Microsoft

Hardware Sizing a Tabular Solution (SQL Server Analysis Services)

SQL Server Technical Article

Authors: John Sirmon, Heidi Steen

Contributors and Technical Reviewers: Karen Aleksanyan, Greg Galloway, Darren Gosbell, Karan Gulati, Chris Kurt, TeoLachev, Greg Low, Akshai Mirchandani, BoyanPenev, Shep Sheppard, Chris Testa-O'Neill

Published: January 2013

Applies to: SQL Server 2012 Analysis Services, Tabular Solutions

Summary: Provides guidance for estimating the hardware requirements needed to support processing and query workloads for an Analysis Services tabular solution.

Copyright

Contents

Introduction

This document provides hardware sizing guidance for in-memory Analysis Services tabular databases so that you can determine the amount of memory and CPU resources required for query and processing workloads in a production environment.

NOTE: This guide is focused exclusively on in-memory tabular solutions. DirectQuery models, which execute queries against a backend relational database, have different resource requirements and are out of scope for this guide.

For in-memory solutions, the best query performance is typically realized on hardware that maximizes the following:

- Amount, bandwidth, and speed of memory
- Fast CPUs
- Onboard cache size

Notice that disk I/O is not a primary factor in sizing hardware for a tabular solution, as the model is optimized for in-memory storage and data access. When evaluating hardware for a tabular solution, your dollars are better spent on the memory subsystem rather than high performance disks.

When sizing hardware for an in-memory database, your focus should be on three operational objectives:

- Estimate the basic storage requirements for an in-memory tabular database (under zero load).
- Estimate memory required for processing, where Analysis Services reads, encodes and loads data into model objects. Processing introduces a temporary, yet significant, load on the memory subsystemfor the duration of the processing operation. You can offset this by designing a processing strategy that includes remote or incremental processing.
- Estimate CPU and memory required for queries issued from client applications, such as Excel pivot reports, Power View reports, or Reporting Services reports. Queries are primarily CPU intensive, but can cause temporary surges in memory usage if a calculation requires data to be decompressed.

To help you evaluate the resources needed to support each objective, we start each section with brief description of how a resource is used, provide estimation techniques you can apply to your own solution, and conclude with key takeaways that summarize important points.

This guide also includes a summary of existing hardware configurations to give you an idea of the range of hardware currently supporting production workloads. Sometimes knowing what works for other people is the most valuable information you can have.

In this guide, server provisioning targets a single system that runs all workloads, using the memory and CPU resources that are local to that system. Scale-out topologies are out of scope for this guide. If you already know that projected database size would require an unreasonable amount of RAM relative to your budget, consider designing a solution that has smaller parts rather than one solution with a very large footprint. Alternatively, consider building a multidimensional solution. Multidimensional models are optimized for disk storage and data access, and can more easily accommodate huge datasets that measure in multiple terabytes.

Hardware Considerations for a Development Environment

Your first interaction with a tabular solution and the hardware it runs on will be with the computer used to develop your project. Before diving into the requirements of a production server, let's take a moment to understand how system resources are used during initial development.

RAM used during processing

Compared to a business class production server, there is usually less RAM available on a development server. Working within the constraints of available RAM means that you'll most likely start your project with a subset of your data that is faster to import and easier to manage.

One of the more compelling reasons for working with a smaller dataset is that it preserves your ability to fully process your Analysis Services solution at any time. **Process Full** is elegant in its simplicity; it clears and reloads all of the objects in your solution, including calculated columns and relationships. It is also the most memory-intensive of all the processing options.

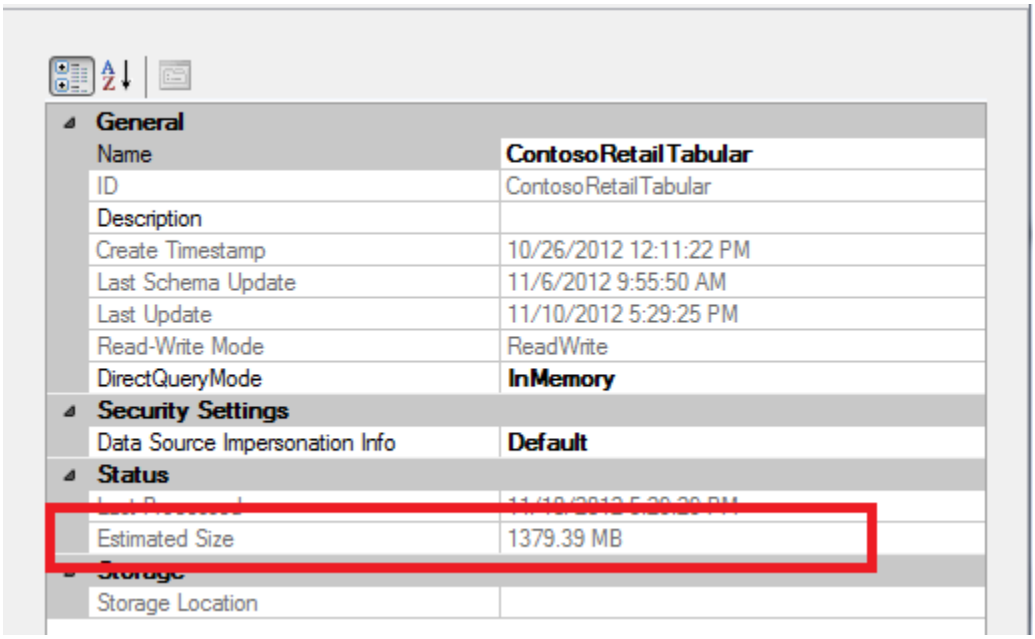
Running a **Process Full** on your solution will require 2 to 3 times more RAM than that required to store the database at steady state. The additional RAM is used to store a shadow copy of the database. After a database has been processed the first time, subsequent processing includes holding a second copy of the database in-memory to service incoming queries while **Process Full** is executing in the background. The shadow copy is released as soon as the final transaction is committed.

Later, when you are closer to deployment, you might want to design a processing strategy that includes incremental processing, remote processing, or batch processing. Using these techniques allows you to prioritize and redistribute the workload, resulting in more efficient resource utilization than what **Process Full** can provide.

Checking memory used for tabular database storage

Tabular databases are stored in memory. As you work with the model, you'll want to check on the memory used by your database size on a regular basis. For tabular solutions, database size is measured by the amount of memory used to store the database at steady state. The easiest way to determine database size is by checking database properties in SQL Server Management Studio (SSMS).

If your tabular model is already built and deployed to a development server, you can determine memory usage for at-rest databases by reading the **Estimated Size** database property in SQL Server Management Studio, and then multiply that number by 2 or 3 to get a rough estimate of total memory needed for testing both processing and query workloads.



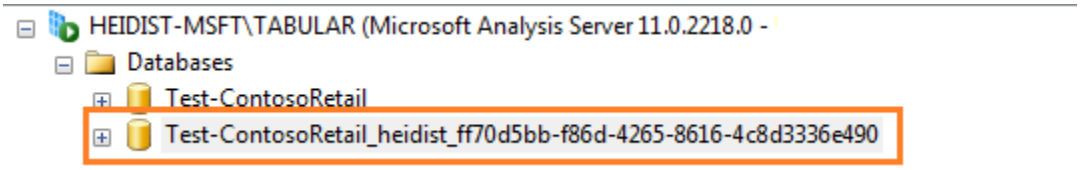
The advantage of using SQL Server Management Studio to estimate database size is that the database is loaded into memory when you access its properties. Using other approaches, for example using Performance Monitor to get memory usage for the **msmdsrv** process at startup, might initially under-report memory usage because you'll get only the memory used by Analysis Services plus the metadata of any databases that it's hosting. Actual data won't be read into memory until the first query is issued to the database.

NOTE: Although reading estimated size from SSMS is the easiest approach, it's not the most accurate approach because the value is estimated at a specific point in time, and then retained for the duration of the connection. Later, we'll present alternative approaches that provide more accuracy.

About the Workspace Database

When creating a tabular model in SQL Server Data Tools, a workspace database takes up memory on your development machine. As you monitor system resource usage on your development machine, remember that artifacts of the development phase, like the workspace database, are not part of production environments.

For this reason, when using Performance Monitor or other tools that report memory usage at the instance level, remember to unload the workspace database so that you get a more accurate assessment of memory usage.



To unload the database, simply close the project in SQL Server Data Tools (SSDT). The workspace database is immediately removed from the server instance you are using for development purposes.

NOTE: A workspace database will not unload if you set the **Workspace retention** option to **Keep in memory**. If this is the case, right-click the database and choose **Detach** to unload the database.

Finally, consider the effects of having multiple workspace databases on a single development server. If multiple developers are using the same workspace database server then you might have multiple copies of the database in memory and on disk.

1. In SSDT, go to **Tools** | **Options** | **Analysis Services**.
2. If "Keep workspace in memory" is set for multiple development machines then these workspaces will collectively consume memory on the database server. Using "Keep workspace databases on disk but unload from memory" is the better choice if multiple developers share the same server.

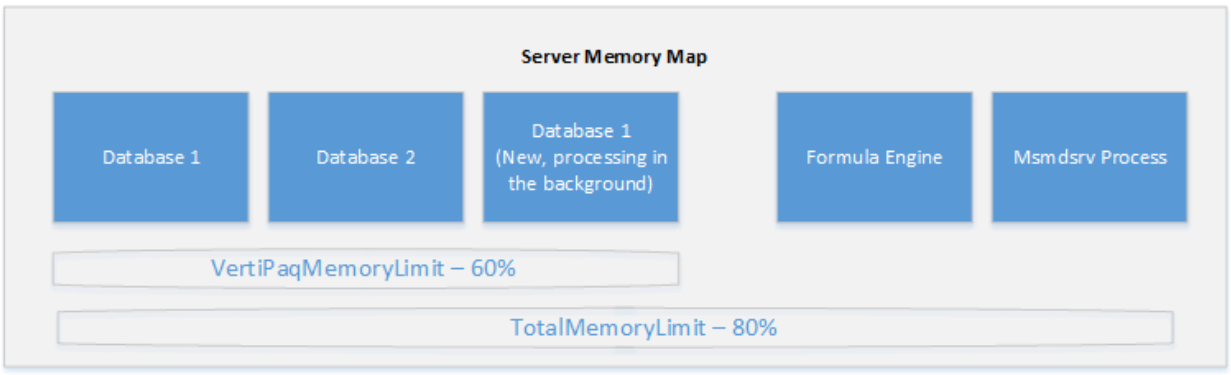
Memory

All in-memory database technologies require large amounts of RAM, so it comes as no surprise that maximizing RAM should be your top priority. Whether you plan to run multiple small to medium size solutions, or just one very large solution, begin your hardware search by looking at systems that offer the most RAM you can afford.

Because query performance is best when the tabular solution fits in memory, RAM must be sufficient to store the entire database. Depending on how you plan to manage and use the database, additional memory up to two or three times the size of your database might be needed for processing, disaster recovery,and for queries against very large datasets that require temporary tables to perform the calculation.

To get the most utilization from business class servers that have more memory than can be immediately used, some customers are using virtual machines and multi-tenant designs to host multiple servers and solutions on a high end server. Using a VM lets you adjust memory incrementally in response to changes in demand, and provides a mechanism for rolling back to previous snapshots if a server goes down. If your hardware has excess capacity at the outset, setting up multiple VMs lets you redistribute extra computing power across multiple logical servers. As the capacity requirements for one VM grows, you can adjust its configuration accordingly, and retire or move other VMs to different machines.

The following illustration provides a visual overview of all the data points you need to collect when estimating memory. It indicates 1-to-n databases, a shadow copy of a database being processed in the background, the formula engine, and **msmdsrv** process. By default, Analysis Services pre-allocates 60% of all RAM, and tops out at 80%. These thresholds are configurable at the instance level.



As you can see, there are variable and fixed components to a tabular instance deployment. If you never load a single data base, you will still need memory for the Formula Engine and msmdsrv process. Each database will place additional demands on the system in the form of data dictionaries, column segments, and query caches (not shared across databases).

> ***Maximum database size***
>
> *There are no theoretical limits on the physical size of the database, except those imposed by the system, which must be able to store all data dictionaries, column segments (this is the storage unit for tabular databases), caches, and the formula enginein memory. The system will page uncompressed data to disk during processing, but you should not count on it for normal server operations.*

Calculate Memory for Databases at Rest

As a first step, you will need to calculate the amount of RAM you will need to simply store the database, irrespective of the processing and query operations that will come later.

For databases that are already built, you can calculate database size in memory using any of the following techniques:

- Run a DMV query that reports on database size
- Read the database file size on disk
- View estimated database size in Management Studio (as mentioned in the previous section)

For projects at a more preliminary stage, including projects with data sizes that exceed the capacity of development machines, you will need to base your estimate on the uncompressed dataset that you are modeling, or build a prototype that uses a subset of your data.

Many customers routinely use PowerPivot for Excel as a tool for testing how much compression they'll get for their raw data. Although PowerPivot uses a slightly different compression algorithm, the compression engine is the same, allowing you to arrive at reasonable estimate when prototyping with the PowerPivot add-in. PowerPivot add-in is available in 32-bit and 64-bit versions. The 64-bit version supports much bigger models. Try to use that version if you can.

Other customers who work with large datasets usually apply filters during import to select a subset of data (for example, filtering on one day's worth of transactions). This approach gives you a smaller and more manageable dataset, while retaining the ability to extrapolate a realistic estimate of the final dataset (all things being equal, if one day's dataset is 20 MB, a month is roughly 600 MB).When choosing a filter, however, it is important to make sure that the subset of the data is still representative of the overall data. For example, it might make more sense to filter by date than by city or state.

If you are contending with tabular model deployment on a system that has little RAM to spare, you can optimize your model to reduce its memory footprint. Common techniques include omitting high cardinality columns that are not necessary in the model. Another tradeoff that a solution architect will consider is using a measure in lieu of a calculated column. A calculated column is evaluated during processing and persisted to memory. As a result, it performs well during query execution. Contrast that with a measure that provides equivalent data, but is generated during query execution and exists only until evicted from cache. The query runs slower due to extra computations, but the benefit is a reduction in persistent storage used on an ongoing basis.

Memory Requirements for Disaster Recovery

When estimating memory usage, remember to budget for database restore to allow for situations where the database becomes corrupted or unusable. Having insufficient memory to perform this operation will limit your recovery options considerably, perhaps requiring you to detach other databases that you'd rather keep online. Ideally, you should have enough RAM to restore and test a backup before removing the one that is being replaced.If the backup is corrupt, unusable, or too stale, you might want to salvage what you can from the original database before deleting it completely.

Having enough memory for restore equates to about 2x the size of the database, about the same amount you would need for full processing. Estimating hardware resources for disaster recovery does

not mean you need an additional 2x RAM on top of that already budgeted for processing; it just gives you one more reason to have RAM that is 2-3x size of your solution.

An alternative approach that others have found effective is to use VMs. You can quickly reload a previous virtual machine snapshot to restore data access to a tabular solution.

Memory Requirements for Program Execution

Analysis Services reserves approximately 150 megabytes (MB) of memory for program execution, Formula Engine, Storage Engine, shared objects, and shared cache. An Analysis Services instance that has no databases will use 150 MB of RAM on the host computer.

In addition, data dictionaries are always locked in memory, never paged to disk (only segment data can be paged to disk, and then only when the system is under memory pressure). The data dictionaries of all databases running on a tabular mode server will load into memory on service start up.

Estimating Memory for Processing

Processing, which includes reading, encoding, and loading compressed data into tabular model objects, is the first resource-intensive workload you'll need to consider. Processing is a memory-intensive operation, requiring at least two times the amount of memory required to store the database when it's finally processed and deployed. Processing uses memory for storing both compressed and uncompressed data while it's being read and encoded.

For exceptionally large databases, you can offload processing to another machine, and then synchronize the fully processed database to a production server. If you refer back to the server map illustration, this approach saves you the memory required for hosting the current database (used for servicing queries) while the new database is built in the background. Although this approach to processing can lower your resource requirements considerably, for the purposes of this guide, we'll assume you are processing the database on the production server.

The most common approach for estimating memory usage for processing is based on a factor of database size at zero load. Memory requirements will vary depending on how well data is compressed.Notably, compression is greater for columns that store repeating values, whereas columns containing distinct values do not compress well.

The technique we recommend for getting preliminary estimates is to use a simple formula for calculating compression and processing requirements. In spite of its limitations, using a formula to is a reasonable place to start if you require a ballpark figure. To improve upon your estimate, you can prototype the model you plan to deploy using a representative sample of the larger dataset:

- Use a formula to get an initial estimate.
- Build a prototype to refine the estimate.

Alternatively, import a subset of rows from each table and then multiply to get an estimated size.

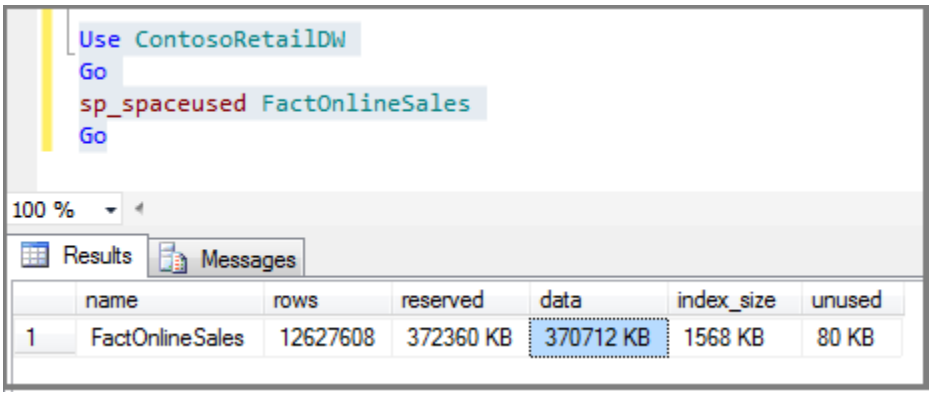
Use a formula to get an initial estimate

The following formula provides an initial estimate of the memory you'll need to process and store all of your data in local memory.

([Size of uncompressed data] /10) * 2

Uncompressed data is the size of the database file, assuming you're importing all or most of the database into a tabular solution.

For a SQL Server database, you can run sp_spaceused to return size information for each table. The following screenshot provides an illustration using the FactOnlineSales table, which measures around 362 megabytes.



Alternatively, to get overall database size, omit the table name and execute **sp_spaceused** as follows:

```
Use ContosoRetailDW;
Go
Exec sp_spaceused;
Go
```

Composition of the formula

Dividing uncompressed data by ten gives you an estimate of model size after the data is compressed. This is how much memory the model will consume once it is deployed to the server.

The denominator (10) was arrived at by averaging the actual compression rates over a wide range of data sets. For your solution, actual compression might be much lower (2 or 3) or much higher (100x), depending on data type, length, and cardinality. Tables or columns containing mostly unique data values (such as fact tables) will experience minimal compression, while other tables might compress by a

lot. For estimation purposes, ten is often cited as an acceptable, middle of the road estimate of overall compression for a database that includes a balance of unique and non-unique values.

The second part of the formula asks you to double the amount of memory used by the database. Doubling memory accounts for processing operations, as the server keeps a second copy of the database in memory to service query requests while processing runs in the background.

Memory used to store the database copy is immediately released after processing concludes, but that doesn't mean memory usage diminishes to just the RAM needed to store the database. Queries and Storage Engine caches also consume memory. On the server, a query devolves into table scans that select and aggregate data, calculations, and other operations.

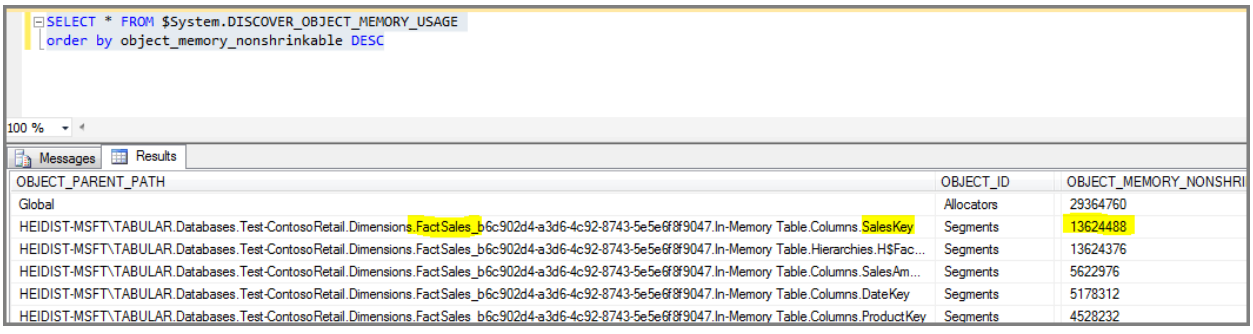
> *As you investigate memory usage by tabular solutions, you might come across other formulas that are more robust mathematically, yet harder to use if you are not deeply familiar with the data. In the end, you might find it more productive to use a simplistic formula for an initial estimate, and then move on to prototype a solution usinga subset of your own data.*

Measure memory used by individual objects

Analysis Services provides Dynamic Management Views (DMVS) that show metadata and system information, particularly memory usage, for model objects.

While not strictly a hardware sizing exercise, knowing which objects take up the most memory gives you the opportunity to identify and possibly remove any resource-intensive objects you don't actually need.

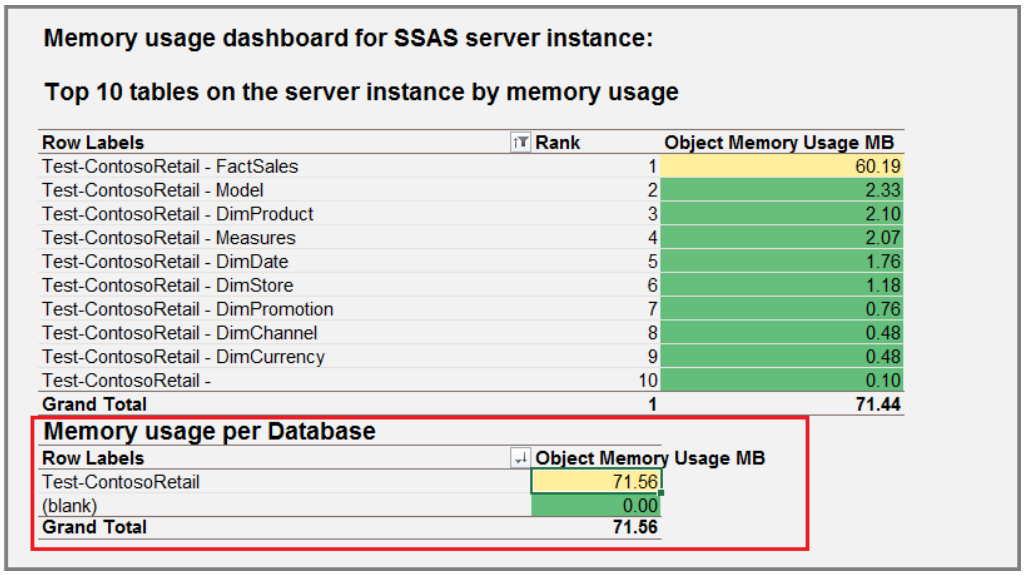
1. Open an MDX query window in SQL Server Management Studio, connecting to the tabular instance running your database.
2. Enter the following DMV command, Select * from $System.discover_object_memory_usage, and click **Execute**. You can order by nonshrinkable memory to see which columns (in this case, the SalesKey in the FactSalesTable) are using the most memory.



While running a DMV is acceptable, we recommend that you download and use a workbook created and published by Kasper De Jonge, an Analysis Services program manager. His workbook uses DMV queries

to report memory usage by object, but improves upon the raw DMV by organizing and presenting the results in a hierarchy that lets you drill down into the details.

Best of all, the workbook reports on database size.



Memory usage dashboard for SSAS server instance:

Top 10 tables on the server instance by memory usage

Row Labels	Rank	Object Memory Usage MB
Test-ContosoRetail - FactSales	1	60.19
Test-ContosoRetail - Model	2	2.33
Test-ContosoRetail - DimProduct	3	2.10
Test-ContosoRetail - Measures	4	2.07
Test-ContosoRetail - DimDate	5	1.76
Test-ContosoRetail - DimStore	6	1.18
Test-ContosoRetail - DimPromotion	7	0.76
Test-ContosoRetail - DimChannel	8	0.48
Test-ContosoRetail - DimCurrency	9	0.48
Test-ContosoRetail -	10	0.10
Grand Total	**1**	**71.44**

Memory usage per Database

Row Labels	Object Memory Usage MB
Test-ContosoRetail	71.56
(blank)	0.00
Grand Total	**71.56**

Refine the estimate by building a prototype that yields a better value for data compression

As we work our way up the continuum of estimation techniques, we arrive at one of the more robust approaches: prototyping using your own data.

The best way to determine how well your data compresses in a tabular solution is to start with some initial data imports. Because the objective is to understand compression behavior, treat this as a prototyping exercise. If you were building a model you planned to keep, you would spend time thinking about model design. For our purposes, you can set those problems aside and focus simply on choosing which tables and columns to import for the purpose of estimating database size.

The following steps approach prototyping from the standpoint of large datasets. If your dataset is not large, you can just run **Process Full**. Otherwise, process just one table at time, and then run **Process Recalc** at the end to process table dependencies.

1. Create a new tabular project using SQL Server Data Tools.
2. Import the largest table from your external data source into a tabular model. If it's a fact table, exclude any columns that are not needed in the model. Usually, the primary key of a fact table is a good candidate for exclusion, as are columns that are only required for ETL processing.
3. Process the table and deploy it to a development server.
4. If deployment succeeds, measure the size of the compressed table in memory. If deployment fails, apply a filter to get a smaller rowset, while ensuring that the filtered rowset is still representative of the overall table.

5. Import and process a second table, deploy the solution, measure memory usage, and then repeat with additional tables.
6. Stop when you have sufficient dataset representation in your model.
7. As a final processing step, run **Process Recalc** to process relationships.
8. Measure the memory used by the database by viewing database properties in SQL Server Management Studio, or by using DMVs if you want to drill into the details. At this point, you now have a solid foundation for projecting how much memory you'll need for the rest of the data.

Calculate actual compression rate

Once you've processed and deployed a solution, you have compressed database files on disk that you can use to calculate a more realistic compression rate. Comparing the file size of compressed data against uncompressed data gives you the actual compression rate for your solution.

After you get an actual compression rate, you can replace the denominator (10) in the 'simple formula' with a more realistic value.

1. Get the file size of the original uncompressed data.
2. Find the \Program Files\Microsoft SQL Server\MSAS11.<instance>\OLAP\DATA\<db folder>, and note the file size.
3. Divide the result from step 1 by the result in step 2 to get the compression ratio.
4. Re-compute the simple formula, replacing 10 with your actual compression rate, to get estimated memory requirements for processing.

NOTE: While this approach is generally reliable, in-memory databases tend to be somewhat larger than database files on disk. In particular, having highly unique columns will cause memory usage to exceed the size of data in the data folder. As a redundant measure, use alternative methods such as the DMV, database property page in Management Studio, or Performance Monitor to further check database size.

Estimating Memory for Querying

Memory requirements for query workloads are by far the most difficult to estimate. Unless your solution includes a report or client application that uses predefined or predictable queries, it is difficult to make a precise calculation. You will need to perform ad hoc testing and load testing, and monitor resource usage while querying from client applications. Power View, Reporting Services reports, and Excel all use memory differently. Furthermore, DAX query construction can adversely affect resource usage. Certain DAX functions (such as EARLIEST, SUMX, and FILTER) are known to cause a temporary yet significant increase in memory usage as temporary tables are created and iterated over to determine which rows to return.

DAX query optimization is beyond the scope of this guide, but other sources are available that cover this material. See the links at the end of this document for more information.

About Concurrency Testing

When you have a large number of users requesting data from a model, memory usage will climb accordingly, but the amount required will vary depending on the query itself. If you have 100 users accessing a report that issues an identical query each time, the query results are likely to be cached and the incremental memory usage will be limited to just that result set. Of course, if each user applies a filter to that query, it's the equivalent of having 100 unique queries, a vast difference in memory consumption.

To truly understand the impact of a large number of requests, you will need to load test your solution. Visual Studio includes a load testing module that lets you simulate multiple user requests at varying intervals for different client applications. The configuration of load tests using this facility is beyond the scope of this guide, but worth mentioning due to its relevance to hardware provisioning. Serious load testing requires a professional toolset. It's not simple to do, but Visual Studio offers one of the better approaches for testing a server under load. For more information, see Working with Load Test (Visual Studio Team Edition), Can your BI solution scale?, and Load Testing Analysis Services.

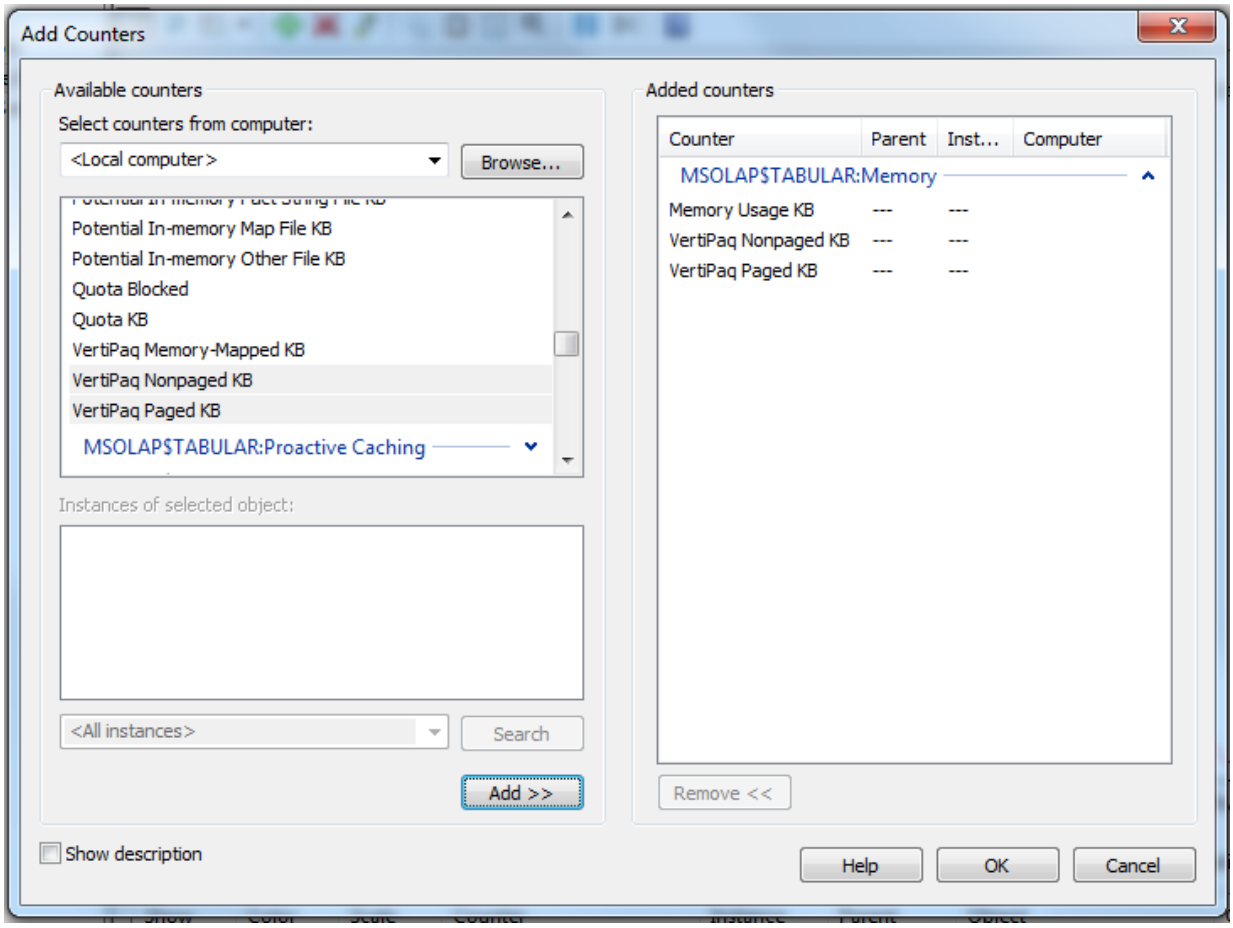
Monitor memory usage during processing and querying

Performance Monitor is an oft-used tool for understanding memory usage and trends at the operating system level. You'll use it regularly to understand memory allocation and release patterns for your tabular solutions.

The following steps will get you started with monitoring RAM usage in Performance Monitor:

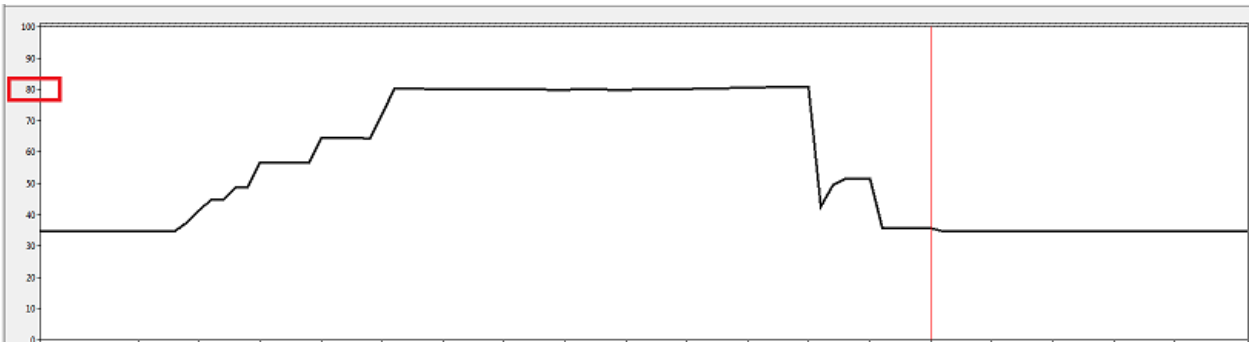
1. Isolate a database at the instance level by detaching other databases so that an instance is running only the database we want to look at. You can detach databases using Management Studio.
2. In Performance Monitor, select the tabular instance (on the system shown in the screenshot, a tabular server is installed as named instance, as localhost\tabular).

3. Expand **MSOLAP$TABULAR:MEMORY** and select **Memory Usage KB**, **VertiPaq Nonpaged KB**, and **VertiPaq Paged KB**.



4. Execute a **Process Full** against the database in Management Studio to understand how memory is used.

The following screen capture shows a typical pattern of increasing memory consumption (the counter shown is for **Memory Usage KB**), leveling out at maximum allowed (80% RAM by default), with memory released as processing concludes.

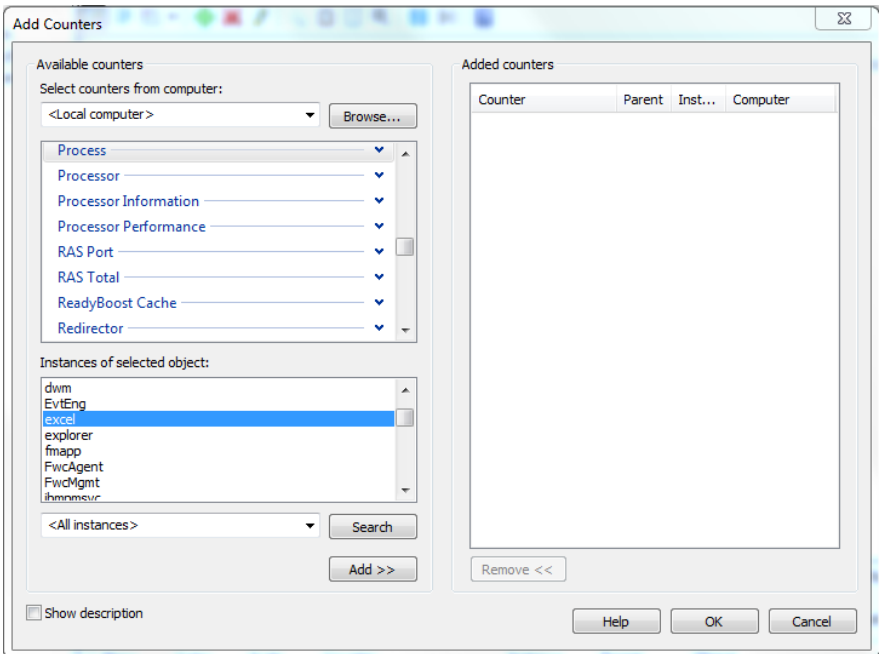


Next, issue queries against the database to understand the query profile of your client application.

If the query can be executed against the compressed data, you won't see a noticeable difference in memory usage on the server. Otherwise, you'll see a transient surge in memory usage as temporary tables are created to store and scan uncompressed data.

Although memory spikes during query execution are sporadic, the same cannot be said for client applications. Client applications will most certainly consume memory as data is retrieved from the tabular model. As part of your investigation, consider adding client processes to the trace to monitor memory usage.

1. On the client computer, start the client applications used to query the model.
2. In Performance Monitor start a new trace.
3. Add counters for client processes.Client processes are listed under the **Process** object in Performance Monitor.
    a. For Management Studio, select **ssms** (not to be confused with **smss**).
    b. For Excel, select **excel**.
    c. For Power View in SharePoint, hosting is in one the SharePoint Service Application AppPool processes (usually w3wp.exe).



Key Points

A tabular solution uses memory during processing, when loading metadata after a service restart, and when loading remaining data on the first query issued against the model. Once a tabular database is loaded, memory usage remains relatively stable unless SSAS needsto build temporary tables during query execution.

Relative to processing and basic storage requirements, memory used for queries tends to be minimal. If you're monitoring a query workload in Performance Monitor, you'll notice that memory usage is often flat for many queries.

A query can result in a temporary but radical uptick in memory usage if a column or table needs to be decompressed during query execution. Certain functions (such as EARLIEST, SUMX, and FILTER) are known to have performance impact. Always test the queries and reports you plan to run in a production environment to understand their performance profile.

When calculating memory usage on a development machine, any workspace databases loaded in memory will skew your measurements. Be sure to unload the workspace database when collecting metrics about memory usage.

Finally, remember to monitor memory used by client applications. A query might be trivial for the Storage Engine, yet bring a client workstation to its knees if a massive amount of data is read into its memory.

Memory Bandwidth and Speed

Moving from a disk-bound architecture to memory-bound architecture shifts performance bottlenecks from disk access to memory access. Just like you needed fast disks for multidimensional models, you'll need fast memory to get the performance you want from of a tabular solution.

As of this writing, we're still testing memory subsystem features to understand where the tradeoffs are. Questions under investigation include identifying changes in query performance when adding more integrated memory controllers versus adding more RAM. In a future update to this paper, we plan to share our findings to these and other questions related to memory performance.

What we do know at this point is memory bandwidth becomes increasingly important as you add cores. The more cores you have, the greater the number of threads requesting read operations from memory. Many server workloads involve scanning large tables with little processing; having sufficientmemory bandwidth lets more of those requests get through at one time.

Bus speed is also crucial. In a few isolated cases, high-end workstations that have a shorter (faster) bus have been known to outperform business class servers.

In the next section, we'll discuss the importance of onboard cache. However, regardless of how much onboard cache you have, reading data from RAM is going to happen regularly. Systems that have high performance memory and extra bandwidth are better equipped to deal with the technical challenges of hosting an in-memory solution. As you evaluate different hardware systems, look for systems that offer better than average memory performance and integrated memory controllers.

Key Points

Tabular solution architecturesare optimized for query performance, predicated on RAM storage being much faster to read from than disk. When evaluating RAM, consider memory designs that balance throughput and speed.

CPU

Selecting a fast CPU with a sufficient number of cores is also a top consideration. In a tabular solution, CPU utilization is greatest when query evaluation is pushed to the Storage Engine. In contrast, CPU bottlenecks are more likely to occur when a query or calculation is pushed to the single-threaded Formula Engine. Because each query is single-threaded in the Formula Engine, on a multi-core system, you might see one processor at maximum utilization while others remain idle.

Constructing queries that only run in the Storage Engine might sound tempting, but it's unrealistic as a design goal. If the point of your model is to provide insights that solve business problems, you'll need to provide queries and expressions that meet business goals, irrespective of query execution mechanics. Furthermore, if you're provisioning a server that hosts self-service BI solutions built and published by other people, query syntax construction is probably beyond your control.

A CPU with clock speeds of at least 2.8 to 3 GHz is your best insurance against queries that execute single-threaded in the Formula Engine, slow queries that are difficult optimize, or suboptimal query syntax created by novice model designers.

Query Load on CPU

In some cases, queries against a tabular model can often be pushed down to the multi-threaded xVelocity Storage Engine, where each query job runs on a separate core. Depending on the size of the dataset and the query itself, you might see greater CPU utilization, which usually equates to better performance.

Queries most likely to run in the Storage Engine are based on simple measures that can be calculated without having to decompress the data first. If a column needs to be decompressed, such as when computing a rank order of all values, a temporary table is created in memory to store and scan the values. This operation consumes memory and CPU, especially if the calculation needs to be handled by the Formula Engine.

Queries that run only in the Formula Engine, such as an evaluation of a SUMX or Filter operation, are single-threaded and a common query performance bottleneck.FILTER iterates over the entire table to determine which rows to return.

Effect of Concurrent Queries on CPU

The number of clients requesting data from the model will also factor heavily into how much CPU resource you'll need. As previously noted, each query moves through the Formula Engine as a single-threaded operation. If you have 100 unique queries running simultaneously, you'll want significantly more cores to handle the load.

Processing Load on CPU

Processing can take a long time to complete, but is typically not considered to be CPU intensive. Each processing job uses one to two cores; one core to read the data, and another core for encoding. Given that each partition within one table must be processed sequentially, the pattern of a processing operation tends to be a small number of cores sustained over a longer period of time. However, if processing many tables in parallel, CPU usage can rise.

Cores

Now that you have a basic understanding of how CPU resources are used, let's move on to specific CPU designs most often used to support medium to large solutions.

For tabular solutions, the most frequently cited CPU designs range from 8 to 16 cores. Performance appears to be better on systems that have fewer sockets. For example, 2 sockets with 8 fast cores, as opposed to 4 sockets with 4 cores. Recall from the previous section the importance of memory bandwidth and speed in data access. If each socket has its own memory controller, then in theory, we should expect that using more cores per socket offers better performance than more sockets with fewer cores.

Equally important, performance gains tend to level off when you exceed 16 cores. Performance doesn't degrade as you add more cores; it just fails to produce the same percentage increase that you achieved previously. This behavior is not specific to tabular solutions; similar outcomes will be encountered when deploying any memory intensive application on a large multi-core system.

The problem is that memory allocations fall behind relative to the threads making memory requests. Contention arises as all cores to read and write to the same shared resource. Operations become serialized, effectively slowing down server performance. The end result is that a CPU might be at 30-40% utilization yet unable to perform any additional processing due to bottlenecks in the memory subsystem. Cores are idled while waiting for memory allocations to catch up.For more information about this behavior, see Detecting Memory Bandwidth Saturation in Threaded Applications.

Other considerations

When evaluating the number of cores, consider associated software costs and limits that increase with the core count. In most cases, software licensing fees go up with the number of cores.

Software licensing fees

Software licensing fees vary based on the number of cores used by SQL Server. As of this writing, the use of more than 20 cores requires an Enterprise edition and a per-core or volume licensing option. See Compute capacity limits by edition to determine the maximum number of cores supported by the SQL Server edition you're using.

Alternatively, you can use the Business Intelligence edition and Server CALs to license by the number of people using the server. Check the Microsoft licensing web site for an explanation of BI edition licensing.

> *Be aware that running multiple SQL Server features on the same hardware is known to slow system performance as the operating system and various services compete for the same resources. Before installing multiple SQL Server features on the same machine, take a look at a server memory calculator that SQL CSS created. It can give you an idea of how well your machine supports the relational engine and operating system. The calculator does not account for Analysis Services, but if you are installing multiple SQL Server feature components, you'll at least understand how to configure the system to provide adequate RAM for the relational engine and operating system.*

Virtual machine limits and licenses

Be aware that using virtual machines will impose new limits on the number of logical cores you can use. In Hyper-V on Windows Server 2012, maximum RAM is 2 TB and maximum virtual cores is 64. Earlier versions of Windows bring that maximum down to 4 cores. For more information, see maximum virtual processor information for Windows Server 2012 and earlier versions. Other virtualization vendors operate under similar constraints; if you are using a different VM technology check the product web site for maximum limits.

Onboard Cache (L1, L2)

Because onboard cache is so much faster than RAM, systems that offer proportionally more L1 and L2 cache are better for a tabular data access.

In SQL Server 2012 SP1, Analysis Services added cache optimizations to speed up performance. To use the onboard cache to its best advantage, data structures providing query results were resized smaller to more easily fit the size constraints of onboard cache. In particular, having a large L2 cache, which is typically shared by all cores on the same socket, has proven to be especially helpful in boosting query performance for a tabular solutions.

The L2 cache is valuable during table scans (i.e., when caches are missed) because the iteration over memory completes sooner when data is found in L2 cache. As the operating system accesses the first

few bytes of a block of data, it can fetch extra bytes into the L2 cache, and as the scan proceeds to bytes further in the block, it will hit them in the L2 cache rather than RAM. Also, queries can benefit from the L2 cache when doing lookups of common dictionary values and relationships.

> *If you are familiar with how Analysis Services manages caching for multidimensional solutions, caching behavior for a tabular solutions is considerably different. Tabular solutions use two types of cache: a Storage Engine cache, and a cell level cache created by MDX queries. Cell-level caching is done by the Formula Engine. It's the same caching used for multidimensional solutions.*
>
> *Storage Engine caches are created during query execution. The Formula Engine will sometimes fire one or more Storage Engine queries, which are then cached by the Storage Engine. If a client executes the same query multiple times, query results are returned from the cache, eliminating the Storage Engine portion of query execution. By default, server configuration properties specify 512 Storage Engine cache slots. You can change this setting in the msmdsrv.ini file if you get too many cache misses.*
>
> *In terms of hardware caching, the operating system fully controls which data structures are stored in onboard cache or RAM, with no intervention from Analysis Services. In terms of solution architecture, what this means to you is that there is no specific action or server reconfiguration on your part that can change hardware caching behavior. Data structures will either be placed in onboard cache or in RAM, depending on resource availability and competition from other applications using the same resources.*

Tip: You can download CPU-Z utility from CPUID.com to determine L1, L2, and L3 cache on your computer.

NUMA

Unlike its multidimensional (MOLAP) counterpart, a tabular solution is not NUMA aware. Neither the Formula Engine nor the Storage Engine will modify execution when running on NUMA machines.

This means that if you have a NUMA machine, you might run into worse performance than if you used a non-NUMA machine with the same number of cores. Typically, this only happens on systems having more than 4 NUMA nodes.

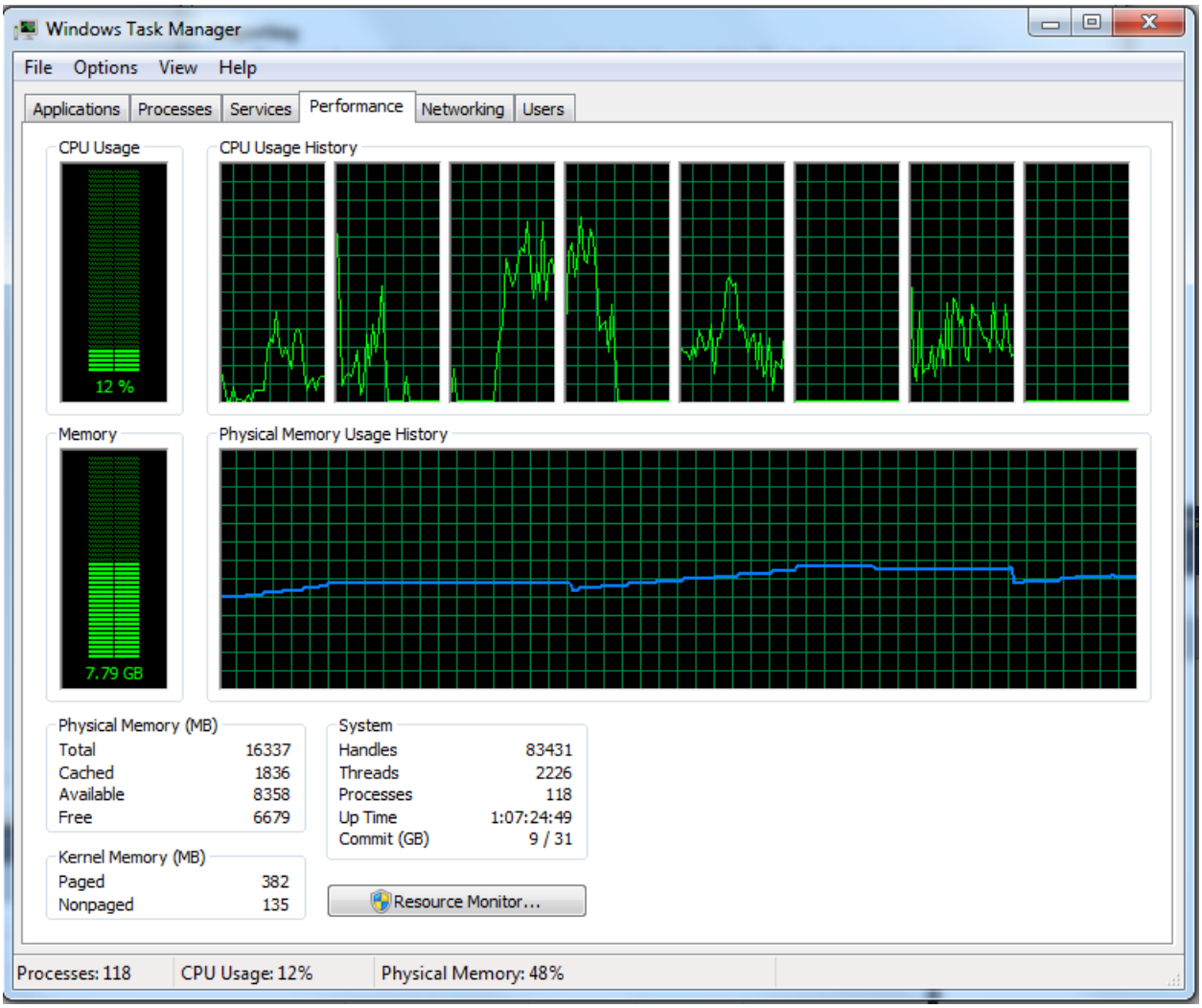
Performance degradation occurs when memory access has to traverse NUMA nodes (i.e., a thread or instruction executing on one node needs something that is executing on another node). When choosing between systems that have the same number of cores and RAM, pick a non-NUMA system if you can.

To offset performance degradation, consider setting processor affinity on Hyper-V VMs, and then installing Analysis Services tabular instances on each VM. For more information about this technique, see Using Tabular Models in a Large-scale Commercial Solution and Forcing NUMA Node affinity for Analysis Services Tabular databases.

Monitor CPU Usage

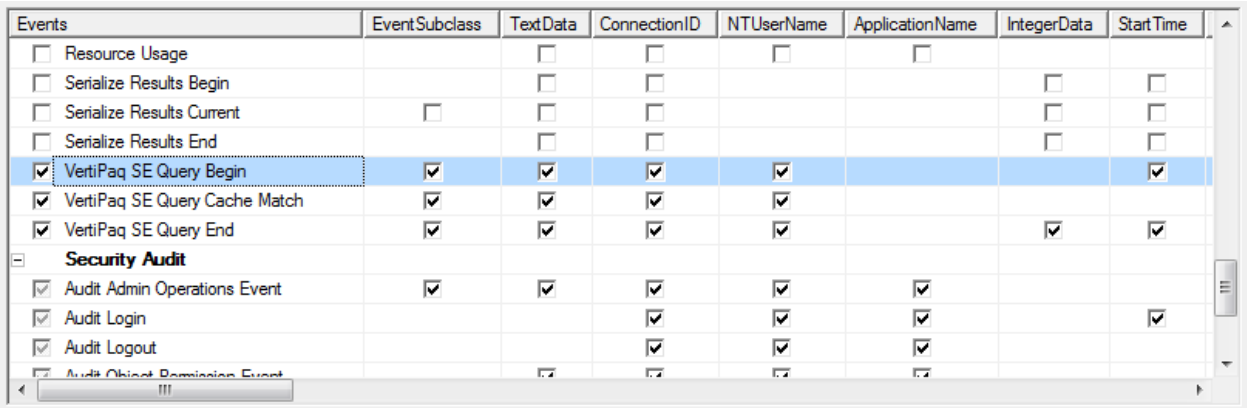
Using Task Manager, you can get an overall sense of CPU utilization during processing and query operations. While Task Manager might not be the best tool to use for performance tuning, it should be adequate for assessing the CPU requirements of your tabular solution. With Task Manager started, run a representative sample of processing and query workloads to see how the system performs.

For example, the following screenshot shows CPU usage while executing a **Process Full** on ContosoRetailDW. Tables are mostly processed in sequence, and as you can see, overall CPU usage is relatively modest.



CPU usage during query execution

Queries that are pushed to the Storage Engine complete much sooner than those that spend additional cycles in the Formula Engine. You can identify which queries are pushed to the Storage Engine by using SQL Server Profiler. When setting up a new trace, click **Show all events**. Events you will want to select appear under Query Processing: **VertiPaq SE Query Begin**, **VertiPaq SE Query Cache Match**, **VertiPaq SE Query End**.



When you run the trace, you can monitor query execution to determine query duration and execution. Queries pushed to the Storage Engine are indicated through the event name. A line has VertiPaq in the name tells you that part of the expression has been pushed down to xVelocity Storage Engine.

EventClass	EventSubclass	TextData	Duration	DatabaseName	ObjectName	SPID	CPUTime
Query Begin	0 - MDXQuery	SELECT NON EMPTY ...		Contoso...		8423	
VertiPaq SE Query Begin	0 - VertiPaq Scan	SET DC_KIND="AUTO...		Contoso...	FactOnlineSales	8423	
VertiPaq SE Query Begin	10 - VertiPaq Scan...	SET DC_KIND="DENS...		Contoso...	FactOnlineSales	8423	
VertiPaq SE Query End	10 - VertiPaq Scan...	SET DC_KIND="DENS...	50	Contoso...	FactOnlineSales	8423	47
VertiPaq SE Query End	0 - VertiPaq Scan	SET DC_KIND="AUTO...	60	Contoso...	FactOnlineSales	8423	47
VertiPaq SE Query Begin	0 - VertiPaq Scan	SET DC_KIND="AUTO...		Contoso...	DimGeography	8423	
VertiPaq SE Query Begin	10 - VertiPaq Scan...	SET DC_KIND="DENS...		Contoso...	DimGeography	8423	
VertiPaq SE Query End	10 - VertiPaq Scan...	SET DC_KIND="DENS...	0	Contoso...	DimGeography	8423	0
VertiPaq SE Query End	0 - VertiPaq Scan	SET DC_KIND="AUTO...	0	Contoso...	DimGeography	8423	0
VertiPaq SE Query Begin	0 - VertiPaq Scan	SET DC_KIND="AUTO...		Contoso...	FactOnlineSales	8423	
VertiPaq SE Query Begin	10 - VertiPaq Scan...	SET DC_KIND="DENS...		Contoso...	FactOnlineSales	8423	
VertiPaq SE Query End	10 - VertiPaq Scan...	SET DC_KIND="DENS...	60	Contoso...	FactOnlineSales	8423	62
VertiPaq SE Query End	0 - VertiPaq Scan	SET DC_KIND="AUTO...	60	Contoso...	FactOnlineSales	8423	62
VertiPaq SE Query Begin	0 - VertiPaq Scan	SET DC_KIND="AUTO...		Contoso...	DimDate	8423	
VertiPaq SE Query Cache M...	0 - VertiPaq Cache...	SET DC_KIND="AUTO...		Contoso...	DimDate	8423	
VertiPaq SE Query End	0 - VertiPaq Scan	SET DC_KIND="AUTO...	0	Contoso...	DimDate	8423	0
VertiPaq SE Query Begin	0 - VertiPaq Scan	SET DC_KIND="AUTO...		Contoso...	DimGeography	8423	

As you monitor query execution to understand hardware requirements, take the time to identify any queries that could benefit from optimization. Queries that take longer than expected will not only frustrate users, it will artificially and unnecessarily raise the level of system resources required by your solution.

Although DAX query optimization and DAX query plan analysis is beyond the scope of this guide, there are links at the end of this document that provide more information.

Key Points

CPU resources are heavily used for queries and some calculations. Simple mathematical computations based on a single column (for example, summing or averaging a numeric column) are pushed to the Storage Engine and executed as a multi-threaded operation on multiple cores. In contrast, a calculation that ranks or sorts values requires the single-threaded Formula Engine, using just one core and possibly lots of memory depending on the size of the temporary table.

Incremental performance gains tend to level off after 16 logical cores. Although a greater number of cores (32 or 64) will definitely increase capacity, you won't see the same gain in performance increase when going beyond 16 cores.

As SSAS Tabular is not NUMA aware, avoid NUMA unless you need it for other applications that run on the same hardware. There will be longer wait times if a request has to traverse NUMA nodes during a read operation.

Finally, when you've narrowed your server selection to a few choices, take a look at the onboard cache and choose the system that offers the larger onboard cache. Query cache optimizations in the tabular engine target the L1 and L2 cache. You gain the most benefit from those optimizations on a system that offers more onboard cache.

Disk I/O

Disk I/O, which normally looms large in any hardware sizing exercise, is less of a concern in tabular solution hardware sizing because given sufficient RAM, tabular solutions are not reading or writing to disk during query execution. Solid performance of table scans, aggregations, and most calculations are predicated on having an ample supply of RAM. If the operating system has to page memory to disk, performance degrades dramatically.

On a properly provisioned server, disk IO occurs infrequently, but at predictable intervals. You'll always see disk I/O activity during processing when reading from a relational database (in comparison, saving the tabular database files to disk is relatively quick). You will also see some I/O after system restart when metadata and data dictionaries are loaded into memory. Data dictionaries are loaded sequentially so this step can take some time if you have a large solution or lots of smaller solutions. You will see I/O activity again when the rest of the data is loaded, typically when the first query is executed. For query workloads, the ideal system should have sufficient memory so that paging to disk does not occur at all.

Although disk I/O is not a hardware investment to maximize, don't discount it entirely. A system that loads many gigabytes of data from disk to memory will perform better if the disk is fast.

NOTE: Paging to disk will only occur if you set the **VertiPaqPagingPolicy** to 1. The default setting is 0, which disallows paging to disk. For more details, Memory Settings in Tabular Instances of Analysis Services.

Hardware Configuration Examples

This section provides a few starter examples of hardware configurations of existing deployments, along with additional notes about the production environment. Over time, we plan to update this table with more examples that demonstrate the breadth of deployment that include in-memory databases.

Deployments described in the following table range from dedicated servers to multi-tenant servers using VM technology running on newer and older hardware. For each deployment, solution architects reported above average performance on the hardware used to run the model.

Model Size	System Information	RAM	Other Details
40 GB	Dell PowerEdge R810, dual 8-core CPU	256 GB	Server runs other SQL Server features as well, including the relational engine and Analysis Services in multidimensional mode. Processing for the tabular solution runs on the same server. **ProcessFull** on a weekly basis, and **ProcessUpdate** nightly.
40 GB	Hewlett-Packard ProLiant DL580 (2)	1 TB	Multi-tenant architecture supporting at least 4 virtual machines will run on the two systems, hosting Analysis Services in tabular mode, Analysis Services in multidimensional mode, SharePoint with Reporting Services Power View, and a SQL Server relational database engine. Decisions about how to allocate memory across all VMs are still pending. Solution design consists of several smaller tabular models, about 10 total, consuming around 40 GB of memory all together.
4 GB	Hewlett-Packard ProLiant BL460 G7 Processor: 2 x Intel X5675 3.07 GHz	96 GB	System is purposely oversized to accommodate expected growth in the data warehouse.
6 GB	Commodity blade servers	16 GB	Using VMs to host multiple smaller solutions. VMs are currently configured to use 16 GB, which can be increased if

			memory-related errors occur.

Anecdotally, we know that tabular models sometimes perform better on faster, newer processors than on high-end server hardware. Workstations that offer more in terms of raw processor performance are often first to market. When evaluating hardware, broaden your search to include workstations that you might not otherwise consider. See this case study to read about one solution that uses high-end workstations for tabular workloads.

Conclusion

In this guide, we reviewed a methodology for estimating memory requirements for a database at steady state, under processing workloads, and under query workloads. We also covered hardware configurations and tradeoffs to get the best price to performance ratio.

In simplest terms, when budgeting hardware for a tabular database, you should maximize these system resources.

- As much RAM as you can afford
- Fast CPU with multiple cores on the fewest number of sockets
- Onboard cache (L2)

Although the trend is to maximize the number of cores, on a tabular server, hardware investment should favor query performance and table scans. Investing in onboard memory, memory speed, and memory bandwidth often yield a better return on investment than upping the number of cores. Also, recall that licensing fees go up as you increase the number of cores.

Memory prices have come down in recent years, but it's still a significant cost, especially at levels you'll be considering for in-memory storage technologies. Systems that offer 64, 128, 256 gigabytes will have other features like onboard memory controllers that can offset the benefits of just adding more memory. You might find that 64 GB with a second memory controller is a better solution that 128 GB with one memory controller.

A few final points to keep in mind:

- If you purchase a high-end machine, you can get immediate use out of excess RAM by distributing available memory across multiple VMs dedicated to different applications and workloads, and then reconfigure memory and cores as capacity requirements increase.
- Remember that compression and query performance are variable, depending on data cardinality, data density, and the types of queries that run on the server. Two different models that both measure 120 GB in size will use resources differently depending on the types of queries submitted to each one. You will need to approach each solution as a unique project and do thorough testing to determine the hardware requirements for each one.

- Continuous monitoring is essential to anticipating future capacity needs. Pay close attention tomemory usage over time, especially if subsequent processing is resulting in larger and larger models, to ensure that query performance stays robust.

For more information:

Tabular Model Solution Deployment

Process Database, Table or Partition

Tabular Model Partitions

Memory Usage

Detecting Memory Bandwidth Saturation in Threaded Applications

About the relativity of large data volumes

VertiPaq vs. Column Store

What is using all that memory on my Analysis Services instance?

Investigating on xVelocity (VertiPaq) column size

Optimizing High Cardinality Columns in VertiPaq

http://en.wikipedia.org/wiki/Memory_bandwidth

Memory Settings in Tabular Instances of Analysis Services

Memory Properties (Books Online)

Memory Considerations about PowerPivot for Excel

How much data can I load into PowerPivot?

Create a memory-efficient Data Model using Excel 2013 and the PowerPivot add-in

http://blogs.msdn.com/b/sqlsakthi/archive/2012/05/19/cool-now-we-have-a-calculator-for-finding-out-a-max-server-memory-value.aspx

http://blogs.msdn.com/b/sqlsakthi/p/max-server-memory-calculator.aspx

Load Testing

(video) Optimizing Your BI Semantic Model for Performance and Scale

(video) Load Testing Analysis Services

Can your BI solution scale?

http://www.tomshardware.com/reviews/ram-speed-tests,1807-2.html

DAX Optimizations

http://mdxdax.blogspot.com/2011/12/dax-query-plan-part-1-introduction.html

http://mdxdax.blogspot.com/2012/01/dax-query-plan-part-2-operator.html

http://mdxdax.blogspot.com/2012/03/dax-query-plan-part-3-vertipaq.html

http://www.powerpivotblog.nl/tune-your-powerpivot-dax-query-dont-use-the-entire-table-in-a-filter-and-replace-sumx-if-possible

http://sqlblog.com/blogs/marco_russo/archive/2011/02/07/powerpivot-filter-condition-optimizations.aspx

http://www.sqlbi.com/articles/optimize-many-to-many-calculation-in-dax-with-summarize-and-cross-table-filtering/

http://sqlblog.com/blogs/marco_russo/archive/2012/09/04/optimize-summarize-with-addcolumns-in-dax-ssas-tabular-dax-powerpivot.aspx

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

- Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?
- Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

Send feedback.