

Analysis Services Operations Guide

SQL Server White Paper

**Writers:** Thomas Kejser, John Sirmon, and Denny Lee

**Technical Writer and Editor:** Heidi Steen, Beth Inghram

**Contributors and Technical Reviewers:**

|  |  |
| --- | --- |
| Kagan Arca | Chris Webb (Crossjoin Consulting) |
| Akshai Mirchandani | Greg Galloway (Artis Consulting) |
| Edward Melomed | Andrew Calvett (UBS) |
| Brad Daniels | Alejandro Leguizamo (SolidQ) |
| Ashvini Sharma | Darren Gosbell (James & Monroe) |
| Sedat Yogurtcuoglu | Marco Russo (Loader) |
| Alexei Khalyako | Alberto Ferrari (SQLBI) |
| Peter Adshead (UBS) | Sanjay Nayyar (IM Group) |
| Willfried Färber (Trivadis) | Marcel Franke (pmOne) |
| Dae Seong Han | Thomas Ivarsson (Sigma AB) |
| Anne Zorner | John Desch |
| Andrea Uggetti | Didier Simon |
| Mike Vovchik | Marius Dumitru |

**Published:** June 2011

**Applies to:** SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2

**Summary:** This white paper describes how operations engineers can test, monitor, capacity plan, and troubleshoot Microsoft SQL Server Analysis Services OLAP solutions in SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2.

Copyright

This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 Microsoft. All rights reserved.

Microsoft, BitLocker, Excel, Hyper-V, MSDN, SQL Server, StreamInsight, Visual Studio, Windows, Windows Server, and Windows Vista are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Contents

[1 Introduction 6](#_Toc294178687)

[2 Configuring the Server 7](#_Toc294178688)

[2.1 Operating System 7](#_Toc294178689)

[2.2 msmdsrv.ini 8](#_Toc294178690)

[2.2.1 A Warning About Configuration Errors 9](#_Toc294178691)

[2.3 Memory Configuration 13](#_Toc294178692)

[2.3.1 Understanding Memory Usage 13](#_Toc294178693)

[2.3.2 Memory Setting for the Analysis Service Process 14](#_Toc294178694)

[2.4 Thread Pool and CPU Configuration 18](#_Toc294178695)

[2.4.1 Thread Pool Sizes 18](#_Toc294178696)

[2.4.2 CoordinatorExecutionMode and CoordinatorQueryMaxThreads 19](#_Toc294178697)

[2.4.3 Multi-User Process Pool Settings 20](#_Toc294178698)

[2.4.4 Hyperthreading 22](#_Toc294178699)

[2.5 Partitioning, Storage Provisioning and Striping 23](#_Toc294178700)

[2.5.1 I/O Pattern 25](#_Toc294178701)

[2.6 Network Configuration 26](#_Toc294178702)

[2.6.1 Use Shared Memory for Local SQL Server Data Sources 26](#_Toc294178703)

[2.6.2 High-Speed Networking Features 29](#_Toc294178704)

[2.6.3 Hyper-V and Networking 31](#_Toc294178705)

[2.6.4 Increase Network Packet Size 32](#_Toc294178706)

[2.6.5 Using Multiple NICs 32](#_Toc294178707)

[2.7 Disabling Flight Recorder 33](#_Toc294178708)

[3 Monitoring and Tuning the Server 34](#_Toc294178709)

[3.1 Performance Monitor 34](#_Toc294178710)

[3.2 Dynamic Management Views 35](#_Toc294178711)

[3.3 Profiler Traces 37](#_Toc294178712)

[3.4 Data Collection Checklist 37](#_Toc294178713)

[3.5 Monitoring and Configuring Memory 37](#_Toc294178714)

[3.6 Monitoring and Tuning I/O and File System Caches 39](#_Toc294178715)

[3.6.1 System Cache vs. Physical I/O 41](#_Toc294178716)

[3.6.2 TempDir Folder 42](#_Toc294178717)

[3.7 Monitoring the Network 43](#_Toc294178718)

[4 Testing Analysis Services Cubes 44](#_Toc294178719)

[4.1 Testing Goals 44](#_Toc294178720)

[4.2 Test Scenarios 47](#_Toc294178721)

[4.3 Load Generation 48](#_Toc294178722)

[4.4 Clearing Caches 49](#_Toc294178723)

[4.5 Actions During Testing for Operations Management 50](#_Toc294178724)

[4.6 Actions After Testing 50](#_Toc294178725)

[5 Security and Auditing 51](#_Toc294178726)

[5.1 Firewalling the Server 51](#_Toc294178727)

[5.2 Encrypting Transmissions on the Network 53](#_Toc294178728)

[5.3 Encrypting Cube Files 54](#_Toc294178729)

[5.4 Securing the Service Account and Segregation of Duties 54](#_Toc294178730)

[5.5 File System Permissions 55](#_Toc294178731)

[6 High Availability and Disaster Recovery 55](#_Toc294178732)

[6.1 Backup/Restore 55](#_Toc294178733)

[6.1.1 Synchronization as Backup 56](#_Toc294178734)

[6.1.2 Robocopy 57](#_Toc294178735)

[6.2 Clustered Servers 58](#_Toc294178736)

[7 Diagnosing and Optimizing 58](#_Toc294178737)

[7.1 Tuning Processing Data 58](#_Toc294178738)

[7.1.1 Optimizing the Relational Engine 59](#_Toc294178739)

[7.2 Tuning Process Index 68](#_Toc294178740)

[7.2.1 CoordinatorBuildMaxThreads 68](#_Toc294178741)

[7.2.2 AggregationMemoryMin and Max 69](#_Toc294178742)

[7.3 Tuning Queries 69](#_Toc294178743)

[7.3.1 Thread Pool Tuning 69](#_Toc294178744)

[7.3.2 Aggregations 71](#_Toc294178745)

[7.3.3 Optimizing Dimensions 72](#_Toc294178746)

[7.3.4 Calculation Script Changes 73](#_Toc294178747)

[7.3.5 Repartitioning 73](#_Toc294178748)

[7.4 Locking and Blocking 76](#_Toc294178749)

[7.4.1 Lock Types 76](#_Toc294178750)

[7.4.2 Server Locks 78](#_Toc294178751)

[7.4.3 Lock Fundamentals 79](#_Toc294178752)

[7.4.4 Locks by Operation 80](#_Toc294178753)

[7.4.5 Investigating Locking and Blocking on the Server 84](#_Toc294178754)

[7.4.6 Deadlocks 87](#_Toc294178755)

[7.5 Scale Out 90](#_Toc294178756)

[7.5.1 Provisioning in a Scaled Out Environment 90](#_Toc294178757)

[7.5.2 Scale-out Processing Architectures 90](#_Toc294178758)

[7.5.3 Query Load Balancing 94](#_Toc294178759)

[7.5.4 ROLAP Scale Out 96](#_Toc294178760)

[8 Server Maintenance 97](#_Toc294178761)

[8.1 Clearing Log Files and Dumps 97](#_Toc294178762)

[8.2 Windows Event Log 97](#_Toc294178763)

[8.3 Defragmenting the File System 98](#_Toc294178764)

[8.4 Running Disk Checks 98](#_Toc294178765)

[9 Special Considerations 100](#_Toc294178766)

[9.1 ROLAP and Real Time 100](#_Toc294178767)

[9.1.1 Cache Coherency 100](#_Toc294178768)

[9.1.2 Manually Set Partition Slicers on ROLAP Partitions 101](#_Toc294178769)

[9.1.3 UDM Design 101](#_Toc294178770)

[9.1.4 Dimension Processing 103](#_Toc294178771)

[9.2 Distinct Count 105](#_Toc294178772)

[9.2.1 Partitioning for Distinct Count 105](#_Toc294178773)

[9.2.2 Optimizing Relational Indexes for Distinct Count 106](#_Toc294178774)

[9.3 Many-to-Many Dimensions 107](#_Toc294178775)

[10 Conclusion 108](#_Toc294178776)

# Introduction

In this guide you will find information on how to test and run Microsoft SQL Server Analysis Services in SQL Server 2005, SQL Server 2008, and SQL Server 2008 R2 in a production environment. The focus of this guide is how you can test, monitor, diagnose, and remove production issues on even the largest scaled cubes. This paper also provides guidance on how to configure the server for best possible performance.

Analysis Services cubes are a very powerful tool in the hands of the business intelligence (BI) developer. They provide an easy way to expose even large data models directly to business users. Unlike traditional, static reporting, where the query workload is known in advance, cubes support ad-hoc queries. Typically, such queries are generated by the Microsoft Excel spreadsheet software – without the business user being aware of the intricacies of the query engine. Because cubes allow such great freedom for users, the power they give to developers comes with responsibility. It is in the interaction between development and operations that the business value of the cubes is realized and where the proper steps can be taken to ensure that the power of the cubes is used responsibly.

It is the goal of this guide to make your operations processes as painless as possible, and to have you run with the best possible performance without any additional development effort to your deployed cubes. In this guide, you will learn how to get the best out of your existing data model by making changes transparent to the data model and by making configuration changes that improve the user experience of the cube.

However, no amount of operational readiness can cure a poorly designed cube. Although this guide shows you where you can make changes transparent to end users, it is important to be aware that there are cases where design change is the only viable path to good performance and reliability. Cubes do not take away the ubiquitous need for informed data modeling. Fortunately, this operations guide has a companion volume targeted at developers: the Analysis Services Performance Guide. We highly recommend that your developers read that white paper and follow the guidance in it.

Cubes do not exist in isolation – they rely on relational data sources to build their data structures. Although a full treatment of good relational data warehouse modeling out of scope for this document, it still provides some pointers on how to tune the database sources feeding the cube. Relational engines vary in their functionality; this paper focuses on guidance for SQL Server data sources. Much of the information here should apply equally to other engines and your DBA should be able to transfer the guidance here to other database systems you run.

In every IT project, preproduction testing is a crucial part of the development and deployment cycle. Even with the most careful design, testing will still be able to shake out errors and avoid production issues. Designing and running a test run of an enterprise cube is time well invested. Hence, this guide includes a description of the test methods available to you.

# Configuring the Server

Installing Analysis Services is relatively straightforward. However, applying certain post-installation configuration options can improve performance and scalability of the cube. This section introduces these settings and provides guidance on how to configure them.

## Operating System

Because Analysis Services uses the file system cache API, it partially relies on code in the Windows Server operating system for performance of its caches. Small changes are made to the file system cache in most versions of Windows Server, and this can have an effect on Analysis Services performance. We have found that patches related to the following software should be applied for best performance.

**Windows Server 2008:**

* [KB 961719](http://support.microsoft.com/kb/961719) – Applications that perform asynchronous cached I/O read requests and that use a disk array that has multiple spindles may encounter a low performance issue in Windows Server 2008, in Windows Essential Business Server 2008, or in Windows Vista SP1

**Windows Server 2008 R2:**

* [KB 979149](http://support.microsoft.com/kb/979149) – A computer that is running Windows 7 or Windows Server 2008 R2 becomes unresponsive when you run a large application
* [KB 976700](http://support.microsoft.com/kb/976700) – An application stops responding, experiences low performance, or experiences high privileged CPU usage if many large I/O operations are performed in Windows 7 or Windows Server 2008 R2
* [KB 982383](http://support.microsoft.com/kb/982383) – You encounter a decrease in I/O performance under a heavy disk I/O load on a Windows Server 2008 R2-based or Windows 7-based computer

**Kerberos-enabled systems:**

For Kerberos-enabled systems, you may need the following additional patches:

* [http://support.microsoft.com/kb/969083/](http://support.microsoft.com/kb/969083/en-us/)
* <http://support.microsoft.com/kb/2285736/>

**References:**

* Configure Monitoring Server for Kerberos delegation - <http://technet.microsoft.com/en-us/library/bb838742(office.12).aspx>
* How to configure SQL Server 2008 Analysis Services and SQL Server 2005 Analysis Services to use Kerberos authentication - <http://support.microsoft.com/kb/917409>
* Manage Kerberos Authentication Issues in a Reporting Services Environment <http://msdn.microsoft.com/en-us/library/ff679930.aspx>

## msmdsrv.ini

You can control the behavior of the Analysis Services engine to a great degree from the **msmdsrv.ini** file. This XML file is found in the **<instance dir>/OLAP/Config** folder. Many of the settings in **msmdsrv.ini** should not be changed without the assistance of Microsoft Product Support, but there are still a large number of configuration options that you can control yourself. It is the aim of this guide to provide you wth the guidance you need to properly configure these settings. The following table provides an overview of the settings available to you and can serve as starting point for exploring reconfiguration options.

|  |  |  |
| --- | --- | --- |
| Setting | Used for | Described in |
| <MemoryHeapType>  <HeapTypeForObjects> | Increasing throughput of high concurrency system | Section 2.3.2.4 |
| <HardMemoryLimit> | Preventing Analysis Services from consuming too much memory | Section 2.3.2 |
| <LowMemoryLimt> | Reserving memory for the Analysis Services process | Section 2.3.2 |
| <PreAllocate> | Locking Analysis Services memory allocation, and improving performance on Windows Server 2003 and potentially Windows Server 2008 | Section 2.3.2.1 |
| <TotalMemoryLimit> | Controlling when Analysis Services starts trimming working set | Section 2.3.2 |
| <CoordinatorExecutionMode>  <CoordinatorQueryMaxThreads> | Setting concurrency of queries in thread pool during query execution | Section 2.4.2 |
| <CoordinatorBuildMaxThreads> | Increasing processing speeds | Section 7.2.1 |
| <AggregationMemoryMin>  <AggregationMemoryMax> | Increasing parallelism of partition processing | Section 7.2.2 |
| <CoordinatorQueryBalancingFactor>  <CoordinatorQueryBoostPriorityLevel> | Preventing single users from monopolizing the system | Section 2.4.3 |
| <LimitSystemFileCacheSizeMB> | Controlling the size of the file system cache | Section 2.3.2 |
| <ThreadPool>  (and subsections) | Increasing thread pools for high concurrency systems | Section 2.4 |
| <BufferMemoryLimit>  <BufferRecordLimit> | Increasing compression during processing, but consuming more memory | Section 2.3.2.3 |
| <DataStorePageSize>  <DataStoreHashPageSize> | Increasing concurrency on a large machine | Section 2.3.2.4 |
| <MinIdleSessionTimeout>  <MaxIdleSessionTimeout>  <IdleOrphanSessionTimeout>  <IdleConnectionTimeout> | Cleaning up idle sessions faster on systems that have many connect/disconnects | Section 2.3.2.5 |
| <Port> | Changing the port that Analysis Services is listening on | Section 5.1 |
| <AllowedBrowsingFolders> | Controlling which folders are viewable by the server administrator | Section 2.5 and Section 5.5 |
| <TempDir> | Controlling where disk spills go | Section 2.5 |
| <BuiltinAdminsAreServerAdmins> | Controlling segregation-of-duties scenarios | Section 5 |
| <ServiceAccountIsServerAdmin> | Controlling segregation-of-duties scenarios | Section 5 |
| <ForceCommitTimeout>  <CommitTimeout> | Killing queries that are blocking a processing operation or killing the processing operation | Section 7.4.4.4 |

### A Warning About Configuration Errors

One of the most common mistakes encountered when tuning Analysis Services is misconfiguration of the **msmdsrv.ini** file. Because of this, changing your **msmdsrv.ini** file settings should be done with extreme care. If you inherit the managing responsibilities of an existing Analysis Services server, one of the first things you should do is to run the **Windif.exe** utility, which comes with Windows Server, against the current **msmdsrv.ini** file with the default **msmdsrv.ini** file. Also, if you upgraded to SQL Server 2008 or SQL Server 2008 R2 from SQL Server 2005, you will likely want to compare the current **msmdsrv.ini** file settings with the default settings for SQL Server 2008 or SQL Server 2008 R2. As described in the Thread Pool section, for optimization, some settings were changed in SQL Server 2008 and SQL Server 2008 R2 from their default values in SQL Server 2005.

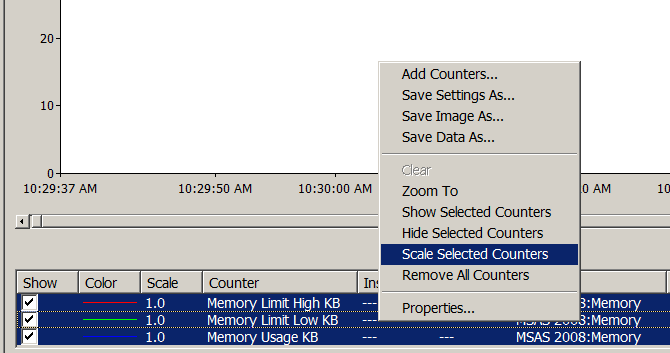
Take the following example from a customer configuration. The customer was experiencing extremely poor query/processing performance. When we looked at the memory counters, we saw this. 

Figure 1 - Scaling Counters

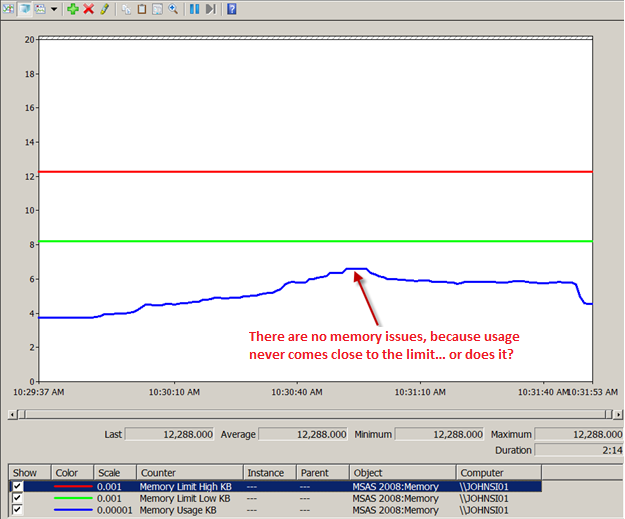


Figure 2 - Memory measurements with scaled counters

At first glance, this screenshot seems to indicate that everything is OK; there is no memory issue. However, in this case, because the customer used the **Scale Selected Counters** option in Windows Performance Monitor incorrectly, it was not possible to compare the three counters. A closer look, after the scale of the counters was corrected, shows this.

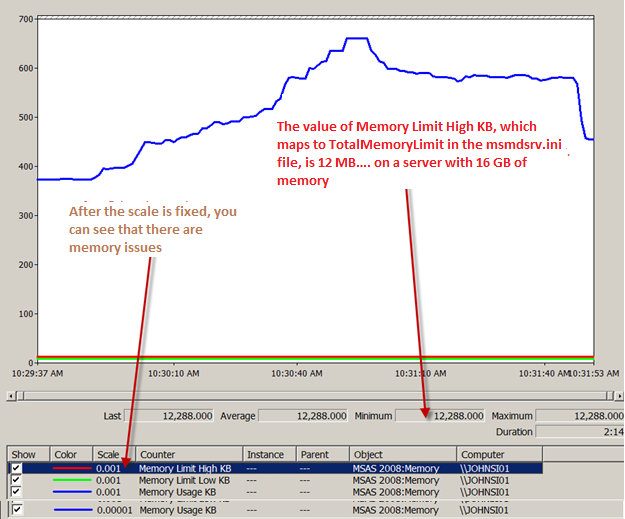


Figure 3 - Properly scaled counters

Now the fact that there is a serious memory problem is clear. Take a look at the actual **Memory Limit High KB** value. In a Performance Monitor log this and **Memory Limit Low KB** will always be constant values that reflect the value of **TotalMemoryLimit** and **LowMemoryLimit** respectively in the **msmdsrv.ini** file. So it appears that someone has modified the **TotalMemoryLimit** from the default 80 percent to an absolute value of 12,288 *bytes*, probably thinking the setting was in megabytes when in reality, the setting is in bytes. As you can see, the results of an incorrectly configured .ini file setting can be disastrous.

The moral of this story is this: Always be extremely careful when you modify your **msmdsrv.ini** file settings. One of the first things the Microsoft Customer Service and Support team does when working on an Analysis Services issue is grab the **msmdsrv.ini** file and compare it with the default .ini file for the customer’s version of Analysis Services. There are numerous file comparison tools you can use, the most obvious being **Windiff.exe**, which comes with Windows Server.

References:

* How to Use the Windiff.exe Utility - <http://support.microsoft.com/kb/159214>

## Memory Configuration

This section describes the memory model of Analysis Services and provides guidance on how to initially configure memory settings for a server. As will become clear in the Monitoring and Tuning section, it is often a good idea to readjust the memory settings as you learn more about the workload that is running on your server.

To understand the tradeoffs you will make during server configuration, it is useful to understand a bit more about how Analysis Services uses memory.

### Understanding Memory Usage

Apart from the executable itself, Analysis Services has several data structures that consume physical memory, including caches that boost the performance of user queries. Memory is allocated using the standard Windows memory API (with the exception of a special case described later) and is part of the Msmdsrv.exe process private bytes and working set. Analysis Services does not use the AWE API to allocate memory, which means that on 32-bit systems you are restricted to either 2 GB or 3 GB of memory for Analysis Services. If you need more memory, we recommend that you move to a 64-bit system. The total memory used by Analysis Services can be monitored in Performance Monitor using either: **MSOLAP:Memory\Memory Usage Kb** or **Process\Private Bytes – msmdsrv.exe.**

Memory allocated by Analysis Services falls into two broad categories: shrinkable and non-shrinkable memory.

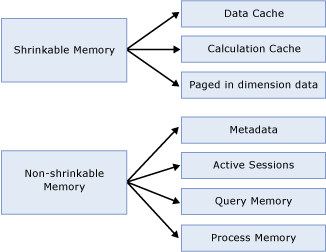


Figure 4 - Memory Structures

*Non-shrinkable memory* includes all the structures that are required to keep the server running: active user sessions, working memory of running queries, metadata about the objects on the server, and the process itself. Analysis Services does not release memory in the non-shrinkable memory spaces – though this memory can see still be paged out by the operating system (but see the section on PreAllocate).

*Shrinkable memory* includes all the caches that gradually build up to increase performance: The formula engine has a calculation cache that tries to keep frequently accessed and calculated cells in memory. Dimensions that are accessed by users are also kept in cache, to speed up access to dimensions, attributes, and hierarchies. The storage engine also caches certain accessed subcubes. (Not all subcubes are cached. For example, the ones used by arbitrary shapes are not kept.) Analysis Services gradually trims its working set if the shrinkable memory caches are not used frequently. This means that the total memory usage of Analysis Services may fluctuate up and down, depending on the server state – this is expected behavior.

Outside of the Analysis Services process space, you have to consider the memory used by the operation system itself, other services, and the memory consumed by the file system cache. Ideally, you want to avoid paging important memory – and as we will see, this may require some configuration tweaks.

**References:**

* Russinovich, Mark and David Solomon: *Windows Internals, 5th edition*. <http://technet.microsoft.com/en-us/sysinternals/bb963901>
* Webb, Chris, Marco Russo, and Alberto Ferrari: *Expert Cube Development with Microsoft SQL Server 2008 Analysis Services* – Chapter 11
* Arbitrary Shapes in AS 2005 - <https://kejserbi.wordpress.com/2006/11/16/arbitrary-shapes-in-as-2005/>

### Memory Setting for the Analysis Service Process

The Analysis Services memory behavior can be controlled by using parameters available in the **msmdsrv**.**ini** file. Except for the **LimitFileSystemCacheMB** setting, all the memory settings behave like this:

* If the setting has a value below 100, the running value is that percent of total RAM on the machine.
* If the setting has a value above 100, the running value is that amount of KB.

For ease of use, we recommend that you use the percentage values. The total memory allocations of Analysis Services process are controlled by the following settings.

**LowMemoryLimit** – This is the amount of memory that Analysis Services will hold on to, without trimming its working set. After memory goes above this value, Analysis Services begins to slowly deallocate memory that is not being used. The configuration value can be read from the Performance Monitor counter: **MSOLAP:Memory\LowMemoryLimit.** Analysis Services does not allocate this memory at startup; however, after the memory is allocated, Analysis Services does not release it.

**PreAllocate** – This optional parameter (with a default of 0) enables you to preallocate memory at service startup. It is described in more detail later in this guide. **PreAllocate** should be set to a value less than or equal to **LowMemoryLimit**.

**TotalMemoryLimit** –When Analysis Services exceeds this memory value, it starts deallocating memory aggressively from shrinkable memory and trimming its working set. Note that this setting is not an upper memory limit for the service; if a large query consumes a lot of resources, the service can still consume memory above this value. This value can also be read from the Performance Monitor counter: **MSOLAP:Memory\TotalMemoryLimit**.

**HardMemoryLimit** – This setting is only available in SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services – not SQL Server 2005 Analysis Services. It is a more aggressive version of **TotalMemoryLimit**. If Analysis Services goes above **HardMemoryLimit**, user sessions are selectively killed to force memory consumption down.

**LimitSystemFileCacheMB** – The Windows file system cache (described later) is actively used by Analysis Services to store frequently used blocks on disk. For some scan-intensive workloads, the file system cache can grow so large that Analysis Services is forced to trim its working set. To avoid this, Analysis Services exposes the Windows API to limit the size of the file system cache with a configuration setting in Msmdsrv.ini. If you choose to use this setting, note that it limits the file system cache for the entire server, not just for the Analysis Services files. This also means that if you run more than one instance of Analysis Services on the server, you should use the same value for **LimitSystemFileCacheMB** for each instance.

#### PreAllocate and Locked Pages

The **PreAllocate** setting found in **msmdsrv.ini** can be used to reserve virtual and/or physical memory for Analysis Services. For installations where Analysis Services coexists with other services on the same machine, setting **PreAllocate** can provide a more stable memory configuration and system throughput.

Note that if the service account used to run Analysis Services also has the **Lock pages in Memory** privilege, **PreAllocate** causes Analysis Services to use large memory pages. Large memory pages are more efficient on a big memory system, but they take longer to allocate. **Lock pages in Memory** is set using Gpedit.msc. Bear in mind that large memory pages cannot be swapped out to the page file. While this can be an advantage from a performance perspective, a high number of allocated large pages can cause the system to become unresponsive.

**Important:** **PreAllocate** has the largest impact on the Windows Server 2003 operating system. With the introduction of Windows Server 2008, memory management has been much improved. We have been testing this setting on Windows Server 2008, but have not measured any significant performance benefits of using **PreAllocate** on this platform. However, you may want still want to make use of the locked pages functionality in Window Server 2008.

To learn more about the effects of **PreAllocate**, see the following technical note:

* Running Microsoft SQL Server 2008 Analysis Services on Windows Server 2008 vs. Windows Server 2003 and Memory Preallocation: Lessons Learned - (<http://sqlcat.com/technicalnotes/archive/2008/07/16/running-microsoft-sql-server-2008-analysis-services-on-windows-server-2008-vs-windows-server-2003-and-memory-preallocation-lessons-learned.aspx>)

**References:**

* How to: Enable the Lock Pages in Memory Option (Windows) - <http://msdn.microsoft.com/en-us/library/ms190730.aspx>

#### AggregationMemory Max/Min

Under the server properties of SQL Server Management Studio and in **msmdsrv.ini**, you will find the following settings:

* **OLAP\Process\AggregationMemoryLimitMin**
* **OLAP\Process\AggregationMemoryLimitMax**

These two settings determine how much memory is allocated for the creation of aggregations and indexes in each partition. When Analysis Services starts partition processing, parallelism is throttled based on the **AggregationMemoryMin/Max** setting. The setting is per partition. For example, if you start five concurrent partition processing jobs with **AggregationMemoryMin** = 10, an estimated 50 percent (5 x 10%) of reserved memory is allocated for processing. If memory runs out, new partition processing jobs are blocked while they wait for memory to become available. On a large memory system, allocating 10 percent of available memory per partition may be too much. In addition, Analysis Services may sometimes misestimate the maximum memory required for the creation of aggregates and indexes. If you process many partitions in parallel on a large memory system, lowering the value of **AggregationMemoryLimitMin** and **AggregationMemoryMax** may increase processing speed. This works because you can drive a higher degree of parallelism during the process index phase.

Like the other Analysis Services memory settings, if this setting has a value greater than 100 it is interpreted as a fixed amount of kilobytes, and if is lower than 100, it is interpreted as a percentage of the memory available to Analysis Services. For machines with large amounts of memory and many partitions, using an absolute kilobyte value for these settings may provide a better control of memory than using a percentage value.

#### BufferMemoryLimit and BufferRecordLimit

**OLAP\Process\BufferMemoryLimit** determines the size of the fact data buffers used during partition data processing. While the default value of the **OLAP\Process\BufferMemoryLimit** is sufficient for most deployments, you may find it useful to alter the property in the following scenario.

If the granularity of your measure group is more summarized than the relational source fact table, you may want to consider increasing the size of the buffers to facilitate data grouping. For example, if the source data has a granularity of day and the measure group has a granularity of month; Analysis Services must group the daily data by month before writing to disk. This grouping occurs within a single buffer and it is flushed to disk after it is full. By increasing the size of the buffer, you decrease the number of times that the buffers are swapped to disk. Because the increased buffer size supports a higher compression ratio, the size of the fact data on disk is decreased, which provides higher performance. However, be aware that high values for the **BufferMemoryLimit** use more memory. If memory runs out, parallelism is decreased.

You can use another configuration setting to control this behavior: **BufferRecordLimit**. This setting is expressed in received records from the data source instead of a **Memory %/Kb** size. The lower of the two takes precedence. For example, if **BufferMemoryLimit** is set to 10 percent of a 32-GB memory space and **BufferRecordLimit** is set to 10 million rows, either 3.2 GB or 10,000,000 times the rowsize is allocated for the merge buffer, whichever is smaller.

#### Heap Settings, DataStorePageSize, and DataStoreHashPageSize

During query execution, Analysis Services generally touches a lot of data and performs many memory allocations. Analysis Services has historically relied on its own heap implementation to achieve the best performance. However, since Windows Server 2003, advances in the Windows Server operating system mean that memory can now be managed more efficiently by the operating system. This turns out to be especially true for multi-user scenarios. Because of this, servers that have many users should generally apply the following changes to the **msmdsrv.ini** file.

|  |  |  |
| --- | --- | --- |
| Setting | Default | Multi-user, faster heap |
| <MemoryHeapType> | 1 | 2 |
| <HeapTypeForObjects> | 1 | 0 |

It is also possible to increase the page size that is used for managing the hash tables Analysis Services uses to look up values. Especially on modern hardware, we have seen the following change yield a significant increase in throughput during query execution.

|  |  |  |
| --- | --- | --- |
| Setting | Default | Bigger pages |
| <DataStorePageSize> | 8192 | 65536 |
| <DataStoreHashPageSize> | 8192 | 65536 |

**References:**

* **KB2004668** - You experience poor performance during indexing and aggregation operations when using SQL Server 2008 Analysis Services - <http://support.microsoft.com/kb/2004668>

#### Clean Up Idle Sessions

Client tools may not always clean up sessions opened on the server. The memory consumed by active sessions is non-shrinkable, and hence is not returned the Analysis Services or operating system process for other purposes. After a session has been idle for some time, Analysis Services considers the session expired and move the memory used to the shrinkable memory space. But the session still consumes memory until it is cleaned up.

Because idle sessions consume valuable memory, you may want to configure the server to be more aggressive about cleaning up idle sessions. There are some **msmdsrv.ini** settings that control how Analysis Services behaves with respect to idle sessions. Note that a value of zero for any of the following settings means that the sessions or connection is kept alive indefinitely.

|  |  |
| --- | --- |
| Setting | Description |
| <MinIdleSessionTimeOut> | This is the minimum amount of time a session is allowed to be idle before Analysis Services considers it ready to destroy. Sessions are destroyed only if there is memory pressure. |
| <MaxIdleSessionTimeout> | This is the time after which the server forcibly destroys the session, regardless of memory pressure. |
| <IdleOrphanSessionTimeout> | This is the timeout that controls sessions that no longer have a connection associated with them. Examples of these are users running a query and then disconnecting from the server. |
| <IdleConnectionTimeout> | This timeout controls how long a connection can be kept open and idle until Analysis Services destroys it. Note that if the connection has no active sessions, **MaxIdleSessionTimeout** eventually marks the session for cleaning and the connection is cleaned with it. |

If your server is under memory pressure and has many users connected, but few executing queries, you should consider lowering **MinIdleSessionTimeOut** and **IdleOrphanSessionTimeout** to clean up idle sessions faster.

## Thread Pool and CPU Configuration

Analysis Services uses thread pools to manage the threads used for queries and processing. The thread management subsystem that Analysis Services uses enables you to fine-tune the number of threads that are created. Tuning the thread pool is a balancing act between CPU overutilization and underutilization: If too many threads are created, unnecessary context switches and contention for system resources lower performance, and too few threads can lead to CPU and disk underutilization, which means that performance is not optimal on the hardware allocated to the process. When you tune your thread pools, it is essential that you benchmark your performance before and after your configuration changes; misconfiguration of the thread pool can often cause unforeseen performance issues.

### Thread Pool Sizes

This section discusses the settings that control thread pool sizes.

**ThreadPool\Process\MaxThreads** determines the maximum number of available threads to Analysis Services during processing and for accessing the I/O system during queries. On large, multiple-CPU machines, the default value in SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services of 64 or 10 multiplied by the number of CPU cores (whichever is higher) may be too low to take advantage of all CPU cores. In SQL Server 2005 Analysis Services the settings for process thread pool were set to the static value of 64. If you are running an installation of SQL Server 2005 Analysis Services, or if you upgraded from SQL Server 2005 Analysis Services to SQL Server 2008 Analysis Services or SQL Server 2008 R2 Analysis Services, it might be a good idea to increase the thread pool.

**ThreadPool\Query\MaxThreads** determines the maximum number of threads available to the Analysis Services formula engine for answering queries. In SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services, the default is 2 multiplied by the number of logical CPU or 10, whichever is higher. In SQL Server 2005 Analysis Services, the default value was fixed at 10. Again, if you are running on SQL Server 2005 Analysis Services or an upgraded SQL Server 2005 Analysis Services, you may want to use the new settings.

### CoordinatorExecutionMode and CoordinatorQueryMaxThreads

Analysis Services uses a centralized storage engine job architecture for both querying and processing operations. When a subcube request or processing command is executed, a coordinator thread is responsible for gathering the data needed to satisfy the request.

The value of **CoordinatorExecutionMode** limits the total number of coordinator jobs that can be executed in parallel by a subcube request in the storage engine. A negative value specifies the maximum number of parallel coordinator jobs that can start per processing core per subcube request. By default **CoordinatorExecutionMode** is set to -4, which means the server is limited to 4 jobs in parallel per core. For example, on a 16-core machine with the default values of **CoordinatorExecutionMode** = -4, a total of 64 concurrent threads can execute per subcube request.

When a subcube is requested, a coordinator thread starts up to satisfy the request. The coordinator first queues up one job for each partition that must be touched. Each of those jobs then continues to queue up more jobs, depending on the total number of segments that must be scanned in the partition. The value of **CoordinatorQueryMaxThreads**, with a default of 16, limits the number of partition jobs that can be executed in parallel for each partition. For example, if the formula engine requests a subcube that requires two partitions to be scanned, the storage engine is limited to a maximum of 32 threads for scanning. Also note that both the coordinator jobs and the scan threads are limited by the maximum number of threads configured in **ThreadPool/Processing/MaxThreads.** The following diagram illustrates the relationship between the different thread pools and the coordinator settings.

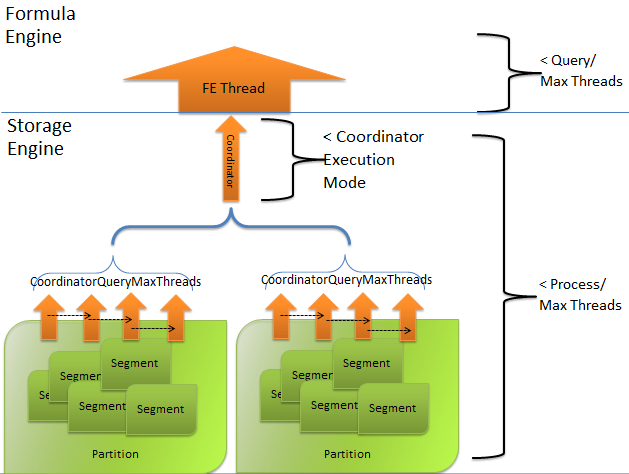


Figure 5 - Coordinator Queries

If you increase the processing thread pool, you should make sure that the **CoordinatorExecutionMode** settings, as well as the **CoordinatorQueryMaxThreads** settings, have values that enable you to make full use of the thread pool.

If the typical query in your system touches many partitions, you should be careful with the **CoordinatorQueryMaxThreads**. For example, if every query touches 10 partitions, a total 160 threads can be used just to answer that query. It will not take many queries to run the thread pool dry under those conditions. In such a case, consider lowering the setting of **CoordinatorQueryMaxThreads**.

### Multi-User Process Pool Settings

In multi-user scenarios, long-running queries can starve other queries; specifically they can consume all available threads, blocking execution of other queries until the longer-running queries complete.

You can reduce the aggressiveness of how each coordinator job allocates threads per segment by modifying **CoordinatorQueryBalancingFactor** and **CoordinatorQueryBoostPriorityLevel** as follows.

|  |  |  |
| --- | --- | --- |
| Setting | Default | Multi-user nonblocking settings |
| CoordinatorQueryBalancingFactor | -1 | 1 |
| CoordinatorQueryBoostPriorityLevel | 3 | 0 |

If you want to understand exactly what these settings do, you need to know a little about the internals of Analysis Services. First, a word of warning: The remainder of this section looks at Analysis Services at a very detailed level. It is perfectly acceptable to take a query workload and test with the default settings and then with the multi-user settings to decide if it is worth making these changes.

With the disclaimer out of the way, let’s look at an example to explain this behavior. On a 45-core Windows Server 2008 server with default .ini file settings, you have a long running query that appears to be blocking many of the other queries being executed by other users. Behind the scenes in Analysis Services, multiple segment jobs (different from coordinator jobs) are created to query the respective segments. A segment of data in Analysis Services is composed of roughly 64,000 records, which are subdivided into pages. There are 256 pages in a segment and 256 records in a page. These records are brought into memory in chunks upon request by queries. Analysis Services determines which pages need to be scanned to retrieve the relevant records for the data requested. These jobs are chained, meaning Job 1 queues Job 2 to the thread pool, Job 1 performs its own job, Job 2 queues Job 3 to the thread pool, Job 2 performs its own job, and so on. Each job has its own thread, or *segment job*.

In our example, using the default settings the first query fires off 720 jobs, scanning a lot of data. The second query fires off 1 job. The first 720 jobs fire off their own 720 jobs, using up all of the threads. This prevents the second query from executing, because no threads are available. This behavior causes blocking of the second and subsequent queries that need threads from the process pool. The multi-user settings (**CoordinatorQueryBalancingFactor**=1, **CoordinatorQueryBoostPriorityLevel**=0) prevent all of the threads from being allocated so the second and third queries can execute their jobs.

Again, each segment job uses one thread. If the long-running query requires scanning of multiple segments, Analysis Services creates the necessary number of threads. In single-user mode, the first query fires off 720 jobs, and the second query fires off 1 job. Each segment job is immediately queued up before the prior segment job begins scanning data. The first 720 jobs fire off their own 720 jobs, preventing second query from executing. In multi-user mode, not all threads are allocated, allowing second query (and third) to execute their jobs.

Be careful modifying these settings; although these settings reduce or stop the blocking of shorter-running queries by longer-running ones, it may slow the response times of individual queries. (In the figure, SSAS stands for SQL Server Analysis Services.)

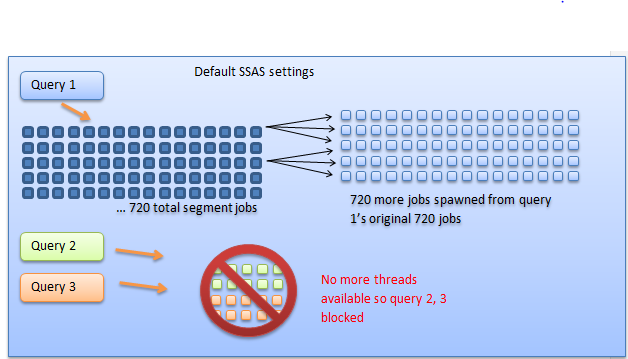


Figure 6 - Default settings

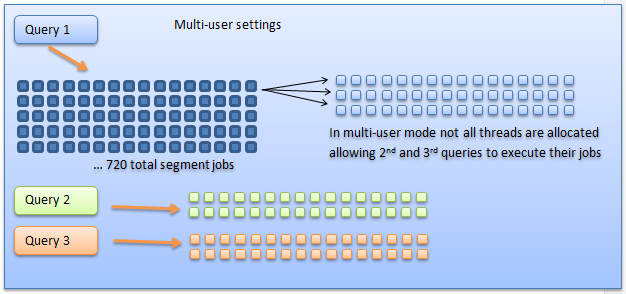


Figure 7 - Multi-user settings

### Hyperthreading

We receive numerous questions from customers around hyperthreading. The Analysis Services development team has no official recommendation on hyperthreading; it is included in this white paper only because our customers ask about it frequently. With that said, we have made the following observations in installations in which hyperthreading is used with Analysis Services:

* If the load on your server is more CPU-bound, turning on hyperthreading offers no improvement, in our experience.
* If the load on your server is more I/O-bound, there may be some benefit to turning on hyperthreading.
* Processors are constantly changing, and many characteristics of behavior and performance related to hyperthreading are going to be specific to the processor.

## Partitioning, Storage Provisioning and Striping

When you configure the storage of a server you are often presented with a series of LUNs for data storage. These are provisioned from either your SAN, internal drives, or NAND memory. NAND memory, in some configuration also known as Solid State Disks/Devices (SSD) is typically a good fit for large cubes, because the latency and I/O pattern favored by these drives are a good match for the Analysis Services storage engine workload.

The question is how to make the best use of the storage for your Analysis Services instance.

Apart from performance and capacity, the following factors must also be considered when designing the disk volumes for Analysis Services:

* Is clustering of the Analysis Services instance required?
* Will you be building a scaled-out environment?

Consult the subsections that follow for guidance in these specific setups. However, the following general guidance applies.

**SAN Mega LUNs:** If you are using a SAN, your storage administrator may be able to provision you a single, large LUN for your cubes. Unless you plan to build a consolidation environment, having such a single, mega-LUN is probably the easiest and most manageable way to provision your storage. First of all, it provides the IOPS as a general resource to the server. Secondly, and additional advantage of a mega-LUN is that you can easily disk align this in Windows Server 2003. For Windows Server 2008 you do not need to worry about disk alignment on newly created volumes.

**Windows Server dynamic disk stripes:** Using Disk Manager it is possible to combine multiple LUNs into a single Windows volume. This is a very simple way to combine multiple, similarly sized, similarly performing LUNs into a single mount point or drive letter. We have tested dynamic disk stripes in Windows Server 2008 R2 all the way up to 100.000 IOPS – and the performance overhead to that level is negligible.

**Note:** You *cannot* use dynamic disk stripes in a cluster. This is discussed in greater detail later in this guide.

**Drive letters versus mount points:** Both drive letters and mount points will work with Analysis Services cubes. If the server is dedicated to a single Analysis Services instance, a drive letter may be the simplest solution. Choosing drive letters versus mount points is often a matter of personal preference, or it can be driven by the standards of your operations team. From the perspective of performance, one is not superior to the other.

**AllowedBrowsingFolders:** Remember that in order for a directory to be visible to administrators of Analysis Services in SQL Server Management Studio, it must be listed in **AllowedBrowsingFolders**, which is available in SQL Server Management Studio by clicking **Server Properties** and then **Advanced Properties**.

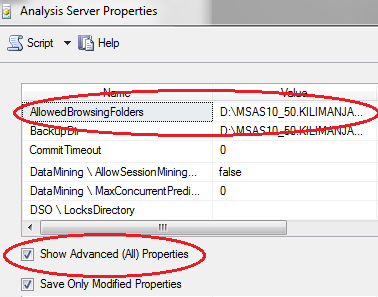


Figure 8 – AllowedBrowsingFolders

The Analysis Services account must also have permission to both read and write to these folders. Just adding them to the **AllowedBrowsingFolders** list is not enough – you must also assign file system permissions.

**MFT versus GPT disk:** If you expect your cubes to be larger than 2 terabytes, you should use a GPT disk. The default, an MFT disk, only allows 2-terabyte partitions.

**TempDir Folder:** The **TempDir** folder, configured in the Advanced Properties of the server, should be moved to the fastest volume you have. This may mean you will share TempDir with cube data, which is fine if you plan capacity accordingly. For more information, see the section about the **TempDir** folder.

**NTFS Allocation Unit Size (AUS):** With careful optimization, it is sometimes possible to get slightly better performance by smartly choosing between 32K or 64K (a few percent). But, unless you are hunting for benchmark performance, just go with 32K. If you have standardized on 64K for other SQL Server services, that will work too.

**Don’t suboptimize storage:** There are a few cases where it makes sense to split your data into multiple volumes – for example if you have different storage types attached to the server (such as NAND for latest data and SATA drives for historical data). However, it is generally *not* a good idea to suboptimize the storage layout of Analysis Services by creating complicated data distributions that span different storage types. The rule of thumb for optimal disk layout is to utilize all disk drives for all cube partitions. Create large pools of disk, presented as single, large volumes.

**Exclude Analysis Services folders from virus scanners:** If you are running a virus scanner on the server, make sure the **Data** folder, **TempDir**, and backup folders are not being scanned or touched by the filter drivers of the antivirus tool. There are no executable files in these Analysis Services folders, and enabling virus scanners on the folder may slow down the disk access speeds significantly.

**Consider Defragmenting the Data Folder:** Analysis Services cubes get very fragmented over time. We have seen cases where defragmenting cube data folders, using the standard disk defragment utility, yielded a substantial performance benefit. Note that defragmenting a drive also has impact on the performance of that drive as it moves the files around. If you choose to defragment an Analysis Services drive, do so in batch window where the service can be taken offline while the defragmentation happens, or plan disk speeds accordingly to make sure the performance impact is acceptable.

**References:**

* White paper: Configuring Dynamic Volumes - <http://technet.microsoft.com/en-us/library/bb727122.aspx>

### I/O Pattern

Because Analysis Services uses bitmap indexes to quickly locate fact data, the I/O generated is mostly random reads. I/O sizes will typically average around 32-KB block sizes.

As with all SQL Server databases systems, we recommend that you test your I/O subsystem before deploying the database. This allows you to measure how close the production system is to your maximum potential throughput. Not running preproduction I/O testing of an Analysis Services server is the equivalent of not knowing how much memory or how many CPU cores your server has.

Due to the threading architecture of the storage engine, the I/O pattern will also be highly multi threaded. Because the NTFS file system cache issues synchronous I/O, each thread will have only one or two outstanding I/O requests. Incidentally, this means that cubes will often run very well on NAND storage that favor this type of I/O pattern.

The Analysis Services I/O pattern can be easily simulated and tested using SQLIO.exe. The following command-line parameter will give you a good indication of the expected performance:

**sqlio -b32 -frandom -o1 -s30 -t256 -kR <path of file>**

Make sure you run on a sufficiently large test file. For more information, refer to the links in the References section.

**References:**

* SQLIO download: <http://www.microsoft.com/downloads/en/details.aspx?familyid=9a8b005b-84e4-4f24-8d65-cb53442d9e19&displaylang=en>
* White paper: Analyzing I/O characteristics and sizing storage for SQL Server Database Applications – <http://sqlcat.com/whitepapers/archive/2010/05/10/analyzing-i-o-characteristics-and-sizing-storage-systems-for-sql-server-database-applications.aspx>
* Blog: The Memory Shell Game – <http://blogs.msdn.com/b/ntdebugging/archive/2007/10/10/the-memory-shell-game.aspx>

## Network Configuration

During the **ProcessData** phase of Analysis Services processing, rows are transferred from the relational database specified in the data source to Analysis Services. If your data source is on a remote server, the data should be retrieved over TCP/IP. It is important to make sure all networking components are configured to support the optimal throughput. If your Ethernet throughput is consistently close to 80 percent of your maximum capacity, adding more network capacity typically speeds up **ProcessData**.

Creating a high-speed network is fairly easy with the networking hardware available today. Specifically most servers come with a Gigabit NICs out of the box. [Infiniband](http://en.wikipedia.org/wiki/Infiniband) and 10-gigabit NICs are also available for even more throughput. With that said, your overall throughput can be limited by routers and other hardware in your network that don’t support the speed of your NIC. You can use the **Networking** tab in Task Manager to quickly determine your link speed. In the following screenshot you can see that the NIC and switch support a maximum of 1Gbps.



Figure 9 - Viewing NIC speed

If you have a 1-Gbps NIC, but only a 100-Mbps switch, Task Manager displays 100 Mbps. Depending on your network topology there may be more to determining your link speed than this, but using Task Manager is a simple way to get a rough idea of the configuration.

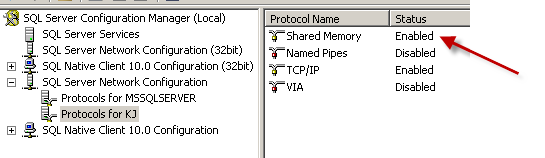
In addition to creating a high-speed network, there are some additional configurations you can change to further speed up network traffic.

### Use Shared Memory for Local SQL Server Data Sources

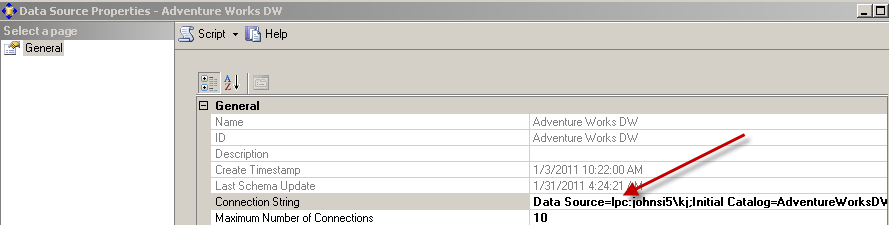
If Analysis Services is on the same physical machine as the data source, and the data source is SQL Server, you should make sure you are exchanging data over the shared memory protocol. Shared memory is much faster than TCP/IP, as it avoids the overhead of the translation layers in the network stack. Shared memory is only possible if the SQL Server database is on the same physical machine as Analysis Services. If you cannot get a high speed network set up in your organization, you can sometimes benefit from running SQL Server and Analysis Services on the same physical machine.

To modify your data source connection to specify shared memory:

* 1. First, make sure that the Shared Memory protocol is enabled in SQL Server Configuration Manager.



* 1. Next, add lpc: to your data source in the connection string to force shared memory.



* 1. After you start processing, you can verify your connection is using shared memory by executing the following SELECT statement.

SELECT session\_id, net\_transport, net\_packet\_size

FROM sys.dm\_exec\_connections

The **net\_transport** for the Analysis Services SPID should show: **Shared memory**.

For more information about shared memory connections, see “Creating a Valid Connection String Using Shared Memory Protocol” (<http://msdn.microsoft.com/en-us/library/ms187662.aspx>).

To compare TCP/IP and shared memory, we ran the following two processing commands.

<!--SQL Server Native Client with TCP-->

<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

<Parallel>

<Process …

<Object>

<DatabaseID>Adventure Works DW 2008R2</DatabaseID>

<CubeID>Adventure Works</CubeID>

</Object>

<Type>ProcessFull</Type>

<DataSource xsi:type="RelationalDataSource">

<ID>Adventure Works DW</ID>

<Name>Adventure Works DW</Name>

<ConnectionString>

Provider=SQLNCLI10.1;Data Source=tcp:johnsi5\kj;

Integrated Security=SSPI;Initial Catalog=AdventureWorksDW2008R2; </ConnectionString>

<ImpersonationInfo>

<ImpersonationMode>ImpersonateCurrentUser</ImpersonationMode>

</ImpersonationInfo>

<Timeout>PT0S</Timeout>

</DataSource>

</Process>

</Parallel>

</Batch>

<!--SQL Native Client with shared memory-->

<Batch xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

<Parallel>

<Process …

<Object>

<DatabaseID>Adventure Works DW 2008R2</DatabaseID>

<CubeID>Adventure Works</CubeID>

</Object>

<Type>ProcessFull</Type>

<DataSource xsi:type="RelationalDataSource">

<ID>Adventure Works DW</ID>

<Name>Adventure Works DW</Name>

<ConnectionString>

Provider=SQLNCLI10.1;Data Source=lpc:johnsi5\kj;

Integrated Security=SSPI;Initial Catalog=AdventureWorksDW2008R2; </ConnectionString>

<ImpersonationInfo>

<ImpersonationMode>ImpersonateCurrentUser</ImpersonationMode>

</ImpersonationInfo>

<Timeout>PT0S</Timeout>

</DataSource>

</Process>

</Parallel>

</Batch>

The first command using TCP/IP maxed out at 112,000 rows per second. Because the data source for the cube was on the same machine as Analysis Services, we were able to use shared memory in the second processing command and get much better throughput: 180,000 max rows/sec.



Figure 10 - Comparing rows/sec throughput

### High-Speed Networking Features

Windows Server 2008 R2 has numerous improvements in network virtualization support that enable enhanced networking support. Windows Server 2008 R2 has also enhanced the support of jumbo packets and TCP offloading that was introduced in Windows Server 2008. Additionally Virtual Machine Queue (VMQ) support was added to allow network routing and data copy processing to be offloaded to a physical NIC. These features were introduced to take advantage of the capabilities found in the 10GbE server NICs.

#### TCP Chimney

TCP Chimney is a networking technology that transfers TCP/IP protocol processing from the CPU to a network adapter during the network data transfer. There have been some issues with enabling TCP Chimney in the past, and for that reason many people recommended turning it off. The technology has matured and many of the issues reported were specific to the NIC manufacturer. Applications that have long-lived connections transferring a lot of data benefit the most from the TCP Chimney feature. Processing Analysis Services data from a remote server falls into this category, so we recommend that you make sure that TCP Chimney is enabled and configured correctly. TCP Chimney can be enabled and disabled in the operating system and in the advanced properties of the network adapter.

To enable TCP Chimney you need to perform the following:

* 1. Enable TCP chimney in the operating system using the **netsh** commands.
  2. Ensure the physical network adapter supports TCP Chimney offload, and then enable it for the adapter in the network driver.

TCP chimney has three modes of operation: Automatic (new in Windows Server 2008 R2), Enabled, and Disabled. The default mode in Windows Server 2008 R2, Automatic, checks to make sure the connections considered for offloading meet the following criteria:

* 10Gbps Ethernet NIC installed and connection established through 10GbE adapter
* Mean round trip link latency is less than 20 milliseconds
* Connection has exchanged at least 130 KB of data

To determine whether TCP Chimney is enabled, run the following from an elevated command prompt.

**netsh int tcp show global**

In the results, check the Chimney Offload State setting.

TCP Global Parameters

----------------------------------------------

Receive-Side Scaling State : enabled

Chimney Offload State : automatic

NetDMA State : enabled

Direct Cache Access (DCA) : disabled

Receive Window Auto-Tuning Level : normal

Add-On Congestion Control Provider : none

ECN Capability : disabled

RFC 1323 Timestamps : disabled

In this example, the results show that TCP Chimney offloading is set to automatic. This means that it is enabled as long as the requirements mentioned earlier are met.

After you verify that TCP Chimney is enabled at the operating system level, check the NIC settings in device manager:

1. Go to start | run and type **devmgmt.msc**.
2. In Device Manager, expand Network Adapters, and right-click the name of the physical NIC adapter, and then click **Properties**.
3. On the Advanced tab, under **Property**, click **TCP Chimney Offload** or **TCP Connection Offload**, and then under **Value**, verify that **Enabled** is displayed. You may need to do this for both IPv4 and IPv6. Note that **TCP Checksum Offload** is not the same as **TCP Chimney Offload**.

If the system has IPsec enabled, no TCP connections are offloaded, and TCP Chimney provides no benefit. There are numerous different configuration options for TCP Chimney offloading that our outside the scope of this white paper. See the references section for a deeper treatment.

**References:**

* Windows Server 2008 R2 Networking Deployment Guide: Deploying High-Speed Networking Features <http://download.microsoft.com/download/8/E/D/8EDE21BC-0E3B-4E14-AAEA-9E2B03917A09/HSN_Deployment_Guide.doc>
* Windows Server 2008 R2 Networking Deployment Guide <http://download.microsoft.com/download/8/E/D/8EDE21BC-0E3B-4E14-AAEA-9E2B03917A09/HSN_Deployment_Guide.doc>

#### Jumbo Frames

Jumbo frames are Ethernet frames with more than 1,500 and up to 9,000 bytes of payload. Jumbo frames have been shown to yield a significant throughput improvement during Analysis Services processing, specifically Process Data. Network throughput is increased while CPU usage is minimized. Jumbo frames are only available on gigabit networks, and all devices in the network path must support them (switches, routers, NICs, and so on).

Most default Ethernet set ups are configured to have MTU sizes of up to 1,500 bytes. Jumbo frames allow you to go up to 9,000 bytes in a single transfer. To enable jumbo frames:

1. Configure all routers to support jumbo frames.
2. Configure the NICs on the Analysis Services machine and the data source machine to support jumbo frames.
   * 1. Click the **Start** button, point to **Run**, and then type **ncpa.cpl**.
     2. Right-click your network connection, and then click **Properties**.
     3. Click **Configure,** and then click the **Advanced** tab. Under **Property**, click **Jumbo Frame**, and then under **Value**, change the value from **Disabled** to **9kb MTU** or **9014**, depending on the NIC.
     4. Click **OK** on all the dialog boxes. After you make the change, the NIC loses network connectivity for a few seconds and you should reboot.
3. After you configure jumbo frames, you can easily test the change by pinging the server with a large transfer:

Ping <servername> -f –l 9000

You should only measure one network packet per ping request in Network Monitor.

### Hyper-V and Networking

If you are running Analysis Services and SQL Server in a Hyper-V virtual machine, there are a few things you should be aware of specific to networking.

When you enable Hyper-V and create an external virtual network, both the guest and the host machine go through a virtual network adapter to connect to the physical network. All traffic goes through your virtual network adapter, and there is additional latency overhead associated with this.

### Increase Network Packet Size

Under the properties of your data source, increasing the network packet size for SQL Server minimizes the protocol overhead require to build many, small packages. The default value for SQL Server 2008 is 4096. With a data warehouse load, a packet size of 32K (in SQL Server, this means assigning the value 32767) can benefit processing. Don’t change the value in SQL Server using **sp\_configure**; instead override it in your data source. This can be set whether you are using TCP/IP or Shared Memory.

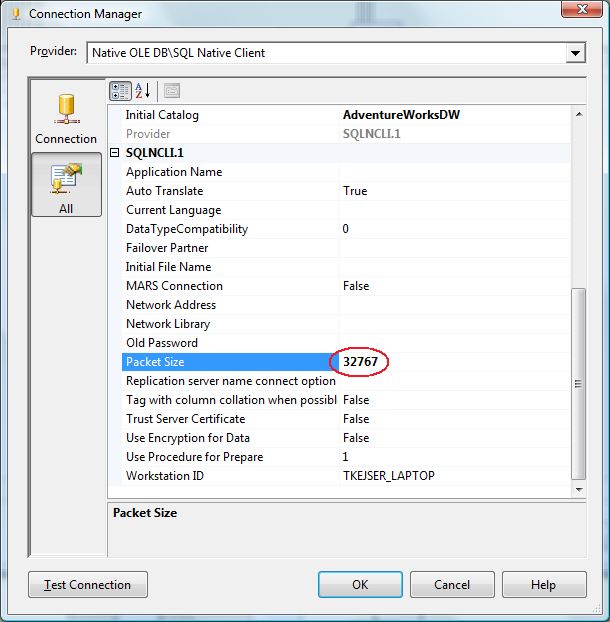


Figure 11 Tuning network packet size

### Using Multiple NICs

If you are hitting a network bottleneck when you process your cube, you can add additional NICs to increase throughput. There are two basic configurations for using multiple NICs on a server. The first and default option is to use the NICs separately. The second option is to use NIC teaming.

**Multiple NICs** can be used individually to concurrently run many multipartition processing commands. Each partition in Analysis Services can refer to a different data source, and it is this feature you make use of. To use multiple NICs, follow these steps:

1. Set up NICs with different IP numbers.
2. Add new data sources.
3. Change your partition setup to reference the new data sources.

First, set up multiple NICs in the source and the target, each with its own IP number. Match each NIC on the cube server with a corresponding NIC on the relational database. Set up the subnet mask so only the desired NIC on the source can be reached from its partner on the cube server. If you are limited in bandwidth on the switches or want to create a very simple setup, you can use this technique with a crossover cable, which we have done successfully

Second, add a data source for each NIC on the data source, and set up each data source to point to its own source NIC.

Third, configure partitions in the cube so that they are balanced across all data sources. For example, if you have 16 partitions and 4 data sources per NIC – point 4 partitions to each data source.

Schematically, the setup looks like this.

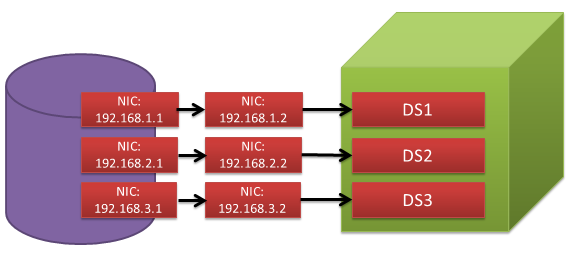


Figure 12 - Using multiple NICs for processing

**NIC teaming:** Another option is to team multiple NICs so they appear as one NIC to Windows Server. It is difficult to make specific recommendations around using teaming, because performance is specific to the hardware and drivers used for teaming. To determine whether NIC teaming would be beneficial to your processing workload, test with it both enabled and disabled, and measure your performance results with both settings.

## Disabling Flight Recorder

Flight Recorder provides a mechanism to record Analysis Services server activity into a short-term log. Flight Recorder provides a great deal of benefit when you are trying to troubleshoot specific querying and processing problems by logging snapshots of common DMV into the log file. However, it introduces an amount of I/O overheard on the server itself.

If you are in a production environment and you are already monitoring the Analysis Services instance using this guide, you do not need Flight Recorder capabilities and you can disable its logging and remove the I/O overhead. The server property that controls whether Flight Recorder is enabled is the **Log\Flight Recorder\Enabled** property. By default, this property is set to **true**.

# Monitoring and Tuning the Server

As part of healthy operations management, you must collect data that allows both reactive and proactive behaviors that increase system stability, performance, and integrity. The temptation is to collect a lot of data in the belief that with more knowledge comes better decisions. This is not always the case, because you may either overload the server with data collection overhead or collect data points that you cannot take any action on.

Hence, for every data point you collect, you should have an idea about what that data collection helps you achieve. This guide provides you with the knowledge you need to interpret the measurements you configure and how to take action on them. For ease of reference, this section summarizes the data collection required. You can use it as a checklist for data collection.

The tool used to collect the data does not matter much; it is often a question of preference and operational procedures. What this guide provides is the source of the data points – how you aggregate and collect them will depend on your organization.

## Performance Monitor

On an Analysis Services server, the minimal collection of data is this.

|  |  |  |
| --- | --- | --- |
| Performance object | Counter | Description |
| Logical Disk | All / All | Collects data about disk load and utilization. |
| Process | Private Bytes – msmdsrv.exe | The memory consumed by the Analysis Services process. |
| MSOLAP:Memory | Memory Usage Kb | Alternative to **Process**. |
| MSOLAP:Connection | Current Connections | Measures the number of open connections to gauge concurrency. |
| MSOLAP:Threads | \* | Allows tuning of thread pools. |
| Memory | Cache Bytes  Standby Cache Normal Priority Bytes  Standby Cache Core Bytes  Standby Cache Reserve Bytes | Estimates the size of the file system cache. |
| Memory | Page Faults / sec | Tests for excessive paging of the server. |
| Memory | Available Bytes | Used to tune memory settings. |
| MSOLAP: Proc Aggregations | temp file bytes written | Measures the spill from processing operations. Ideally, cube and hardware should be balanced to make this 0. |
| MSOLAP: Proc Indexes | Current Partitions  Rows/sec | Measures speed and concurrency of process index. |
| MSOLAP: Processing | Rows read/sec  Rows written/sec | Measures speed of relation read and efficiency of merge buffers. |
| MSOLAP:MDX | \* | Used by cube tuners to determine whether calculation scripts or MDX queries can be improved. |
| System | File Read Bytes/sec | Measure bytes read from the file system cache. |
| System | File Read Operations/sec | Measures IOPS from file system cache |
| Network Interface | Bytes Received/sec  Bytes Sent/sec | Contains the capacity plan to follow if NIC speed slows. See section 2.6.5. |
| TCPv4 and TCPv6 | Segments / sec  Segments Retransmitted/sec | Discovers unstable connections. |

In most cases, you can collect this information every 15 seconds without causing measurable impact to the system.

If you are using Systems Center Operations Manager (SCOM) to monitor servers, it is a good idea to add this data collection to your monitoring. Alternatively, you can use the built in performance monitor in Windows Server, but that will require that you harvest the counters from the servers yourself.

## Dynamic Management Views

Starting with SQL Server 2008 Analysis Services, there is a new set of monitoring tools available to you: dynamic management views (DMVs). These views provide information about the operations of the service that you cannot find in SQL Server Profiler or Performance Monitor.

The following table lists the most useful DMVs collect on a regular basis from the server.

|  |  |
| --- | --- |
| DMV | Description |
| $system.discover\_commands | Lists all currently running commands on the server, including commands run by the server itself. |
| $system.discover\_sessions | List all sessions on the server. It is used to map commands and connections together. |
| $system.discover\_connections | Lists current open connections. |
| $system.discover\_memoryusage | Lists all memory allocations in the server. |
| $system.discover\_locks | Lists currently held and requested locks. |

Your capture rate of the data depends on the system you are running and how quickly your operations team response to events. **$system.discover\_locks** and **$system.discover\_memoryusage** are expensive DMVs to query – and gathering them too often consumes significant CPU resources on the server. Capturing the DMVs once every few minutes is probably enough for most operations management purposes. If you choose to capture them more often, measure the impact on your server, which will depend on the concurrency of executing sessions, memory sizes, and cube design.

Depending on how often you query the DMV, you can generate a lot of data. It is often a good idea to keep the recent data at a high granularity and aggregate older data. Using a tool like Microsoft StreamInsight enables you to perform such historical aggregation in real time.

Capturing the DMV data enables you to monitor the progress of queries and alert you to heavy resource consumers early. Here are some examples of issues you can identify when you use DMVs:

* Queries that consume a lot of memory
* Queries that have been blocked for a long time
* Locks that are held for a long time
* Sessions that have run for a long time, or consumed a lot of I/O
* Connections that are transmitting a large amount of data over the network

One way to collect this data is to use a linked server from the SQL Server Database Engine to Analysis Services and use the Database Engine as the storage for the data you collect. You may also be able to configure your favorite monitoring tool to do the same. Unfortunately, Analysis Services does not ship with a fully automated tool for this data collection.

The following example collection scripts can get you started with a basic data collection framework:

select SESSION\_SPID /\* Key in commands \*/

/\* Monitor for large values and large different with below \*/

, COMMAND\_ELAPSED\_TIME\_MS

, COMMAND\_CPU\_TIME\_MS /\* Monitor for large values \*/

, COMMAND\_READS /\* Monitor for large values \*/

, COMMAND\_WRITES /\* Monitor for large values \*/

, COMMAND\_TEXT /\* Track any problems to the query \*/

from $system.discover\_commands

select SESSION\_SPID /\* Join to SPID in Sessions \*/

, SESSION\_CONNECTION\_ID /\* Join to connection\_id in Connections \*/

, SESSION\_USER\_NAME /\* Finds the authenticated user \*/

, SESSION\_CURRENT\_DATABASE

from $system.discover\_sessions

select CONNECTION\_ID /\* Key in connections \*/

, CONNECTION\_HOST\_NAME /\* Find the machine the user is coming from \*/

, CONNECTION\_ELAPSED\_TIME\_MS

, CONNECTION\_IDLE\_TIME\_MS /\* Find clients not closing connections \*/

/\* Monitor the below for large values \*/

, CONNECTION\_DATA\_BYTES\_SENT

, CONNECTION\_DATA\_BYTES\_RECEIVED

from $system.discover\_connections

select SPID

, MemoryName

, MemoryAllocated

, MemoryUsed

, Shrinkable

, ObjectParentPath

, ObjectId

from $system.discover\_memoryusage

where SPID <> 0

**References:**

* MSDN library reference on Analysis Services DMVs: <http://msdn.microsoft.com/en-us/library/ee301466.aspx>
* CodePlex ResMon tool, which captures detailed information about cubes: <http://sqlsrvanalysissrvcs.codeplex.com/wikipage?title=ResMon%20Cube%20Documentation>

## Profiler Traces

Analysis Services exposes a large number of events through the profiler API. Collecting every single one during normal operations not only generates high data collection volumes, it also consumes significant CPU and network traffic. While it is possible to dig into great detail of the server, you should measure to impact on the system while running the trace. If every event is traced, a lot of trace information is generated: Make sure you either clean up historical records or have enough space to hold the trace data. Very detailed SQL Server Profiler traces are best reserved for situations where you need to do diagnostics on the server or where you have enough CPU and storage capacity to collect lots of details.

**References:**

* ASTrace – a tool to collect profiler traces into SQL Server for further analysis: <http://msftasprodsamples.codeplex.com/wikipage?title=SS2005%21Readme%20for%20ASTrace%20Utility%20Sample&referringTitle=Home>

## Data Collection Checklist

You can use the following checklist to make sure you have covered the basic data collection needs of the server:

* Windows Server-specific Performance Monitor counters
* Analysis Services-specific Performance Monitor counters
* Data Management Views collected:
  + $system.discover\_commands
  + $system.discover\_sessions
  + $system.discover\_connections
  + $system.discover\_memoryusage
  + $system.discover\_locks

## Monitoring and Configuring Memory

To discover the best memory configuration for your Analysis Services instance, you need to collect some data about the typical usage of the system.

First of all, you should start up the server with Analysis Services disabled and make sure that all the other services you need for normal operations in a state that is typical for your standard server configuration. What typical is varies by environment –you are looking to measure how much memory will be available to Analysis Services after the operating system and other services (such as virus scanners and monitoring tools) have taken their share. Note down the value of the Performance Monitor counter **Memory/Available Bytes**.

Secondly, start Analysis Services and run a typical user workload on the server. You can for example use queries from your test harness. For this test, also make sure **PreAllocate** is set to 0. While the workload is running, run the following query.

SELECT \* FROM $SYSTEM.DISCOVER\_MEMORYUSAGE

If you have any long-running, high-memory consuming queries, you should also measure how much memory they consume while they execute.

Copy the data into an Excel spreadsheet for further analysis. You can also use the CodePlex project ResMon cube (see reference section) to periodically log snapshots of this DMV and browse memory usage trends summarized in a cube.

With the data collected in the previous section, you can define some values needed to set the memory:

**[Physical RAM]** = The total physical memory on the box

**[Available RAM]** = The value of the Performance Monitor counter **Memory/Available** Bytes as noted down earlier.

**[Non Shrinkable Memory]** = The sum of the **MemoryUsed** column from $SYSTEM.DISCOVER\_MEMORYUSAGE where the column **Shrinkable** is **False.**

**[Valuable Objects]** = The sum of all objects that you want to reserve memory for from $SYSTEM.DISCOVER\_MEMORYUSAGE where column **Shrinkable** is **True**. For example, you will most likely want to reserve memory for all dimensions and attributes. If you have a small cube, this value may simply be everything that is marked as shrinkable.

**[Worst Queries Memory]** = The amount of memory used by the all the most demanding queries you expect will run concurrently. You can use the DMV **$system.discover\_memoryusage** to measure this on a known workload.

With the preceding values you can calculate the memory configuration for Analysis Services:

**LowMemoryLimit** = [Non Shrinkable Memory] + [Valuable Objects] / [Total RAM]But keep a gap of at least 20 percent between this value and **HighMemoryLimit** to allow the memory cleaners to release memory at a good rate.

**HighMemoryLimit** = ([Available RAM] - [Worst Queries Memory]) / [Total RAM]

**HardMemoryLimit** = [Available RAM] / [Total RAM]

**LimitSystemFileCacheMB =** [Available RAM] - LowMemoryLimit

Note that it may not always be possible to measure all the components that make up these equations. In such case, your best bet is to guesstimate them. The idea behind this method is that Analysis Services will always hold on to enough memory to keep the valuable objects in memory. What is valuable depends on your particular setup and query set. The rest of the available memory is shared between the file system cache and the Analysis Services caches; if you use the operating systems memory usage optimizations, the ideal balance between Analysis Services and the file system cache is adjusted dynamically.

If Analysis Services coexists with other services on the machine, take their maximum memory consumption into consideration. When you calculate **[Available RAM]**, subtract the maximum memory use by other large memory consumers, such as the SQL Server relational engine. Also, make sure those other memory consumers have their maximum memory settings adjusted to allow space for Analysis Services.

The following diagram illustrates the different uses of memory.



Figure 13 - Memory Settings

**References:**

* CodePlex ResMon project - <http://sqlsrvanalysissrvcs.codeplex.com/wikipage?title=ResMon%20Cube%20Documentation>

## Monitoring and Tuning I/O and File System Caches

To tune the Analysis Services I/O subsystem, it is helpful to understand how a user query is translated into I/O requests. The following diagram illustrates the caching and I/O architecture of Analysis Services.



Figure 14 - I/O stack

From the perspective of the I/O system, the storage engine requests data from an offset in a file. The building blocks of the storage used by Analysis Services are the files used to store dimension and fact data. Dimension data is typically cached by the storage engine. Hence, the most frequently requested files from the storage system are measure group data, and they have the following file names: **\*.fact.data, \*.aggs.rigid.data** and **\*.agg.flex.data**. Because Analysis Services uses buffered I/O, frequently requested blocks in files are typically retained by the NTFS file system cache. Note that this means that the memory used for the file system caching of data comes from outside the Analysis Services working set.

When a block from a file is requested by Analysis Services, the path taken by the operating system depends on the state of the cache.



Figure 15 - Accessing a file in the NTFS cache

In the preceding illustration, Analysis Services executes the ReadFileEx call to the Windows API.

* If the block is not in the cache, an I/O request is issued (2), the file is put in the cache (3), and the data is returned to Analysis Services (5).
* If he block is already in the cache (3), it is simply returned to Analysis Services (5).
* If a block is not frequently accessed or if there is memory pressure, the NTFS file system cache may choose to place the block in the Standby Cache (4) and eventually remove the file from memory. If the file is requested before it is removed, it is brought back into the System Cache (3) and then returned to Analysis Services (5).
  + Note that in Windows Server 2003, the file is simply removed from the system cache, bypassing the standby cache.

### System Cache vs. Physical I/O

Because Analysis Services uses the NTFS cache, not every I/O request reaches the I/O subsystem. This means that even with a clear storage engine and formula engine cache, query performance will still vary depending on the state of the file system cache. In the worst case, a query will run on a clean file system cache and every I/O request will hit the physical disk. In the best case, every I/O request will be served by the memory cache. This depends on the amount of data requested by the query.

It is useful to know the ratio between the I/O issues to the disk and the I/O served by the file system cache. This helps you capacity plan for growing cube. Small cubes, less than the size of the server memory, are typically served mostly from the file system cache. However, as cubes grow larger and exceed the file system cache size, you will eventually have to assist cube performance with a fast disk system. Measuring the current cache hit ratio helps shed light on this.

To measure the I/O served by the file system cache, use Performance Monitor to measure **System:File Read Bytes/sec** and **System File Read Operations/sec**. These counters will give you an estimate of the number of I/O operations and file bytes that are served from memory. By comparing this with **Logical Disk / Disk reads /sec** you can measure the ratio between memory buffered and physical I/O. As the amount of data in the cube grows, you will often see that the amount of disk access begins to grow in proportion to memory access. By extrapolating from historical trends, you can then decide what the best ratio between IOPS and RAM sizes should be going forward.

To measure the total memory consumption of Analysis Services, you will also have to take the file system cache into account. The total memory used by analysis services is the sum of:

* **Process – msmdsrv.exe / Private Bytes** – The memory used by Analysis Services itself
* **Memory – Cache Bytes** – The files currently live in cache
* **Memory – Standby Cache Normal Priority Bytes** – The files that can be freed from the cache

However, note that the memory counters also measure other files’ caches in the NTFS file system cache. If you are running Analysis Services together with other services on the machine, the file system counter may not accurately reflect the caches used by Analysis Services.

### TempDir Folder

When memory is scarce on the server, Analysis Services spills operations to disk. Examples of such operation are:

* Dimension processing
  + ByTable processing commands that don’t fit memory
  + Processing of large dimension where the hash tables created exceed available memory
* Aggregation processing
* ROLAP dimension attribute stores

**Dimension processing:** To optimize dimension processing, we recommend that you use the techniques described later in this document to avoid spills and speed up processing. **ByTable** should generally only be used if you can keep the dimension data in memory during processing.

**Aggregation processing:** During cube processing, aggregation buffers described in configuration section determine the amount of memory that is available to build aggregations for a given partition. If the aggregation buffers are too small, Analysis Services supplements the aggregation buffers with temporary files. To monitor any temporary files used during aggregation, review **MSOLAP:Proc Aggregations\Temp file bytes written/sec**. You should try to design your cube in such a way that memory spills to temp files does not occur, that is, keep the temp bytes written at zero. There are several techniques available to avoid memory spills during aggregation processing:

* Only create aggregations that are small enough to fit in memory during processing.
* Process fewer partitions in parallel.
* Add more memory to the machine or allocating more memory to Analysis Services.

It is generally a good idea to split **ProcessData** and **ProcessIndex** operations into two different processing jobs. **ProcessData** typically consumes less memory than **ProcessIndex** and you run many **ProcessData** operations in parallel. During **ProcessIndex**, you can then decrease concurrency if you are short on memory, to avoid the disk spill.

**ROLAP dimensions:** In general, you should try to avoid ROLAP dimensions; MOLAP stores are much faster for dimension access. However, the requirement for ROLAP dimensions is often driven by a lack of sufficient memory to hold the attribute stores requires for drillthrough actions, which returns data using a degenerate ROLAP dimension. If this is your scenario, you may not be able to avoid spills to the **TempDir** folder.

If you cannot design the cube to fit all operations in memory, spilling to **TempDir** may be your only option. If that is the case, we recommend that you place the **TempDir** folder on a fast LUN. You can either use a LUN backed by caches (for example in a SAN) or one that sits on fast storage – for example, NAND devices.

## Monitoring the Network

One way to easily monitor network throughput is through a performance monitor trace. Windows Performance Monitor requests the current value of performance counters at specified time intervals and logs them into a trace log (.blg or .csv).

The following counters will help you isolate any problems in the network layer.

**Processing: Rows read/sec** – this is the rate of rows read from your data source. This is one of the most important counters to monitor when you measure your throughput from Analysis Services to your relational data source. When you monitor this counter, you will want to view the trace using a line chart, as it gives you a better idea of your throughput relative to time. It is reasonable to expect tens of thousands of rows per second per network connection to a SQL Server data source. Third-party sources may experience significantly slower throughput.

**Bytes received/sec** and **Bytes sent/sec** - These two counters enable you to measure how far you are from the theoretical NIC speeds. It allows capacity planning for faster NIC.

**TCPv4 and TCPv6 Segments/sec** and **Segments retransmitted/sec** – These counters enable you to discover unstable network connections. The ratio between segments retransmitted and segments for both TCPv4 and TCPv6 should not exceed 3-4 percent on a stable network.

# Testing Analysis Services Cubes

As you prepare for user acceptance and preproduction testing of a cube, you should first consider what a cube is and what that means for user queries. Depending on your background and role in the development and deployment cycle, there are different ways to look at this.

As a database administrator, you can think of a cube as a database that can accept *any* query from users, and where the response time from any such query is expected to be “reasonable” – a term that is often vaguely defined. In many cases, you can optimize response time for specific queries using aggregates (which for relational DBAs is similar to index tuning), and testing should give you an early idea of good aggregation candidates. But even with aggregates, you must also consider the worst case: You should expect to see leaf-level scan queries. Such queries , which can be easily expressed by tools like Excel, may end up touching every piece of data the cube. Depending on your design, this can be a significant amount of data. You should consider what you want to do with such queries. There are multiple options: For example, you may choose to scale your hardware to handle them in a decent response time, or you may simply choose to cancel them. In either case, as you prepare for testing, make sure such queries are part the test suite and that you observe what happens to the Analysis Services instance when they run. You should also understand what a “reasonable” response time is for your end user and make that part of the test suite.

As a BI developer, you can look at the cube as your description of the multidimensional space in which queries can be expressed. A part of this space will be instantiated with data structures on disk supporting it: dimensions, measure groups, and their partitions. However, some of this multidimensional space will be served by calculations or ad-hoc memory structures, for example: MDX calculations, many-to-many dimensions, and custom rollups. Whenever queries are not served directly by instantiated data, there is a potential, query-time calculation price to be paid, this may show up as bad response time. As the cube developer, you should make sure that the testing covers these cases. Hence, as the BI-developer, you should make sure the test queries also stress noninstantiated data. This is a valuable exercise, because you can use it to measure the impact on users of complex calculations and then adjust the data model accordingly.

Your approach to testing will depend on which situation you find yourself in. If you are developing a new system, you can work directly towards the test goals driven by business requirements. However, if you are trying to improve an existing system, it is an advantage to acquire a test baseline first.

## Testing Goals

Before you design a test harness, you should decide what your testing goals will be. Testing not only allows you to find functional bugs in the system, it also helps quantify the scalability and potential bottlenecks that may be hard to diagnose and fix in a busy production environment. If you do not know what your scale-barrier is or where they system might break, it becomes hard to act with confidence when you tune the final production system.

Consider what characteristics are reasonable to expect from the BI systems and document these expectations as part of your test plan. The definition of *reasonable* depends to a large extent on your familiarity with similar systems, the skills of your cube designers, and the hardware you run on. It is often a good idea to get a second opinion on what test values are reachable – for example from a neutral third party that has experience with similar systems. This helps you set expectations properly with both developers and business users.

BI systems vary in the characteristics organizations require of them – not everyone needs scalability to thousands of users, tens of terabytes, near-zero downtime, and guaranteed subsecond response time. While all these goals can be achieved for most cases, it is not always cheap to acquire the skills required to design a system to support them. Consider what your system needs to do for your organization, and avoid overdesigning in the areas where you don’t need the highest requirements. For example: you may decide that you need very fast response times, but that you also want a very low-cost server that can run in a shared storage environment. For such a scenario, you may want to reduce the data in the cube to a size that will fit in memory, eliminating the need for the majority of I/O operations and providing fast scan times even for poorly filtered queries.

Here is a table of potential test goals you should consider. If they are relevant for your organization’s requirements, you should tailor them to reflect those requirements.

|  |  |  |
| --- | --- | --- |
| Test Goal | Description | Example goal |
| Scalability | How many concurrent users should be supported by the system? | “Must support 10,000 concurrently connected users, of which 1,000 run queries simultaneously.” |
| Performance/ throughput | How fast should queries return to the client? This may require you to classify queries into different complexities.  Not all queries can be answered quickly and it will often be wise to consult an expert cube designer to liaise with users to understand what query patterns can be expected and what the complexity of answering these queries will be.  Another way to look at this test goal is to measure the throughput in queries answered per second in a mixed workload. | “Simple queries returning a single product group for a given year should return in less than 1 second – even at full user concurrency.”  “Queries that touch no more than 20% of the fact rows should run in less than 30 seconds. Most other queries touching a small part of the cube should return in around 10 seconds. With our workload, we expect throughput to be around 50 queries returned per second.”  “User queries requesting the end-of-month currency rate conversion should return in no more than 20 seconds. Queries that do not require currency conversion should return in less than 5 seconds.” |
| Data Sizes | What is the granularity of each dimension attribute? How much data will each measure group contain?  Note that the cube designers will often have been considering this and may already know the answer. | “The largest customer dimension will contain 30 million rows and have 10 attributes and two user hierarchies. The largest non-key attribute will have 1 million members.”  “The largest measure group is sales, with 1 billion rows. The second largest is purchases, with 100 million rows. All other measure groups are trivial in size.” |
| Target Server  Platforms | Which server model do you want to run on? It is often a good idea to test on both that server and an even bigger server class. This enables you to quantify the benefits of upgrading. | “Must run on 2-socket 6-core Nehalem machine with 32 GB of RAM.”  “Must be able to scale to 4-socket Nehalem 8-core machine with 256 GB of RAM. |
| Target I/O system | Which I/O system do you want to use? What characteristics will that system have? | “Must run on corporate SAN and use no more than 1,000 random IOPS at 32,000 block sizes at 6ms latency.”  “Will run on dedicated NAND devices that support 80,000 IOPS at 100 µs latency.” |
| Target network infrastructure | Which network connectivity will be available between users and Analysis Services, and between Analysis Services and the data sources?  Note that you may have to simulate these network conditions in a lab. | “In the worst case scenario, users will connect over a 100ms latency WAN link with a maximum bandwidth of 10Mbit/sec.”  “There will be a 10Gbit dedicated network available between the data source and the cube.” |
| Processing Speeds | How fast should rows be brought into the cube and how often? | “Dimensions should be fully processed every night within 30 minutes.”  “Two times during the day, 100,000,000 rows should be added to the sales measure group. This should take no longer than 15 minutes.” |

## Test Scenarios

Based on the considerations from the previous section you should be able to create a user workload that represents typical user behavior and that enables you to measure whether you are meeting your testing goals.

Typical user behavior and well-written queries are unfortunately not the only queries you will receive in most systems. As part of the test phase, you should also try to flush out potential production issues before they arise. We recommend that you make sure your test workload contains the following types of queries and tests them thoroughly :

* Queries that touch several dimensions at the same time
* Enough queries to test every MDX expression in the calculation script
* Queries that exercise many-to-many dimensions
* Queries that exercise custom rollups
* Queries that exercise parent/child dimensions
* Queries that exercise distinct count measure groups
* Queries that crossjoin attributes from dimensions that contain more than 100,000 members
* Queries that touch every single partition in the database
* Queries that touch a large subset of partitions in the database (for example, current year)
* Queries that return a lot of data to the client ( for example, more than 100,000 rows)
* Queries that use cube security versus queries that do not use it
* Queries executing concurrently with processing operations – if this is part of your design

You should test on the full dataset for the production cube. If you don’t, the results will not be representative of real-life operations. This is especially true for cubes that are larger than the memory on the machine they will eventually run on.

Of course, you should still make sure that you have plenty of queries in the test scenarios that represent typical user behaviors – running on a workload that only showcases the slowest-performing parts of the cube will not represent a real production environment (unless of course, the entire cube is poorly designed).

As you run the tests, you will discover that certain queries are more disruptive than others. One goal of testing is to discover what such queries look like, so that you can either scale the system to deal with them or provide guidance for users so that they can avoid exercising the cube in this way if possible.

Part of your test scenarios should also aim to observe the cubes behavior as user concurrency grows. You should work with BI developers and business users to understand what the worst-case scenario for user concurrency is. Testing at that concurrency will shake out poorly scalable designs and help you configure the cube and hardware for best performance and stability.

## Load Generation

It is hard to create a load that actually looks like the expected production load – it requires significant experience and communication with end users to come up with a fully representative set of queries. But after you have a set of queries that match user behavior, you can feed them into a test harness. Analysis Services does not ship with a test harness out of the box, but there are several solutions available that help you get started:

**ascmd** – You can use this command-line tool to run a set of queries against Analysis Services. It ships with the Analysis Services samples and is maintained on CodePlex.

**Visual Studio** – You can configure Microsoft Visual Studio to generate load against Analysis Services, and you can also use Visual Studio to visually analyze that load.

**Third-party tools** – You can use tools such as HP LoadRunner to generate high-concurrency load. Note that Analysis Services also supports an HTTP-based interface, which means it may be possible to use web stress tools to generate load.

**Roll your own:** We have seen customers write their own test harnesses using .NET and the ADOMD.NET interface to Analysis Services. Using the .NET threading libraries, it is possible to generate a lot of user load from a single load client.

No matter which load tool you use, you should make sure you collect the runtime of all queries and the Performance Monitor counters for all runs. This data enables you to measure the effect of any changes you make during your test runs. When you generate user workload there are also some other factors to consider.

First of all, you should test both a sequential run and a parallel run of queries. The sequential run gives you the best possible run time of the query while no other users are on the system. The parallel run enables you to shake out issues with the cube that are the result of many users running concurrently.

Second, you should make sure the test scenarios contain a sufficient number of queries so that you will be able to run the test scenario for some time. To stress the server properly, and to avoid querying the same hotspot values over and over again, queries should touch a variety of data in the cube. If all your queries touch a small set of dimension values, it will hardly represent a real production run. One way to spread queries over a wider set of cells in the cube is to use query templates. Each template can be used to generate a set of queries that are all variants of the same general user behavior.

Third, your test harness should be able to create reproducible tests. If you are using code that generates many queries from a small set of templates, – make sure that it generates the same queries on every test run. If not, you introduce an element of randomness in the test that makes it hard to compare different runs.

**References:**

* Ascmd.exe on MSDN - <http://msdn.microsoft.com/en-us/library/ms365187%28v=sql.100%29.aspx>
* Analysis Services Community samples - <http://sqlsrvanalysissrvcs.codeplex.com/>
  + Describes how to use **ascmd** for load generation
  + Contains Visual Studio sample code that achieves a similar effect
* HP LoadRunner - [https://h10078.www1.hp.com/cda/hpms/display/main/hpms\_content.jsp?zn=bto&cp=1-11-126-17^8\_4000\_100](https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17%5e8_4000_100)

## Clearing Caches

To make test runs reproducible, it is important that each run start with the server in the same state as the previous run. To do this, you must clear out any caches created by earlier runs.

There are three caches in Analysis Services that you should be aware of:

* The formula engine cache
* The storage engine cache
* The file system cache

**Clearing formula engine and storage engine caches:** The first two caches can be cleared with the XMLA **ClearCache** command. This command can be executed using the **ascmd** command-line utility:

<ClearCache

xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

<Object>

<DatabaseID><database name></DatabaseID>

</Object>

</ClearCache>

**Clearing file system caches:** The file system cache is a bit harder to get rid of because it resides inside Windows itself.

If you have created a separate Windows volume for the cube database, you can dismount the volume itself using the following command:

**fsutil.exe volume dismount** < Drive Letter | Mount Point >

This clears the file system cache for this drive letter or mount point. If the cube database resides only on this location, running this command results in a clean file system cache.

Alternatively, you can use the utility **RAMMap** from sysinternals. This utility not only allows you to read the file system cache content, it also allows you to purge it. On the **empty** menu, click **Empty System Working Set**, and then click **Empty Standby List**. This clears the file system cache for the entire system. Note that when **RAMMap** starts up, it temporarily freezes the system while it reads the memory content – this can take some time on a large machine. Hence, **RAMMap** should be used with care.

There is currently a CodePlex project called ASStoredProcedures found at: <http://asstoredprocedures.codeplex.com/wikipage?title=FileSystemCache>. This project contains code for a utility that enables you to clear the file system cache using a stored procedure that you can run directly on Analysis Services.

Note that neither **FSUTIL** nor **RAMMap** should be used in production cubes –both cause disruption to users connected to the cube. Also note that neither **RAMMap** or ASStoredProcedures is supported by Microsoft.

## Actions During Testing for Operations Management

While your test team is creating and running the test harness, your operations team can also take steps to prepare for deployment.

**Test your data collection setup:** Test runs give you a unique chance to try out your data collection procedures before you go into production. The data collection you perform can also be used to drive early feedback to the development team.

**Understand server utilization:** While you test, you can get an early insight into server utilization. You will be able to measure the memory usage of the cube and the way the number of users maps to I/O load and CPU utilization. If the cube is larger than memory, you can also measure the effect of concurrency and leaf level scan on the I/O subsystem. Remember to measure the worst-case examples described earlier to understand what the impact on the system is.

**Early thread tuning:** During testing, you can discover threading bottlenecks, as described in this guide. This enables you to go into production with pretuned settings that improve user experience, scalability, and hardware utilization of the solution.

## Actions After Testing

When testing is complete, you have reports that describe the run time of each query, and you also have a greater understanding of the server utilization. This is a good time to review the design of the cube with the BI developers using the numbers you collected during testing. Often, some easy wins can be harvested at this point.

When you put a cube into production, it is important to understand the long-term effects of users building spreadsheets and reports referencing it. Consider the dependencies that are generated as the cube is successfully deployed in ad-hoc data structures across the organization. The data model created and exposed by the cube will be linked into spreadsheets and reports, and it becomes hard to make data model changes without disturbing users. From an operational perspective, preproduction testing is typically your last chance to request cheap data model changes from your BI developers before business users inevitably lock themselves into the data structures that unlock their data. Notice that this is different from typical, static reporting and development cycles. With static reports, the BI developers are in control of the dependencies ;if you give users Excel or other ad-hoc access to cubes, that control is lost. Explorative data power comes at a price.

# Security and Auditing

Cubes often contain the very heart of your business data – the source of your decision making. That means that you have to consider potential attack vectors that a malicious user or intruder could use to acquire access to your data. The following table provides an overview of the potential vulnerabilities and the countermeasures you can take. Note that most environments will not need all of these countermeasures – it depends on the data security and on the attack vectors that are possible on your network.

|  |  |
| --- | --- |
| Attack vector | Countermeasure |
| Other services listening on the server | Firewall all ports other than those used by Analysis Services |
| Sniff TCP/IP packets to client  Sniff TCP/IP packets during processing | Configure IPsec or SSL encryption  Configure SQL Server Protocol Encryption |
| Steal physical media containing cubes | Encrypt file system used to store cubes |
| Compromise the service account | Configure minimum privileges to the service  account  Require a strong password |
| Access the file system as a logged-in user | Secure file system with minimal privileges |

A full treatment of security configuration is outside the scope of this guide, but the following sections provide references that serve as a starting point for further reading and give you an overview of the options available to you.

## Firewalling the Server

By default, Analysis Services communicates with clients on port 2383. If you want to change this, access the properties of the server.

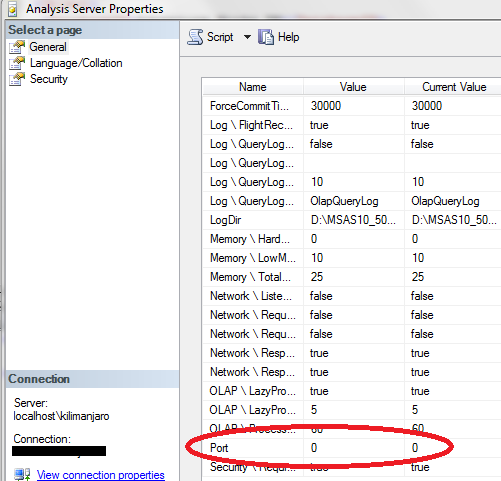


Figure 16 - Changing the port used for Analysis Services

Bear in mind that you will need to open the port assigned here in your firewall for TCP/IP traffic. If you leave the default value of 0, Analysis Services will use port 2382.

If you are using named instances, your client application may also need to access the SQL Server browser service. The browser service allows clients to resolve instance names to port numbers, and it listens on TCP port 2382. Note that it is possible to configure the connection string for Analysis Services in such a way that you will not need the browser service port open. To connect directly to the port that Analysis Services is listening on, use this format **[Server name}:[Port].** For example, to connect to an instance listening on port 2384 on server MyServer, use **MyServer:2384**.

Analysis Services can be set up to use HTTP to communicate with clients. If you choose this option, follow the guidelines for configuring Microsoft Internet Information Services. In this case, you will typically need to open either port 80 or port 443.

**References:**

* How to: Configure Windows Firewall for Analysis Services Access - <http://msdn.microsoft.com/en-us/library/ms174937.aspx>
* Resolving Common Connectivity Issues in SQL Server 2005 Analysis Services Connectivity Scenarios - <http://msdn.microsoft.com/en-us/library/cc917670.aspx>
  + Also applies to SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services
* Configuring HTTP Access to SQL Server 2005 Analysis Services on Microsoft Windows 2003 - <http://technet.microsoft.com/en-us/library/cc917711.aspx>
  + Also applies to SQL Server 2008 Analysis Services, SQL Server 2008 R2 Analysis Services, Windows Server 2008, and Windows Server 2008 R2
* Analysis Services 2005 protocol - XMLA over TCP/IP - <http://www.mosha.com/msolap/articles/as2005_protocol.htm>

## Encrypting Transmissions on the Network

Analysis Services communicates in a compressed and encrypted format on the network. You may still want to use IPsec to restrict which machines can listen in on the network traffic, but the communication channel itself is already encrypted by Analysis Services.

If you have configured Analysis Services to communicate over HTTP, the communication can be secured using the SSL protocol. However, be aware that you may have to acquire a certificate to use SSL encryption over public networks. Also, note that SSL encryption normally uses port 443, not port 80, to communicate. This difference may require changes in the firewall configuration. Using the HTTP protocol also allows you to run secured lines to parties outside the corporate network – for example, in an extranet setup.

Depending on your network configuration, you may also be concerned about network packets getting sniffed during cube processing. To avoid this, you again have to encrypt traffic. Again, you can use IPsec to achieve this. Another option is to use protocol encryption in SQL Server, which is described in the references.

**References:**

* How to configure SQL Server 2008 Analysis Services and SQL Server 2005 Analysis Services to use Kerberos authentication - <http://support.microsoft.com/kb/917409>
* Windows Firewall with Advanced Security: Step-by-Step Guide: Deploying Windows Firewall and IPsec Policies - <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=0b937897-ce39-498e-bb37-751c00f197d9&displaylang=en>
* How To Configure IPsec Tunneling in Windows Server 2003 - <http://support.microsoft.com/kb/816514>
* How to enable SSL encryption for an instance of SQL Server by using Microsoft Management Console - <http://support.microsoft.com/kb/316898>

## Encrypting Cube Files

Some security standards require you to secure the media that the data is stored on, to prevent intruders from physically stealing the data. If you have MOLAP cubes on the server, media security may be a concern to you. Because Analysis Services, unlike the relational engine, does not ship with native encryption of MOLAP cube data, you must rely on encryption outside the engine. You can use Windows File System encryption (Windows Server 2003) or BitLocker (on Windows Server 2008 and Windows Server 2008 R2) to encrypt the drive used to store cubes. Be careful, though; encrypting MOLAP data can have a big impact on performance, especially during processing and schema modification of the cube. We have seen performance drops of up to 50 percent when processing dimensions on encrypted drives – though less so for fact processing. Weigh the requirement to encrypt data carefully against the desired performance characteristics of the cube.

**References:**

* Encrypting File System in Windows XP and Windows Server 2003 - <http://technet.microsoft.com/en-us/library/bb457065.aspx>
* BitLocker Drive Encryption - <http://technet.microsoft.com/en-us/library/cc731549%28WS.10%29.aspx>

## Securing the Service Account and Segregation of Duties

To secure the service account for Analysis Services, it is useful to first understand the different security roles that exist at a server level.

**The** **service account** is the account that runs the msmdsrv.exe file. It is configured during installation and can be changed using SQL Server Configuration Manager. The service account is the account used to access all files required by Analysis Services, including MOLAP stores. For your convenience, a local group **SQLServerMSASUser$<Instance Name>** iscreated that has the right privileges on the binaries required to run Analysis Services. If you configure the service account during installation, the account will automatically be added to this group. If you later choose to change the service account in SQL Server Configuration Manager, you must manually update the group membership.

**The server administrator role** has privileges to change server settings, and to create, back up, restore, and delete databases. Members of this account can also control security on all databases in the instance. It is the DBA role of the instance and almost equivalent to the **sysadmin** role for SQL Server. Membership in the server administrator role is configured in the properties of the server in SQL Server Management Studio.

In a secure environment, you should run Analysis Services under a dedicated service account. You can configure this during installation of the service.

If your environment requires segregation of duties between those who configure the service account and those who administer the server, you need to make some changes to the msmdsrv.ini file:

* By default, local administrators are members of the server administrator role. To remove this association, set **<BuiltinAdminsAreServerAdmins>** to 0.
* By default, the service account for Analysis Services is a member of the server administrator role. Remove this association by setting **<ServiceAccountIsServerAdmin>** to 0.

However, keep in mind that a local administrator could still access the msmdsrv.ini file and alter the changes you have made, so you should audit for this possibility.

## File System Permissions

As mentioned earlier, the Analysis Services installer will create the Windows NT group **SQLServerMSASUser$<Instance Name>** and add the service account to this group. At install time, this group is also granted access to the **Data**, backup, log, and **TempDir** folders.

If you at a later time add more folders to the instance to hold data, backups, or log files, you will need to grant the **SQLServerMSASUser$<Instance Name>** groupread and write access to the new folders. If you move the TempDir folder, you will also need to assign the group read/write permission to the new location. No other users need permissions on these folders for Analysis Services to operate.

In the configuration of the server you will also find the **AllowedBrowsingFolders** setting. This setting, a pipe-separated list of directories, limits the visible folders that server administrators can see when they configure storage locations Analysis Services data. **AllowedBrowsingFolders** is *not* a security feature, because server administrator can change the values in it to reflect anything that the service account can see. However, it does serve as a convenient filter to display only a subset of the folders that the service account can access. Note that server administrators cannot directly access the data visible through the **AllowedBrowsingFolders** setting, but they can write backups in the folders, restore form the folders, move **TempDir** there, and set the storage location of databases, dimensions, and partitions to those folders.

# High Availability and Disaster Recovery

High availability and disaster recovery are not robustly integrated as part of Analysis Services for large-scale cubes. In this section, you learn about the combination of methods you can use to achieve these goals within an enterprise environment.

To ensure disaster recovery, use the built-in Backup/Restore method, Analysis Services Synch, or Robocopy to ensure that you have multiple copies of the database. You can achieve high availability by deploying your Analysis Services environment onto clustered servers. Also note that by using a combination of multiple copies, clustering, and scale out (section 7.4), you can achieve both high availability and disaster recovery for your Analysis Services environment. In a large-scale environment, the scale-out method generally provides the best of use hardware resources while also providing both backup and high availability.

## Backup/Restore

Cubes are data structures on disk and as such, they contain information that you may want to back up.

**Analysis Services backup –** Analysis Services has a built-in backup functionality that generates a single backup file from a cube database. Analysis Services backup speeds have been significantly improved in SQL Server 2008 and for most solutions, you can simply use this built-in backup and restore functionality. As with all backup solutions, you should of course test the restore speed.

**SAN based backup** – If you use a SAN, you can often make backups of a LUN using the storage system itself. This backup process is transparent to Analysis Services and typically operates on the LUN level. You should coordinate with your SAN administrator to make sure the correct LUNs are backed up, including all relevant data folders used to store the cube. You should also make sure that the SAN backup utility uses a VDI/VSS compliant tool to call to Windows before the backup is taken. If you do not use a VDI/VSS tool, you risk getting chkdsk errors when the LUN is restored.

**File based backup copy** – In SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services, you can attach database if you have a copy of the data folder the database resides in. This means that a detached copy of files in a stale cube can serve as a backup of the database. This option is available only with SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services. You can use the same tools that you use for scale-out cubes (for example, Robocopy or NiceCopy). Restoring in this case means copying the files back to the server and attaching the database.

**Don’t back up** – While this may sound like a silly option, it does have merit in some disaster recovery scenarios. Cubes are built on relational data sources. If these sources are guaranteed to be available and securely backed up, it may be faster to simply reprocess the cube than to restore it from backup. If you use this option, make sure that no data resides in the cubes that cannot be re-created from the relational source (for example, data loaded via the SQL Server Integration Services Analysis Services destination). Of course, you should also make sure that the cube structure itself is available, including all aggregation and partition designs that have been changed on the production server from the standard deployment script.

This is particularly true for ROLAP cubes. In this case (as in all the previous scenarios), you should *always* maintain an updated backup of your Analysis Services project that allows for a redeployment of the solution (with the subsequent required processing), in case your backup media suffers any kind of physical corruption.

### Synchronization as Backup

If you are backing up small or medium-size databases, the Analysis Services synchronization feature is an operationally easy method. It synchronizes databases from source to target Analysis Service servers. The process scans the differences between the two databases and transfers only the files that have been modified. There is overhead associated with scanning and verifying the metadata between the two different databases, which can increase the time it takes to synchronize. This overhead becomes increasingly apparent in relation to the size and number of partitions of the databases.

Here are some important factors to consider when you work with synchronization:

* At the end of the synchronization process, a Write lock must be applied to the target server. If this lock is queued up behind a long running query, it can prevent users from querying the target database.
* During synchronization, a read commit lock is applied to the source server, which prevents users from processing new data but allows multiple synchronizations to occur from the same source server.
* For enterprise-size databases, the synchronization method can be expensive and low-performing. Because some operations are single-threaded (such as the delete operation), having high-performance servers and disk subsystems may not result in faster synchronization times. Because of this, we recommended that for enterprise size databases you use alternate methods, such as Robocopy (discussed later in this guide) or hardware-based solutions (for example, SAN clones and snapshots).
* When executing multiple synchronization operations against a single server, you may get the best performance (by minimizing lock contentions) by queuing up the synchronization requests and then running them serially.
* For the same locking contention reasons, plan your synchronizations for down times, when querying and processing are not occurring on the affected servers. While there are locks in place to prevent overwrites, processes such as long-running queries and processing may prevent the synchronization from completing in a timely fashion.

For more information, see the “Analysis Services Synch Method” section in Analysis Services Synchronization Best Practices technical note (<http://sqlcat.com/technicalnotes/archive/2008/03/16/analysis-services-synchronization-best-practices.aspx>).

### Robocopy

The basic principle behind the Robocopy method is to use a fast copy utility, such as Robocopy, to copy the OLAP data folder from one server to another. By following the sample script noted in the technical note, Sample Robocopy Script to customer synchronize Analysis Services databases (<http://sqlcat.com/technicalnotes/archive/2008/01/17/sample-robocopy-script-to-customer-synchronize-analysis-services-databases.aspx>), you can perform delta copies (that is, copy only the data that has changed) of the OLAP data folder from one source to another target source in parallel. This method is often employed in enterprise environments because the key factor here is fast, robust data file transfer.

However, the key disadvantages of using this approach include:

* You must stop and then restart (in SQL Server 2005 Analysis Services) or detach (in SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services) your Analysis Services servers when you use a fast copy utility.
* You cannot use the synchronization feature and Robocopy together.
* This approach makes the assumption that there is only one database on that instance; this is usually okay because of its size.
* Some functionality is lost if you use this method, including, but not limited to, writeback, ROLAP, and real-time updates.

Nevertheless, this is an especially effective method for query server / processing server architectures that involve only one database per server.

**References:**

* Scale-Out Querying with Analysis Services (<http://sqlcat.com/whitepapers/archive/2007/12/16/scale-out-querying-with-analysis-services.aspx>)

## Clustered Servers

Analysis Services can participate in a Windows failover cluster using a shared disk subsystem, such as a SAN. When provisioning storage for a cluster, there are some things you should be aware of.

It is not possible to use dynamic disk stripes. In this case, you have two options for spreading the cubes over all available LUN:

* Have your SAN administrator configure a mega-LUN.
* Selectively place partitions on different LUNs and then manually balance the load between these LUNs.

**Mega LUN:** Most SANs today can stripe multiple, smaller LUNs into a large mega-LUN. Talk to your SAN administrator about this option.

**Selective placement:** If you need to use multiple LUNs for a single cube, you should try to manually balance I/O traffic between these LUNs. One way to achieve this is to partition the cube into roughly equal-sized slices based on a dimension key. Very often, the only way to achieve roughly equal balance like this is to implement a two-layer partitioning on both date and the secondary, balancing key. For more information, see the Repartitioningsection in this guide.

# Diagnosing and Optimizing

This section discusses how to troubleshoot problems and implement changes that can be made transparently in the cube structures to improve performance. Many of these changes are already documented in the Analysis Services Performance Guide. But some additional considerations apply from an operations perspective.

## Tuning Processing Data

During **ProcessData** operations, Analysis Services uses the processing thread pool for worker threads used by the storage engine.

**ProcessData** operations use three parallel threads to perform the following tasks:

* Query the data source and extract data
* Look up dimension keys in dimension stores and populate the processing buffer
* Write the processing buffer to disk when it is full

You can usually increase throughput of **ProcessData** by tuning the network stack and processing more partitions in parallel. However, you may still benefit from tuning the process pool.

To optimize these settings for the **ProcessData** phase, check your Performance Monitor counter on the object **MSOLAP: Threads** and use the following table for guidance.

|  |  |
| --- | --- |
| Situation | Action |
| **Processing pool job queue length** > 0 and  **Processing pool idle threads** = 0  for longer periods during processing. | Increase **ThreadPool\Process\MaxThreads** and then retest. |
| Both **Processing pool job queue length** > 0and **Processing pool idle threads** > 0 at same time during processing. | Decrease **CoordinatorExecutionMode** (for example, change it from -4 to -8) and then retest. |

You can use the **Processor –% Processor Time – Total** counter as a rough indicator of how much you should change these settings. For example, if your CPU load is 50 percent, you can double **ThreadPool\Process\MaxThreads** to see whether this also doubles your CPU usage. It is possible to get to 100 percent CPU load in a system without bottlenecks, though you may want to leave some headroom for growth. Keep in mind that increased parallelism of processing also has an effect on queries running at the system. Ideally, use a separate processing server or a dedicated processing time window where no one is querying Analysis Services. If this is not an option, as you dedicate more CPU power and threads to processing, less CPU will be used for query responses. Because processing consumes threads from the same pool as query subcube requests, you should also be careful that you don’t run the process thread pool dry if you process and query at the same time. If you are processing the cubes during a set processing window with no users on the box, this will of course not be an issue.

**References:**

* SQL Server 2005 Analysis Services (SSAS) Server Properties (<http://technet.microsoft.com/en-us/library/cc966526.aspx>)

### Optimizing the Relational Engine

In addition to looking at Analysis Services configurations, and settings, you can also look at the relational engine when you are planning improvements to your Analysis Services installation. This section focuses on working with relational data from SQL Server 2005, SQL Server 2008, or SQL Server 2008 R2. Although Analysis Services can be used with any OLE DB or .NET driver enabled database (such as Oracle or Teradata), the advice here may not apply to such third-party environments. However, if you are a third-party DBA, you may be able to translate the techniques discussed here to similar ones in your own environment.

#### Relational Indexing for Partition Processing

While you generally want each cube partition to touch at most one relational partition, the reverse is not true. It is perfectly viable to have to have more than one cube partition accessing the same relational partition. As an example, a relational source that is partitioned by year with a cube that is partitioned by month can still provide optimal processing performance.

If you do not have a one-to-one relationship between relational and cube partitions, you generally want an index to support the fact processing query. The best choice of index for this purpose is a clustered index; if your load strategy allows you to maintain such an index, this is what you should aim for.

When a partition processing query is supported by an index the plan should look like this.

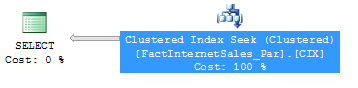


Figure 24 Supporting Measure Group processing with an index

**References:**

* Top 10 SQL Server 2005 Performance Issues for Data Warehouse and Reporting Applications (<http://sqlcat.com/top10lists/archive/2007/11/21/top-10-sql-server-2005-performance-issues-for-data-warehouse-and-reporting-applications.aspx>)
* Ben-Gan, Itzik and Lubor Kollar, *Inside Microsoft SQL Server 2005: T-SQL Querying.* Redmond, Washington: Microsoft Press, 2006.

#### Relational Indexing for Dimension Processing

If you follow a dimensional star schema design (which we recommend for large cubes), most dimension processing queries should run relatively fast and take only a tiny portion of the total cube processing time. But if you have very large dimensions with millions of rows or dimensions with lots of attributes, some performance can be still be gained by indexing the relational dimension table. To understand the best indexing strategy, it is useful to know which form dimensions processing queries take. The number of queries generated by Analysis Services depends on the attribute relationships defined in the dimension. For each attribute in the dimension, the following query is generated during processing.

**SELECT DISTINCT** <attribute>, [<related attribute> […n] ]  
**FROM** <dimension table>

Consider the following example dimension, with **CustomerID** as the key attribute.



Figure 17 – Example Customer Dimension – Attribute relationships

The following queries are generated during dimension processing.

SELECT DISTINCT Country FROM Dim.Customer

SELECT DISTINCT State, Country FROM Dim.Customer

SELECT DISTINCT City, State FROM Dim.Customer

SELECT DISTINCT Zip, City FROM Dim.Customer

SELECT DISTINCT Name FROM Dim.Customer

SELECT DISTINCT Gender FROM Dim.Customer

SELECT DISTINCT Age FROM Dim.Customer  
SELECT DISTINCT CustomerID, Name, Gender, Age, Zip FROM Dim.Customer

The indexing strategy you apply depends on which attribute you are trying to optimize for. To illustrate a tuning process, this section walks through some typical cases.

**Key attribute:** the key attribute in a dimension, in this example **CustomerID,** can be optimized by creating a unique, clustered index on the key. Typically, using such a clustered index is also the best strategy for relational user access to the table – so your DBA will be happy if you do this. In this example, the following index helps with key processing.

CREATE UNIQUE CLUSTERED INDEX CIX\_CustomerID ON Dim.Customer (CustomerID)

This will create the following, optimal query plan.

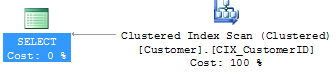


Figure 18 - A good key processing plan

**High cardinality attributes:** For high cardinality attributes, like **Name**, you need a nonclustered index. Notice the DISTINCT in the SELECT query generated by Analysis Services.

SELECT DISTINCT Name FROM Dim.Customer

DISTINCT generally forces the relational engine to perform a sort to remove duplicates in the returned dataset. The sort operation results in a plan that looks like this.

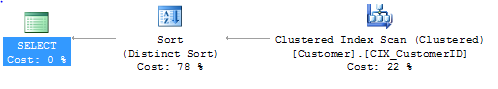


Figure 19 - Expensive sort plan during dimension processing

If this plan takes a long time to run, which could be the case for large dimension, consider creating an index that helps remove the sort. In this example, you can create this index.

CREATE INDEX IX\_Customer\_Name ON Dim.Customer (Name)

This index generates the following, much better plan.

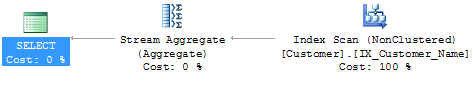


Figure 20 - A fast high cardinality attribute processing plan

**Low cardinality attributes:** For attributes that are part of a large dimension but low granularity, even the preceding index optimization may result in an expensive plan. Consider the **City** attribute in the customer dimension example. There are very few cities compared to the total number of customers in the dimension. For example, you want to optimize for the following query.

SELECT DISTINCT City, State FROM Dim.Customer

Creating a multi-column index on **City** and **State** removes the sort operation required to return DISTINCT rows – resulting in a plan very similar to the optimization performed earlier with the **Name** attribute. This is better than running the query with no indexed access. But it still results in touching one row per customer – which is far from optimal considering that there are very few cities in the table.

If you have SQL Server Enterprise, you can optimize the SELECT query even further by creating an indexed view like this.

CREATE VIEW Dim.Customer\_CityState

WITH SCHEMABINDING

AS

SELECT City, State, COUNT\_BIG(\*) AS NumRows FROM Dim.Customer

GROUP BY City,State

**GO**

CREATE UNIQUE CLUSTERED INDEX CIX\_CityState

ON Dim.Customer\_CityState (City, State)

SQL Server maintains this aggregate view on disk, and it stores only the distinct values of **City** and **State –** exactly the data structure you are looking for. Running the dimension processing query now results in the following optimal plan.

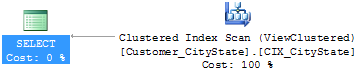


Figure 21 - Using indexed views to optimize for low-cardinality attributes

#### Overoptimizing and Wasting Time

It is possible to tune the relational engine to cut down time on processing significantly, especially for partition processing and large dimensions. However, bear in mind that every time you add an index to a table, you add one more data structure that must be maintained when users modify rows in that table. Relational indexing, like aggregation design in a cube and much of BI and data warehousing, is tradeoff between data modification speed and user query performance. There is typically a sweet spot in this space that will depend on your workload. Different people have different skills, and the perception of where that sweet spot lies will change with experience. As you get closer to the optimal solution, increased the tuning effort will often reach a point of diminishing returns where the speed of the system moves asymptotically towards the optimum balance. Monitor your own tuning efforts and try to understand when you are getting close to that flatline behavior. As tempting as full tuning exercises can be to the technically savvy, not every system needs benchmark performance.

#### Using Index FILLFACTOR = 100 and Data Compression

If page splitting occurs in an index, the pages of the index may end up less than 100 percent full. The effect is that SQL Server will be reading more database pages than necessary when scanning the index.

You can check for index pages are not full by querying the SQL Server DMV **sys.dm\_db\_index\_physical\_stats**. If the column **avg\_page\_space\_used\_in\_percent** is significantly lower than 100 percent, a FILLFACTOR 100 rebuild of the index may be in order. It is not always possible to rebuild the index like this, but this trick has the ability to reduce I/O. For stale data, rebuilding the indexes on the table is often a good idea before you mark the data as read-only.

In SQL Server 2008 you can use either row or page compression to further reduce the amount of I/O required by the relational database to serve the fact process query. Compression has a CPU overhead, but reduction in I/O operations is often worth it.

**References:**

* Data Compression: Strategy, Capacity Planning and Best Practices - <http://msdn.microsoft.com/en-us/library/dd894051%28v=sql.100%29.aspx>

#### Eliminating Database Locking Overhead

When SQL Server scans an index or table, page locks are acquired while the rows are being read. This ensures that many users can access the table concurrently. However, for data warehouse workloads, this page level locking is not always the optimal strategy – especially when large data retrieval queries like fact processing access the data.

By measuring the Perfmon counter **MSSQL:Locks – Lock Requests / Sec** and looking for **LCK** events in **sys.dm\_os\_wait\_stats**, you can see how much locking overhead you are incurring during processing.

To eliminate this locking overhead, you have three options:

* Option 1: Set the relational database in Read Only mode before processing.
* Option 2: Build the fact indexes with ALLOW\_PAGE\_LOCKS = OFF and ALLOW\_ROW\_LOCKS = OFF.
* Option 3: Process through a view, specifying the WITH (NOLOCK) or WITH (TABLOCK)query hint.

**Option 1** may not always fit your scenario, because setting the database to read-only mode requires exclusive access to the database. However, it is a quick and easy way to completely remove any lock waits you may have.

**Option 2** is often a good strategy for data warehouses. Because SQL Server Read locks (S-locks) are compatible with other S-locks, two readers can access the same table twice, without requiring the fine granularity of page and row locking. If insert operations are only done during batch time, relying solely on table locks may be a viable option. To disable row and page locking on a table and index, rebuild ALL by using a statement like this one.

ALTER INDEX ALL ON FactInternetSales REBUILD

WITH (ALLOW\_PAGE\_LOCKS = OFF, ALLOW\_ROW\_LOCKS = OFF)

**Option 3** is a very useful technique. Processing through a view provides you with an extra layer of abstraction on top of the database –a good design strategy. In the view definition you can add a NOLOCK or TABLOCK hint to remove database locking overhead during processing. This has the advantage of making your locking elimination independent of how indexes are built and managed.

CREATE VIEW vFactInternetSales

AS

SELECT [ProductKey], [OrderDateKey], [DueDateKey]

,[ShipDateKey], [CustomerKey], [PromotionKey]

,[CurrencyKey], [SalesTerritoryKey], [SalesOrderNumber]

,[SalesOrderLineNumber], [RevisionNumber], [OrderQuantity]

,[UnitPrice], [ExtendedAmount], [UnitPriceDiscountPct]

,[DiscountAmount], [ProductStandardCost], [TotalProductCost]

,[SalesAmount], [TaxAmt], [Freight]

,[CarrierTrackingNumber] ,[CustomerPONumber]

FROM [dbo].[FactInternetSales] WITH (NOLOCK)

If you use the **NOLOCK** hint, beware of the dirty reads that can occur. For more information about locking behaviors, see SET TRANSACTION ISOLATION LEVEL (<http://technet.microsoft.com/en-us/library/ms173763.aspx>) in SQL Server Books Online.

#### Forcing Degree of Parallelism in SQL Server

During **ProcessData**, a partition processing task is limited by the speed achievable from a single network connection the data source. These speeds can vary between a few thousand rows per second for legacy data sources, to around 100,000 rows per second from SQL Server. Perhaps that is not fast enough for you, and you have followed the guidance in this document and the Analysis Services Performance Guide to allow multiple partitions to process in parallel – scaling **ProcessData** nearly linearly.

We have seen customer reach 6.5 million rows per second into a cube by processing 64 partitions concurrently. Bear in mind what it takes to transport millions of rows out of a relational database every second. Each processing query will be selecting data from a big table, aggressively fetching data as fast as the network stack and I/O subsystem can deliver them. When SQL Server receives just a single request for all rows in a large fact table, it will spawn multiple threads inside the database engine to serve that it as fast as possible – utilizing all server resources. But what the DBA of the relational engine may not know, is that in a few milliseconds, your cube design is set up to ask for *another* one of those large tables – concurrently. This presents the relational engine with a problem: how many threads each query should be assigned to optimize the total throughput of the system, without sacrificing overall performance of individual processing commands. It is easy to see that race conditions may create all sorts on interesting situations to further complicate this. If parallelism overloads the Database Engine, SQL Server must resort to context switching between the active tasks, continuously redistributing the scarce CPU resources and wasting CPU time with scheduling. You can measure this happening as a high **SOS\_SCHEDULER\_YIELD** wait in **sys.dm\_os\_wait\_stats**.

Fortunately, you can defend yourself against excessive parallelism by working together with the cube designer to understand how many partitions are expected to process at the same time. You can then assign a smaller subset of the CPU cores in the relational database for each partition. Carefully designing for this parallelism can have a large impact. We have seen cube process data speeds more than double when the assigned CPU resources are carefully controlled.

You have one or two ways to partition server resources for optimal process data speeds, depending on which version of SQL Server you run.

**Instance reconfiguration** – Using **sp\_configure**, you can limit the number of CPU resources a single query can consume. For example, consider a cube that processes eight partitions in parallel from a computer running SQL Server with 16 cores. To distribute the processing tasks equally across cores, you would configure like this.

EXEC sp\_configure 'show advanced options', 1

RECONFIGURE

EXEC sp\_configure 'max degree of parallelism', 2

RECONFIGURE

Unfortunately, this is a brute-force approach, which has the side effect of changing the behavior of the entire instance of SQL Server. It may not be a problem if the instance is dedicated for cube processing, but it is still a crude technique. Unfortunately, this is the only option available to you on SQL Server 2005.

**Resource Governor** – If you run SQL Server 2008 or SQL Server 2008 R2, you can use Resource Governor to control processing queries. This is the most elegant solution, because it can operate on each data source view individually.

The first step is to create a resource pool and a workload group to control the Analysis Services connections. For example, to the following statement limits each ProcessData task to 2 CPU cores and a maximum memory grant of 10 percent per query.

CREATE RESOURCE POOL [cube\_process] WITH(

min\_cpu\_percent=0

, max\_cpu\_percent=100

, min\_memory\_percent=0

, max\_memory\_percent=100)

GO

CREATE WORKLOAD GROUP [process\_group] WITH(

group\_max\_requests=0

, importance=Medium

, request\_max\_cpu\_time\_sec=0

, **request\_max\_memory\_grant\_percent=10**

, request\_memory\_grant\_timeout\_sec=0

, **max\_dop=2**)

USING [cube\_process]

GO

The next step is to design a classifier function that recognizes the incoming cube process requests. There are several ways to recognize cube connection. One is to use the host name. Another is to use application names (which you can set in the connection string in the data source view in the cube). The following example recognizes all Analysis Services connections that use the default values in the connection string.

USE [master]

GO

CREATE FUNCTION fnClassifier()

RETURNS sysname

WITH SCHEMABINDING

AS

BEGIN

DECLARE @group SYSNAME

IF APP\_NAME() LIKE '%Analysis Services%' BEGIN

SET @group= 'process\_group'

END

ELSE BEGIN

SET @group = 'default'

END

RETURN (@group)

END

Make sure you test the classifier function before applying it in production. After you are certain that the function works, enable it.

ALTER RESOURCE GOVERNOR WITH (CLASSIFIER\_FUNCTION = [dbo].[fnClassifier]);

GO

ALTER RESOURCE GOVERNOR RECONFIGURE;

#### Loading from Oracle

Analysis Services is commonly deployed in heterogeneous environments, with Oracle being one of the main data sources. Because cubes often need to read a lot of data from the source system during processing, it is important that you use high speed drivers to extract this data. We have found that the native Oracle drivers provide a reasonable performance, especially if many partitions are processed in parallel.

However, we have also found that data can be extracted from a SQL Server data source at 5-10 times the speed of the same Oracle source. The SQL Server driver SQLNLCI is optimized for very high extraction speeds – sometimes reaching up to 80,000-100,000 rows per second from a single TCP/IP connection, depending on the source schema). We have tested processing speeds on top of SQL Server all the way to 6.1 million rows per second.

Consider that SQL Server Integration Services, part of the same SKU as Analysis Services, has a high-speed Oracle driver available for download. This driver is optimized for high-speed extraction from Oracle. We have found that the following architecture often provides faster processing performance than processing directly on top of Oracle.



Figure 22 - Fast Processing on Oracle

**References:**

* Microsoft Connectors Version 1.1 for Oracle and Teradata by Attunity - <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=6732934c-2eea-4a7f-85a8-8ba102e6b631>
* The Data Loading Performance Guide - <http://msdn.microsoft.com/en-us/library/dd425070.aspx>
  + Describes how to move lots of data into SQL Server

## Tuning Process Index

As with **ProcessData** workloads, you can often increase speed of **ProcessIndex** by running more partitions in parallel. However, if this option is not available to you, the thread settings can provide an extra benefit. When you measure CPU utilization with the counter **Processor –% Processor Time – Total** and you find utilization is less than 100 percent with no I/O bottlenecks, there is a good chance that you can increase the speed of the **ProcessIndex** phase further by using the techniques in this section.

### CoordinatorBuildMaxThreads

When a single partition is scanned, the amount of threads used to scan extents is limited by the **CoordinatorBuildMaxThreads** setting. The setting determines the maximum number of threads allocated per aggregation processing job. It is the absolute number of threads that can be run by an aggregation processing job. Keep in mind that you are still limited by the number of threads in the [process thread pool](file:///C:\Users\Beth\Documents\Vend\Docs\SQL%20White%20papers\SSASPerfGuide2008_v1_(2).docx#_Adjusting_ThreadPool_and) when processing, so increasing this value may require increasing the threads available to the process thread pool too.



Figure 29 CoordinatorBuildMaxThreads

If you are not able to drive high parallelism by using more partitions, you can change the **CoordinatorBuildMaxThreads** value. Increasing this allows you to use more threads per partition while building aggregations for each partition during the **ProcessIndex** phase

### AggregationMemoryMin and Max

As you increase parallelism of **ProcessIndex**, you may run into memory bottlenecks. On a server with more than around ten **Process ndex** jobs running in parallel, you may need to adjust **AggregationMemoryMin** and **AggregationMemoryMax** to get optimal results. For more information, see the Memory section of this guide.

## Tuning Queries

Query optimization is mostly covered in the Analysis Services Performance Guide and will generally involve design or application level changes. However, there are some optimizations that can be made in a production cube that are transparent to users and BI developers. These are described here.

### Thread Pool Tuning

Analysis Services uses the parsing thread pools, the query thread pool, and the process thread pool for query workloads on Analysis Services. A listener thread listens for a client request on the TCP/IP port specified in the Analysis Services properties. When a query comes in, the listener thread brokers the request to one of the parsing thread pools. The parsing thread pools either execute the request immediately or send the request off to the query or process thread pool.

Worker threads from the query pool check the data and calculation caches to see whether the request can be served from cache. If the request contains calculations that need to be handled by the formula engine, worker threads in the query pool are use to perform the calculation and store the results. If the query needs to go to disk to retrieve aggregations or scan a partition, worker threads from the processing pool are used to satisfy the request and store the results in cache.

There are settings in the **msmdsrv.ini** file that allow users to tune the behavior of the thread pools involved in query workloads. Guidance on using them is provided in this section.

#### Parsing Thread Pools

The Analysis Services protocol uses Simple Object Analysis Protocol (SOAP) and XMLA for Analysis (XMLA) with TCP/IP or HTTP/HTTPS as the underlying transport mechanism. After a client connects to Analysis Services and establishes a connection, commands are then forwarded from the connection’s input buffers to one of the two parsing thread pools where the XMLA parser begins parsing the XMLA while analyzing the SOAP headers to maintain the session state of the command.

There are two parsing thread pools: the short-command parsing pool and the long-command parsing pool. The short-command parsing pool is used for commands that can be executed immediately and the long-command parsing pool is used for commands that require more system resources and generally take longer to complete. Requests longer than one package are dispatched to the long-parsing thread pool, and one-package requests go to the short-parsing pool. If a request is a quick command, like a DISCOVER, the parsing thread is used to execute it. If the request is a larger operation, like an MDX query or an MDSCHEMA\_MEMBERS, it is queued up to the query thread pool.

For most Analysis Services configurations you should not modify any of the settings in the short-parsing or long-parsing thread pools. However, with the information provided here, you can imagine a workload where either the short-parsing or long-parsing thread pools run dry. We have only seen one such workload, at very high concurrency, so it is unlikely that you will need to tune the parsing thread pool. If you do have a problem with one the parsing thread pools, it will show up as values consistently higher than zero in **MSOLAP/Thread - Short parsing job queue length** or **MSOLAP/Thread – Long parsing job queue length.**

#### Query Thread Pool Settings

Although modifying the **ThreadPool\Process\MaxThreads** and **ThreadPool\Query\MaxThreads** properties can increase parallelism during querying, you must also take into account the additional impact of **CoordinatorExecutionMode** as described in the configuration section.

In practical terms, the balancing of jobs and threads can be tricky. If you want to increase parallelism, it is important to first narrow down parallelism as the bottleneck. To help you determine whether this is the case, it is helpful to monitor the following performance counters:

* **Threads\Query pool job queue length**—The number of jobs in the queue of the query thread pool. A nonzero value means that the number of query jobs has exceeded the number of available query threads. In this scenario, you may consider increasing the number of query threads. However, if CPU utilization is already very high, increasing the number of threads only adds to context switches and degrades performance.
* **Threads\Query pool busy threads**—The number of busy threads in the query thread pool. Note that this counter is broken in some versions of Analysis Services and does not display the correct value. The value can also be derived from the size of the thread pool minus the **Threads\Query pool idle threads** counter.
* **Threads\Query pool idle threads**—The number of idle threads in the query thread pool.

Any indication of queues in any of these thread pools with a CPU load less than 100 percent indicate a potential option for tuning the thread pool.

**References**

* Analysis Services Query Performance Top 10 Best Practices (<http://www.microsoft.com/technet/prodtechnol/sql/bestpractice/ssasqptb.mspx>)

### Aggregations

Aggregations behave very much like indexes in a relational database. They are physical data structures that are used to answer frequently asked queries in the database. Adding aggregations to a cube enables you to speed up the overall throughput, at the cost of more disk space and increased processing times.

Note that just as with relational indexing, not every single, potentially helpful aggregate should be created. When the space of potential aggregates grows large, one aggregate being read may push another out of memory and case disk thrashing. Analysis Services may also struggle to find the best aggregate among many matching a given query.

A good rule of thumb is that no measure group should have more than 30 percent of its storage space dedicated to aggregates. To discover which aggregates are most useful, you should collect the **query subcube verbose** event using SQL Server Profiler while running a representative workload. By summing the run time of each unique subcube you can get a good overview of the most valuable aggregations to create. For more information about how to create aggregates, see the Analysis Services Performance Guide.

There are some aggregates that you generally always want to create, namely the ones that are used directly by sets defined in the calculation script. You can identify those by tracing queries going to the cube after a process event. The first user to run a query or connect to the cube after a processing event may trigger **query subcube verbose** events that have SPID = 0 – this is the calculation script itself requesting data to instantiate any named sets defined there.

**References:**

* Analysis Services 2005 Aggregation Design Strategy - <http://sqlcat.com/technicalnotes/archive/2007/09/11/analysis-services-2005-aggregation-design-strategy.aspx>
  + Also applies to SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services
* Aggregation Manager on CodePlex - <http://bidshelper.codeplex.com/wikipage?title=Aggregation%20Manager&referringTitle=Home>
  + Makes the task of designing aggregations easier
* BIDS Helper - <http://bidshelper.codeplex>
  + Assists with many common tasks, including aggregation design

### Optimizing Dimensions

Good dimension design is key to good performance of large cubes. Your developers should design dimensions that fit the needs of the business users. However, there are some minor changes you can often make on a live system that can yield a significant performance benefit. The changes are not 100 percent transparent to business users, but they may well be acceptable or even improve the behavior of the cube. Be sure to coordinate with your BI developers to understand whether you can make any of the changes discussed here in the cube.

**Removing the (All) Level** – In some dimensions, it does not make sense to ask for the (All) level in a query. The classic example is a date dimension. If the (All) level is present, and no default has been set, a user connecting to the cube may inadvertently ask for the sum of all years. This number is rarely a useful value and just wastes resources at the server.

**Setting default members** – Whenever a user issues an MDX statement, every hierarchy not explicitly mentioned in the query uses its default member value. By default, this member is the (All) level in the hierarchy. The (All) level may not be the typical usage scenario, which will cause the user to reissue the query with a new value. It is sometimes useful to set another default value in the dimension that more accurately reflects the most common usage scenario. This can be done with a simple modification to the calculation script of the cube. For example, the following command sets a new default member in the Date dimension.

ALTER CUBE [Adventure Works]UPDATE

DIMENSION [Date], DEFAULT\_MEMBER='[Date].[Date].&[2000]'

**Scope the All Level to NULL** –Removing the (All) level and setting default members can be confusing to users of Excel. Another option is to force the (All) level to be NULL for the dimensions where querying that level makes no sense. You can use a SCOPE statement for that. For more information, see the performance guide.

**AttributeHierarchyEnabled** – This property, when set to **false**, makes the property invisible as an attribute hierarchy to users browsing the cube. This reduces the metadata overhead of the cube. Not all attributes can be disabled like this, but you may be able to remove some of them after working with the users.

Note that there are many other optimizations that can be done on dimensions, and the Analysis Services Performance Guide contains more detailed information.

### Calculation Script Changes

The calculation script of a cube contains the MDX statements that make up the nonmaterialized part of the cube. Optimizing the calculation script is a very large topic that is outside the scope of this document. What you should know it that such changes can often be made transparently to the user and that the gains, depending on the initial design, can often be substantial.

You should generally collect the performance counters under **MSOLAP:MDX** because these can be used by cube developers to determine whether there are potential gains to be had. A typical indicator of a slow calculation script is a large ratio between the **MSOLAP:MDX/Number of cell-by-cell evaluation nodes** and **MSOLAP:MDX/Number of bulk-mode evaluation nodes.**

### Repartitioning

Partitioning of cubes can be used to both increase processing and query speeds. For example, if you struggle with process data speeds, splitting up the nightly batch into multiple partitions can increase concurrency of the processing operation. This technique is documented in the Analysis Services Performance Guide.

Partitions can also be used to selectively move data to different I/O subsystems. An example of this is a customer that keeps the latest data in the cube on NAND devices and moves old and infrequently accessed data to SATA disks. You can move partition around using SQL Server Management Studio in the properties pane of the partition. Alternatively, you can use XML or scripting to move partition data by executing a query like this.

<Alter ObjectExpansion="ExpandFull"

xmlns="http://schemas.microsoft.com/analysisservices/2003/engine">

<Object>

<DatabaseID>Adventure Works DW</DatabaseID>

<CubeID>Adventure Works DW</CubeID>

<MeasureGroupID>Fact Reseller Sales</MeasureGroupID>

<PartitionID>Reseller\_Sales\_2001</PartitionID>

</Object>

<ObjectDefinition>

<Partition>

<ID>Reseller\_Sales\_2001</ID>

<Name>Reseller\_Sales\_2001</Name>

<StorageLocation>**D:\MSAS10\_50.KILIMANJARO\OLAP\2001\**</StorageLocation>

</Partition>

</ObjectDefinition>

</Alter>

Be aware that after you have changed the storage location of the partition that marks the partition as unprocessed and empty, you must reprocess the partition to physically move the data. Note that Analysis Services creates a folder under the path you specify, and this folder is named using a GUID – not easy to decode for a human. To keep track of where your moved partitions are, it is therefore an advantage to precreate folders with human-readable names to hold the data.

For large cubes, it is often a good idea to implement a “matrix” partitioning scheme: partition on both date and some other key. The date partitioning is used to selectively delete or merge old partitions. The other key can be used to achieve parallelism during partition processing and to restrict certain users to a subset of the partitions. For example, consider a retailer that operates in US, Europe, and Asia. You might decide to partition like this.



Figure 23 - Example of matrix partitioning

If the retailer grows, they may choose to split the region partitions into smaller partitions to increase parallelism of load further and to limit the worst-case scans that a user can perform. For cubes that are expected to grow dramatically, it is a good idea to choose a partition key that grows with the business and gives you options for extending the matrix partitioning strategy appropriately. The following table contains examples of such partitioning keys.

|  |  |  |
| --- | --- | --- |
| Industry | Example partition key | Source of data proliferation |
| Web Retail | Customer key | Adding customers and transactions |
| Store Retail | Store key | Adding new stores |
| Data Hosting | Host ID or rack location | Adding a new server |
| Telecommunications | Switch ID or country code or area code | Expanding into new geographical regions or adding new services |
| Computerized manufacturing | Production Line ID or Machine ID | Adding production lines or (for machines) sensors |
| Investment Banking | Stock Exchange or financial instrument | Adding new financial instruments, products, or markets |
| Retail Banking | Credit Card Number or Customer Key | Increasing customer transactions |
| Online Gaming | Game Key or Player Key | Adding new games or players |

Sometimes it is not possible to come up with a good distribution of the keys across the partitions. Perhaps you just don’t have a good key candidate that fits the description in the previous paragraph, or perhaps the distribution of the key is unknown at design time. In such cases, a brute-force approach can be used: Partition on the hash value of a key that has a high enough cardinality and where there is little skew. Users will have to touch all the hash buckets as they query the cube, but at least you can perform parallel loading. If you expect every query to touch many partitions, it is important that you pay special attention to the **CoordinatorQueryBalancingFactor** described earlier.

As you add more partitions, the metadata overhead of managing the cube grows exponentially. As a rule of thumb, you should therefore seek to keep the number of partitions in the cube in the low thousands. This affects **ProcessUpdate** and **ProcessAdd** operations on dimensions, which have to traverse the metadata dependencies to update the cube when dimensions change. For large cubes, prefer larger partitions over creating too many partitions. This also means that you can safely ignore the Analysis Management Objects warning (AMO) in Visual Studio that partition sizes should not exceed 20 million rows. We have measured the effect of large partition sizes, and found that they show negligible performance differences compared to smaller partition sizes. Therefore, the reduction in partition management overhead justifies relaxing the guidelines on partition sizes, particularly when you have large numbers of partitions.

#### Adding and Managing Partitions

Managing partitions on a large cube quickly becomes a large administrative task if done manually through SQL Server Management Studio. We recommend that you create scripts to help you manage partitioning of the cube in an automated manner instead.

The best way to manage a large cube is to use a relational database to hold the metadata about the desired partitioning scheme of the cube and use XMLA to keep the cube structure in sync with the metadata in the relational source. Every time you request metadata from a cube, you have to run a DISCOVER command - some of these commands are expensive on a large cube with many partitions. You should design partition management code to extract the metadata from the cube in a single bulk operation instead of multiple small DISCOVER commands. Note that when you use AMO (and the SQL Server Integration Services task) to access the cube, some amount of error handling is built into the .NET library. This error handling code executes DISCOVER commands and be quite chatty with the server. A good example of this is the partition add function in AMO. This code will first check to see whether the partition already exists (it raises an error if it does) using a DISCOVER command. After that it will issue the CREATE command, creating the partition in the cube. This technique turns out to be a highly inefficient way to add many new partitions on cube with thousands of existing partitions. Instead, it is faster to first read all the partitions using a single XMLA command, discover which partitions are missing ,and then run an XMLA command to create each missing partition directly – avoiding the error checking that AMO.NET does.

In summary, design partition management code carefully and test it using SQL Server Profiler to trace the commands you generate as the code runs. Because it provides so much feedback to the server, partition management code is very expensive, and it is often surprising to customers how easy it is to create inefficient operations on a big cube. That is the price of the safety net that ADOMD.NET gives you. Using XMLA to carefully manage partitions is often a better solution for a large cube.

## Locking and Blocking

Locks are used to manage concurrent operations. In an Analysis Services deployment, you’ll find that locks are used pervasively in discovery and execution to ensure the stability of underlying data structures while a query or process is running.

Occasionally, locks taken for one operation can block other processes or queries from running. A common example is when a query takes a long time to execute, and another session issues a write operation involving the same objects. The query holds locks, causing the write operation to wait for the query to complete. Additional new queries issued on other sessions are then blocked as they wait for the original query operation to complete, potentially causing the entire service to stop handling commands until the original long-running query completes.

This section takes a closer look at how locks are used in different scenarios, and how to diagnose and address any performance problems or service disruptions that arise from locking behaviors. One of the tools at your disposal is the new SQL Server Profiler lock events that were introduced in SQL Server 2008 R2 Service Pack 1 (SP1). Finally, this section looks at deadlocks and provides some recommendations for resolving them if they begin to occur too frequently.

### Lock Types

In Analysis Services, the most commonly used lock types are Read, Write, Commit\_Read, and Commit\_Write. These locks are used for discovery and execution on the server. SQL Server Profiler, queries, and dynamic management views (DMVs) use them to synchronize their work. Other lock types (such as LOCK\_NONE, LOCK\_SESSION\_LOCK, LOCK\_ABORTABLE, LOCK\_INPROGRESS, LOCK\_INVALID) are used peripherally as helper functions for concurrency management. Because they are peripheral to this discussion, this section focuses on the main lock types instead.

The following table lists these locks and briefly describes how they are used.

|  |  |  |
| --- | --- | --- |
| Lock type | Object | Usage |
| Read | All | Read locks are used for reading metadata objects, ensuring that these objects cannot be modified while they are being used.  Read locks are shared, meaning that multiple transactions can take Read locks on the same object. |
| Write | All | Write locks are used for create, alter, and update operations.  Write locks are exclusive. Exclusive locks prevent concurrent transactions from taking Read or Write locks on the object at the same time. |
| Commit\_Read | Database,  Server,  Server Proxy | Commit\_Read locks are shared, but they block write operations that are waiting to commit changes to disk.  Database Objects: Commit\_Read locks are used for the following:   * Query operations to ensure that no commit transactions overwrite any of the metadata objects used in the query. The lock is held for the duration of a query. * During processing, while acquiring Read and Write locks on other objects. * At the beginning of a session to calculate user permissions for a given database. * During some discover operations, such as those that read from a database object.   Server Objects: Commit\_Read locks are used at the beginning of a session to compute session security. Commit\_Read is also used at the start of a transaction to read the master version map.  Server Proxy Objects: Commit\_Read locks are used at the beginning of SQL, DMX, and MDX queries to prevent changes to assemblies or administrative role while the objects are retrieved. |
| Commit\_Write | Database,  Server,  Server Proxy | Commit\_Write locks are exclusive locks on an object, taken to prevent access to that object while it is being updated. Commit\_Write is the primary mechanism for ensuring “one version of the truth” for queries.  Database Objects: A Commit\_Write is used in a commit transaction that creates, updates, or deletes DDL structures in the database. In a commit transaction, Commit\_Write locks can be taken on multiple databases at the same time, assuming the transaction includes them (for example, deleting multiple databases in one transaction).  Server Object: A Commit\_Write is used to update the master version map. More information about the master version map is in the next section on server locks.  Server Proxy Object: A Commit\_Write is taken whenever there are changes to assemblies or membership of the server role. |

### Server Locks

In Analysis Services, there are two server lock objects, the Server and the Server Proxy, each used for very specific purposes.

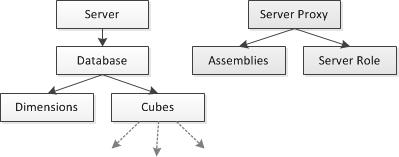


Figure 24 - Server Lock Hierarchies

The Server object takes a commit lock for operations such as processing or querying that traverse the object hierarchy. Typically, commit locks on the server are very brief, taken primarily whenever the server reads the master version map (Master.vmp) file or updates it so that it contains newer versions of objects changed by a transaction.

The master version map is a list of object identifiers and current version number for all of the major objects recognized by the server (that is, Database, Cube, Measure Groups, Partitions, Dimensions and Assemblies). Minor objects, such as hierarchy levels or attributes, do not appear in the list. The master version map identifies which object version to use at the start of a query or process. The version number of a major object changes each time you process or update it. Only the Server object reads and writes to the Master.vmp file.

When reading the master version map, the server takes a Commit\_Read to protect against changes to the file while it is being read. A Commit\_Write is taken on the Server object to update the master version map, by merging newer version information that was created in a transaction.

The Server Proxy object is used for administrative locks, for example when you make changes to the membership of the Server role or to assemblies used by the database.

### Lock Fundamentals

This section explores some of the foundational concepts that describe locking behavior in Analysis Services. Like all locking mechanisms, the purpose of locks in Analysis Services is to protect metadata or data from the effects of concurrency.

The rules or principles of how locks are used can be distilled into the following points:

* Locks protect metadata; latches protect data. Latches are lightweight locks that are rapidly acquired and released. They are used for atomic operations, like reading a data value out of system data.
* Read locks are taken for objects that feed into a transaction; Write locks are taken for objects that are changed by the transaction. Throughout the rest of this section, we’ll see how this principle is applied in different operations.

Consider the case of dimension processing. Objects like data sources and data source views that provide data to the dimension (that is, objects that the dimension depends on) take a Read lock. Objects that are changed by the operation, such as partitions, take a Write lock. When a dimension is processed, the partitions need to be unprocessed and then reprocessed, so a Write lock is taken on the partitions to update the dimension-related data within each partition.

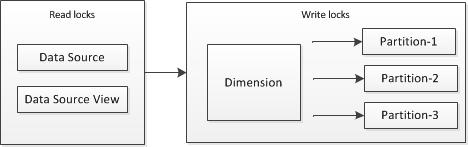


Figure 25 - Locks during dimension processing

Compare the preceding illustration to the following one, which shows only Read locks used for partition processing. Because no major objects depend on a partition, the only Write lock in play is on the partition itself.

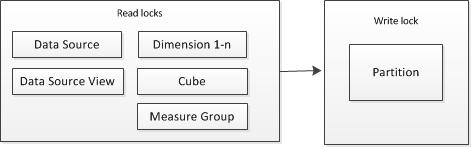


Figure 26 - Locks during partition processing

### Locks by Operation

This section explains the locks taken by different operations on the server. This list of locks is not exhaustive:

* Begin Session
* Queries
* Discover
* Processing
* DDL operations
* Rollback

Other operations such as lazy processing, sessions, and proactive caching pose interesting challenges in terms of understanding locks. The reason is discussed at the end of this section, but these other operations are not discussed in detail. Also missing from the list is writeback, Because writeback is a hybrid of query and process operations, which are covered individually.

#### Begin Session

When a session starts, Analysis Services determines which databases the user has permission to use. Performing this task requires taking a Commit\_Read lock on the database. If a database is not specified, session security is computed for each database on the server until a database is found that the session has read access to. Locks are released after session security is computed. Occasionally, the commit lock is acquired and released so quickly that it never shows up in a trace.

#### Queries

All queries run on a database and in a session. A Commit\_Read lock is taken simultaneously on the Server Proxy object and database when the query starts. The lock is held for the duration of the query.

The Commit\_Read lock on a database held by query is sometimes blocks processing operations.

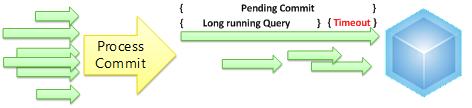


Figure 27 - Write locks blocking queries

In the preceding example, the long-running query prevents the Commit\_Write lock required by the processing operation (yellow arrow) from being taken. New readers requiring the Commit\_Read lock on the database then become blocked behind the processing command. It is this combination of events that can turn a three-second query into a three-minute (or longer) query.

#### Discovers

There are two types of discover operations that use locks: those that request metadata about instances or objects (such as MDSCHEMA\_CUBE), and those that are part of a dynamic management view (DMV) query.

The Discover method on metadata objects behaves very similarly to a query in that it takes a Commit\_Read lock on the database.

Discovers issued for DMV operations don’t take Read locks. Instead, DMVs use interlocking methods to synchronize access to system data that is created and maintained by the server. This approach is sufficient because DMV discovers do not read from the database structure. As such, the protection offered by locks is overkill. Consider DISCOVER\_OBJECT\_MEMORY\_USAGE. It is a DMV query that returns shrinkable and unshrinkable memory used by all objects on the server at given point in time. Because it is a fast read of system data directly on the server, DISCOVER\_OBJECT\_MEMORY\_USAGE uses interlocked access over memory objects to retrieve this information.

#### Processing

For processing, locks are acquired in two phases, first during object acquisition and again during schedule processing. After these first two phases, the data can be processed and the changes committed.

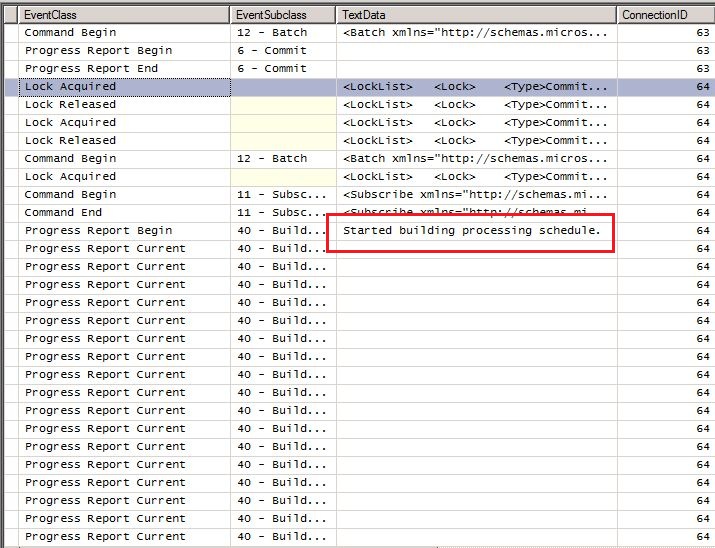
**Phase 1: Object Acquisition**

1. A Commit\_Read lock is acquired on the database while the objects are retrieved.
2. The objects used in processing are looked up.
3. The objects are identified, and then the Commit\_Read lock on the database is released.

**Phase 2: Schedule Processing**

This phase finds the objects on which the dimension or partition depends, as well as those objects that depend on it. In practice, building the schedule is an iterative process to ensure that all of the dependencies are identified. After all of the dependencies are understood, the execution workflow moves forward to the processing phase. The following list summarizes the events in schedule processing.

1. Scheduler builds a dependency graph that identifies all of the objects involved in the processing command. Based on this graph, it builds a schedule of operations.



1. Scheduler acquires a Commit\_Read on the database, and then it acquires Read and Write locks on the objects.
2. Scheduler generates the jobs and is now free to start executing the jobs.

**Phase 3: Process**

After scheduling, there are no Commit\_Read locks, just Write locks on the affected objects, and read locks on objects. Only Read and Write locks are used to do the work. Read and Write locks prevent other transactions from updating any objects used in the transaction. At the same time, objects that are related but not included in the current transaction can be updated. For example, suppose two different dimensions are being processed in parallel. If each dimension is referenced by different cubes and if each dimension is fully independent of the other, there is no conflict when the transaction is committed. Had a Commit\_Write lock been held on the database, this type of parallel processing would not be possible.

**Phase 4: Commit**

When a Commit transaction starts, a Commit\_Write lock is taken on the database and held for the duration of the transaction. No new Commit\_Read locks are accepted while Commit\_Write is pending. Any new Begin Sessions may encounter a connection timeout. New queries are queued or canceled, depending on timeout settings.

The Commit transaction waits for the query queue to drain (that is, it waits for Commit\_Read locks to be released). At this point, reading from the property settings defined on the server, it might use one of the following properties to cancel a query if it is taking too long:

**ForceCommitTimeout** cancels transactions that hold Commit\_Read locks. By default, this property is set to 30 seconds. However, it is important to remember that cancelations are not always instantaneous. Sometimes it takes several minutes to release commit locks.

**CommitTimeout** cancels transactions requesting Commit\_Write locks, in effect prioritizing queries over processing. The server uses whichever timeout occurs first. If **ForceCommitTimeout** occurs sooner than **CommitTimeout**, cancelation is called on long-running queries instead of the write operation.

**Note:** If the client application has its own retry logic, it can reissue a command in response to a connection timeout and the connection will succeed if there are available threads.

A commit transaction also takes a Commit\_Write lock on the server. This is very short and occurs when the transaction version map is merged into the master version map that keeps track of which object versions are the master versions. Because this is a Write lock, it must wait for Read locks to clear – this wait time is controlled by **ForceCommitTimeout** and **CommitTimeout**. A commit transaction creates new versions of the master objects, ensuring consistent results for all queries.

After the Master.vmp file is updated, the server deletes unused version data files. At this point, the Commit\_Write lock is released.

#### DDL Operations

For operations that create, alter, or delete metadata objects, locks are acquired once. It is similar to the processing workflow but without the scheduling phase. It finds the objects that the object depends on and locks them using Read locks, and Write locks are taken on the objects that depend on the object that is being modified. Write locks are also acquired on the objects that are created, updated, or deleted in the transaction.

#### Rollback

For rollback, there is a server latch that protects Master.vmp, but a rollback by itself does not take a Commit\_Read or Commit\_Write lock on the database. In a rollback, nothing is changing, so no commit locks are required. However, any Read and Write locks that were taken by the transaction are still held during the rollback, but they are released after the rollback has been completed. Rollback deletes the unused versions of any objects created by the original transaction.

#### Session Transactions, Proactive Caching, and Lazy Processing

In terms of locking, sessions, proactive caching and lazy processing are tricky because they have breakable locks. Pending a write operation from an administrator, Lock Manager cancels Write locks in a session, proactive caching, or lazy processing, and it lets the Write Commit prevail.

Sessions can have Write locks that don’t conflict. Snapshots and checkpoints are used to manage write operations for session cubes. The classic example is the grouping behavior in Excel where new dimensions are created in-session, on the fly, to support ad-hoc data structures. The new dimensions are always rolled back eventually, but they can be problematic during their lifetime. If an error occurs in session, the server might perform a partial rollback to achieve a stable state; sometimes these rollback operations have unintended consequences.

#### Synchronization, Backup and Restore, and Attach

The locking mechanism for synchronization, restore, and attach are already documented in the “Analysis Services Synchronization Best Practices” article on the SQLCAT web site (http://sqlcat.com). Restated from that article, the basic workflow of lock acquisition and release for synchronization is as follows.

During synchronization, a Write lock is applied before the merge of the files on the target server, and a read commit lock is applied to the source database while files are transferred.

* The Write lock is applied on the target server (when a target database exists), preventing users from querying and/or writing in database. The lock is released from the target database after the metadata is validated, the merge has been performed and the transaction is committed.
* The read commit lock is taken on the source server to prevent processing from committing new data, but it allows queries to run, including other source synchronization. Because of this, multiple servers can be synchronized at the same time from the same source. The lock is released at about the same moment as the Write lock, as it is taken in this distributed transaction.

For synchronization only, there is also an additional Commit\_Write lock while the databases are merged. This lock can be held for a long period of time while the database is created and overwritten.

For backup, there is only a Commit\_Read on the database.

A restore operation uses a more complex series of locks. It acquires a Write lock on the database that is being restored, ensuring the integrity of the database as its being restored, but blocking queries while Restore is being processed. Best practice recommendations suggest using two databases with different names so that you can minimize query downtime.

The basic workflow for all of these operations is as follows:

1. Write locks are taken on the database that is being synchronized, restored, or attached.
2. Files are extracted and changes committed while processing.
3. The locks are released.

### Investigating Locking and Blocking on the Server

This section describes the tools and techniques for monitoring locks and removing locking problems. One of the tools discussed is the new lock events that are introduced in SQL Server 2008 R2 SP1. The other tool is DMVs (and the DISCOVER\_LOCKS schema rowset in particular), which are discussed with emphasis on how it fits into a troubleshooting scenario.

Locking and blocking problems manifest themselves in a server environment in different ways. As with all performance problems, this one is a matter of degree. You might have a blocking issue in your environment that resolves so quickly, it never registers as a problem that requires a solution. At the other end of the spectrum, blocking can become so severe that users are locked out of the system. Connection requests fail; queries either time out or fail to start altogether.

In these extreme scenarios, it is difficult to clearly diagnose the problem if you do not have an available thread to connect SQL Server Profiler or if you are unable to run a DMV query that tells you what is going on. In this case, the only way to confirm a locking problem is to work with a Microsoft support engineer to analyze a memory dump. The engineer can tell you whether your server unavailability is due to locks, indicated by **PCLockManager::Wait** on multiple threads.

#### Using DMV Queries and XMLA to Cancel a Blocked Transaction

If your situation is less dire, you can use a DMV query and XML for Analysis (XMLA) to unblock your server. Given a free thread for processing a new connection request, you can connect to the server using an administrator account, run a DMV query to get the list of locks, and then try to kill the blocking SPID by running this XMLA command.

<Cancel xmlns="<http://schemas.microsoft.com/analysisservices/2003/engine>">

<SPID>nnnn</SPID>

</Cancel>

To get the SPID, you can use SQL Server Management Studio to connect to the server and then issue a DISCOVER\_LOCKS statement as an MDX query.

Select \* from $SYSTEM.DISCOVER\_LOCKS

This DMV query returns a snapshot of the locks used at specific point in time. DISCOVER\_LOCKS returns a rowset directly from Lock Manager. As such, the data you get back might not be as clear or easy to follow as the trace events in SQL Server Profiler. The following example contains a snapshot of the locks held during a query but no information about the query itself.

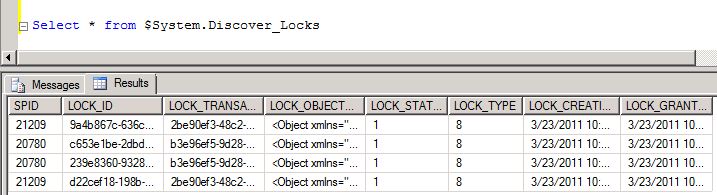


Figure 28 - Output of $system.discover\_locks

Unfortunately, there is no mitigation for this. You cannot run DISCOVER\_LOCKS in conjunction with other DMV statements to get additional insight into the timing of the lock acquisition-release lifecycle relative to the transactions running on your server. You can only run the MDX SELECT statement and then act on the information it provides.

#### Using SQL Server Profiler to Analyze Lock Contention

SQL Server Profiler offers numerous advantages over DMV queries in terms of depth and breadth of information, but it comes at a cost. It is often not feasible to run SQL Server Profiler in a production environment because of the additional resource demands it places on server. But if you can use it, and if you are running SQL Server 2008 R2 SP1, you can add the new Lock Acquired, Lock Released, and Lock Waiting events to a trace to understand the locking activity on your server.

SQL Server 2008 R2 SP1adds the following events to the Locks event category in SQL Server Profiler: Lock Acquired, Lock Released, Lock Waiting.

Like other lock events, they are not enabled by default. You must select the **Show all columns** check box before you can select events in the Locks category. To get an idea of how locks are acquired and released in the course of a transaction be sure to add Command Begin and Command End to the trace. If you want to view the MDX executed, you should also include the Query Begin and Query End.

In Analysis Services, it is normal to see a large number of acquired and released lock events. Every transaction that includes Read or Write operations on a major object requests a lock to perform that action. Lock Waiting is less common, but by itself is not symptomatic of a problem. It merely indicates that a queue has formed for transactions that are requesting the same object. If the queue is not long and operations complete quickly, the queue drains and query or processing tasks proceed with only a small delay.

The following illustration shows a Lock Waiting event and the XML structures that identify which transaction currently holds a lock on the object, and which transactions are waiting for that same object. In the illustration:

* **<HoldList>** shows which transaction is currently holding a Commit\_Write lock on the database.
* **<WaitList>** indicates that another transaction is waiting for a Commit\_Write lock on the same database.

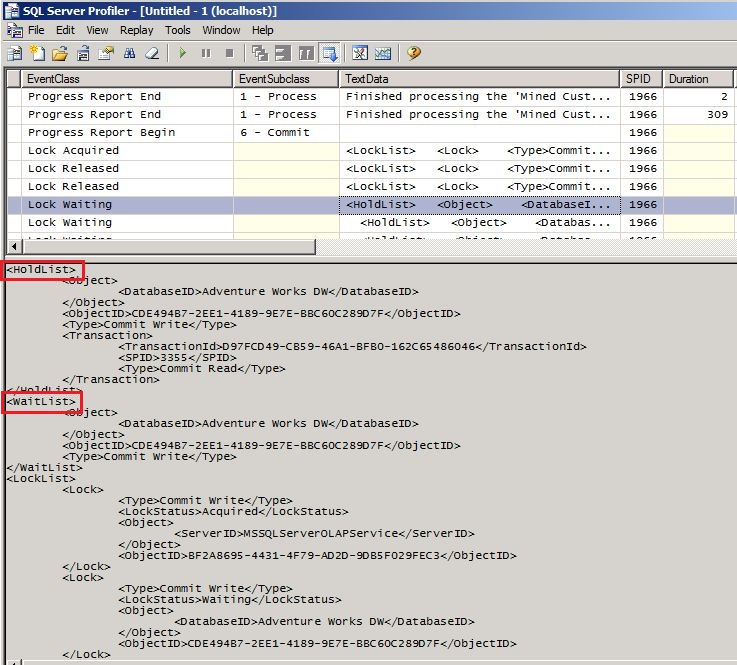


Figure 29 – SQL Server Profiler output of lock events

### Deadlocks

A deadlock in Analysis Services is lock contention between two or more sessions, where operations in either session can’t move forward because each is waiting to acquire a lock held by the other session.

Analysis Services has deadlock detection but it only works for locks (Read, Write, Commit\_Read, Commit\_Write). It won’t detect deadlocks for latches. When a deadlock is detected, Analysis Services stops the transaction in one of the sessions so that the other transaction can complete.

Deadlocks are typically a boundary case. If you can reproduce it, you can run a SQL Server Profiler trace and use the Deadlock event to determine the point of contention. In SQL Server Profiler, a deadlock looks like the following:

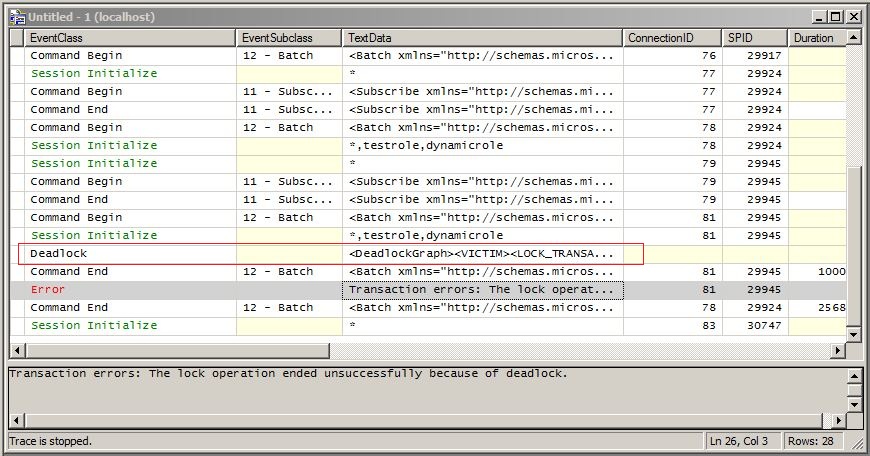


Figure 30 - Deadlock events in profiler

A deadlock event uses an XML structure called a deadlockgraph to report on which sessions, transactions, and objects created the event. The <VICTIM> node in the first few lines identifies which transaction was sacrificed for the purpose of ending the deadlock. The remainder of the graph enumerates the lock type and status for each object. In the following example, you can see which objects are requested and whether the lock was granted or waiting.

<DeadlockGraph>

<VICTIM>

<LOCK\_TRANSACTION\_ID>E0BF8927-F827-4814-83C1-98CA4C7F5413</LOCK\_TRANSACTION\_ID>

<SPID>29945</SPID>

</VICTIM>

<LOCKS>

<Lock>s

<LOCK\_OBJECT\_ID><Object><DatabaseID>FoodMart 2008</DatabaseID><DimensionID>Promotion</DimensionID></Object></LOCK\_OBJECT\_ID>

<LOCK\_ID>326321DF-4C08-43AA-9AEC-6C73440814F4</LOCK\_ID>

<LOCK\_TRANSACTION\_ID>E0BF8927-F827-4814-83C1-98CA4C7F5413</LOCK\_TRANSACTION\_ID>

<SPID>29945</SPID>

<LOCK\_TYPE>2</LOCK\_TYPE> ---- Lock\_Type 2 is a Read lock

<LOCK\_STATUS>1</LOCK\_STATUS> ---- Lock\_Status 1 is ‘acquired’

</Lock>

<Lock>

<LOCK\_OBJECT\_ID><Object><DatabaseID>FoodMart 2008</DatabaseID> <DimensionID>Product</DimensionID></Object></LOCK\_OBJECT\_ID>

<LOCK\_ID>21C7722B-C759-461A-8195-1C4F5A88C227</LOCK\_ID>

<LOCK\_TRANSACTION\_ID>E0BF8927-F827-4814-83C1-98CA4C7F5413</LOCK\_TRANSACTION\_ID>

<SPID>29945</SPID>

<LOCK\_TYPE>2</LOCK\_TYPE> ---- Lock\_Read

<LOCK\_STATUS>0</LOCK\_STATUS> ---- waiting on Product, which is locked by 29924

</Lock>

<Lock>

<LOCK\_OBJECT\_ID><Object><DatabaseID>FoodMart 2008</DatabaseID> <DimensionID>Promotion</DimensionID></Object></LOCK\_OBJECT\_ID>

<LOCK\_ID>7F15875F-4CCB-4717-AE11-5F8DD48229D0</LOCK\_ID>

<LOCK\_TRANSACTION\_ID>1D3C42F3-E875-409E-96A0-B4911355675D</LOCK\_TRANSACTION\_ID>

<SPID>29924</SPID>

<LOCK\_TYPE>4</LOCK\_TYPE> ---- Lock\_Write

<LOCK\_STATUS>0</LOCK\_STATUS> ---- waiting on Promotion, which is locked by 29945

</Lock>

<Lock><LOCK\_OBJECT\_ID><Object><DatabaseID>FoodMart 2008</DatabaseID> <DimensionID>Product</DimensionID></Object></LOCK\_OBJECT\_ID>

<LOCK\_ID>96B09D08-F0AE-4FD2-8703-D33CAD6B90F1</LOCK\_ID>

<LOCK\_TRANSACTION\_ID>1D3C42F3-E875-409E-96A0-B4911355675D</LOCK\_TRANSACTION\_ID>

<SPID>29924</SPID>

<LOCK\_TYPE>4</LOCK\_TYPE> ---- Lock\_Write

<LOCK\_STATUS>1</LOCK\_STATUS> ---- granted

</Lock><

/LOCKS>

</DeadlockGraph>

Deadlocks should be rare events, if they occur at all. Persistent deadlocks indicate a need for redesigning your processing strategy. You might need to speed up processing by making greater use of partitions, or you might need to take a closer look at other processing options or schedules to see whether you can eliminate the conflict. For more information about these recommendations, see the Analysis Services Performance Guide (<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3be0488d-e7aa-4078-a050-ae39912d2e43&displaylang=en>).

**References:**

* Analysis Services Synchronization Best Practices -<http://sqlcat.com/technicalnotes/archive/2008/03/16/analysis-services-synchronization-best-practices.aspx>
* Deadlock Troubleshooting in SQL Server Analysis Services (SSAS) - <http://blogs.msdn.com/b/sql_pfe_blog/archive/2009/08/27/deadlock-troubleshooting-in-sql-server-analysis-services-ssas.aspx>
* SSAS: Processing, ForceCommitTimeout and "the operation has been cancelled" - <http://geekswithblogs.net/darrengosbell/archive/2007/04/24/SSAS-Processing-ForceCommitTimeout-and-quotthe-operation-has-been-cancelledquot.aspx>
* Locking and Unlocking Databases (XMLA) - <http://msdn.microsoft.com/en-us/library/ms186690.aspx>

## Scale Out

Analysis Services currently supports up to 64 cores in a scale-up configuration. If you want to go beyond that scale, you will have to design for scale-out. There are also other drivers for scale-out; for example, it may simply be cheaper to use multiple, smaller machines to achieve high user concurrency. Another consideration is processing and query workloads. If you expect to spend a lot of time processing or if you are designing for real time, it is often useful to scale-out the processing on a different server than the query servers.

Scale-out architectures can also be used to achieve high availability. If you have multiple query servers in a scale-out farm, some of them can fail but the system will remain online.

Although the full details of designing a scale-out Analysis Services farm is outside the scope of this guide, it is useful to understand the tradeoffs and potential architectures that can be applied.

### Provisioning in a Scaled Out Environment

In a scale-out environment you can use either read-only LUN and the attach and detach functionality of SQL Server 2008 Analysis Services or SAN snapshots (which can also be used in SQL Server 2005 Analysis Services) to host multiple copies of the same database on many machines.

When you update a scale-out read-only farm, a Windows volume has to be dismounted and mounted every time you update an Analysis Services database. This means that no matter which disk technology you use the scale out, the smallest unit you can update without disturbing other cubes is a Windows volume. If you have multiple databases in the scale-out environment, it is therefore an advantage to have each database live on its own Windows volume. This separation allows you to update the databases independently of each other if you are running SQL Server 2008 Analysis Services and SQL Server 2008 R2 Analysis Services.

### Scale-out Processing Architectures

There are basically three different architectures you can use in a scale-out configuration:

* Dedicated processing architecture
* Query/processing flipping architecture
* ROLAP

The three architectures have different tradeoffs that you have to consider, which this section describes. Note: It is also possible to combine these architectures in different hybrids, but that is outside the scope of this document. Understanding the tradeoffs for each will help you make the right design decisions.

#### Dedicated Processing Architecture

In the dedicated processing architecture, an instance of Analysis Services is reserved to process all new, incoming data. After processing is done, the result is copied to query servers. The advantage of this architecture is that the query servers can respond to the queries without being affected by the processing operation. A lock is required only when data is updated or added to the cube.



Figure 31 - Dedicated Processing Architecture

In a dedicated processing architecture, consider how to get the data from the processing instance to the query servers. There are several ways to achieve this.

**Analysis Services Cube Synchronization:** By using this built-in Analysis Services functionality, you can move the delta data directly to the query servers.

**Robocopy or NiceCopy:** By using a high-speed copying program, you can quickly synchronize each query instance with its own copy of the changed data. This method is generally faster than cube synchronization, but it requires you to set up your own copy scripts.

**SAN Snapshots or Storage Mirrors:** Using SAN technology, it is possible to automatically maintain copies of the data LUN on the processing servers. These copies can then be mounted on the query servers when the data is updated.

**SAN Read -Only LUN:** Using this technique, which is available only in SQL Server 2008 and SQL Server 2008 R2, you can use read-only LUN to move the data from the processing instance to the query servers. A read-only LUN can be shared between multiple servers, and hence enables you to use more than query server on the same physical disk.

Both SAN snapshots and read-only LUN strategies may require careful design of the storage system bandwidth. If your cube is small enough to fit in memory, you will not see much I/O activity and this technique will work very well out of the box. However, if the cube is large and cannot fit in memory, Analysis Services will have to do I/O operations. As you add more and more query servers to the same SAN, you may end up creating a bottleneck in the storage processors on the SAN to serve all this I/O. You should make sure that the SAN is capable of supporting the required throughput. If the I/O throughput is not sufficient, you may end up with a scale-out solution that performs worse than a scale-up.

If you are worried about I/O bandwidth, the Robocopy/NiceCopy solution or the cube synchronization solution may work better for you. In these solutions you can have dedicated storage on each query server . However, you have to make sure there is enough bandwidth on the network to run multiple copies over the network. You may have to use dedicated network cards for such a setup.

The dedicated processing architectures can also be used to achieve high availability. However, you need a way to protect the processing server, to avoid a single point of failure. You can have either a standby processing server or an extra (disabled) instance on one of the query servers that can be used to take over the role of processing service in the case of hardware failure. Another alternative is to use clustering on the processing server ,although this may waste hardware resources on the passive node.

**References:**

* Scale-Out Querying for Analysis Services with Read-Only Databases - <http://sqlcat.com/whitepapers/archive/2010/06/08/scale-out-querying-for-analysis-services-with-read-only-databases.aspx>
* Sample Robocopy Script to Synchronize Analysis Services Databases - <http://sqlcat.com/technicalnotes/archive/2008/01/17/sample-robocopy-script-to-customer-synchronize-analysis-services-databases.aspx>
* Scale-Out Querying with Analysis Services - <http://sqlcat.com/whitepapers/archive/2007/12/16/scale-out-querying-with-analysis-services.aspx>
* Scale-Out Querying with Analysis Services Using SAN Snapshots - <http://sqlcat.com/whitepapers/archive/2007/11/19/scale-out-querying-with-analysis-services-using-san-snapshots.aspx>
* Analysis Services Synchronization Best Practices - <http://sqlcat.com/technicalnotes/archive/2008/03/16/analysis-services-synchronization-best-practices.aspx>
* SQL Velocity – Scalable Shared Data base - <http://sqlvelocity.typepad.com/blog/2010/09/scalable-shared-data-base-part-1.html>

#### Query/Processing Flipping Architecture

The dedicated processing server architecture solves most scale-out cases. However, the time required to move the data from the processing server to the query servers may be restrictive if updates happen at intervals shorter than a few hours. Even with SAN snapshots or read-only LUN, it will still take some time to dismount the LUN, set it online on the query server, and finally mount the updated cube on the query servers. In the query/processing flipping architecture, each instance of Analysis Services performs its own processing, as illustrated in the following figure.



Figure 32 - Query and processing flipping

Because each server does it own processing, it is possible that some servers will have more recent data than others. This means that one user executing a query may get a later version of the data than another use executing the same query concurrently. However, for many scenarios where you are going near-real time, such a state of loose synchronization may is an acceptable tradeoff. If the tradeoff is not acceptable, you can work around it with careful load balancing – at the price of adding some extra latency to the updates.

In the preceding diagram, you can see that the source system receives more processing requests than in the dedicated processing architecture. You should scale the source accordingly and consider the network bandwidth required to read the source data more than once.

The query/processing flipping architecture also has a build in high availability solution. If one of the servers fails, the system can remain online but with additional load on the rest of the servers.

### Query Load Balancing

In any scale-out architecture with more than one query server, you need to have a load balancing mechanism in place. The load balancing mechanism serves two purposes. First, it enables you to distribute queries equally across all query servers, achieving the scale-out effect. Second, it enables you to selectively take query servers offline, gracefully draining them, while they are being refreshed with new data.

When you use any load-balancing solution, be aware that the data caches on each of the servers in the load-balancing architecture will be in different states depending on the clients it is currently serving. This results in differences in response times for the same query, depending on where it executes.

There are several load balancing strategies to consider. These are treated in the following subsections. As you choose the load balancer, bear in mind the granularity of the load balancing and how this affects the process to query server switching. This is especially important if you use a dedicated processing architecture. For example, the Windows Network Load Balancing solution balances users between each Analysis Services Instance in the scale-out farm. This means that when you have to drain users from a query server and update the server with the latest version of the cube, the entire instance has to be drained. If you host more than one database per instance, this means that if one database is updated the other databases in the same instance must also be taken offline. Client load balancing and Analysis Services Load Balancer may be better solutions for you if you want to load-balance databases individually.

#### Client Load Balancing

In the client load balancing, each client knows which query server it will use. Implementing this strategy requires client-side code that can intelligently choose the right query server and then modify the connection string accordingly. Excel add-ins are an example of this type of client-side code. Note that you will have to develop your own load balancer to achieve this.

#### Hardware-Level Load Balancing

Using technologies such as load balancers from F5, it is possible to implement load balancing directly in the network layer of the architecture. This makes the load balancing transparent to both Analysis Services and the client application. If you choose to go down this route, make sure that the load-balance appliance enables you to affinitize client connections. When clients create session objects, state is stored on Analysis Services. If a later client request, relying on the same session state, is redirected to a different server, the OLE DB provider throws an error. However, even if you run affinity, you may still have to force clients off the server when processing needs to commit. For more information about the **ForceCommitTimeout** setting, see the locking section.

#### Windows Network Load Balancing

The Microsoft load-balancing solution is Network Load Balancing (NLB), which is a feature of the Windows Server operating system. With NLB, you can create an NLB cluster of Analysis Services servers running in multiple-host mode. When an NLB cluster of Analysis Services servers is running in multiple-host mode, incoming requests are load balanced among the Analysis Services servers.

#### Analysis Services Load Balancer

Analysis Services is used extensively inside Microsoft to serve our business users with data. As part of the initiative to scale out our Analysis Services farms a new load balancing solution was built. The advantages of this solution are that you can load balance on the database level and that you use a web API to control each database and the users connected to it. This customized Analysis Services load balancing solution also allows fine control over the load balancing algorithm used. Be aware that moving large datasets over the web API has a bandwidth overhead, depending on how much data is requested by user queries. Measure this bandwidth overhead as part of the cube test phase.

**References:**

* Analysis Services Load Balancing Solution - <http://sqlcat.com/technicalnotes/archive/2010/02/08/aslb-setup.aspx>

### ROLAP Scale Out

With the query/processing architecture you can get the update latency of cubes down to around 30 minutes, depending on workload. But if you want to refresh data faster than that, you have to either go fully ROLAP or use a hybrid of one of the strategies discussed earlier and ROLAP partitions.

In a pure ROLAP setup, you only process the dimensions and redirect all measure group queries directly to a relational database. The following diagram illustrates this.



Figure 33 - ROLAP scale-out

In a ROLAP system like this, you have to consider the special requirements mentioned later in this document. You should also make sure that your relational data store is scaled to support multiple Analysis Services query servers.

An interesting hybrid between MOLAP and ROLAP can be built by combining the ROLAP scale-out with either the dedicated processing architecture or the query/processing flipping architecture. You can store data that changes less frequently in MOLAP partitions, which you either process or copy to the query servers. Data that changes frequently can be stored in ROLAP partitions, redirecting queries directly to the relational source. Such a setup can achieve very low update latencies, all the way down to a few seconds, while maintaining the benefits of MOLAP compression.

**References:**

* Analysis Services ROLAP for SQL Server Data Warehouses - <http://sqlcat.com/whitepapers/archive/2010/08/23/analysis-services-rolap-for-sql-server-data-warehouses.aspx>

# Server Maintenance

When you move an Analysis Services instance to production, there are some regular maintenance tasks you should configure. This section describes those tasks.

## Clearing Log Files and Dumps

During server operations, Analysis Services generates a log file containing data about the operation. This log file is located in the folder described by the LogDir in Msmdsrv**.**ini – the default location being <Install dir>\OLAP\Log. This log grows extremely slowly and you should generally not need to clean it up. If you *do* need to reclaim the disk spaced used by the log file, you have to stop the service to delete it.

In the **<**Install Dir>\OLAP\log folder, you may also find files with extension \*.mdmp. These are minidump files generated by the Analysis Services and are typically a few megabytes each. These files get generated when undetected deadlocks happen inside the process or when there is a problem with the service. The files are used by Microsoft Support to investigate stability issues and errors in the server. If you are experiencing any such behavior, you should collect these minidump files for use during case investigation. Periodically check for these files, and clean them up after any Microsoft Support case you have open is resolved.

## Windows Event Log

Analysis Services uses the Windows event logs to report server errors, warnings, and information. The Application Log is used for most messages, but the System Log is also used for events that are related to Service Manager.

Depending on your server configuration, event logs may be configured to be cleaned manually. Make sure that this is a regular part of your maintenance. Alternatively, you can configure the event log to overwrite older events when the log is close to full. In both cases, make sure you have enough disk space to hold the full event log. The following illustration shows how to configure the event log to overwrite older events.

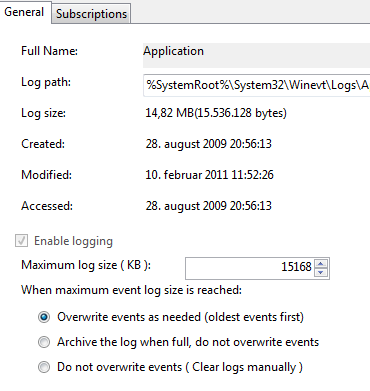


Figure 34 - Recycling the event log

## Defragmenting the File System

As described in the I/O section, there can be an advantage in defragmenting the files storing a cube, especially after a lot of changes to the partitions and dimensions. Running disk defrag will have a measurable impact on your disk subsystem performance, and depending on the hardware you run on, this may affect user response time. You can consider running defragmentation on the server during off-peak hours or in batch windows.

Note that the defrag utility retains the work done, even when it does not run to completion. This means that you can do partial defragmentation spread over time.

## Running Disk Checks

Running disk checks (using ChkDsk.exe) on the Analysis Services volume gives you the confidence that no undetected I/O corruption has occurred. How often you want to do this depends on how often you expect the I/O subsystem to create such errors without detecting them – this varies by vendor and disk model.

Note that ChkDsk.exe can run for a very long time on a large disk volume, and that it will have an impact on your I/O speeds. Because of this, you may want to use a SAN snapshot of the LUN and run ChkDsk.exe on another machine that mounts the snapshot.

In both cases, you should be able to detect disk corruption without touching the live system. If you detect irreparable corruption, you should consider restoring the backups as per the previous section.

**References:**

* Chkdsk (<http://technet.microsoft.com/en-us/library/bb491051.aspx>)

# Special Considerations

Using certain features of Analysis Services cubes can lead you down some design paths that require extra attention to succeed. This section describes these special scenarios and the considerations that apply when you encounter them.

## ROLAP and Real Time

This section deals with issues that are specific to BI environments that do not have clearly defined batch windows for loading data. If you have a cube that updates data at the same time that ETL jobs are running on the source data or while users are connected, you need to pay special attention to certain configuration parameters.

As described in the Locking and Blocking section, processing operations generally take an instance-wide lock. This will typically prevent you from designing MOLAP systems that are updated more frequently than approximately every 30 minutes . Such a refresh frequency may not be enough, and if this is your scenario, ROLAP is the path forward, and you should be aware of the special considerations that apply. You should also be aware that a ROLAP partition can put significant load on the underlying relational source, which means you should involve the DBA function to understand this workload and tune for it.

### Cache Coherency

By default, the storage engine of Analysis Services caches ROLAP subcubes in the same way it caches MOLAP subcubes. If the relational data changes frequently, this means that queries that touch ROLAP partitions may use a combination of the relational source and the storage engine cache to generate the response. If the relational source has changed since the cache entry was added to the storage engine, – this combination of source data can lead to results that are transactionally inconsistent from the perspective of the user because they represent an intermediate state of the system. There are of course ways to resolve this coherency issue, depending on your scenario.

**Changing the connection string–** It is possible to add the parameter **Real Time OLAP=true** to the connection string when the cube is accessed. Setting this parameter to **true** causes all relevant storage caches to be refreshed for every query run on that connection – including the caches generated by MOLAP queries. Note that this change can cause a significant impact on both query performance and concurrency. You should test it carefully. However, it gives you the most up-to-date results possible from the cube, because Analysis Services is essentially used as a thin MDX wrapper on top of the relational source in this mode.

**Blowing away caches** **at regular intervals** – you can either use an XMLA script to clear the cache or you can use query notifications (set in the ProActive caching properties of the partition). This allows you to clear the storage engine caches at regular intervals. Assuming you time this cache clearing with relational data loads, this gives users a consistent view that is updated every time the cache is cleared. Although this does not give you the same refresh frequency as the **Real Time OLAP=true** setting, it does have a smaller impact on user query performance and concurrency.

As you can see from these two options, going towards real-time cubes requires you to carefully consider tradeoffs between refresh frequencies and performance. Full coherency is possible but expensive. However, you can get a loose coherency that is much cheaper. Analysis Services supports both paradigms.

### Manually Set Partition Slicers on ROLAP Partitions

When Analysis Services processes the index on a MOLAP partition, it collects data about the attributes in that partition. Assuming the data matches only one attribute value, an automatic slicer is set on the partition, eliminating it from scans that do not include that attribute value. For example, if you process a partition that has data from December 2008 only, Analysis Services detects this slicing and only accesses that partition when queries request data in that time range.

Because ROLAP data resides outside of Analysis Services, the automatic slicer functionality is not used. Unless you set the slicer manually (which can be done from both Visual Studio and SQL Server Management Studio) every query has to touch every ROLAP partition. It is therefore a good practice to always set slices on ROLAP partitions when they are available.

### UDM Design

When you design for ROLAP access, it is generally a good idea to keep the UDM as simple as possible. The gives the relational engine the best possible conditions for optimizing for the query workload. The following table lists some optimizations you should consider when switching a cube to ROLAP mode.

|  |  |
| --- | --- |
| Existing feature usage | ROLAP redesign |
| Reference dimensions | Switch to a pure star schema to eliminate unnecessary joins and provide relational engine with optimal conditions for query execution. |
| Parent/child dimensions | Normalize the parent-child dimension (for more information about how to do this, see the references for this section). |
| Many-to-many dimensions | Reduce intermediate table sizes using matrix compression. |
| Query binding of partitions | Switch to table binding. Consider binding to a view if queries are needed. |
| Query binding of dimensions | Implement the result of the query in the relational source instead. |
| Aggregates | Consider reducing the number of aggregates. Be aware of conditions and features in the relational engine so that you fully understand the tradeoffs. For example, in SQL Server 2008 R2, it is often a good idea to focus on aggregates that are targeted only at the leaf level and/or **[all]** level of attributes). |
| MDX calculations | Optimize carefully, and avoid cell-by-cell operations in large ROLAP partitions. |
| ROLAP dimensions | If at all possible, use MOLAP dimensions. MOLAP dimensions have much better performance than ROLAP dimensions and if you run regular **ProcessAdd** operations, you can keep them up to date at short refresh intervals. |

You should work closely with the BI developers when troubleshooting ROLAP cubes. It is imperative to get the design right and follow the guidance in the Analysis Services Performance Guide.

**References:**

* Analysis Services Parent-Child Dimension Naturalizer on CodePlex –<http://pcdimnaturalize.codeplex.com/>
  + Also available from BIDS helper: <http://bidshelper.codeplex.com>
* Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques – <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3494E712-C90B-4A4E-AD45-01009C15C665&displaylang=en>
* Analysis Services ROLAP for SQL Server Data Warehouses- <http://sqlcat.com/whitepapers/archive/2010/08/23/analysis-services-rolap-for-sql-server-data-warehouses.aspx>

### Dimension Processing

Real-time dimension processing can present a special challenge. In a batch-style warehouse, you are in control of when inserts and updates happen – which means you can typically process the dimension after the relational source is done refreshing data. But if you are designing a cube on top of a real-time source, the relational data may change while you are processing a dimension. Dimension processing is by default executed as many concurrent SQL Server queries, as described in the Optimizing Processing section. Consider this sequence of events:

1. The customer dimension contains customers from all of the United States, but no customers from the rest of the world.
2. Dimension **ProcessAdd** starts.
3. Analysis Services sends query **SELECT DISTINCT Zip, City FROM Dimension** to the relational source. This query reads all current attribute **Zip** and **City** values.
4. The relational source inserts a new row, **City** = **Copenhagen**, in **Country = Denmark.**
5. Analysis Services, reading the next level of the hierarchy, sends the query **SELECT DISTINCT City, Country**.
6. The **City** member **Copenhagen** Is returned in the second query, but because it was not returned in the first, Analysis Services throws an error.

While this scenario may sound uncommon, we have seen it at several customers that design real-time systems. There are some ways to avoid these conditions.

**ByTable processing-** By setting the **ProcessingGroup** property of the dimension to be **ByTable** you will change howAnalysis Services behaves during dimension processing. Instead of sending multiple SELECT DISTINCT queries, the processing task will instead request the entire table with one query. This allows you to get a consistent view of the dimension, even under concurrency in the relational source. However, this setting has a drawback, namely that you will need to keep all the source dimension data in memory while the dimension is processing. If the server is under memory while this happens, paging can occur, which may cause a slowdown of the entire system.

**MARS and Snapshot -** If you are processing on top of a SQL Server data source, you can use Multiple Active Result Sets (MARS) and snapshot isolation to process the dimension and get a consistent view of the data even under updates.

Configuring MARS and Snapshot processing requires a few configuration changes to the data source view and relational database. First, in the data source properties, change the data source view to use snapshot isolation.

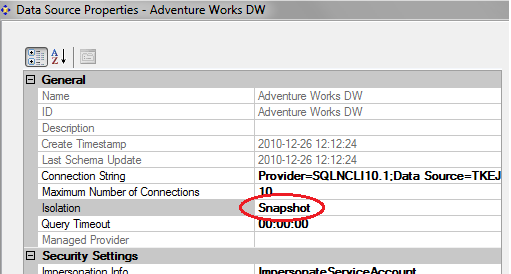


Figure 35 - Setting the Data Source to Snapshot

Second, enable MARS in the connection string of the data source view.

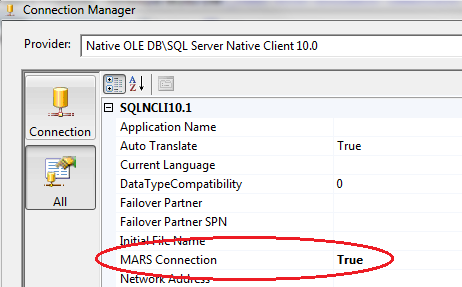


Figure 36 - Setting MARS in a DSV

And finally, enable either snapshot or read committed snapshot isolation in the SQL Server database.

ALTER DATABASE [Database]

SET READ\_COMMITTED\_SNAPSHOT ON

Processing now uses MARS, and snapshots generate a consistent view of the dimension during processing.

Understand that maintaining the snapshot during processing, as well as streaming the data through MARS, does not provide the same performance as the default processing option.

**Maintaining consistency relationally –** If you want to both maintain the processing speed and avoid memory consumption in the Analysis Services service, you have to design your data model to support real-time processing. There are several ways to do this, including the following:

* Add a timestamp to the rows in the dimension table that shows when they are inserted. During processing, only read the rows higher than a certain timestamp.
* Create a database snapshot of the relational source before processing.
* Manually create a copy of the source table before processing on top of the copy. The original can then be updated while the copy is being accessed by the cube.

**References:**

* Multiple Active Result Sets (MARS) - <http://msdn.microsoft.com/en-us/library/ms345109(v=sql.90).aspx>
* Using ByAttribute or ByTable Processing Group Property with Analysis Services 2005 - <http://blogs.msdn.com/b/sqlcat/archive/2007/10/19/using-byattribute-or-bytable-processing-group-property-with-analysis-services-2005.aspx?wa=wsignin1.0>

## Distinct Count

Distinct count measures behave differently than other measures. Because data in a distinct count measure is not additive, a lot more information must be stored on disk. According to best practice, a measure group that has a distinct count measure should only have that single measure and no others. While additive measures compress very well, the same is not true for distinct count measures. This means that leaf-level data of the measure group takes up more disk space.

Targeting good aggregates for distinct count measures can also be difficult. Although aggregates for additive measures can be used by queries at higher granularities than the aggregate, the same does not apply for distinct count measures.

The combined effects of big measure groups and less useful aggregates means that queries that run against distinct count data often cause a significant amount of I/O operations and simply run longer than other queries. This is expected and part of the nature of distinct count data. However, there are some optimizations you can make that can greatly speed up both queries and processing of distinct count measures.

### Partitioning for Distinct Count

Recall that for additive measures, it is generally recommended that you partition by time and sometimes by another dimension. This partition strategy is described in the Nonbreaking Cube Changes section. Distinct count measures are an exception to this rule of thumb. When it comes to distinct count, it is often a good idea to partition by the values of the distinct count measure itself. Analysis Services keeps track of the measure values in each partition, and assuming the intervals are not overlapping, it can benefit from some parallelism optimizations. The basic idea is to create partitions, typically one per CPU core in the machine, that each contain an equal-sized, nonoverlapping interval of measure values. The following picture illustrates these partitions.



Figure 37 - Distinct Count Partitioning on a 4-Core Server

You can still apply a date based partition schema in addition to the distinct count partitioning. But if you do, make sure that queries do not cross the granularity level of this date range, or you lose part of the optimization. For example, if you do not have queries across years, you may benefit by partitioning by both year and the distinct count measure. Conversely, if you have queries that ask for data at the year level, you should not partition by month and the distinct count measure.

The white paper in the References section describes the partition strategy for distinct count measures in much more detail.

**References:**

* Analysis Services Distinct Count Optimization - <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=65df6ebf-9d1c-405f-84b1-08f492af52dd&displaylang=en>
  + Describes the partition strategy that speeds up queries and processing for Distinct Count Measure groups

### Optimizing Relational Indexes for Distinct Count

Analysis Services adds an ORDER BY clause to the distinct count processing queries have. For example, if you create a distinct count measure on **CustomerPONumber** in **FactInternetSales,** you get this query while processing.

SELECT … FROM FactInternetSales

ORDER BY [CustomerPONumber]

If your partition contains a large amount of rows, ordering the data can take a long time. Without supporting indexes, the query plan looks something like this.

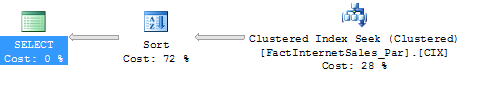


Figure 25 Relational sorting caused by distinct count

Notice the long time spent on the Sort operation? By creating a clustered index sorted on the distinct count column (in this case **CustomerPONumber**), you can eliminate this sort operation and get a query plan that looks like this.

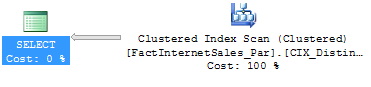


Figure 26 Distinct count query supported by a good index

Of course, this index needs to be maintained. But having it in place speeds up the processing queries.

## Many-to-Many Dimensions

Many-to-many dimensions are a powerful feature of Analysis Services cubes. They enable easy solutions for some complex, yet common scenarios in dimensional modeling. When cubes resolve many-to-many queries, the join with the intermediate table is done in Analysis Services memory and during query time. If the intermediate table is large, especially if it larger than memory, these queries can take a long time to respond. We recommend that you use many-to-many dimensions only if the intermediate table fits in memory. The links in the References section describe some techniques that enable you to reduce the memory consumption of the intermediate table.

**References:**

* Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques – <http://www.microsoft.com/downloads/en/details.aspx?FamilyID=3494E712-C90B-4A4E-AD45-01009C15C665&displaylang=en>
* BIDS Helper has tools to estimate benefits of the Many-to-many compression described in the section:
  + <http://bidshelper.codeplex.com/wikipage?title=Many-to-Many%20Matrix%20Compression>
* Many-to-many project - <http://www.sqlbi.com/manytomany.aspx>
  + Design patterns for many-to-many dimensions

# Conclusion

This document provides the means to monitor, capacity plan, and diagnose SQL Server 2008 Analysis Services operations issues.

For more information, see:

<http://sqlcat.com/>: SQL Customer Advisory Team

<http://www.microsoft.com/sqlserver/>: SQL Server website

<http://technet.microsoft.com/en-us/sqlserver/>: SQL Server TechCenter

<http://msdn.microsoft.com/en-us/sqlserver/>: SQL Server DevCenter

If you have any suggestions or comments, please do not hesitate to contact the authors. You can reach Thomas Kejser at [tkejser@microsoft.com](mailto:tkejser@microsoft.com), Denny Lee at dennyl@microsoft.com, and John Sirmon at [johnsi@microsoft.com](mailto:johnsi@microsoft.com).

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](mailto:sqlfback@microsoft.com?subject=White%20Paper%20Feedback:%20SQL%20Server%202008%20R2%20Analysis%20Services%20Operations%20Guide).