available for purchase. There are likely to be many empty cells for items not purchased for any given order.

Previously, I mentioned the performance impact of cell-by-cell calculations. When you have sparse data, specifying a NON_EMPTY algorithm can also speed up performance, but there's more than one code path for this algorithm. A slower code path can cause performance degradation. This behavior was improved in SSAS 2008 but can still occur. The above counters can help you identify when that's the case. More details can be found in the NON_EMPTY_BEHAVIOR section of the Analysis Services MOLAP Performance Guide for SQL Server 2012 and 2014.

Improving MDX Performance

So, to wrap up this section, what are some primary strategies for optimizing MDX performance? First, proper aggregations and partitions can make a big difference, especially if the bottleneck is in the storage engine. To reuse our analogy, aggregation strategy is similar to SQL index strategy. There's a sweet spot. Too many aggregations can be just as bad as too few. Aggregations provide pre-calculated data that can improve query performance. However, aggregations must be maintained and too many will increase the resource requirements during processing as well as storage requirements for all this extra data.

If the bottleneck appears to be in the formula engine, watch for cell-by-cell calculation, and eliminate empty cells and tuples using the recommended strategies in the linked whitepapers mentioned above.

Aside from the counters just mentioned, how do you know where the main bottleneck might be—in the Formula Engine or Storage Engine? Where should you focus your performance troubleshooting? I'll help you further pinpoint those efforts in the next section.

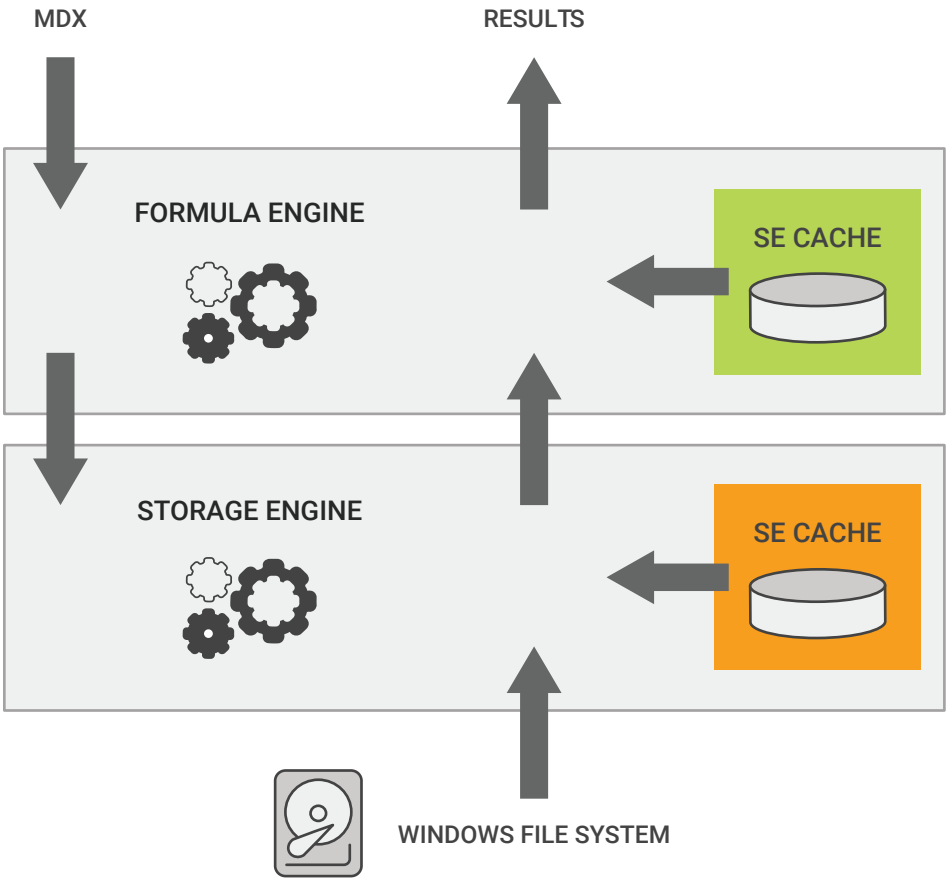
Proper aggregations can make a big difference in optimizing MDX performance, especially if the bottleneck is in the storage engine.

Part 4

TRACKING TRACE EVENTS

In the last section, I talked about how to detect performance issues related to unoptimized MDX and different strategies based on the type of issue. A big part of ensuring you're as efficient as possible with your troubleshooting efforts is to first identify whether the bottleneck is in the Formula Engine (FE) or Storage Engine (SE). Technically, unoptimized MDX itself is likely to manifest as an FE bottleneck, where an SE bottleneck really resides more with the underlying cube architecture. That said, all this really ties back to the first section and the anatomy of an MDX query.

In part 1, I walked through the anatomy of an MDX query and described what role the FE and SE play in the handling of a request. When troubleshooting slow MDX performance, it's not likely for the bottleneck to be entirely with the FE or the SE. So, how do you tell where the biggest bottleneck is, so you can ensure you get the most out of your tuning efforts?



Anatomy of an MDX Query

Where Is SSAS Spending Time?

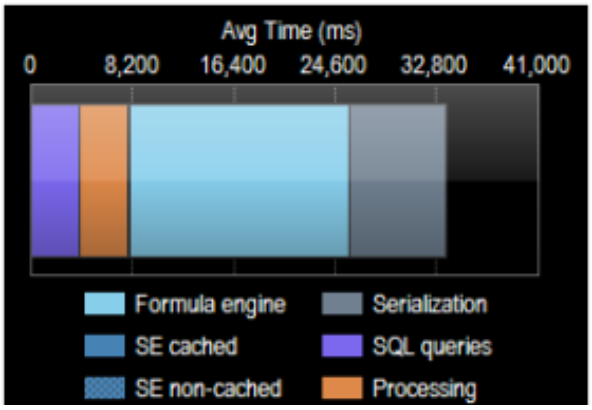
To put it simply, if SSAS is spending most of its time in the FE while handling a request, that's where you should start your troubleshooting, likewise for the SE. But how can you accurately measure something like this? The most accurate way if you're using native tools is a trace. Certain trace events in SSAS map closely to the activity we covered in the "anatomy of MDX" discussion.

SSAS Trace Events

- **Command Begin/End:** This signifies the beginning/end of an XMLA command. This is often associated with processing activity.
- **Query Begin/End:** This encapsulates MDX, or other queries like DMX, SQL, etc., from request submission to results returned.
- **Query Subcube/Verbose:** The FE requests data from the SE .
- **Get Data From Cache:** There was a cache hit to either the SE or FE caches. The event subclass will tell you which.
- **Progress Report Begin/End:** This indicates SE file system activity either for processing or querying. The event subclass will tell you which.
- **Get Data From Aggregation:** This indicates the SE made use of an aggregation to return data from the file system.

There are other events and additional detail provided within the trace, but this list covers the primary events we've discussed. Note there can be many of these events—excluding the command or query begin and end—for a single request. Tracking these events can get a bit complicated fairly quickly, especially if there are multiple requests hitting the server at the same time. However, if you're troubleshooting a particular query and run a trace, you have the information to add up the duration for each event and come up with an idea of where most of the time is being spent.

Fortunately, SQL Sentry does all of this for you. You might have noticed a rather unique chart on the SQL Sentry dashboard. This chart monitors all this activity for you continuously at a fraction of the overhead a Profiler trace tends to impart. It summarizes this information in real time and retains history, so you can quickly and easily see where most of the time is being spent. It even breaks the information down into more detail for you. Serialization is handled by the FE and is related to NON_EMPTY behavior. Processing and SQL queries are handled by the SE. By the way, SQL queries are likely due to data reads for processing but can also be seen if you're using ROLAP partitions.



SQL Sentry Dashboard Chart Detail

For a further breakdown of this activity on a query-by-query basis in SQL Sentry, refer to the Top Queries tab. Not only will you see whether your query experienced more FE or SE time, but you can expand the view for a complete breakdown by Measure Group, Partition, Aggregation, and Dimension. More information on this functionality is available here.

Text Data	Duration	CPU	CPU %	FE Time	SE Time	FE Time %	SE Time %	SE Cache Ops	SE Non-cache Ops
SELECT NON EMPTY {[Date].[Day of Week].me...	2,957	2,438	2%		2,129		100%	66	37
SELECT NON EMPTY {[Date].[Day of Week].me...	5,037	4,156	3%		3,800		100%	124	61
WITH MEMBER [Measures].[Employee Rank] AS ...	5,271	16		5,271		100%			1
WITH MEMBER [Measures].[Employee Rank] AS ...	7,884	16		7,882	2	100%			1
SELECT { [Measures].[Internet Sales Amount] ,...	4,596	94			1,345		100%	9	6
with member measures.daysto10000 as iif(coll:...	19,947	19,078	13%	16,638	1,723	91%	9%	11,344	36

Top Queries Tab in SQL Sentry

Part 5

TROUBLESHOOTING BOTTLENECKS

Up to this point, I've been focusing on common activity in SSAS, how to identify the bottleneck, and how to resolve the bottleneck. In the last section, I specifically discussed how to identify whether the Storage Engine (SE) or Formula Engine (FE) was the larger bottleneck. Now I want to talk more about identifying bottlenecks with the server's physical resources themselves. The four basic areas of server resources to investigate are:

- CPU
- Memory
- Disk
- Network

A lot of these issues, and methods for identifying them, are common to Windows Server performance in general, but I'll include pertinent SSAS-specific performance details and metrics, too.

CPU

Any Windows server includes a handful of counters to indicate the bottleneck with your server might be related to the CPU(s):

- Processor: % Processor Time
- System: Context Switches/sec
- System: Processor Queue Length

While no magic number indicates what "good" CPU utilization is, you generally want to get the most utilization without overburdening the system, so look for sustained periods of 100% utilization on one or all cores to suggest a bottleneck here. The other two counters, when abnormally high for sustained periods, usually suggest an issue with too many parallel operations on the server.

So, let's look at some SSAS performance metrics that shed some light on processor utilization. Two sets of SSAS performance counters apply to the formula and storage engines:

- Threads: Query pool
- Threads: Processing pool
- Threads: Processing I/O pool

Don't let the names confuse you. The query pool refers to FE activity. The processing pool not only refers to processing, but any SE activity. Prior to SSAS 2012, there

was no Processing I/O pool. It was introduced to further break out processing activity from read activity in support of query requests by the SE. Query-related read activity is now represented by the Processing I/O pool. Each of these categories contains counters for Busy, Idle, Job Rate, and Queue Length. They allow you to see the thread activity for each engine. If you're seeing consistently high queue lengths, but not high CPU utilization, you might want to adjust your MaxThreads and/or CoordinatorExecutionMode properties for your SSAS instance. Remember the FE is single threaded, so increasing the query pool setting might not improve performance of any one query but might improve the performance in handling multiple simultaneous requests.

Memory

Three groups of metrics are relevant in monitoring MOLAP SSAS memory:

- **Overall usage—**how much total memory is SSAS using on the server.
- **Cache hit ratios—**how efficient is the use of that memory.
- **Cache activity—**what is happening to the memory.

Memory monitoring is very different in Tabular mode as the entire Storage Engine operates in–memory. Here we'll review MOLAP memory usage.

SSAS Memory Usage

- Memory: Memory Usage KB

This is the total memory usage for the server process and should be the same as the Process: Private Bytes counter for msmdsrv.exe.

NOTE: Do NOT rely on Task Manager for an accurate picture of memory usage.

- Memory: Cleaner Memory KB
- Memory: Cleaner Memory shrinkable KB
- Memory: Cleaner Memory nonshrinkable KB

These counters refer to the background cleaner for SSAS. The first counter refers to the amount of memory known to the background cleaner. This memory is then divided into shrinkable and nonshrinkable memory. This describes what portion of the known memory is subject to purging by the cleaner based on memory limits. The cleaner value is likely to be a bit lower than the total usage value, but it's important to know because this value lets you know how much room you have to work with when it comes to memory management. The limits the cleaner works with are defined by properties indicated by the following two counters:

- Memory: Memory Limit Low KB
- Memory: Memory Limit High KB

A great explanation of these properties and counters, along with real-world examples of their use is covered in Greg Gonzalez's blog post "Analysis Services Memory Limits."

SSAS Cache Hit Ratios

Remember the FE and SE each have caches. There are the Calculation and Flat caches for the FE and Dimension and Measure Group caches for the SE. The counters that allow you to determine cache efficiency are all in the Storage Engine Query category:

- Calculation cache lookups/sec, hits/sec
- Flat cache lookups/sec, hits/sec
- Dimension cache lookups/sec, hits/sec
- Measure group cache lookups/sec, hits/sec

While there's no persistent cache hit ratio counter itself for these caches, as there is for SQL Server, these metrics let you to calculate the ratio for each cache for a given point in time.

SSAS Cache Activity

One last group of counters to consider relates to overall cache activity:

- Cache: Inserts/sec
- Cache: Evictions/sec
- Cache: KB added/sec
- Memory: Cleaner: Memory shrunk KB/sec

These metrics give a fairly direct indicator of memory pressure on the server. If the Evictions/sec and/or Cleaner: Memory shrunk KB/sec are consistently non-zero, you likely have memory pressure on the server. The Cleaner: Memory shrunk counter indicates you're exceeding your defined memory limits described earlier.

Improving Cache Usage

In addition to improperly configured memory limits, as discussed in Greg Gonzalez's blog referenced earlier, another common SSAS memory-related performance issue is a cold cache. In Part 2 of this series, we discussed cube processing and data in cache becoming invalidated and flushed after processing occurs. This means the next time a query is executed, no data will be in cache, resulting in queries to the file system, and a dramatic hit to the query's performance. A common scenario involves nightly processing that leaves the cache cold in the morning. The first person in the office, which oftentimes is someone high up the management chain, runs an important report that takes forever to run. The point here is some of the people you least want to experience performance issues might be the ones most likely to see them under this scenario. So, what do you do?

The answer is cache warming. There are different ways to warm the cache. The simplest is to run a couple of the most commonly used queries after processing to pull the most likely needed data into cache.

Disk

To determine whether the disk system is a bottleneck for your SSAS instance, you need to first verify SSAS is indeed accessing the disk system. I mentioned in the first post that the SE accesses the file system. So, check the previously mentioned counters to verify the SE is active. A more specific counter is:

- MSAS: Storage Engine Query: Queries from file/sec

Remember, unlike the relational engine, SSAS data may be in the Windows file cache. This means even when the above counter is non-zero, it alone doesn't guarantee physical disk IO. You'll want to examine the following three counters to get an idea of how much of the SE activity is actually reading from disk as opposed to the Windows file cache:

- MSAS: Storage Engine Query: Data bytes/sec
- Physical Disk: Disk Read Bytes/sec
- Cache: Copy Reads/sec

Be sure to account for activity outside of SSAS when using these metrics. For a much more detailed explanation with great screen shots of SQL Sentry for SSAS, see Greg Gonzalez's blog post on the subject.

So, once you've determined SSAS is incurring physical disk IO, what should you check to ensure the disks are performing optimally? There are many different counters available for disk performance, and some are more useful than others.

For a long time, disk queue length was considered an important metric, but as server storage grew to include more and more spindles, was moved to SANs, or included SSDs, this metric has become less meaningful.

A more universal indicator for disk performance is latency.

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

Optimally these values should remain below 10 ms. As you approach latency 20 ms to 30 ms or more, you're going to notice performance issues related to the disk system. With SSDs, latency should be even lower. This isn't specific to SSAS, but more of a general server guideline.

Improving Disk Performance

Partitions are essential to optimizing disk performance. Partitions based on the way the data is most likely queried, such as by timeframes, will help reduce the amount of data that must be retrieved from disk for a given query. Proper partitioning can also help you take advantage of parallelism when multiple queries are submitted as well as when processing. Remember to distribute your partitions properly to spread the load across multiple disks.

You might also want to disable Flight Recorder. Without going into the pros and cons of why you do or don't want Flight Recorder, many articles suggest a performance improvement by disabling it. It's basically a file-based trace on your system and will increase IO. It can be disabled in the properties for the SSAS instance.

A couple of other disk performance tactics worth mentioning are not specific to SSAS. First note the location of your cube's files. If they're sharing spindles with system files, or other busy databases, you're likely to run into contention.

The same principle applies to SAN allocation. This is often harder to investigate without the help of your SAN administrator but be sure you're not sharing busy spindles on the SAN, either.

Network

Just as with the relational engine, the network probably offers the least visibility. The problem is just as likely to be outside your server. The network itself can be slow, or the bottleneck might be on the client end. Some metrics can identify whether the problem is a local network issue.

With any Windows Server, look at:

- Network Interface: Bytes Received/sec
- Network Interface: Bytes Sent/sec
- Network Interface: Output Queue Length

This will give you an idea of the traffic on the NIC(s), and let you know if there's a backup in output. For SSAS, we can at least identify what kind of traffic we are sending through the pipe:

- Processing: Rows read/sec—tells the rate of rows read from all RDBMSs
- Storage Engine Query: Rows sent/sec—tell the rate of rows sent from the server to clients

This should give you better visibility into how your network cards are performing and how much of that is related to SSAS activity.

SQL SENTRY: NEXT STEP IN OPTIMIZING SSAS

Having read the guidelines in this whitepaper, you have a much better understanding of how SSAS works under the covers and how to identify SSAS performance bottlenecks. If you're already a SQL Sentry user, you should find this whitepaper has served to jump-start your use of the product to quickly interpret the information being provided and optimize your SSAS performance. If you haven't yet tried the product, what are you waiting for?

Meet the Author

Steven Wright

Steven Wright (@SQL_Steve) is a database sales engineering manager at SolarWinds. He holds several Microsoft and other industry certifications, as well as a Microsoft Professional Program Data Science Certificate.

ABOUT SOLARWINDS

SolarWinds (NYSE:SWI) is a leading provider of powerful and affordable IT management software. Our products give organizations worldwide—regardless of type, size, or complexity—the power to monitor and manage their IT services, infrastructures, and applications; whether on-premises, in the cloud, or via hybrid models. We continuously engage with technology professionals—IT service and operations professionals, DevOps professionals, and managed services providers (MSPs)—to understand the challenges they face in maintaining high-performing and highly available IT infrastructures and applications. The insights we gain from them, in places like our THWACK community, allow us to solve well-understood IT management challenges in the ways technology professionals want them solved. Our focus on the user and commitment to excellence in end-to-end hybrid IT management has established SolarWinds as a worldwide leader in solutions for network and IT service management, application performance, and managed services. Learn more today at www.solarwinds.com.

solarwinds

For additional information, please contact SolarWinds at 866.530.8100 or email sales@solarwinds.com.
To locate an international reseller near you, visit http://www.solarwinds.com/partners/reseller_locator.aspx