SQLCAT's Guide to: BI and Analytics

Microsoft SQLCAT Team

Guide & Reference

Microsoft

SQLCAT's Guide to: BI and Analytics

Microsoft SQLCAT Team

Summary: This e-book is a collection of some of the more popular technical content that was available on the old SQLCAT.COM site. It covers SQL Server technology ranging from SQL Server 2005 to SQL Server 2012. However, this is not all the content that was available on SQLCAT.COM. To see additional content from that site you can follow the SQLCAT blog which will point to additional content. For more comprehensive content on SQL Server, see the MSDN library.

Category: Guide & Reference
Applies to: SQL Server Business Intelligence and Analytics
Source: SQLCAT Blog
E-book publication date: September 2013

Contents

Section 1: Administration

Running Microsoft SQL Server 2008 Analysis Services on Windows Server 2008 vs. Windows Server 2003 and Memory Preallocation: Lessons Learned

Introduction

SQL Server 2005 Service Pack 2 introduced a new advanced configuration setting for Analysis Services that enables an administrator to specify that a certain percentage of memory be allocated to Analysis Services when the service starts (memory preallocation). This configuration setting was introduced because the Microsoft Windows Server 2003 operating system did not scale well with many small virtual memory allocations due to memory fragmentation and inefficiencies in retrieving information already located in memory. With the use of this configuration setting with 64-bit versions of Windows Server 2003 on multiprocessor computers (specifically computers with 16, 32, and 64 cores), the SQL Customer Advisor Team (SQL CAT) observed significant performance gains during Analysis Services processing and querying when memory use substantially exceeds 3 GB. Indeed, when Unisys and Microsoft achieved a record-setting benchmark for Analysis Services cube processing, the memory preallocation configuration setting for Analysis Services was set to 20 rather than to the default setting of 0 (resulting in 20% of physical memory being allocated to Analysis Services). For more information about this benchmark, see Data Warehouse Performance: Unisys and Microsoft Achieve Record-setting Benchmark.

The memory manager in Windows Server 2008 includes many performance and scalability enhancements, one of which is a change in the algorithm for scanning pages in memory. Additional memory manager improvements include the following:

· Kernel virtual address space is dynamically allocated to improve scalability when a greater number of processors and larger memory configurations are used. The sizes of paged and non-paged memory pools are no longer fixed, so manually reconfiguring systems to prevent resource imbalances is not required. Previously, the sizes of some resources, such as memory pools, were set by registry keys or by SKU.

· Kernel virtual address space is made more available in x86 architectures by more efficiently using space in kernel-mode stacks in recursive calls into the kernel.

· When a kernel-mode component allocates more than a page of memory, the memory manager now uses the pool memory between the end of that allocation and the next page boundary to satisfy other memory requests.

· SuperFetch, an adaptive page prefetch technique that analyzes data usage patterns and preloads data into memory according to those patterns, was added.

· Numerous enhancements that make writing to the page file faster and more efficient have been added. The memory manager now writes more data in each operation, aligns pages with their neighbors, and does not write pages that are completely zero.

· There is now greater coordination between the memory manager and the cache manager for write operations.

· Prefetch-style clustering of pages to satisfy page faults and populate the system cache was added.

· The capabilities of NUMA architectures are more fully utilized.

For more information about the improvements in the Windows Server 2008 memory manager, see:

· Advances in Memory Management for Windows

· Memory Management, Dynamic Kernel Addressing, Memory Priorities and I/O Handling

· Designing Applications for High Performance

· Inside Windows Server 2008 Kernel Changes

We decided to test and compare Analysis Services partition and dimension processing using SQL Server 2008 Analysis Services RC0 running on Windows Server 2008 and Windows Server 2003 to assess the impact of these memory manager improvements on the type of hardware and with the type of data sets with which we have previously seen significant performance benefits with the memory preallocation configuration setting.

Executive Summary

Due to the improvements in the Windows Server 2008 memory manager related to the change in the algorithm for scanning pages in memory, SQL Server 2008 Analysis Services performed equally well during both partition and dimension processing with or without memory preallocation when running on Windows Server 2008. However, SQL Server 2008 Analysis Services performed substantially better during both partition and dimension processing with the use of the memory preallocation configuration setting when running on Windows Server 2003. Specifically, the processing performance of Analysis Services 2008 without memory preallocation on Windows Server 2008 was virtually identical to the processing performance of Analysis Services 2008 on Windows Server 2003 with memory preallocation. With the hardware that we used in our tests and with the Analysis Services objects that we processed, we observed a performance benefit of approximately 100% during partition processing and a performance benefit of between 30-40% during dimension processing.

If you use memory preallocation with SQL Server 2008 (or SQL Server 2005), use a value that is low enough to ensure that sufficient memory remains for other processes on the computer (avoiding paging) and high enough for Analysis Services (use the peak value for the Process: Private Bytes counter for the msmdsrv instance to establish this value).

Note: While we did not specifically test SQL Server 2005 Analysis Services for this technical note, the results discussed here apply to Analysis Services 2005 as well as to Analysis Services 2008 because the underlying code base for the portion of Analysis Services 2008 that we tested has not changed.

Test Environment

For our testing, we used the following:

· A Unisys ES7000/one Enterprise Server with 32 dual-core x64 processors and 128 GB of RAM that was connected to two HP EVA 8000 arrays

· Windows Server 2003 Datacenter Edition and Windows Server 2008 Datacenter Edition

· SQL Server 2008 RC0

· An internal Microsoft sales database, containing approximately 1 terabyte of relational data

· An Analysis Services cube based on this relational database, which resulted in a cube that was approximately 180 GB in size. The dimensions were approximately 60 GB in size and the partitions were approximately 120 GB in size (and contained no aggregations).

Note: We used the Microsoft Enterprise Engineering Center, with its extensive hardware inventory and team expertise to build the test system for this test.

Database Engine Configuration

For our processing tests, we set the minimum and maximum memory for the relational engine to 20 GB. We left all other settings at their defaults. We located the data and log files for the relational database on different LUNs from the Analysis Services dimension and partition files.

Analysis Services Configuration

In Analysis Services, we set memory preallocation to 40 (<PreAllocate>40</PreAllocate>) in the msmdsrv.ini file (which preallocates 40% of the 128 GB of memory on our test computer to Analysis Services at startup). We experimented with enabling large page support, but found no measurable performance benefit. Large pages require that the Analysis Services service account have the lock pages in memory user directly on the local computer. Large page support enables server applications to establish contiguous large-page memory regions. For more information about large page support and restrictions when using large pages, see Large Page Support.

Caution: When we enabled large page support for Analysis Services on Windows Server 2003 after the computer had been in operation for some time, and then restarted Analysis Services, this service took longer to restart than without large page support (from several minutes to occasionally much longer, sometimes requiring a system restart due to the inability of this service to acquire the specified amount of contiguous large pages because of memory fragmentation). However, we rarely saw this symptom with Windows Server 2008. When we enabled large page support for Analysis Services with Windows Server 2008, this service would restart quickly.

Test Methodology

For our processing tests, we processed the following Analysis Services objects:

· A single measure group containing 158 partitions. The size of these partitions on disk was approximately 9 GB. We fully processed this measure group after unprocessing all of the partitions in the measure group.

· A dimension containing 89 attributes, with the key attribute containing approximately 26 million distinct members. The size of this dimension on disk was approximately 38 GB. We fully processed this dimension after unprocessing the dimension.

· A dimension containing 47 attributes, with the key attribute containing approximately 75,000,000 distinct members. The size of this dimension on disk was approximately 18 GB. We incrementally processed this previously processed dimension (process update).

With each processing test, we always worked with warm file system cache (we did not reboot the computer between tests). That is, for each test, we tested once with the cold file system cache and then tested three times with the warm file cache to confirm that our test results were reproducible.

Note: We chose large dimensions and measure groups to attempt to maximize the potential impact of memory preallocation. The performance benefit of memory preallocation with smaller dimensions and partitions was not tested.

Test Results

In our tests, we observed the following:

· Using default settings, the processing performance of Analysis Services 2008 on Windows Server 2008 was always superior to the processing performance of Analysis Services 2008 on Windows Server 2003 (see Figure 1).

· Using memory preallocation with Analysis Services 2008 on Windows Server 2003 enabled us to match the processing performance of Analysis Services 2008 on Windows Server 2008 with default settings (See Figure 2).

· Using memory preallocation with Analysis Services 2008 on Windows Server 2008 provided no measurable performance benefit (See Figure 3).

Note: The results that we saw may not apply across all hardware platforms or with all data sets. We verified our test results on an HP Superdome with 32 IA64 single-core processors and 64 GB of RAM to compare the x64 and IA64 platforms with respect to this configuration setting. We did not verify our test results with SQL Server 2005, but do expect that we would see very similar results with this version of SQL Server. We also did not verify our results on any other hardware platforms.

Caution: If you choose to preallocate memory with Analysis Services 2008 (or Analysis Services 2005) when running on Windows Server 2003, test thoroughly to determine if there is a performance benefit in your environment. If you do preallocate memory, you must ensure that sufficient memory remains for the operating system and other services on your computer.

Partition Processing

During our partition processing tests, we observed performance improvements with memory preallocation of approximately 100% (see Figure 1).

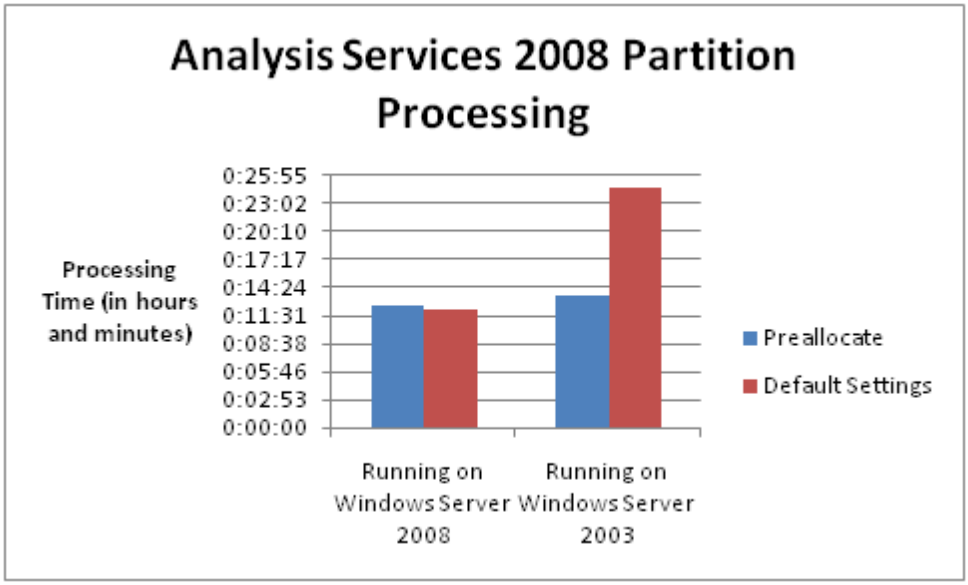


Figure 1: Analysis Services 2008 Partition Processing

Dimension Processing

During our dimension processing tests, we observed performance improvements with memory preallocation of between 30-40% (see Figures 2 and 3).

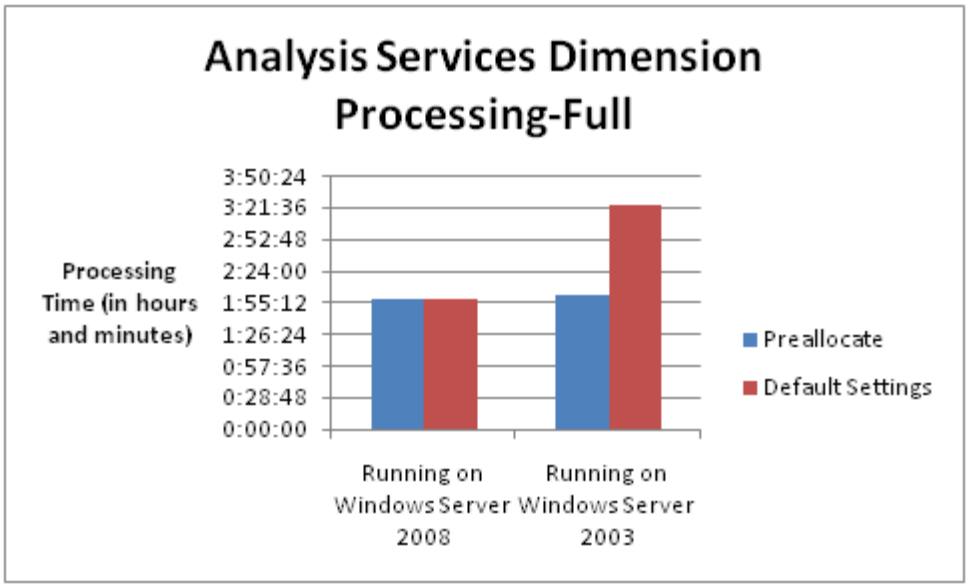


Figure 2: Analysis Services 2008 Dimension Processing—Full

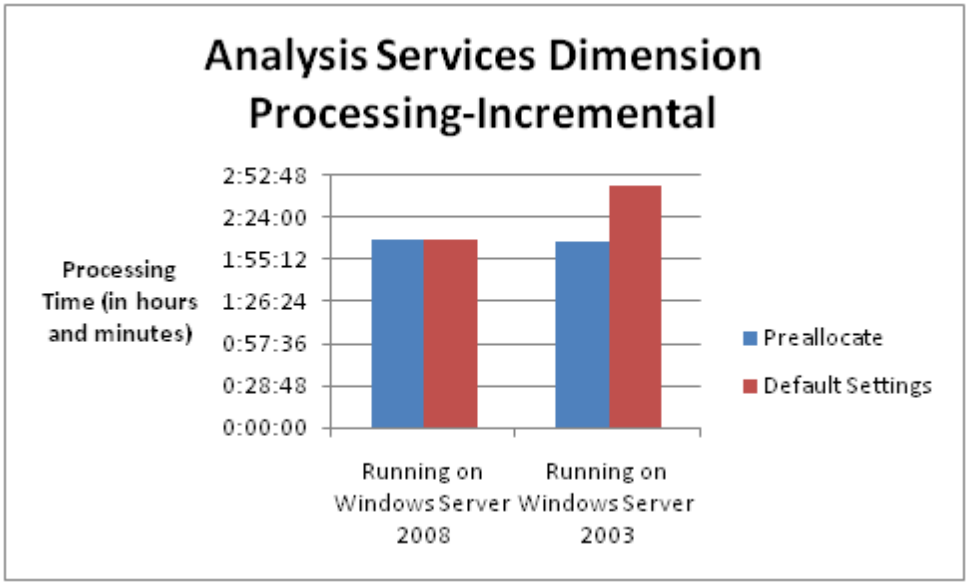


Figure 3: Analysis Services 2008 Dimension Processing—Incremental

Note: While we did not include MDX querying in this memory preallocation test, we observed the benefit of memory preallocation with SQL Server 2005 Analysis Services on Windows Server 2003 when we tested the performance of MDX queries involving many-to-many dimensions. For more information, see Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques. We expect the results that we saw in this test when running Analysis Services 2008 on Windows Server 2008 to apply equally to MDX queries, but more testing is required in this area.

Determining the Appropriate Value for Memory Preallocation

If you use memory preallocation with SQL Server 2008 (or SQL Server 2005), use a value that is low enough to ensure that sufficient memory remains for other processes on the computer (avoiding paging) and high enough for Analysis Services (use the peak value for the Process: Private Bytes counter for the msmdsrv instance to establish this value).

· If you set the memory preallocation value too high, you will starve other processes on the server and the result will be memory paging. If you are running the SQL Server relational engine on the same server as Analysis Services, you should also limit the maximum amount of memory used by the SQL Server relational engine. While there is no hard limit for how much memory you should leave for the operating system, 6-8 GB is generally sufficient if no other services are running on the same server.

· Set the memory preallocation to a value that is high enough to ensure that Analysis Service preallocates sufficient memory for Analysis Services query and processing activity on the server such that it does not need to request any additional address space. To determine this value, monitor the peak value for the Process: Private Bytes counter for the msmdsrv instance. The peak value establishes the maximum value for memory preallocation that you need to set. When you cannot set the preallocation value this high, set it as high as possible without starving other processes and the operating system.

Conclusion

For optimum performance with SQL Server 2008 Analysis Services default settings, use Windows Server 2008. To match the performance of SQL Server 2008 Analysis Services running on Windows Server 2008 when running SQL Server 2008 Analysis Services on Windows Server 2003, consider using memory preallocation. These conclusions apply to SQL Server 2005 as well as to SQL Server 2008.

Scripts for Stress Testing Analysis Services using ASCMD

Scripts for Using ASCMD for Stress Testing at http://www.codeplex.com/SQLSrvAnalysisSrvcs

This project on Codeplex includes two scripts for using ASCMD for stress testing. One script is optimized for serially executing a stream of queries and the second script is for executing multiple streams of queries in parallel. You can control the number of queries, the number of streams, and the think time between queries.

A Solution for Collecting Analysis Services Performance Data From Many Sources For Performance Analysis

A Solution for Collecting Analysis Services Performance Data for Performance Analysis
at http://www.codeplex.com/SQLSrvAnalysisSrvcs

This codeplex solution automates the collection of SQL Server 2008 performance data from any or all of the following data sources ans stores the collected data into a single SQL Server 2008 relational database:

- SQL Server Profiler Analysis Services trace data
- Performance Monitor counters
- Analysis Services Dynamic Memory Views (DMVs)

This data is collected using an Analysis Services server-side trace, several Integration Services packages, a custom performance monitor collector in Management Data Warehouse, and Transact-SQL stored procedures. This codeplex project also include sample Reporting Services reports utilizing SQL Server stored procedures that correlate and analyze the collected data.These reports enable you to:

- Determine your slowest MDX queries for a specified time period
- Compare the performance of a specified query during different time periods
- Analyze a specific query to determine if it is storage engine or formula engine bound, to view the aggregations utilized, the number of subcubes and the number of partitions
- Analyze processing performance
- Correlate performance monitor counters with the execution of a specific query or processing operation
- Correlate DMVs with the execution of a specific query or processing operation

This solution was updated on 3/6/2009 and again on 3/31/2009 to:

- Consolidate all environment variables into a single script file - this dramatically simplifies installation and operation
- Fix bugs related to named instances - including, but not limited to the following - Performance Monitor counters for Analysis Services named instances must be entirely in capital letters in order to work in the Management Data Warehouse performance counter data collector.

Compress Many-to-Many C# Utility

http://sqlsrvanalysissrvcs.codeplex.com/

This project creates a utility that enables you to easily implement the many-to-many compression technique discussed in the Analysis Services Many-to-Many Dimensions: Query Performance Optimization Techniques best practices white paper. You can download this white paper from the SQL Server Best Practices web site at: http://technet.microsoft.com/en-us/sqlserver/bb671430.aspx

Create a Processing Log Script Project

http://sqlsrvanalysissrvcs.codeplex.com/

This project contains a script that enables you to create a system-wide processing log file for monitoring / auditing purposes. It also contains a script that enables you to delete a system-wide processing file.

Powershell Scripts for SQL Server 2008 Analysis Services DMVs

http://sqlsrvanalysissrvcs.codeplex.com/

These sample scripts demonstrate how to use Powershell to query SQL Server 2008 Analysis Services DMV's. The four sample scripts enable you to cancel all queries running longer than a specified length of time, cancel a particular session/SPID/connection, retrieve the Analysis Services version, or return the results of any DMV query.

Sample Robocopy Script to customer synchronize Analysis Services databases

Authors: Eric Jacobsen, Denny Lee
Contributors: Mike Vovchik

Technical Reviewers: Guillaume Fourrat, Richard Tkachuk, Thomas Kejser

Applicable to: SQL Server 2005 Analysis Services

Introduction

As a quick follow up to the SQL Server Best Practices article Scale-Out Querying with Analysis Services, below is an example cmd script on how to robocopy the OLAP databases from one server to another. As a quick refresher, within this whitepaper we called out the technique of using robocopy as a method to synchronize an OLAP database between servers as a viable alternative to the Analysis Services' synchronize method. The advantages of using this approach includes the fact that you can copy the database to multiple servers in parallel and it can be faster provided that you are not blocked by the constraints noted below. A disadvantage to this approach is that it will require stopping/restarting the Analysis Services service in order for the server to recognize the OLAP database once it has been copied over. This approach is typically used within environments where there is only one AS database on the server, though it can work for multiple databases provided there are no issues with you shutting out user access to multiple databases to stop/restart the server. You can review the whitepaper linked above for more details.

Robocopy

You can get a copy of robocopy from the Windows Server 2003 Resource Kit Tools. Typically in a simple network configuration, robocopy will use about 1/2 the available bandwidth which implies that you can optimally robocopy your OLAP database to two query servers in parallel.

Important

Please note, you should not intermix the use of this robocopy technique with the AS Synchronize command. Once you start using the robocopy method, it is advisable to stay with it as this is a one-way only synchronize technique (one source to multiple targets).

Recall, the purpose of this technique is to provide you with a fast alternate method to provide **query-only** OLAP databases on separate AS instances. The Analysis Services synchronize command does far more than just copy these Analysis Services files. The command also scans metadata and touches all of the partitions of the two databases between the servers to ensure a two-way synchronization ability (source and target can interchange), regeneration of the .db.xml file (containing among other things the data source connection strings), and cryptokey. As well, there are some features such as ROLAP, writeback, real-time updates, and some data mining features that cannot be cloned via the robocopy method (i.e. to use these features, you will need to use the AS Synchronize command).

Sample Script

The sample robocopy script is as can be seen below in which it is copying the various Analysis Services files with 21 processes in parallel (i.e. count the number of “start %cmd%” statements in the example script below).

Please note, this script is provided "AS IS" with no warranties and confers no rights. This is a sample example script of how to do custom synchronizing of Analysis Services databases. In your environment you may want to use this as a template for your own script, add error handling, and reduce the number of threads copying in parallel.

```
REM
REM Synchronize OLAP Data Folder
REM
REM    Make a copy of the entire OLAP data folder
REM    Run many instances of robocopy in parallel to speed it up.
REM
REM    Author: Eric Jacobsen Jun-2006
REM    Update: Denny Lee, Thomas Kejser Jan-2008

setlocal
REM Enter Source Directory Here
set SRC=

REM Enter Destination Directory Here
set DST=

REM Enter Log file Here
SET LOG=

REM Robocopy Template
set CMD=robocopy %SRC% %DST%

REM Robocopy Options
REM   If you do not want a log file, remove the "/LOG+:%LOG%" below
set OPT=/Z /S /LOG+:%LOG%

start /MIN "Copying Flexible Aggregations" %CMD% *.agg.flex.data %OPT%
start /MIN "Copying Flexible Mapping Files" %CMD% *.agg.flex.map %OPT%
start /MIN "Copying Flexible Aggregation Headers" %CMD% *.agg.flex.map.hdr
%OPT%
start /MIN "Copying Fact Mapping Files" %CMD% *.fact.map %OPT%
start /MIN "Copying Fact Mapping Headers" %CMD% *.fact.map.hdr %OPT%
start /MIN "Copying Flexible Data Headers" %CMD% *.flex.data.hdr %OPT%
start /MIN "Copying AH Store Files" %CMD% *.ahstore %OPT%
start /MIN "Copying AS Store Files" %CMD% *.asstore %OPT%
start /MIN "Copying AStore Files" %CMD% *.astore %OPT%
start /MIN "Copying SStore Files" %CMD% *.sstore %OPT%
start /MIN "Copying DStore Files" %CMD% *.dstore %OPT%
start /MIN "Copying Fact Data Files" %CMD% *.fact.data %OPT%
```

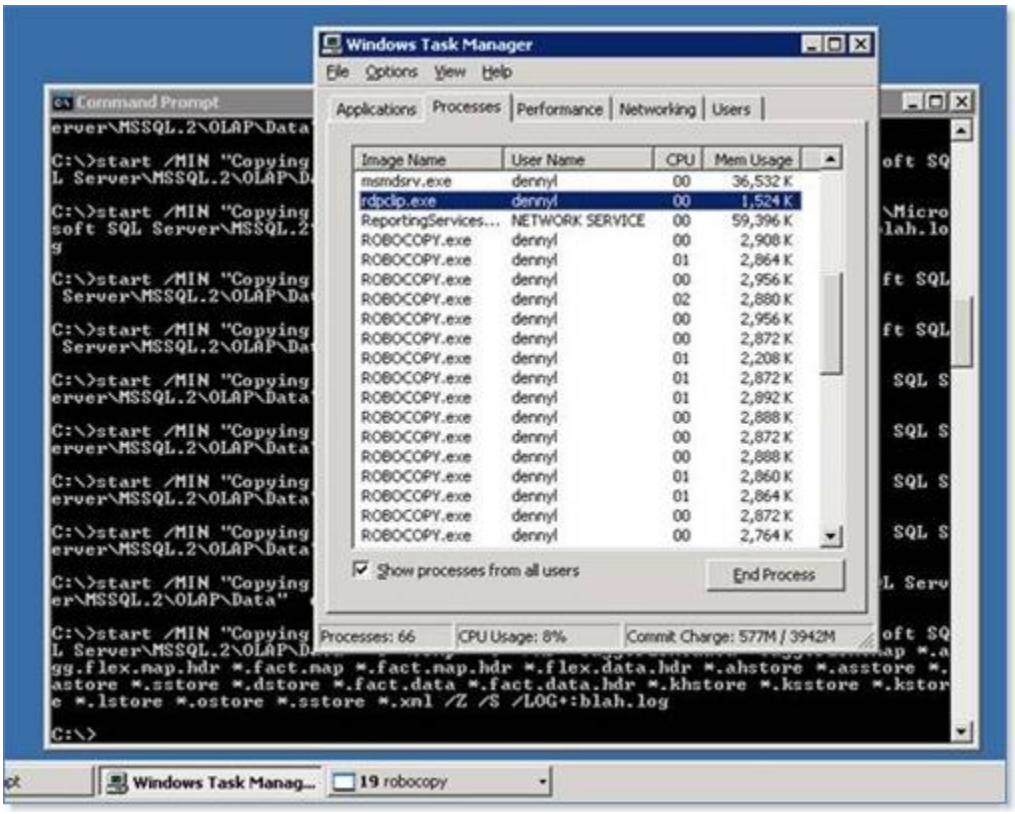
```
start /MIN "Copying Fact Data Header Files" %CMD% *.fact.data.hdr %OPT%
start /MIN "Copying KH Store Files" %CMD% *.khstore %OPT%
start /MIN "Copying KS Store Files" %CMD% *.ksstore %OPT%
start /MIN "Copying KStore Files" %CMD% *.kstore %OPT%
start /MIN "Copying LStore Files" %CMD% *.lstore %OPT%
start /MIN "Copying OStore Files" %CMD% *.ostore %OPT%
start /MIN "Copying SStore Files" %CMD% *.sstore %OPT%
start /MIN "Copying XML Files" %CMD% *.xml %OPT%
start /MIN "Copying All other Files" %CMD% *.* /xf *.agg.flex.data
*.agg.flex.map *.agg.flex.map.hdr *.fact.map *.fact.map.hdr *.flex.data.hdr
*.ahstore *.asstore *.astore *.sstore *.dstore *.fact.data *.fact.data.hdr
*.khstore *.ksstore *.kstore *.lstore *.ostore *.sstore *.xml %OPT%
```

As you can see with this script, you will need to provide the three entries of:

· SRC: The source AS data folder

· DST: The destination (target) AS data folder

· LOG: This is the log file just in case you want to record all of the files being copied. Note, since all 21 processes are running concurrently, your log files will be interlaced with copy completions from all 21 robocopy executions. To turn off the log file, please go to the next line and remove the "/LOG+:%LOG%" statement.



Above is a screenshot of the robocopy method to synchronize the two different Analysis Services databases. As you can see, the cmd window executes the 21 separate processes, 19 of which are showing up within the Task Manager and minimized within the 19 robocopy cmd windows.

Closing Remarks

As noted in the introduction, this robocopy method of synchronizing database is an alternative to the AS Synchronize method. It is very effective for the specific scenarios described within the Scale-Out Querying with Analysis Services whitepaper. But before using this method, remember that this method does have some limits (query only, one-way sync only, etc.) and should never be used intermixed with the AS Synchronize method.

Section 2: Analysis Services Performance

Analysis Services Synchronization Best Practices

Introduction

When administering your enterprise Analysis Services environment, it is generally a best practice to create separate querying, processing, and synchronization windows to ensure that these processes do not interfere with each other. This is a common approach for enterprise SQL environments where techniques such as clustered servers and read-only databases are used for querying and the various replication techniques or custom SQL Server Integration Services (SSIS) packages are used to perform SQL database synchronization. The key to having dedicated servers for querying and processing is the ability to effectively synchronize data between different servers.

Background

As noted in the Analysis Services Processing Best Practices white paper, long-running Analysis Services queries that are executed concurrently with processing can prevent processing from completing on the Analysis Services database. Long-running queries prevent Analysis Services from taking an exclusive lock on the database; therefore, processing must wait until the queries complete.

To prevent processing and querying from interfering with each other, one tactic is to create separate querying and processing windows on your Analysis Services server. A common approach is during the business day (such as 8 A.M. – 6 P.M.) the Analysis Services server is for querying only. After 6 P.M., processing and other operational tasks like backup and patches can be done. The problem with this methodology is that many customers require that processing occur multiple times throughout the day. As well, with global customers and/or users, there may very few opportunities for a processing window. An effective solution to this problem is to have a separate querying and processing architecture, as shown in the following figure (derived from the above linked paper).

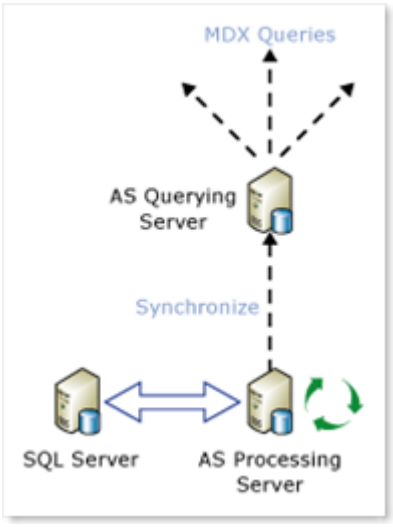


Figure 1: Querying/processing Analysis Server server architecture

The basic idea is that you set up your regular Analysis Services server, which acts as your processing server and connects to your relational data source (in this case a SQL Server database). After processing has completed, you can replicate your database from the processing server to a separate Analysis Services server that acts as a query server.

User queries ping the Analysis Services query server while processing can occur on the Analysis Services processing server. For more information on this approach, see the Scale-Out Querying with Analysis Services white paper, which describes this approach in detail. Another white paper, Scale-Out Querying with Analysis Services Using SAN Snapshots, resolves the separation of query and processing tasks by using SAN snapshots.

Whichever approach you decide to use, it is important to separate the querying and processing tasks so that you will not encounter maintenance issues associated with long-running queries, incomplete queries, and incomplete processing tasks.

Synchronization Techniques

There are four main techniques for synchronizing an Analysis Services database from one server to another. In general the attach/detach, backup/restore, and synchronize methods are great for individual database synchronizations. The robocopy method is best for one database per Analysis Services instance environments because you must stop/restart the service.

Analysis Services Synch Method

Included as part of Analysis Services, the Analysis Services Synch method is a common way to synchronize an Analysis Services database. What is great about this particular method is that it is easy to operate and does not require a lot of development to get it to work in your production environment. As well, you do not need multiple versions of your database (more on this later) because this method synchronizes with an existing database. This process scans the differences between two databases on the two servers and transfers over only files that have been modified. A past disadvantage of this method is that it could take a long time to complete. For example, in one scenario, a 10-GB database with about 250,000 files took around 11 hours to complete (if at all) in Analysis Services 2005. Using the same database and hardware on Analysis Services 2008 CTP5, the synchronization now takes around 40 minutes.

The performance of this new synchronization is quite impressive and is due to the changes to the underlying file system as part of Analysis Services 2008. But by digging into the details of this synchronization, we discovered that when we ran the synchronization between the servers when no database existed on the target server, the synchronization finished in less than 10 minutes. That is, there is an approximately 30 minute difference in synchronization time when a database exists as compared to when a database does not exist. There is an overhead to scan and verify the metadata when the database exists on both source and target servers, but we had not expected it to be this much.

When we ran the synchronization in SQL Server Profiler, we discovered that there were a large number of OBJECT DELETE statements to delete the metadata objects, which occurs when deleting existing database objects as well as when instantiating new database objects. Deleting these temporary files took about 25 minutes. After we deleted the database on the target server, it also took 25 minutes to validate that the act of deletion uses most of the time. The good news is that based on these results, it took only five minutes for Analysis Services to scan and verify the metadata of the source and target databases. But the question remains why it took so long for the deletion tasks to complete. While there were 250,000 files within the database and the deletion process is an operating system single-threaded process, we still did not expect the process to take this long. We have synchronized very large databases with even more files that did not see such an extreme difference in time. On further analysis, this issue appeared to be specific to this particular database and was an indicator that the underlying file system needed to be optimized (in this case, the SAN needed to be reconfigured) so that the deletion could work faster. For more information on optimizing SQL interactions with disk I/O, see the Predeployment I/O Best Practices white paper.

During synchronization, a number of locks are placed on the databases to ensure validity. A write lock is applied before the transfer of the files and a read commit lock is applied to the source DB at almost the same moment. The write lock is released from the target Analysis Services database once the metadata is validated and the transaction is committed. The read commit lock is released at about the same moment as the write lock as it is taken in this distributed transaction. This means that during the synchronization process a write lock is applied on the target server (when a target database exists), preventing users from querying and/or writing over the database. There is only a read commit lock on the source server, which prevents processing from committing new data but allows queries to run so multiple servers can be synchronized at the same time. In the end, this means that while synchronization has improved quite a bit for SQL Server 2008 Analysis Services (and the performance degradations seen were specific to this database and disk configuration issues), you still need a maintenance window because users cannot query the database for a short while. This maintenance window must be a bit longer if you must synchronize from one source to multiple target servers.

Backup/Restore Database

The backup/restore method is the most common data recovery method for an Analysis Services database and can be adapted as a synchronization technique. After you back up your database on your source server, you can copy it to multiple target servers, and then execute the restore method on to your target servers. In addition, when you restore your database, you can rename the database on your target servers so that you can keep two different versions of the database. This synchronization process would be in the form of:

1. Back up the database [Foodmart] on the source server.

2. Point the Report UI to the [Foodmart] database on the target server.

3. Copy the database to your target servers.

4. Restore the database with a new name, [Foodmart V2].

5. Point the Report UI to the [Foodmart V2] database on the target server.

The advantage of this approach is that if your restoration of the [Foodmart V2] database fails, users can still query the original [Foodmart] database. As well, your backup is your data redundancy and synchronization methodology so it is slightly easier to maintain. A disadvantage with this approach is that you need two copies of the database—this requires twice the disk space you originally required. For enterprise environments, the main disadvantages involve time—the time required to robocopy the full database backup from one server to another (versus just copying the delta) and the time to restore the database.

Note that when you are restoring the database on the target server, a read lock is first initiated and then released for validation, a write lock is initiated if the database exists, restoration of files occurs, the write lock is released, and then the read lock is released. This mechanism of locks ensures the integrity of the database as its being restored. At the same time, this also means that the database cannot be queried during the restore. This is the reason we suggest using two databases with different names so that you can minimize query downtime. At the same time, if you have a wide query downtime window (or your UI cannot change connection strings easily), you can always restore over the existing database instead of having two instances.

Attach/Detach Database

As part of SQL Server 2008 Analysis Services (recently introduced in the SQL Server 2008 CTP6 release), you have the option to attach/detach your Analysis Services database and set it to read-only. This makes it safer for your Analysis Services databases because you can ensure that no one can make changes to your database. The great thing about attach/detach (analogous to the SQL attach/detach feature) is that you can detach your database from your processing server, copy it over to your query server, and then simply attach it. The synchronization process for attach/detach involves:

1) Complete processing the [Foodmart] database on the source server.

2) Detach the [Foodmart] database on the source server.

3) Robocopy the database from the source server to target server(s).

4) Re-attach the[Foodmart] database on the source server.

5) Detach the old [Foodmart] database on the target server.

6) Attach the new [Foodmart] database on the target server; you can attach it as read-only usage mode within the Attach command.

The advantage of this approach is that while you have to robocopy the full database (versus the delta files), this involves simply attaching a database. Since the attached database on the target server is read-only, you can also set your SAN mounts to be optimized for reads so your storage engine queries will execute faster.

Similar to the backup/restore technique, the attach/detach method requires more disk space because you need two copies of the database. As well, you must detach the database on the target server before attaching the new one on the target server (you cannot attach with a different name) because of the database ID issue noted in Renaming Olap Databases Issues within Analysis Services.

Note that in the near future as we complete tests (including performance numbers) with real customer implementations, we will have more on the read-only feature, which allows multiple query servers to query a single database on a shared SAN disk space.

Robocopy Method

There is already a bit written on the robocopy method, which can be found in the following articles:

- Sample Robocopy Script to customer synchronize Analysis Services databases
- Scale-Out Querying with Analysis Services

A general summary is that this method enables you to copy over only the delta files (that is, the changed files) of your Analysis Services database from your source to target servers (similar to the Synch method). In enterprise environments, this is especially handy because you want to copy over only the changed files and you want to do it faster. Robocopy is an especially fast and effective tool for copying files from one server to another and it can also be run in parallel if you want to upload to multiple query servers concurrently. Disadvantages of this method include the

need to stop/restart your Analysis Services server, the Synch and robocopy methods do not intermix, and some functionality (such as writeback, ROLAP, real-time updates, and so on) is lost. This methodology is especially effective for query/processing architectures that involve only one database per server.

Operational Maintenance Window

As you can see, all of the above techniques are quite useful and have their advantages and disadvantages. But irrelevant of the technique you use to synchronize your database, all of these methods require some operational maintenance window to allow for the synchronization to complete. An effective way to get around this is to have two instances (as noted Scale-Out Querying with Analysis Services) but this requires two instances of the same database and control of the UI connection strings.

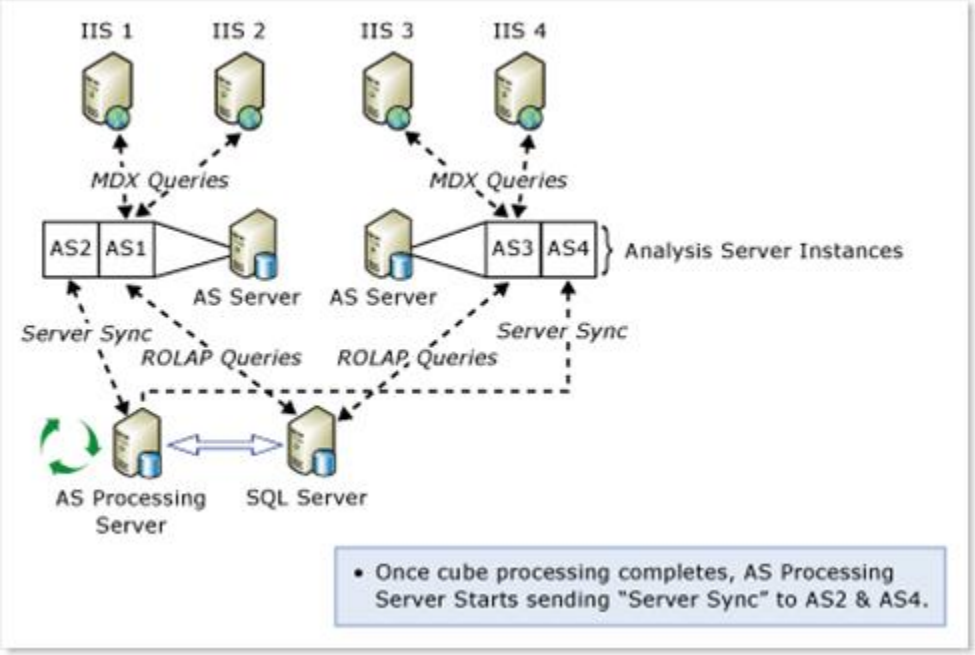


Figure 2: Queries to active instances (AS1, AS3) while synchronization occurs to the idle instances (AS2, AS4)

Regardless of the technique you choose, it is important to remember that your techniques of synchronization require clear cut-off times between processing, querying, and synchronization so that it is possible to perform each of these tasks without interference from the other. One important note for this is that you should ensure that lazy processing is disabled and use **ProcessIndexes** after your **ProcessUpdate**. **ProcessIndexes** completes the index and aggregation build process much faster than lazy aggregation and there is a definitive cut-off time so you know when you can start your synchronization or allow queries to run.

Discussion

As you can see, there are a number of ways to synchronize your Analysis Services database from your source to target servers. The robocopy method is extremely fast and you can transfer files to multiple servers concurrently but has its own set of disadvantages; it is most effective for the scenario where there is a single database on an Analysis Services server. For SQL Server 2008 Analysis Services, the backup/restore and synchronization methods have great

improvements in performance but require space and/or maintenance windows during which queries cannot be executed.

Note: The numbers quoted in this technical note are specific to a single customer database; we expect your system to see improved performance though not necessarily to the magnitude seen here. As well, the SQL Server 2008 Analysis Services read-only (attach/detach) feature introduces a new and intriguing way to synchronize and an optimized way to query as well. The read-only feature allows multiple servers to query the same single Analysis Services database to allow for more concurrent queries—more on this topic in a later document. All of these techniques involve some form of a maintenance window; hence it is important for you to create clear-cut windows for querying, processing, and synchronization processes for your enterprise Analysis Services environment. This will minimize the conflicts for your enterprise Analysis Services environments and make your system a little easier to manage.

Appendix

A quick note on lazy processing

We are calling out lazy processing in the context of synchronization because while lazy processing is occurring in the background it may interfere with synchronization processes and this could result in a non-queryable database on the target server. Lazy processing performs the task of building indexes and aggregations for dimensions and measure group partitions at a lower priority to reduce foreground processing time and to allow users to query the cube sooner. For lazy processing to occur, you must switch the **ProcessingMode = LazyAggregations** of your measure group partitions; by default this value is **Regular** (lazy processing is turned off). When processing a dimension with flexible aggregations such as parent-child or virtual dimension by using the processing enumeration of **ProcessUpdate** (such as to take into account of member name or hierarchy changes), lazy processing is initiated to ensure that the aggregations are rebuilt on the associated measure group partitions. The problem with this, as noted above, is that if synchronization is occurring concurrently with lazy processing, this may result in a non-queryable database due to file lock conflicts. While you can monitor lazy processing by using the SQL Server Profiler, there is no AMO method that allows one to automatically monitor the creation of lazy aggregations. It is also quite difficult to estimate the time it takes for lazy processing to complete due to the size and cardinality of your cube. As well, since lazy processing is a background process, the processing time is dependent on what is executing in the foreground at the time. Therefore, to ensure that synchronization does not interfere with processing, it may make sense to shut off lazy processing completely (**ProcessingMode = Regular**). To ensure that the aggregations of your measure group partitions are rebuilt when processing a dimension with flexible aggregations, use the processing enumeration of **ProcessIndexes** after processing your dimensions with **ProcessUpdate**. While the processing will occur in the foreground, this allows you to have a clear cut-off time for when your dimension and measure group partitions have completed processing. After this is done, you can perform the task of synchronizing your cube without conflicting with the lazy processing.

Microsoft SQL Server 2008 Analysis Services Consolidation Best Practices

This technical note describes the consolidation options, considerations, and best practices for Microsoft® SQL Server® 2008 Analysis Services that the SQL CAT team has derived from working with Analysis Services customers. In this technical note, we also discuss an Analysis Services load balancer

solution that one of our internal customers uses to load balance queries from Microsoft Excel® and SQL Server Reporting Services clients across multiple query servers to increase query performance and provide availability.

Why Consolidate

Our customers are consolidating Microsoft SQL Server 2008 Analysis Services for the following reasons:

- Better utilization of hardware resources, either of existing servers or in conjunction with the acquisition of new, more powerful servers
- Reduction of sprawl, particularly of many small to medium-sized Analysis Services cubes
- Increased importance of availability, as business intelligence solutions become more business critical
- Green IT, with fewer more powerful servers utilizing less energy resources than more, less powerful servers

Consolidation Solutions

In working with our customers, we have implemented either one or both of the following Analysis Services consolidation solutions:

- Co-locating the SQL Server relational data warehouse with the Analysis Services instance on a single physical server (but NOT with the Online Transactional Processing (OLTP) system).
- Co-locating multiple Analysis Services databases onto a single physical server, either within a single Analysis Services instance or through the use of multiple Analysis Services instances

Co-Locating the Relational Data Warehouse with the Analysis Services Instance

Frequently, an Analysis Services database is located on a dedicated Analysis server and Analysis Services processing queries are sent across the wire over TCP/IP or named pipes to a SQL Server relational data warehouse on its own, dedicated server. In this scenario, network congestion can be the processing performance bottleneck, particularly when the volume of data being loaded into the Analysis Services database for each processing operation is large. If this is the case, the ASYNC_NETWORKIO wait will appear high in the SQL Server wait stats list.

Network congestion can be addressed by improving the network connection performance in one or more of the following manners:

- Using Jumbo frames and tuning network parameters
- Utilizing a dedicated network connection or employing faster Ethernet connections, such as 10-gigabit Ethernet and InfiniBand
- Reducing the amount of data involved with each processing operation, such as by utilizing incremental updates
- Network congestion can also be addressed by co-locating the relational data warehouse onto the same physical server as the Analysis Services database. Co-location facilitates can improve processing performance because Analysis Services can connect to SQL Server by using the shared memory network protocol. The shared memory protocol is enabled by default on the

SQL Server relational engine and, if it is enabled as a client protocol, is always used by Analysis Services to communicate with the relational engine on the same physical server.

Co-locating the SQL Server relational data warehouse with the Analysis Services instance on a single physical server is particularly appealing as servers become more powerful (multiple cores and more memory) and less expensive. On these more powerful servers, dedicating one powerful server for the relational data warehouse and another powerful server for the Analysis Services instance may not fully utilize the processor and memory resources on either server. Co-locating the relational data warehouse with the Analysis Services instance may enable better utilization of available hardware resources. Co-location requires that you address and resolve resource contention issues. These issues are discussed later in this paper.

Co-Locating Multiple Analysis Services Databases onto a Single Server (Using a Single Instance or Multiple Instances)

To more fully utilize these more powerful and less expensive servers, it is tempting for IT to consolidate multiple small and medium-sized Analysis Services databases onto a single physical server. In many environments, there are numerous small to medium-sized cubes spread out across the enterprise that do not fully utilize the hardware resources of the servers on which they are running. Furthermore, managing and monitoring this sprawl requires significant human effort and power resources to keep running on a 24 x 7 basis. This phenomenon is exacerbated as older servers are retired and newer servers are purchased.

Consolidating Analysis Services databases onto a single physical server reduces the number of servers, which results in better utilization of hardware and a reduction in power usage. Managed appropriately, such consolidation also results in a reduction in management and monitoring costs by the IT department.

Considerations

Each of these consolidation solutions addresses specific needs. These consolidation solutions can be used separately or can used in combination. There are some common considerations for these consolidation solutions and some unique considerations for each solution.

Common Considerations When Consolidating

Consolidation will result in competition for server resources such as processor, memory, and I/O resources. In some circumstances, it may also result in locking issues or inefficient use of existing resources. When you are consolidating separate systems onto a single server platform, proper control of the utilization and sharing of resources is vital to the success of the consolidation effort. The following resources are affected by consolidation:

- Memory – when multiple processes run on a single server, all share the available memory on the server. The use of substantial memory by any single process may leave insufficient memory for other processes.
- Processor – when multiple processes run on a single server, all share the available processor resources on the server. The use of substantial processor resources by any single process may leave insufficient processor cycles for other processes. On large multi-processor servers, the relational engine may generate parallel execution plans for each individual query in a processing operation. When there are a large number of concurrent relational queries, which is typical of

Analysis Services processing operations, excess parallelism can result in contention for processor resources and inefficient use of L1 and L2 caches.
- I/O – when multiple processes run on a single server, all share I/O resources on the server. In the absence of proper disk layout and an adequate disk I/O subsystem, there is contention for the same I/O resources.

Considerations When Co-Locating the Relational Data Warehouse with the Analysis Services Instance

Co-locating the SQL Server relational data warehouse with the Analysis Services instance on a single physical server can result in significant contention for hardware resources during processing as both the SQL Server relational instance and the Analysis Services instance each require processor, memory, and I/O resources to complete the processing operation. The SQL Server relational engine and the Analysis Services multidimensional engine will each, by default, attempt to utilize all available server resources. This may result in contention for these resources during processing. For example, the SQL Server relational engine may utilize a substantial percentage of memory on the server and be slow to give up these allocated memory resources to Analysis Services when memory resources are needed during the indexing and aggregation phase of the Analysis Services processing operation.

Additionally, this resource contention may negatively affect querying, if querying is occurring concurrently with processing. For example, contention for memory resources may result in the clearing of Analysis Services caches that would otherwise be used to improve query performance. Additionally, locking during processing may result in short queries taking longer than expected and new connections failing when a long-running user query blocks the commit phase of a processing operation. When processing is affecting query performance, utilizing separate query and processing servers can resolve this resource contention. The use of separate query and processing servers is discussed later in this article.

Considerations When Co-Locating Multiple Analysis Services Databases onto a Single Server (Using a Single Instance or Multiple Instances)

Co-locating multiple Analysis Services databases onto a single server can result in several types of resource contention issues:

- The resources required to process any Analysis Services database can negatively impact the query performance of all other databases on the server while the processing of any database is occurring. With multiple databases on a single physical server, processing operations may affect querying for longer periods of time, and you will generally not want to process too many databases simultaneously (due to resource requirements of each processing operation).
- A single resource-intensive user query (such as multi-threaded storage engine intensive queries) against any single cube in any database can consume a significant portion of the resources on the entire server and negatively affect the performance of all other user queries on that server. For more information, see the Analysis Services Performance Guide.

Co-locating multiple Analysis Services databases onto a single server can result in several types of maintenance issues. When you consolidate multiple databases into a single physical server, down time related to operating system or hardware issues will negatively affect more databases and consequently more users. For example, applying an operating system service pack or upgrading hardware could take all of the databases on the consolidated physical server offline for a period of time. Furthermore, if you consolidate using a single instance of Analysis Services for all databases, instance management (such as

SQL Server service packs and cumulative updates) can also impact availability of the databases in a consolidated environment.

The best practices discussed in the remainder of this paper describe how we have resolved these resource contention issues.

Best Practices

The following best practices describe the consolidation solutions that we have successfully implemented in customer environments to realize the benefits and mitigate the risks inherent in each of these consolidation solutions. Utilizing these best practices, we have increased processing performance and hardware utilization, decreased power consumption, simplified manageability through standardization, and increased availability.

Co-Locating the Relational Data Warehouse with the Analysis Services Instance

The challenge when the relational data source is collocated with the Analysis Services instance is to realize the benefits of improved processing performance and increased hardware utilization without impacting user queries. We implemented this consolidation option in two different ways to avoid resource contention with user queries.

Utilizing a Processing Window

A processing window is a period of time during which the Analysis Services instances on a single server are resolving few or no user queries and one that is long enough for all required processing operations to complete. A typical scenario is an environment where querying is not occurring overnight (although the definition of overnight is getting shorter). The benefit of this scenario is that the co-location of the SQL Server relational warehouse with the Analysis Services instance on a single physical server enables you to reduce the size of this processing window by increasing the speed of processing. If you have such a processing window, you will be able to complete processing during a period in which there is little or no resource contention on the server related to querying. However, as you place more Analysis Services databases on a single server, you need either more server resources or more time.

When you have such a processing window, the resource contention is only between the SQL Server relational engine and the SQL Server analysis engine for processing operations, and it does not affect a significant number of user queries. Solving contention between these two processes during the Analysis Services processing operation requires the application of a number of techniques.

Restricting Memory and Processor Utilization in the SQL Server Relational Engine

You can restrict the amount of memory and processors resources used by the SQL Server relational engine to ensure sufficient resources are available to Analysis Services. You can also use disk layout techniques to limit I/O contention between the SQL Server relational engine and Analysis Services.

Restricting Memory Utilization in the SQL Server Relational Engine

The SQL Server relational engine uses memory in its buffer pool and memory outside of its buffer pool.

- You can directly restrict the memory used by SQL Server in its buffer pool.
- You can indirectly restrict the memory used for connections by limiting the number of connections.
- You cannot restrict the amount of memory used by SQL Server for threads, extended stored procedures, and the CLR.

To manage the size of the SQL Server buffer pool, you can use:

- The max server memory setting of the **sp_configure** system stored procedure to place an upper limit the amount of buffer pool memory used by an instance of the SQL Server relational engine.
- Resource Governor in SQL Server 2008 to set different amounts of buffer pool memory within an instance on a per workload basis, guaranteeing a minimal amount of memory for each workload. For more information about Resource Governor, see Managing SQL Server Workloads with Resource Governor and Using the Resource Governor.

You can use these options together. For example, if you have users querying the relational data warehouse for reporting purposes and Analysis Services querying the relational data warehouse for processing purposes, you can use the max server memory setting to set an upper limit on usage of buffer pool memory and then use Resource Governor to guarantee the Analysis Services processing queries (or the reporting queries) a specified amount of memory resources.

> **Important**: You should use the capabilities built into the SQL Server relational engine to restrict its use of memory; you should not use Windows® System Resource Manager (WSRM) for restricting memory use by the SQL Server relational engine. For more information about WSRM and SQL Server, see Scaling Up Your Data Warehouse with SQL Server 2008 and Consolidation Using SQL Server 2008.

Restricting Processor Utilization in the SQL Server Relational Engine

You can restrict the use of processor resources by the SQL Server relational engine either by limiting the use of all processor resources by the SQL Server relational engine or by allocating specific processors to the SQL Server relational engine:

- Use WSRM to limit the amount of processor resources utilized by the SQL Server relational engine across all processors. WSRM is an optional (and free) component of the Enterprise and Datacenter editions Windows Server® 2003 and of all editions of Windows Server 2008. WSRM uses Windows Management Instrumentation (WMI) to monitor running processes, monitoring the memory size (virtual and physical) and processor utilization, and to prevent any process from exceeding specified upper limits. WSRM is our preferred approach to regulating the utilization of processor resources by all processes. For more information about WSRM, see WSRM for Windows Server 2008 R2, WSRM for Windows Server 2008, and WSRM for Windows Server 2003.
- You can also use processor affinity within the SQL Server relational engine to restrict processor utilization to specific processors by means of two affinity mask options, affinity mask and affinity I/O mask, thus leaving remaining processor cores for the SQL Server analysis engine.

> **Important**: On large multi-processor servers, you may need to set MAXDOP to 4 (or less) to prevent excessive parallelization of Analysis Services processing queries by the SQL Server relational engine.

> **Note**: If you are running SQL Server 2008, you can also use Resource Governor to throttle CPU consumption by processing queries.

Tip: You can use WMI to dynamically configure these resource utilization settings during processing and then reconfigure them when processing completes. For more information, see WMI Provider for Configuration Management Concepts.

Restricting Memory and Processor Resource Utilization in Analysis Services

You cannot directly restrict the SQL Server analysis engine to specific memory and processor resources.

Utilizing Settings in Analysis Services to Restrict Memory and Processor Utilization

Analysis Services does not support processor affinity, and the max memory settings for Analysis Services are soft settings, not hard settings. By soft, we mean that when specified maximum memory settings are exceeded, objects in cache will be evicted to attempt to avoid exceeding the memory settings, but you can still blow through these memory settings under a variety of circumstances. For more information, see the SQL Server Analysis Services 2008 Performance Guide. There is a HardMemoryLimit setting in the msmdsrv.ini file in SQL Server 2008 Analysis Services that enables you to specify a threshold beyond which Analysis Services will start cancelling operations if the memory exceeds a specified threshold. However, the use of this setting will not prevent Analysis Services from exceeding the specified threshold under all circumstances.

The buffered file mode that Analysis Services uses can cause the Windows system file cache to grow quite large and starve the relational engine of all memory resources during Analysis Services processing, resulting in an apparent "hang" during processing. If you encounter this scenario, you can limit the amount of system file cache used by Analysis Services by modifying the LimitSystemFileCacheSizeMB setting in the msmdsrv.ini file. Limiting the amount of system file cache used by Analysis Services will ensure that memory remains available for the relational engine.

Utilizing WSRM to Restrict Memory and Processor Utilization in Analysis Services

WSRM is our preferred approach to place hard restrictions on the usage of memory and processor resources used by the Analysis Services process.

Important: When you utilize WSRM to place a hard restriction on the usage of memory resources by Analysis Services, the Analysis Services process will still see the total amount of memory on the server. Therefore, you need to adjust the values for the **TotalMemoryLimit** and **LowMemoryLimit** properties to reflect the appropriate percentage of the total physical memory on the server that WSRM is allowing Analysis Services to utilize in order for these settings to function as designed.

Note: If you utilize the Preallocate setting in the msmdsrv.ini file, do not grant the Lock Pages in Memory local user policy to the Analysis Services service account if you utilize WSRM. If you use Local System as the service account, be aware that Local System has this privilege by default.

Resolving Disk Contention and Maximizing I/O Resources

You can resolve disk contention between the relational engine and Analysis Services by utilizing separate LUNs, utilizing one set of LUNs for Analysis Services and another set for the relational engine. For more information, see Accelerating Microsoft adCenter with Microsoft SQL Server 2008 Analysis Services.

Maximizing Processing Performance in a Consolidated Environment

For many consolidated environments, you may want to separate the **Process Data** and **Process Index** phase of the Analysis Services processing operation to improve the performance of the entire processing operation. Separating these processing operations enables you to reconfigure the SQL Server relational engine after the **Process Data** phase of the processing operation completes to enable all server

resources to be made available to Analysis Services during the **Process Index** phase of the processing operation. The **Process Index** phase is much more processor and memory intensive than the **Process Data** phase from a single thread perspective. By running the **Process Index** phase separately, you can parallelize the indexing operation to make use of all available resources, because the **Process Index** phase does not require access to the SQL Server relational engine

Utilizing a Dedicated Processing Server

In the absence of a processing window, we implemented this consolidation solution by performing all processing on physical servers not utilized for querying (called processing servers) and directing all user queries to one or more physical servers not utilized for processing (called query servers). Utilizing separate servers for querying and processing eliminates the resource contention during processing that reduces query performance while enabling you to maximize processing performance.

Utilizing a dedicated processing server and one or more query servers may, at first, seem inconsistent with consolidation. Indeed, it is if you are only looking at consolidating a very small number of Analysis Services databases. However, if you are looking to consolidate dozens of older, department-sized servers onto a small number of newer, server-class servers, consolidating to a small number of servers works very well – each server with its own standardized role.

Note: When utilizing a dedicated processing server, we have frequently consolidated the relational data warehouse on the processing server.

This solution requires you to use one of several options to synchronize the newly processed objects from the processing server to the query server(s):

- **Analysis Services Synch Method** – You can use the Analysis Services synch method to perform an online synchronization of the newly processed database on the processing server with the database on one or more query servers. This method synchronizes only the changed files between the processing server and the query server(s), provided that your folder structures match between the source and destination databases. If the folder structure does not match, all files are copied and synchronized rather than only the changed files.
- **Backup and Restore Method** – You can back up the newly processed database on the processing server, copy the backup file to the query server(s), and then restore the database while querying is occurring (a database lock is required at the end of the restore operation). To maximize query performance during restoration, you can restore the database on the query server using a different database name and then, after the restore is complete, drop the existing database and rename the newly restored database. This latter option also enables you to perform any quality checks on the restored database before you present it to users.
- **Attach and Detach Method** – You can detach the newly processed database from the processing server, copy the detached database folder to the query server(s), and then attach it to the query server(s). You cannot attach over an existing database, and you cannot change the name of the database while you are attaching. You can attach it as read/write to a single query server or as read-only to multiple query servers. This latter option is quite useful if your query workload is processor and memory intensive rather than I/O intensive. You can attach the newly processed database to a physical or a virtual server.
- **Robocopy Method** – You can use Robocopy to copy only the delta files from your processing server to your query server(s). This method, however, requires you to either stop the Analysis Services engine or detach the database that is being synchronized. For more information, see

Sample Robocopy Script. In Windows 7 and Windows Server 2008, Robocopy has been enhanced with the multi-thread support. For syntax, see Robocopy.

For more information about these options, see Analysis Services Synchronization Best Practices.

Co-Locating Multiple Analysis Services Databases onto a Single Server

In order to realize the benefits of increased hardware utilization when co-locating multiple Analysis Services databases onto a single server, we have implemented this consolidation option either with a single Analysis Services instance or with multiple Analysis Services instances. When consolidating multiple databases, we have generally utilized multiple query servers and one or more processing servers, and we have co-located the relational database source with its corresponding processing server. When we utilize multiple query servers, we use a number of techniques to load-balance user queries.

Important: We have only implemented this type of consolidation with small to medium sized Analysis Services databases; consolidating multiple very large or very complex databases onto a single instance simply has not made sense from a resource contention point of view. While the definition of a small or medium sized Analysis Services database is at best imprecise, our definition includes databases against which individual queries do not consume a significant portion of the server resources to resolve and which are primarily constrained by concurrency rather than query complexity. For more information about resource usage by queries, see the SQL Server Analysis Services 2008 Performance Guide.

Utilizing a Single Instance or Multiple Instances

We have rarely seen the need to spread Analysis Services databases across multiple instances of Analysis Services on a single physical server to improve query performance. We have used multiple instances of Analysis Services on a single instance when users are querying a database in one instance while the same database in a second instance is being synchronized. This configuration is used to enable user queries to be directed to the newly synchronized database on the second instance without downtime.

Note: Multiple instances of the SQL Server relational engine are sometimes used to manage the priority of queries. With Analysis Services, you can use the CoordinatorQueryBalancingFactor and CoordinatorQueryBoostPriorityLevel properties to prioritize long-running queries versus short queries. For more information, see the SQL Server Analysis Services 2008 Performance Guide.

When choosing between querying all databases in a single instance versus spreading the databases across multiple instances on a single server, consider the following:

- Utilizing a single instance rather than multiple instances for the same number of databases generally uses fewer system resources.
- All data security operations can be performed with database level permissions such that each database can have a different set of database administrators and users with read permissions.
- Processing and synchronization operations are generally scripted and performed by SQL Server Agent or other scheduling tools, and they are executed using server-level permissions. For these types of operations, it generally does not make a difference whether a single or multiple instances are used.
- If you have different sets of instance-level administrators, you need multiple instances. For example, to perform a trace of data within a database or to clear the cache of a database, you must be a server-level administrator. If you need to allow the administrators of a database to

trace queries in their database without being able to see the data in another database, you will need to place that database within another instance.
- If you have different instance-level properties for different databases, you need multiple instances. For example, you may want to have different priorities for long-running queries on one instance versus another instance. We have not, however, found this need to date in the customer organizations with which we have worked.

Utilizing Multiple Processing Servers and Query Servers

With multiple Analysis databases on a single physical server, the processing of any single database can affect the query performance of any query against any database on the server. A long-running query against a database can prevent the processing of that database from completing, unless the long-running query is forcibly terminated. Furthermore, within a single instance, a long-running query can prevent a processing operation from completing its commit phase and block subsequent queries as well as all new connections until the commit operation completes successfully or times out.

When consolidating databases onto a query server, we have generally utilized one or more processing servers and multiple query servers, and we have co-located the relational database source with its corresponding processing server. These servers provide the following benefits:

- Processing servers – We have sometimes utilized multiple processing servers so that multiple databases can easily be processed simultaneously. Processing databases on servers that are not being used for querying insulates the query performance of all Analysis databases from resource contention due to the processing of any single database. Your need for one or more processing servers will depend upon the frequency with which each Analysis database is processed and the number of databases that you are processing. Co-locating the relational data warehouse with its corresponding processing server increases processing performance and server utilization for the reasons previously discussed.
- Query servers – When using separate processing and query servers, we have always utilized multiple query servers for performance and availability. With multiple query servers, we can load balance our workload across multiple query servers to maximize performance or take a server offline without impacting availability.

Synchronizing a processed database from its processing server to multiple query servers adds minimal additional complexity or synchronization time compared to synchronization with a single query server. You can synchronize all query servers either simultaneously or in two (or more) groups. For example, you could synchronize a processed database to your first group of query servers while directing all new queries against that database to the second group of query servers. If you use Server Sync, this approach allows existing queries against that database to complete while synchronization is occurring and new queries to execute against the servers that are not being synchronized – to maximize query performance. After synchronization is complete for the first group, all queries against that database can then be directed to the first group of query servers so that the newest data is being queried while the second group of query servers is synchronized. This approach is illustrated in figures 1 and 2.

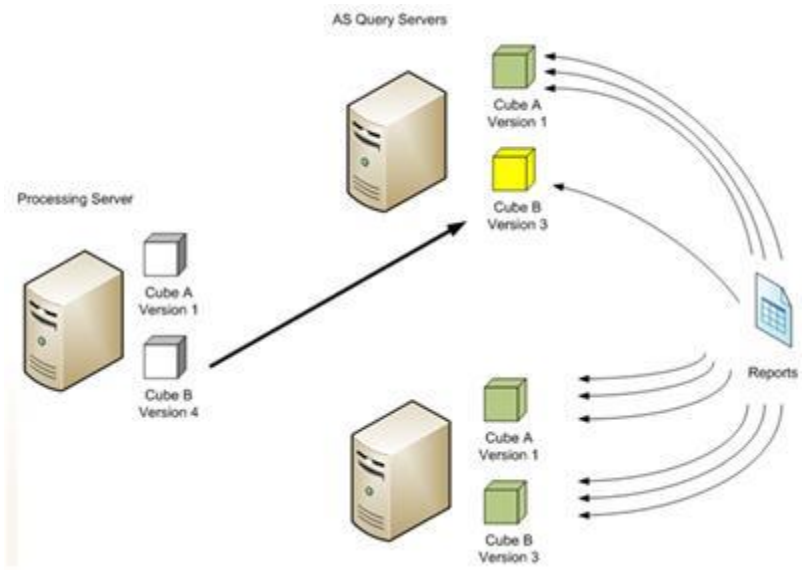


Figure 1: Synchronizing First Query Server

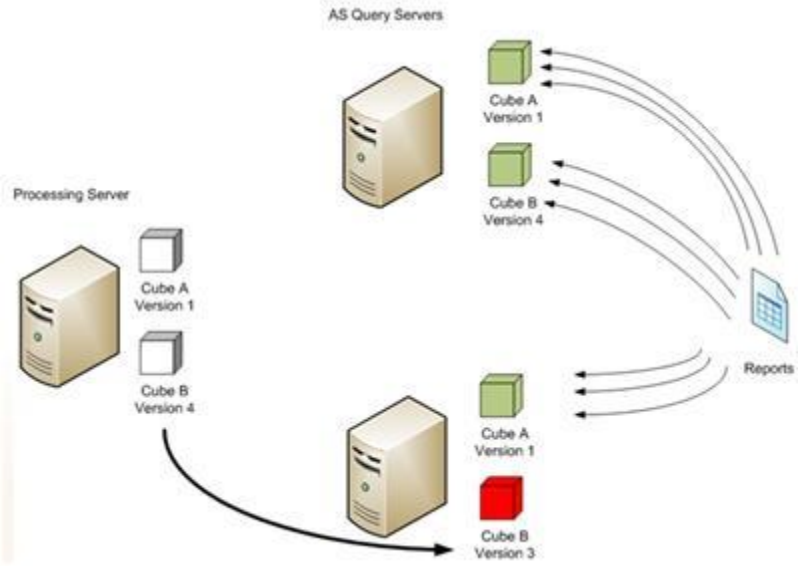


Figure 2: Synchronizing Second Query Server

Load Balancing User Queries for Performance and Availability

When you utilize multiple query servers, you need to load balance the user queries across the query servers for performance and availability. We are taken several approaches to load balance user queries across multiple query servers.

Load Balancing at the Presentation Layer

One approach is to load balance at the presentation layer. This approach utilizes multiple Web servers, with some Web servers directing MDX queries to one query server and other Web servers directing MDX queries to a second query server. This approach is discussed in Scale-Out Querying with Analysis Services and is illustrated in figure 3.

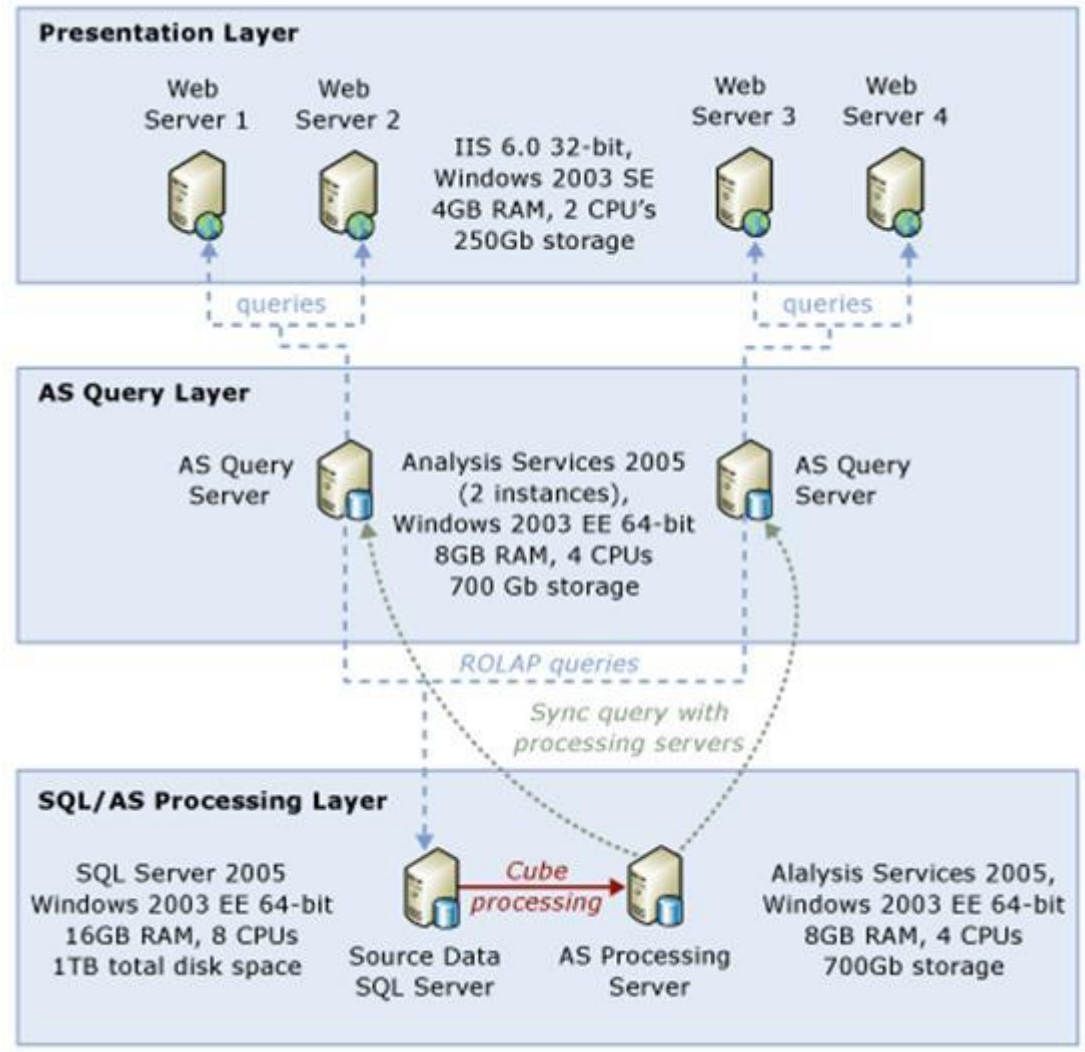


Figure 3: Query Scale-Out Topology with Multiple Web Servers

This approach works well as long as the workload directed to each Web server is evenly balanced, but it does not provide load balancing where that is not the case. For example, suppose some users are connecting from Excel to one query server and through Reporting Services to a second query server. In this scenario, the load from each set of users may well not be balanced.

Load Balancing using Network Load Balancing Software or Hardware Solutions

Another approach is to use network load balancing software, such as Microsoft's Network Load Balancing (NLB) software solution, or network load balancing hardware, such F5.

Using Network Load Balancing Software

NLB is an optional component in the Windows Server 2008 and Windows Server 2008 R2 operating systems and is built into Windows Server 2003. NLB load balances network traffic among multiple servers in an IP cluster. For more information, see Windows Server 2008 NLB and Windows Server 2003 NLB.

NLB is a software solution to create a cluster of machines, each with two IP addresses – a private unique one and a virtual one that is shared by all machines in the cluster. Each machine in the cluster runs an algorithm that determines whose turn is next in responding to requests. These machines also exchange heartbeats with each other, so they all know if one server goes down and won't allocate any more requests to such a machine until it is healthy again.

NLB supports a number of affinity options depending on your version of Windows Server. When no affinity is specified, all network requests are load-balanced across the cluster without respect to their source. In general, when affinity is enabled, all client requests from the same IP address are responded to by the same cluster host.

However, NLB has several limitations when working with multiple Analysis Services query servers, particularly when each query server has a subset of the Analysis Services databases to which queries are being directed (for example, all databases are located on at least two servers for availability, but all databases are not located on all servers):

- NLB load balances at the IP address level, rather than at the database level. For NLB to load balance across multiple Analysis Services query servers, each must contain the same set of databases.
- NLB monitors availability at the server level, rather than at the Analysis Service process or database level. As a result, if the Analysis Services process is not responding for any reason, NLB will continue to forward user queries to the nonresponsive Analysis Services query server. In addition, as an Analysis Services database is being updated or is out-of-date, NLB has no ability to remove that database from the cluster. Rather, an NLB administrator would have to remove the entire server from the cluster during an Analysis Service synchronization process.
- NLB has no ability to monitor processor load on each of the servers in an NLB cluster and direct user queries to the server with the least load.

Due to these limitations, NLB is not our recommended solution for load balancing user requests across multiple Analysis Services query servers.

Using Network Load Balancing Hardware

Network load balancing hardware solutions have more sophisticated affinity options, such as cookie persistence, and they include dynamic load balancing algorithms that track server performance. However, these solutions do not solve the problem of removing a single database on an Analysis Services query server from the cluster while continuing to direct user queries to the other databases on that query server. In addition, these hardware solutions do not, out of the box, monitor the availability of the Analysis Services process itself.

Due to these limitations, using network load balancing hardware solutions is not our recommended solution for load balancing user requests across multiple Analysis Services query servers.

Load Balancing Using a Custom Load Balancing Solution

When using multiple query servers with multiple databases on each server, we have found the need to load balance queries at the database level as well as the need to monitor service availability at the Analysis Services level. To accomplish this, we wrote a custom Analysis Services load balancing solution (ASLB) that consists of a load balancing Web service backed by a SQL Server load balancing metadata

database to load balance MDX queries across multiple query servers. The metadata database contains information about each Analysis Services query server to which queries can be redirected by the Web service.

Note: This load balancing Web service can be clustered using NLB for availability, and the load balancing metadata database can also be mirrored for availability.

The ALSB solution solves the problems previously discussed as follows:

- **Process availability monitoring** – A SQL Server Agent job monitors the availability of the Analysis Service process on each query server. If the process fails to respond, this information is recorded in the metadata database so that no further user queries are directed to that instance of Analysis Services until it is healthy.
- **Resource Utilization Monitoring** – A SQL Server Agent job monitors the processor utilization on each query server and records this information into the metadata database. The Web service uses this information to direct user queries to a query server that is not overloaded and which has a copy of the requested database.
- **HTTP Redirection** – As illustrated in figure 4, the user application (such as Excel or Reporting Services) connects to the ASLB Web service specifying a particular Analysis Services database in the URL. The Web service performs a table lookup to retrieve the name of a query server that is running, that contains the most up-to-date version of the requested database, and that has the lowest workload. The load balancing Web service constructs the URL for the appropriate

Analysis server and redirects the user application to that URL.

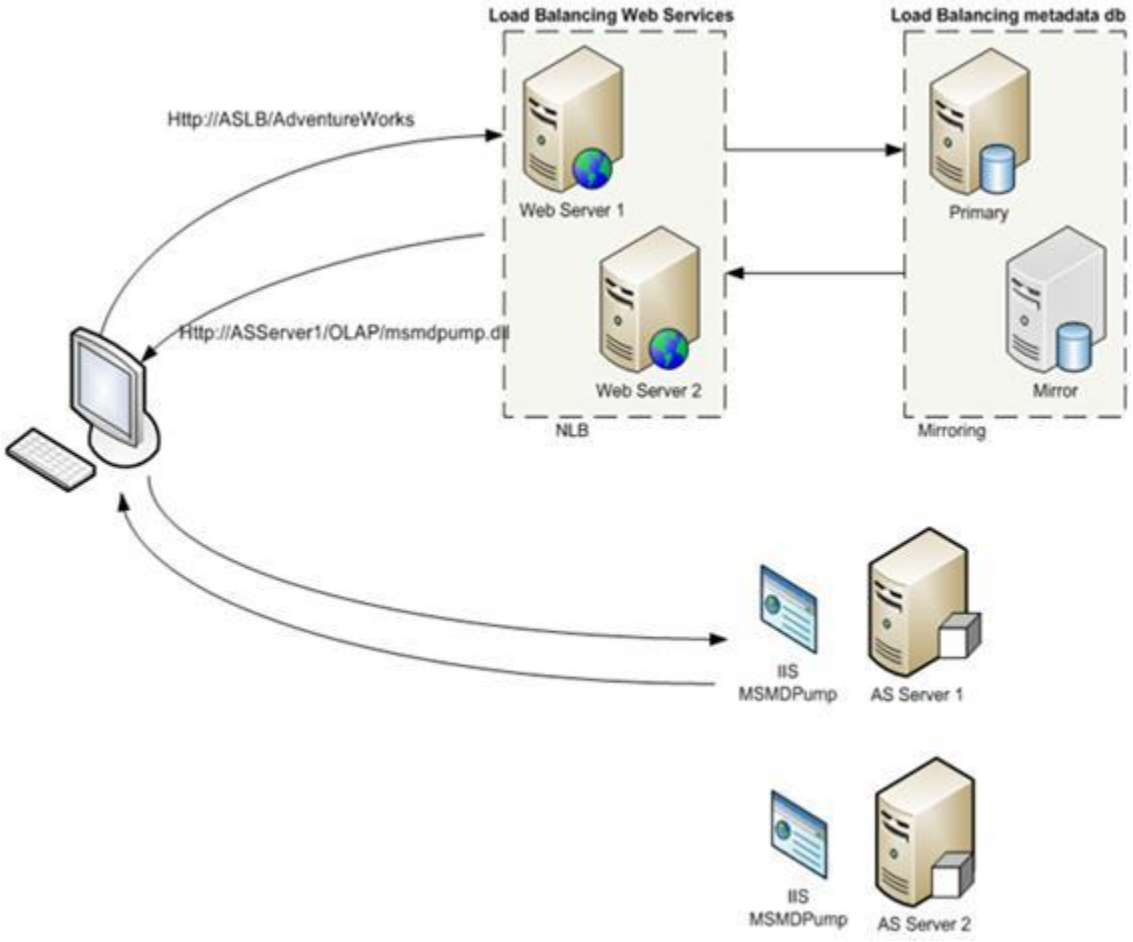


Figure 4: Individual Query Using the Analysis Services Load Balancer

- **Removing individual databases from a cluster** – ASLB enables you to remove a database from being available to accept new queries by simply modifying a value in the appropriate table in the metadata database. Figure 5 illustrates this concept during the synchronization of cube B from the processing server to a query server, directing all new queries to the second query server until server synchronization completes. Meanwhile, queries to all other databases on the first query server continue unabated, enabling you to drain a server workload without terminating running queries.

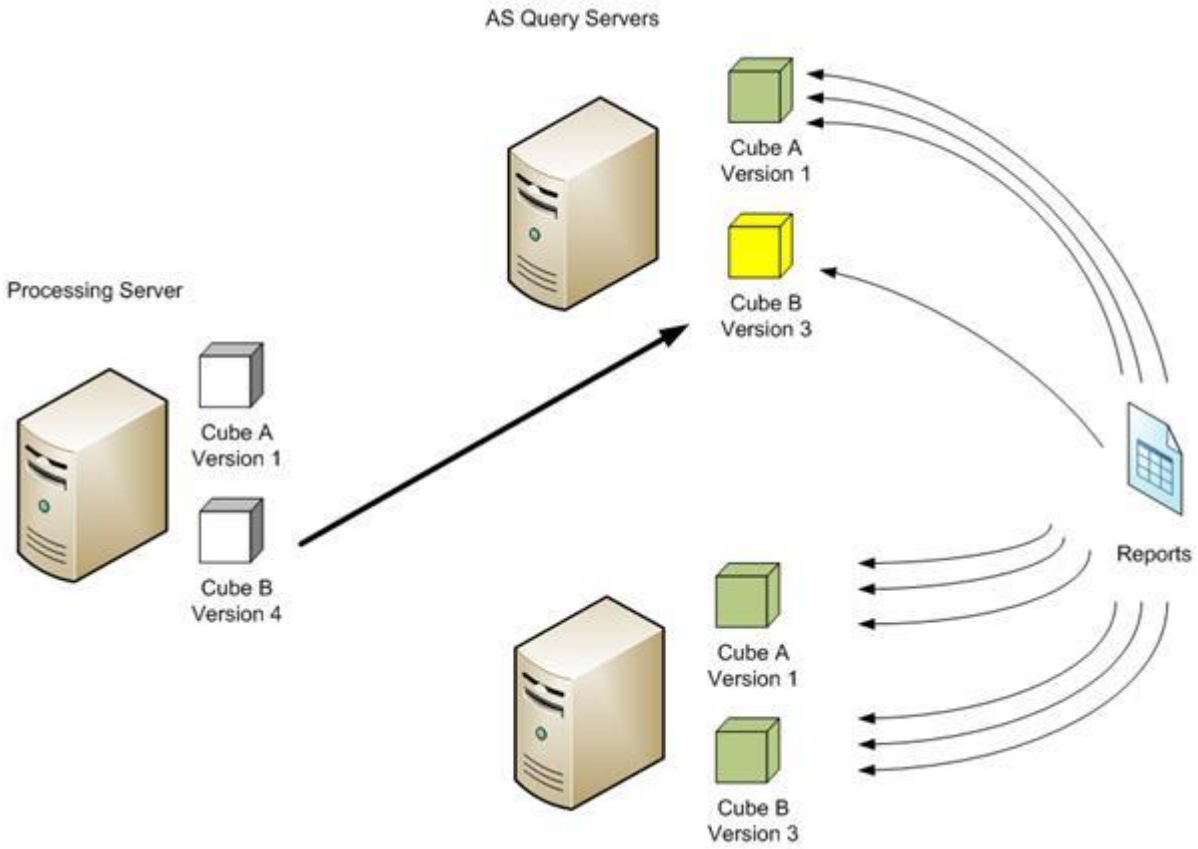


Figure 5: Querying and Processing Using the Analysis Services Load Balancer

- **Utilizing different databases on each server** – If you utilize more than two query servers, each query server does not have to have the same list of Analysis Services databases because ASLB has a record in the metadata database of which servers have which databases. For example, if you have four query servers, you can have your most heavily queried databases on all four query servers and your least frequently queried servers on only two of the query servers. This configuration provides load balancing and availability benefits. Finally, you can also have Analysis Services databases on multiple query servers and use ASLB to load balance among some of these query servers while leaving other query servers "in reserve" – for example, bringing them online for a particular database only during peak time periods. The query servers "in reserve" can also be virtual servers that are provisioned automatically based on demand (more on this in another paper).
-
-

Important: This solution uncovered several bugs that have been fixed in the most recent cumulative updates to SQL Server 2008 Service Pack 1 (SP1). To receive these bug fixes, you should patch your SQL Server using Service Pack 1 Cumulative update 4 or newer. For more information about the newest cumulative updates, see Knowledge Base article #970365.

Note: The code for the ASLB solution as well as an installation and configuration document is available for download on the Microsoft download center at http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=c7737e80-8dfd-4cb6-a27e-69bbe03b2f9e.

Links

SQL Server Consolidation at Microsoft

http://technet.microsoft.com/en-us/library/dd557540.aspx

*Green IT in Practice: SQL Server Consolidation in Microsoft IT*http://msdn.microsoft.com/en-us/architecture/dd393309.aspx

Conclusions

Server consolidation in the Analysis Services environment can be used to increase processing performance, fully utilize server resources, reduce management costs, and save energy. Analysis Services server consolidation may consist of co-locating the relational data warehouse with the Analysis Services instance and / or standardizing on Analysis Services processing and query servers. To realize these benefits and mitigate the risks, you must manage the resource contention inherent in Analysis Services consolidation. Properly managed, Analysis Services server consolidation can be used to increase performance as well as availability.

Analysis Services Distinct Count Optimization Using Solid State Devices

A Real Practices Case Study at Xbox LIVE

Authors: Joseph Szymanski, Tyson Solberg, Denny Lee

Technical Reviewers: Lindsey Allen, Akshai Mirchandani, Heidi Steen

Executive Summary

To expand on the distinct count optimization techniques provided in the Analysis Services Distinct Count Optimization white paper, this technical note shows how using solid state devices (SSDs) can improve distinct count measures. We recount the experiences of the Microsoft Entertainment and Devices Data Warehousing Team (known for Xbox, Xbox LIVE, XNA, and Zune) in our analysis of applying SSDs to a real-world, distinct count heavy, Microsoft SQL Server Analysis Services customer environment. The key conclusion is that enterprise SSD devices, when combined with a well optimized Analysis Services MOLAP cube, will drastically improve the performance and scalability of the cube when it accesses distinct count measures. It can also improve non-distinct count measures, if the calculations being performed rely heavily on storage-engine calculations.

Purpose

Analysis Services distinct count measures are extremely expensive in all aspects of an Analysis Services solution – the time requirements for processing the data, the long query durations, and large storage space requirements. Often the best approach is to convince the analysts using your cubes to use alternate measures or calculations. However, in many cases the distinct count-based Key Performance Indicators (KPIs) are key components of business analytics systems. In such cases, the focus has to move from "Are you *sure* you need distinct count?" to "How can we make distinct count queries fast(er)?"

One way to improve the performance of distinct count measures is to change the business problem that they attempt to measure (for example, limiting the time range for distinct counts to specific years or months rather than all years). However, when you have exhausted your ability to reduce the solution's complexity by simplifying business processes, you can turn to other techniques to improve your solution. This paper endeavors to share several that we came across in real-world enterprise usage scenarios.

As originally described in Analysis Services Distinct Count Optimization, distinct count calculations are Storage-Engine-heavy, resulting in heavy disk I/O utilization (that is, higher latencies and slower throughput). This leads us to the first and foremost among those techniques: the use of solid state devices (SSDs), which have taken the world by storm. Given the high initial cost, relative newness, and lower capacity per device, it is often difficult to find a good fit for them in the enterprise. However, Analysis Services dovetails well with the SSD story because SSDs offer ultra-high read speeds for both random *and* sequential I/O. Analysis Services queries generally access a large amount of relatively static data and SSDs provide the speed and flexibility required. For example:

- Analysis Services scenarios are well suited for SSDs because most are designed for fast read performance.
- The biggest benefit that SSDs offer over physical disks is that they provide random I/O read speed that is nearly as high as sequential I/O read speed – orders of magnitude faster than spin disks.
- Data stored in MOLAP cubes is an ideal target for SSDs: most SSDs are rather low in capacity, but Analysis Services MOLAP cubes are generally small in size when compared to their data warehouse source. For example, in our sample case we have a 10-terabyte warehouse, which would cost an extraordinary amount of money to move to SSDs, but a 160 GB cube, which would be very easy and inexpensive to move to SSDs.
- While the initial costs of SSDs are higher than those of spin disks, the overall lifetime costs of SSDs are comparable to spin disks because of cooling costs, differences in power consumption, and general maintenance costs associated with your storage (SSDs typically have lower maintenance costs).

Scenario

This paper covers the business scenario in which business users seek to improve the performance of their Analysis Services distinct count operations. As an example, the Microsoft SQLCAT team worked with the people at the Microsoft Entertainment and Devices Data Warehousing who needed to build their critical KPIs based on distinct counts. We decided that a significant research effort to find a way to make their distinct count queries run faster through design methodologies and the use of SSDs was worthwhile.

The engineering team, examining the scenario at hand, endeavored to find out the following:

a) Can distinct counts be made to run faster: *a.* Given a faster I/O system? *b.* Given better cube design?

b) Will such improvements be cost effective?

c) Will such improvements improve or hinder scalability, compared with the current design?

Datasets

The database is a real-world production dataset with the following characteristics.

Dataset	Sizes
MOLAP Cube	120 GB
SQL DW	10.0 terabytes

The cube design, for the distinct counts, is relatively simplistic, as noted in Figure 1.

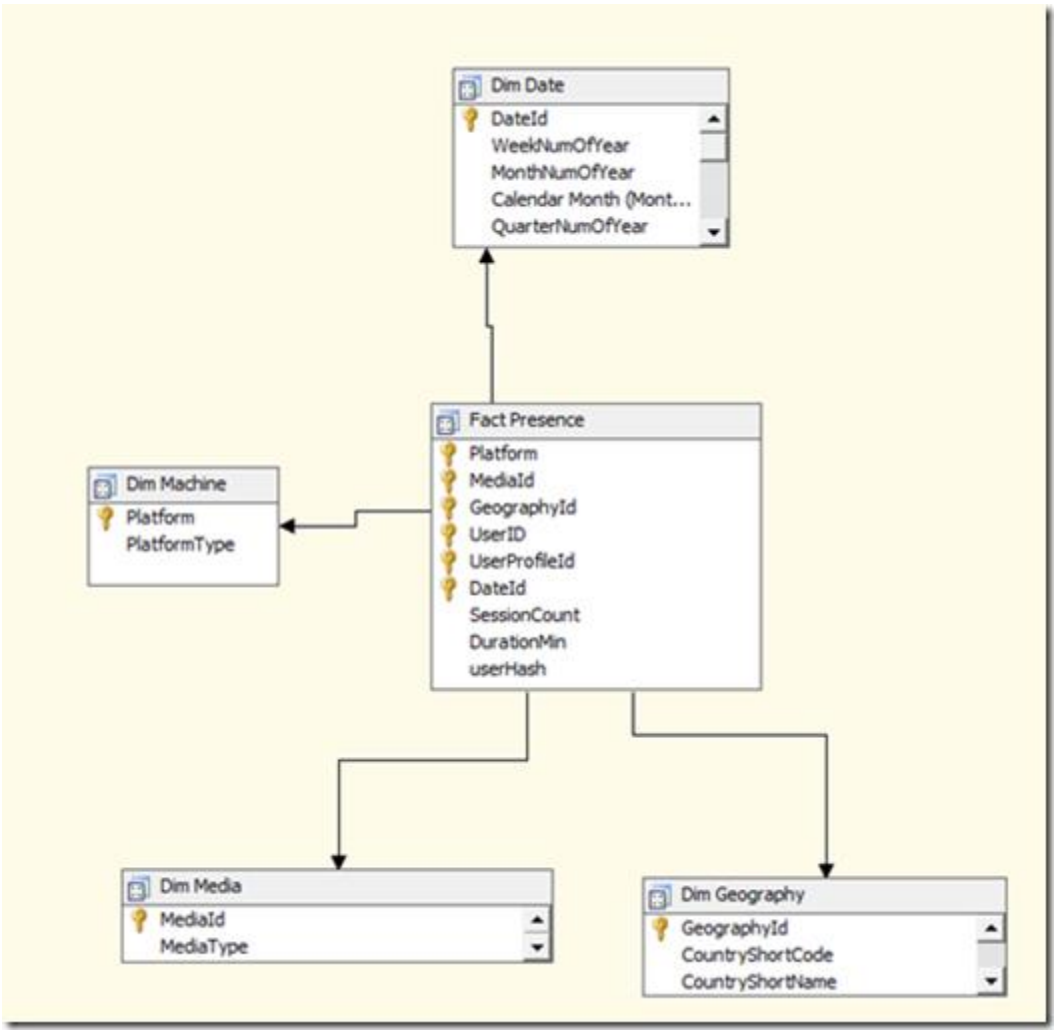


Figure 1: Usage Cube Schema

OLAP Query Characteristics

One of the primary business drivers that needed an answer is: "How many distinct users used the service for <a time period>?" To get this answer, we focused on a single simple MDX statement and sought to make it run as fast as possible from cold cache (that is, we cleared all existing caches to force the data to load from the pertinent device).

```
SELECT [Measures].[Online Unique Users] ON 0,
[Date].[Date].[Date] ON 1
FROM [Usage]
WHERE
[Date].[Calendar Month].[Calendar Month].&[2009]&[1]
```

The measure "Online Unique Users" is a distinct count measure that, for the time period of the month selected, scans through 300 million SQL records in the MOLAP cube dataset.

Test Hardware

We utilized a number of servers to perform our query tests.

User Acceptance Testing Server

HP DL580 G5 4-socket, quad core, 2.4 GHz, 64 GB RAM server connected to a high end SAN array; Chart Label: "UAT SAN"

Development Server

Dell R710 2-socket, quad core, 2.93 GHz, 72 GB RAM

Disk Arrays

- 1 Fusion-io ioDrive PCI-E SSD device; Chart Label: "Dev SSD"
- Two Dell MD1000 enclosures (16 x 750 GB 7200RPM drives); Chart Label: "Dev Hard Drives"

SQL Customer Advisory Team Server

Dell R905 4-socket, quad core, AMD server with 4 Fusion-io ioDrive PCI-E SSD device; Chart Label: "SQLCAT SSD"

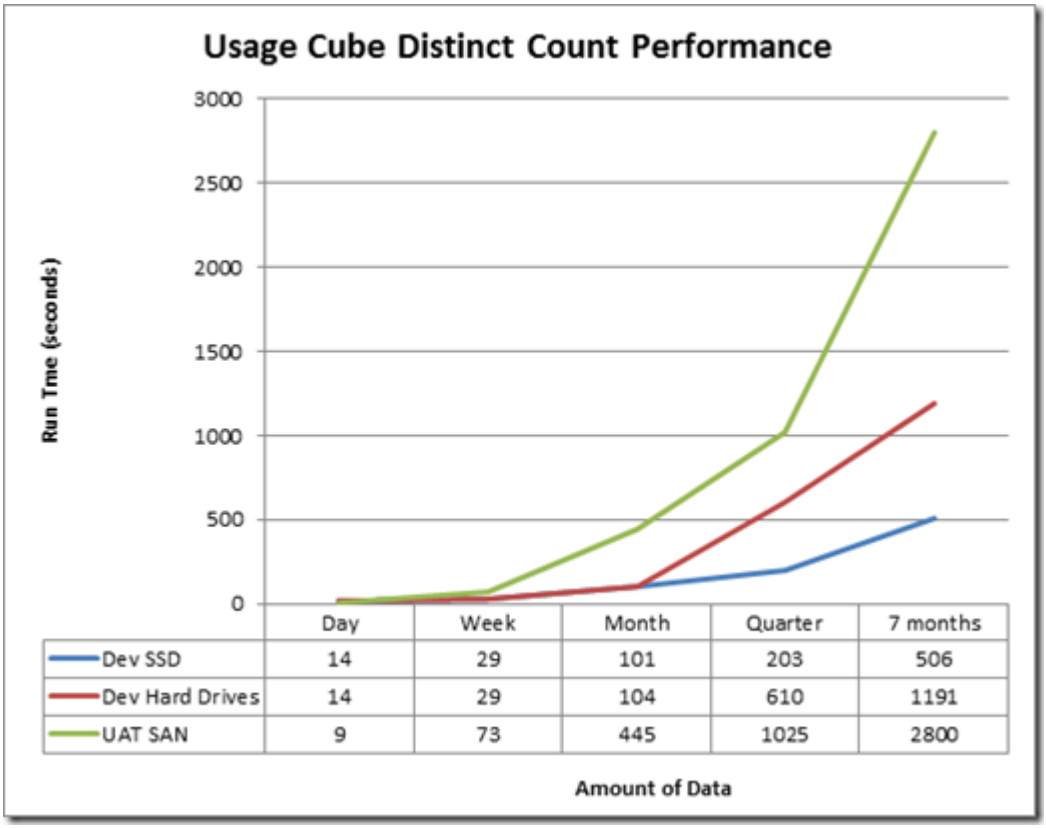
We'd like to thank Fusion-io, Dell, and Hewlett-Packard for the use of and support with their hardware.

Analysis

As originally described in Analysis Services Distinct Count Optimization, distinct count calculations are Storage-Engine-heavy, resulting in heavy disk I/O utilization. Therefore, the original hypothesis for this case study was modeled after the reasonable argument: "If we are disk I/O bound, and Analysis Services provides a 100 percent random read load, SSDs should drastically improve performance."

Initial Query Performance Comparison Between SSDs and Spin Disks

Let's start by comparing the query performance between SSDs and spin disks (local hard drives and SAN).



	Day	Week	Month	Quarter	7 months
Dev SSD	14	29	101	203	506
Dev Hard Drives	14	29	104	610	1191
UAT SAN	9	73	445	1025	2800

Figure 2: Initial Performance Optimization (Lower Is Better)

As can be seen in Figure 2, the use of Fusion-io SSDs resulted in dramatically faster query performance when compared to local drives or SAN drives. For example, for the seven-month query, SSDs were 5.5 times faster than the SAN hardware.

But as with any successful BI performance improvement, Analysis Services users also asked more resource-intensive questions such as: "For the last year, how many distinct people did <X>?" That is, these questions resulted in distinct count queries that covered a year's worth of data across a substantially larger dataset. In an attempt to find further performance improvements, the engineering team profiled the cube during query time, in production, and found a very odd thing: Disk usage and CPU usage were nowhere near the limits of what the system was able to handle.

This finding was counterintuitive, because all tests were run on a system that was dedicated to this exercise (that is, the system had no other users or processes running concurrently); it was not a matter of resource contention. We then worked with the SQLCAT team and other experts to find out why Analysis Services was not using more system resources to query the cube, starting with an investigation of how the partitions were defined.

The Move to an Optimally Parallel Cube

As noted in Analysis Services Distinct Count Optimization, partitioning significantly improves distinct count query performance. By creating distinct buckets based on distinct value and time, you can significantly improve distinct count query performance by forcing the Analysis Services Storage Engine to fire off many more threads – one for each partition – and therefore more quickly calculate the distinct value. But if partitions are designed with an uneven number of distinct values (such as in Figure 3), the query may ultimately become single-threaded (even though all four partitions are being queried) because the Analysis Services Storage Engine is waiting for the largest partition (data file with values from 1,500 to 20,000) to complete its calculations. This behavior explained the puzzling results around disk and CPU consumption in the earlier tests.

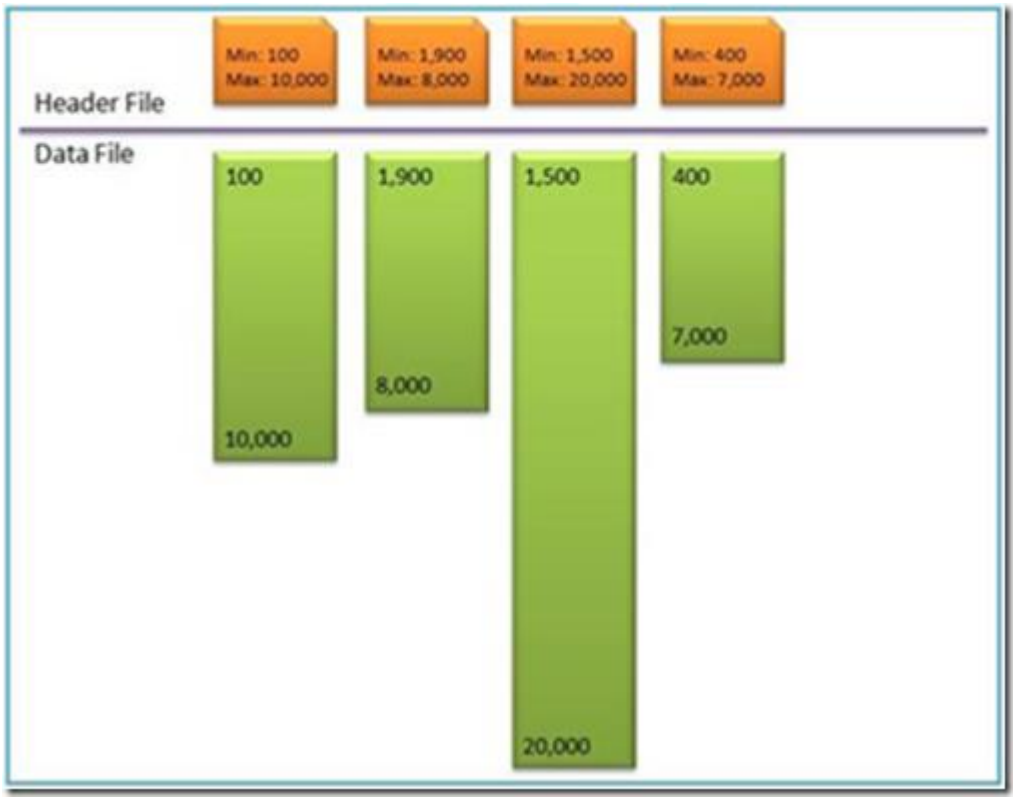


Figure 3: Unevenly Distributed Partitions

A more evenly distributed set of distinct values within the partitions results in all four threads completing at approximately the same time, resulting in minimal spinning and wait time while the calculations are completed. Clearly, parallel queries across the partitions had to be part of our solution.

Figure 4: Evenly Distributed Partitions

Figure 4 shows the even distribution of data among the partitions, which is a key concept for distinct count query optimization. The Analysis Services Storage Engine will initially query the header file to determine which data partitions to query for the range of distinct count values. This way, the storage engine queries only partitions that have the values required to complete the calculation.

After extensive testing, we rediscovered some important rules and added them to the distinct count partitioning strategy to ensure all distinct count queries are optimally parallel:

The distinct count measure must be directly contained in the query.

If you partition your cube by the *hash of a UserID* distinct value, it is important that your query perform a distinct count of the hash of the UserID – not the distinct count of the UserID itself. For fast distinct count query performance, it is important for the distinct count value itself to be placed in its own periodic partition (for example, User 1 repeatedly shows up in only month 1 partition 1, User 100 in month 1 partition 2, and so on) and for the values to be non-overlapping (for example, Users 1-99 in month 1 partition 1, Users 100-199 in month 1, partition 2, and so on). The hashing will cause the records in the same range to be distributed across multiple partitions, therefore losing the non-overlapping behavior. Even if the UserID and the hash of the UserID have the same distribution of data, and even if you partition data by the latter, the header files contain only the range of values associated with the hash of the UserID. This ultimately means that the Analysis Services Storage Engine must query all of the partitions to perform the distinct on the UserID. For more information about these concepts, see the white paper Analysis Services Distinct Count Optimization.

The distinct count values need to be *continuous.*

As implied in Figures 3 and 4, each partition has a continuous range of values so that the partition contains the values from 100 – 20,000 (in this example). Based on the empirical evidence we gathered in our testing for this case, it appears that distinct count query performance improves if the values within the partitions are continuous.

After we followed these two rules, we were easily able to improve query parallelism with *very few changes*.

More specifically, we analyzed our data size, selected a month as the coarse time grain for the distinct count measure group partitions, and then sub-selected the data, per month, into <n> partitions, where *n* is the number of physical CPU cores on the OLAP hardware. We made this decision after we identified a number of options, tested them, and found this particular grain to be the best for our set of data. Other than the partition changes, the cube design stayed the same, and we did not alter any aggregations for this cube. Note, we had followed the established guidelines of the SQLCAT white paper Analysis Services Distinct Count Optimization.

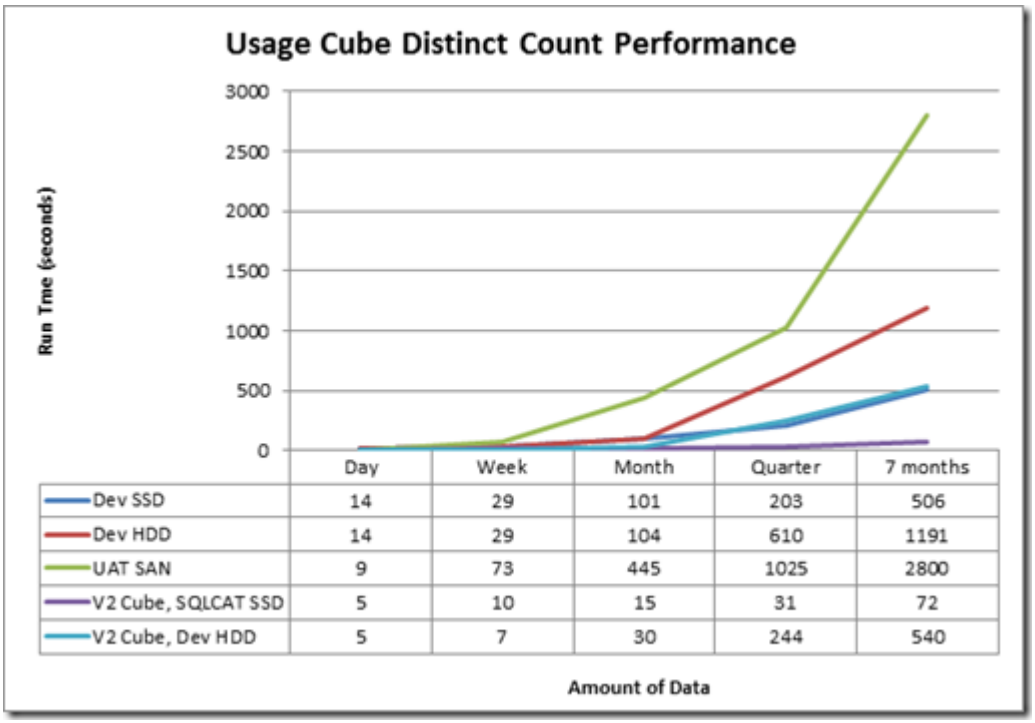
Note: To allow for more repeatable distinct count query comparison, the cube used here contained no aggregations on distinct count measures.

The following lists various combinations of measure group slicer queries and distinct count measures.

SQL Query WHERE clause	**Analysis Services distinct count member**	**Is the query optimally parallel? If not, why?**
`WHERE userid % 16 = 0`	`userid`	**NO:** Query does not return a continuous dataset.
`WHERE CAST(HASHBYTES('SHA1',CAST(userid AS VARCHAR)) AS BIGINT) BETWEEN a AND b`	`userid`	**NO:** The Analysis Services member "userid" is not contained directly in the query.
`WHERE userid BETWEEN a AND b`	`userid`	**YES**
`WHERE userid % 16 = 0`	`CAST(HASHBYTES('SHA1',CAST (userid AS varchar)) AS bigint)`	**NO:** Query does not return a continuous dataset.
`WHERE CAST(HASHBYTES('SHA1',CAST(userid AS VARCHAR)) AS BIGINT) BETWEEN a AND b`	`CAST(HASHBYTES('SHA1',CAST (userid AS varchar)) AS bigint)`	**YES**
`WHERE userid BETWEEN a AND b`	`CAST(HASHBYTES('SHA1',CAST (userid AS varchar)) AS bigint)`	**NO:** The Analysis Services member <Hash of Userid> is not directly in the query.

Now That We Have an Optimally Parallel Cube…

The results were stunning, as shown by the "V2" lines for both SSDs and hard disk drives (HDDs) (where "V2" is the version 2 cube, which follows the optimizations discussed earlier in this paper).



	Day	Week	Month	Quarter	7 months
Dev SSD	14	29	101	203	506
Dev HDD	14	29	104	610	1191
UAT SAN	9	73	445	1025	2800
V2 Cube, SQLCAT SSD	5	10	15	31	72
V2 Cube, Dev HDD	5	7	30	244	540

Figure 5: Performance after enabling "multi-threaded mode".

(Note that in Figure 5, the V2 cube performed at the same speed, on SSDs, in all environments. We show only the SQLCAT line for simplicity.)

The conclusion is that by adding enough I/O (through SSDs so that I/O was no longer a bottleneck), we were able to find and resolve the algorithmic issues, enabling incredible performance. The key is that we never would have found these algorithmic issues without first removing the I/O channel bottlenecks by the use of SSDs.

But Wait, Processes Get in the Way!

At this point, due to operational issues, we were initially unable to go live with the SSD servers. It was agreed that given these results, we should go with the easier-to-implement DAS HDD solution, which offered similar performance. Specifically, our challenge to going live was that, due to a lack of SSD enterprise standards as of mid-2009, the supportability story was too complicated to be sustainable across a large number of servers.

In retrospect, not waiting for SSDs was a serious mistake. The cube went live and user satisfaction plummeted. What went wrong? The results shown in Figure 5 were correct, but somehow, performance was awful.

Parallel User Load

Figure 5, though accurate, shows query response times for a single user only. It does show accurately that, with a solidly parallel cube, good DAS can be nearly as fast as SSDs for reasonably large MOLAP cubes. But a more in-depth analysis found that, in our initial analysis, we failed to consider a highly parallel user load and failed to benchmark a large enough multi-user parallel Analysis Services query load.

Had this been done sooner, the patterns illustrated in Figures 6 and 7 would have been found.

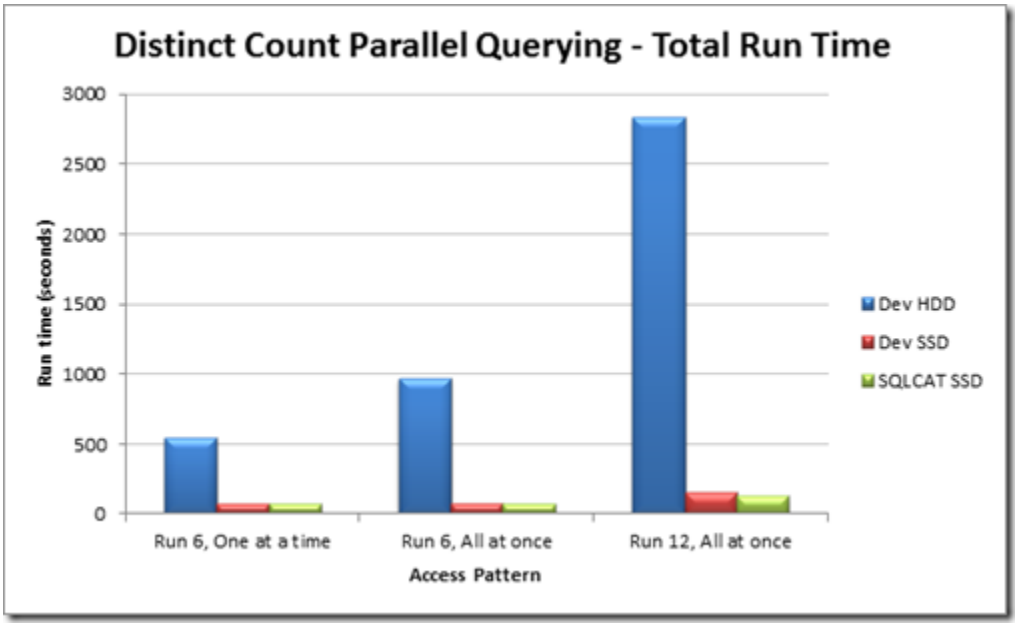


Figure 6: Distinct count querying comparison of different access patterns

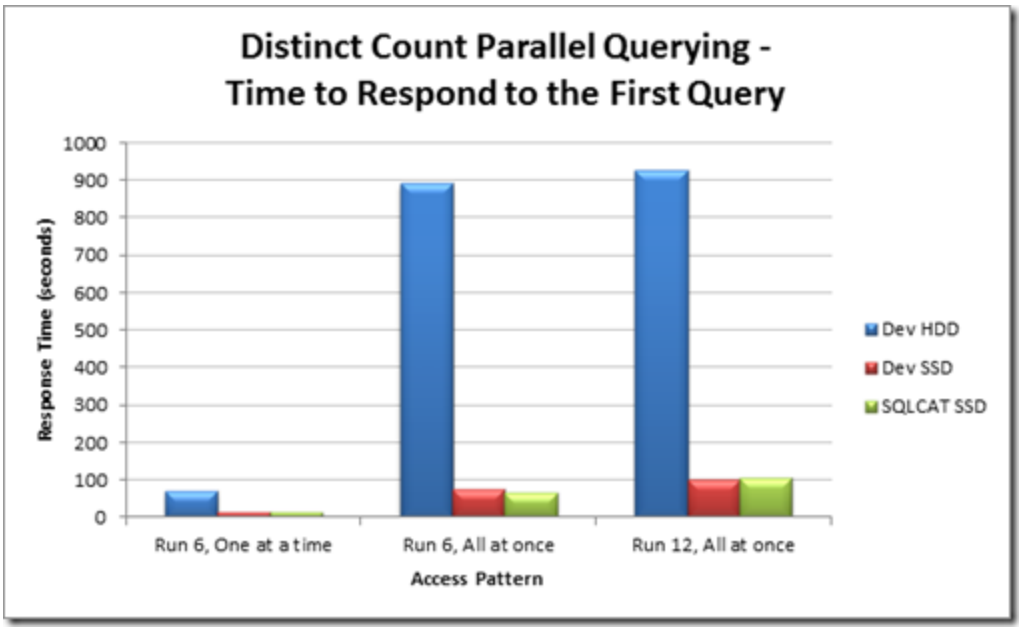


Figure 7: Distinct count query comparison of different access patterns (time to respond to first query)

To execute the parallelization scenario, the engineering team used a tool to execute multiple Analysis Services queries in parallel, with the following characteristics:

- Each query selected a distinct dataset – no overlaps between data.
- Each query was run in two concurrency modes: all at the same time, and with 30 seconds between queries, to simulate a real user load.
- Three access patterns were selected and timed: Running six queries serially, running six queries concurrently, and running twelve queries concurrently.
- Each test run was executed multiple times, clearing caches between runs; the times indicated are averages.
- Two measures were recorded: the time for the first submitted query to complete, and the total time for all queries to complete.

After we captured and analyzed this data, our conclusion was clear: Single-user queries benefitted from the use of SSDs. As soon as the number of concurrent business users increased, the benefit of SSD became even more apparent.

From a technical perspective, SSDs allow many more threads of execution to run in parallel without incurring huge I/O wait times, because their random I/O throughput is basically the same as its sequential I/O throughput. This benefit is relevant because multiple independent queries, serviced simultaneously, implicitly cause random I/O at the disk level, and unlike rotating disks, SSD devices do not slow down under random I/O. Though rotating disks slow down to a three-digit number of I/O operations per second when access is random, high-end SSD devices continue to deliver five-digit number of I/O operations per second, sequential or random. This directly translates into more parallel queries, and therefore more concurrent users, per server when its I/O system is based on high end SSD technology.

Recommendations

Here is a list of conclusions we drew from the work we did for the Microsoft Entertainment and Devices Data Warehousing Team. Most have been discussed in this paper, but some just general best practices we want to share with you:

- Remove the I/O bottlenecks by adding fast enough underlying disk I/O. Their absence makes it easier to find algorithmic bottlenecks in SQL Server Analysis Services implementations.
- The Analysis Services workload is well suited, for distinct counts (and Storage-Engine-heavy query loads), to an SSD I/O backend.
- When you evaluate changes to an Analysis Services cube, testing single-user query performance is *not* enough. If you do not create an independently parallel load, you are not properly simulating usage patterns, because your users are creating parallel loads.
- It is critical to be aware of your production workload, to monitor the queries being run and the performance of the system servicing the queries.
- Even a simple query can stress Analysis Services distinct count performance – it is critical to consider the size of the dataset that a distinct count query returns to accurately assess the query's performance.
- Follow these rules for making sure Analysis Services can parallelize distinct count queries, in addition to the existing standards and practices for partitioning:
    - Make sure that the distinct count attribute is directly used in the partitioning query.
    - Make sure that the partitioning query function (for all sub-partitions in a single time period) is continuous. Using a hash function and BETWEEN is one way to do this that works well.
- When benchmarking, if you are testing cold-cache scenarios, be sure that you run multiple times and clear *all* caches between runs. Don't accept a result as true until you can reproduce it.

Summary

The results of our analysis are clear: SSD technology has significant benefits for MOLAP-based Analysis Services solutions. Because concurrent users implicitly create random I/O patterns, solid-state devices enable greater scalability and per-user performance. In the past, before the advent of SSD technology, getting very high end parallel random I/O performance required a complex and very expensive solution. SSDs offer these benefits without the prohibitively high cost.

Excel, Cube Formulas, Analysis Services, Performance, Network Latency, and Connection Strings

This technical note discusses the usage of three connection string settings to improve the performance of Excel reports over slow network connections against Analysis Services cubes.

Problem Overview

Microsoft, as a worldwide company, has users spread all over the globe that use a custom Microsoft® Office 2007 Excel solution to perform analysis of business data that is stored in Microsoft SQL Server® 2008 Analysis Services cubes that are physically located in Redmond, Washington, United States. To facilitate the analysis of this business data, the Microsoft IT department has developed approximately 25 Excel reports, each of which contains approximately 5,000 cube formulas.

These reports are used by a wide range of users, with the data returned to these reports restricted based on the Windows® user account of the person executing the report. Analysis Services dynamic dimension security is used to ensure that the data returned for each report from the Analysis Services cubes is only the data that they have permission to view. The reports perform very well for users located in Redmond, Washington, United States and who are directly connected to the Microsoft corporate network here in Redmond. For quite some time, users located in other parts of the world have experienced a significant degradation in performance correlated with the performance characteristics of the network connection to Analysis Services cubes in Redmond. This significant degradation is also experienced by users working from home here in the greater Seattle area in Washington, United States, and connecting to the corporate network via RAS. The SQL CAT team worked with the Microsoft IT department to understand the reasons for this performance degradation and develop a solution that significantly reduces this performance degradation for all remote users.

Problem Analysis

We began by analyzing the Excel reports executed against Analysis Services, comparing the performance characteristics of these reports over different types of network connections. In performing this analysis, we used SQL Server Profiler and Performance Monitor to begin to understand the performance differences over different types of connections and to identify the source of the differences in performance. The first thing that we discovered is what you would probably expect – namely that the performance degradation over slower network connections was directly related to the amount of network traffic transmitted between Excel and the Analysis Services cube. As we dug deeper, however, our analysis became more interesting. We discovered the following:

- Security restrictions did not result in faster-running reports, even though these restrictions significantly reduced the amount of data available to the report. Excel reports executed by users that were restricted via Analysis Services security to a small slice of the cube did not run the

same report significantly faster than users with little or no restrictions on the slice of the cube that they could see. To understand why this was true, we analyzed the execution of a single report by these different types of users :

- When we analyzed report execution by using Performance Monitor, we saw that the amount of data exchanged between Excel and Analysis Services was not proportionate to the slice of the cube that a user had permission to view. Specifically, when the Excel report was run by a user that was restricted via Analysis Services dynamic security to a small slice of the cube, the amount of data exchanged between Excel and Analysis Services was not proportionally less than when the same report was run by a user with no restrictions.

- When we analyzed report execution by using SQL Server Profiler, we saw that if the Excel report was run by a user that was restricted via Analysis Services dynamic security to only a slice of the cube, and if a cube formula in the report attempted to access data outside that slice, Analysis Services threw an error. After the error occurred, Excel closed the connection to Analysis Services and opened a new connection to continue resolving the remaining cube formulas in the report. Each time a new connection was opened, a new set of Discover requests were sent to Analysis Services and the results of these requests sent to Excel. In Excel 2007, with the Microsoft Analysis Services OLE DB Provider for Microsoft SQL Server 2008, 11 Discover requests were executed (synchronously) for each new connection. As a result, the smaller the slice of the cube to which a user had permission to view data, the greater the number of errors and, consequently, the greater the number of new connections opened and associated Discover requests issued to resolve the cube formulas in the report. The slower the connection, the longer the total time to resolve the cube formulas in the report.

    - **Important**: In our lab, we investigated the performance implications of upgrading to SQL Server 2008 R2 Analysis Services and Office 2010 Excel. We discovered that such an upgrade did not improve performance over slow networks. Indeed, the amount of data on the wire actually increased slightly for each new connection, resulting in a slight decrease in performance – particularly for the most restricted users. The reason for this additional data on the wire is that the Microsoft Analysis Services OLE DB Provider for Microsoft SQL Server 2008 R2 issues two new Discover requests for each new Analysis Services connection from Excel (increasing from 11 to 13 Discover requests per connection). One of these new requests enables the use of a more optimized axis format for cellsets and the other detects whether Excel is connecting to SQL Server PowerPivot for SharePoint or SQL Server 2008 R2 Analysis Services.

- Analysis Services was returning both unformatted and formatted cell values in the result set returned in response to the MDX query issued by Excel for each cube formula. By default, Analysis Services returns both the formatted and unformatted value for each cell. Additional server-side formatting is also supported, including font style, fill color, and text color. However, returning the formatted value places more data on the wire than is actually required, because this formatting can also be done in Excel. For more information, see Using Cell Properties (http://msdn.microsoft.com/en-us/library/ms145573(SQL.90).aspx) in SQL Server 2005 Books Online.

- Analysis Services uses a proprietary binary encoding method to return the XML data to Excel rather than using UTF-8 XML encoding. In general, this implementation saves CPU time, space, and memory on the server. However, because this implementation stores all strings in Unicode, strings on the wire require twice as much space as UTF-8 XML encoding does. The greater the number of strings involved in the exchange of data between Excel and Analysis Services, the greater the performance impact of this default encoding method. The Discover requests for each new connection are strings. For more information about these encoding methods, see Mosha Pasumansky's blog post Analysis Services 2005 protocol - XMLA over TCP/IP (http://sqlblog.com/blogs/mosha/archive/2005/12/02/analysis-services-2005-protocol-xmla-over-tcp-ip.aspx) and protocol format connection string information Microsoft OLE DB Provider for SQL Server Analysis Services Connection String (http://msdn.microsoft.com/en-us/library/ee209205.aspx) on MSDN.

Solution

To reduce the amount of traffic on the wire for these Excel reports, our solution is to use connection string properties to modify the default behavior of Excel and Analysis Services.

Connection String Settings

We changed the following connection string settings for each of these reports for all users.

Missing Member Errors

We changed the value of the MdxMissingMemberMode XMLA property in the connection string to ignore missing members in MDX queries. For more information about this property, see Supported XMLA Properties (http://msdn.microsoft.com/en-us/library/ms186627.aspx) on MSDN. With Excel, changing this setting avoids the resetting of the connection to Analysis Services within a report each time a cube formula receives an error from Analysis Services due to restricted security access to a cube member referenced on a report.

Note: Excel sets this property to "Error" by default, which overrides this setting if set as a standard connection string setting. To override an Analysis Services connection property that is set by Excel, you must specify the setting as an extended property (Extended Properties="MDXMissingMemberMode=Ignore"). The extended properties are applied after the Excel

properties have been set using the OLE DB APIs – so extended property settings override what Excel sets.

Important: You should avoid changing the default value for this setting if you use PivotTables in Excel, because Excel needs to be notified when members vanish from server dimensions so that it can refresh PivotTables appropriately (that is, so that Excel can remove those members from the pivot views as well, even though this isn't always obvious and immediately visible to the end users). Also, if you change the default, invalid cube sets no longer throw an error in Analysis Services. So you must ensure that your cube sets are actually valid.

Formatting of Result Sets

We disabled formatting of result sets returned to Excel from Analysis Services and now perform all formatting in Excel. For information about how to enable or disable OLAP server formatting in an Excel PivotTable report, see Enable or Disable OLAP server formatting (http://office.microsoft.com/en-us/excel-help/design-the-layout-and-format-of-a-pivottable-report-HP010168032.aspx#BMinclude_olap_server_formatting) in Microsoft Office documentation online.

Encoding of Result Sets

We changed the encoding method for communication between Excel from Analysis Services from binary to UTF-8 ("Protocol Format=XML") by modifying the connection string used by Excel to connect to Analysis Services.

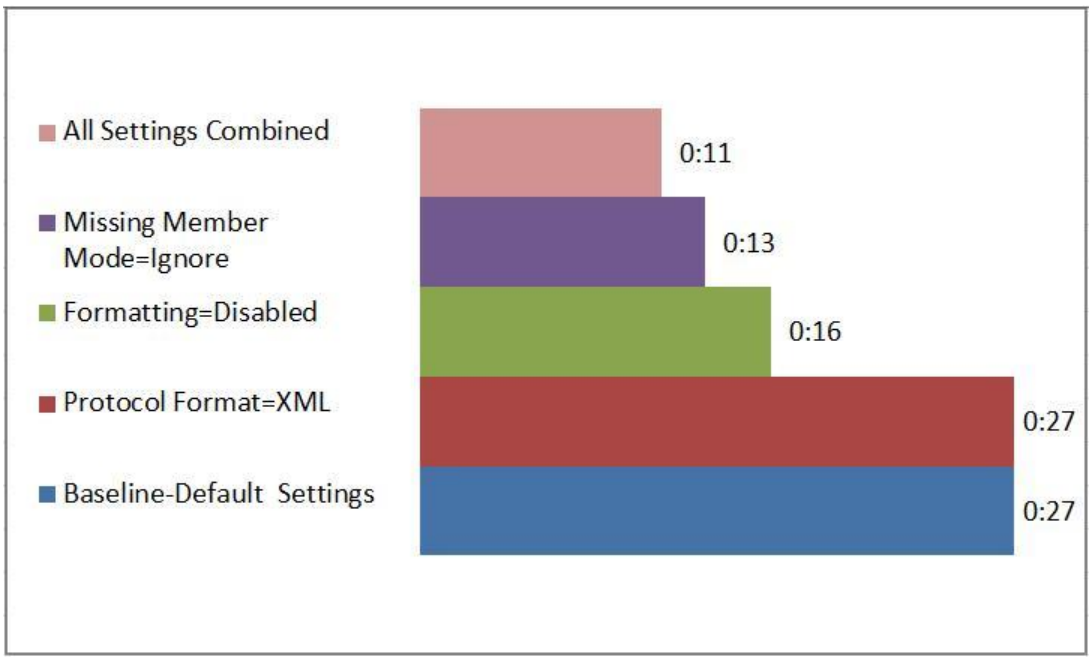
Performance Improvements

The following graphs show the performance improvement associated with each of these settings as well as with all three of these settings combined in four scenarios:

- Wired connection on the Redmond campus
- Wireless connection on the Redmond campus
- RAS connection from home in Redmond to the Redmond campus
- Wired connection from Dublin, Ireland to the Redmond campus

Wired-Redmond

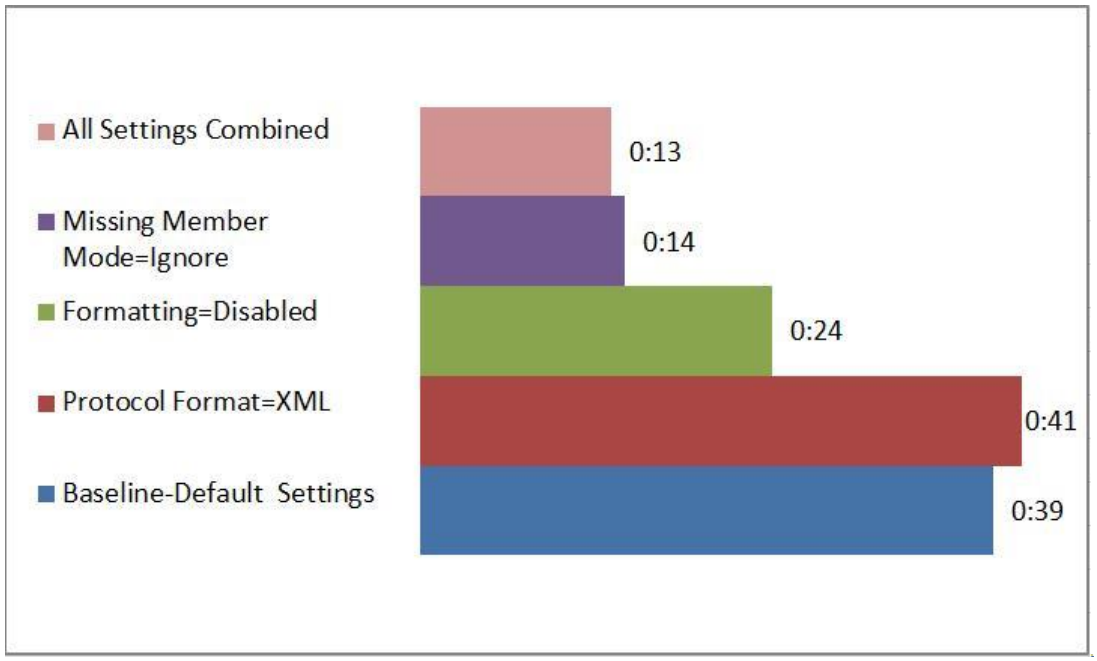
The following graph displays the performance of a standard MSIT report by a typical user (restricted access to only a slice of the cube) executed in my office in Redmond, Washington, United States over a wired network connection to the Analysis Services cube. This report took 27 seconds to execute using the default settings and 11 seconds with all three of the amended settings discussed in this note. For this type of report, the use of these settings increased performance by approximately 250 percent. Notice that the both the elimination of formatting and the ignoring of missing members had a significant impact on performance in our environment, with the change in protocol encoding format showing no impact on performance.



Report Performance in Redmond, Washington, United States over wired connection (in minutes)

Wireless-Redmond

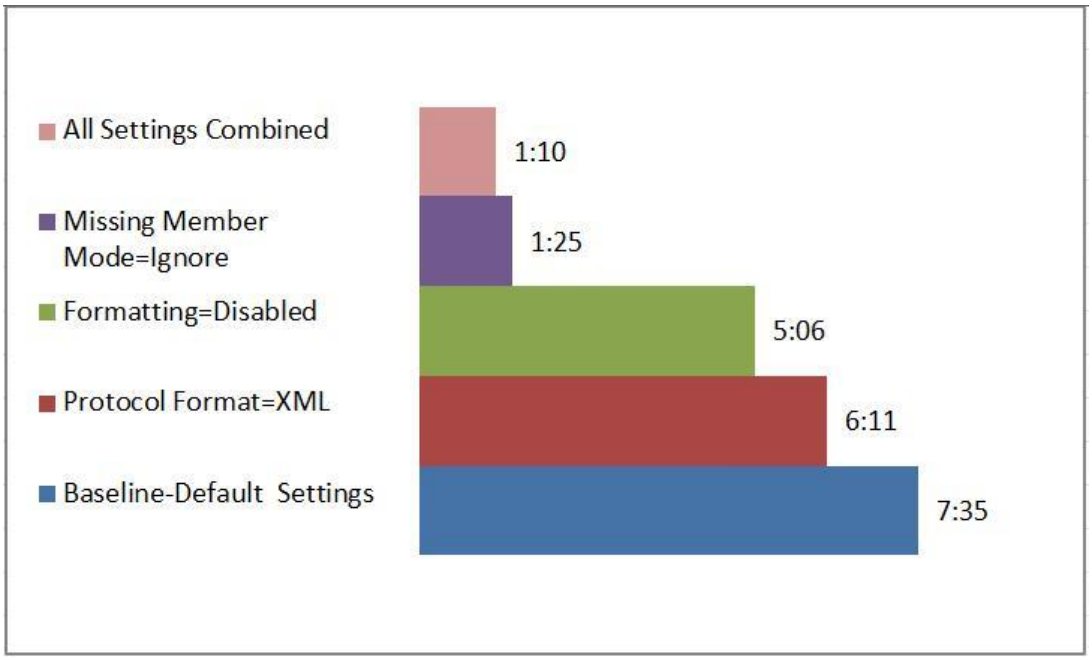
The following graph displays the performance of a standard MSIT report by a typical user (restricted access to only a slice of the cube) executed in my office in Redmond, Washington, United States over a wireless network connection to the Analysis Services cube. This report took 39 seconds to execute using the default settings and 13 seconds with all three amended settings. For this type of report, the use of these settings increased performance by approximately 300 percent. Notice that the both the elimination of formatting and the ignoring of missing members had a significant impact on performance in our environment, with the change in protocol encoding format showing a slight degradation in performance.



Report Performance in Redmond, Washington, United States over wireless connection (in minutes)

RAS-Redmond

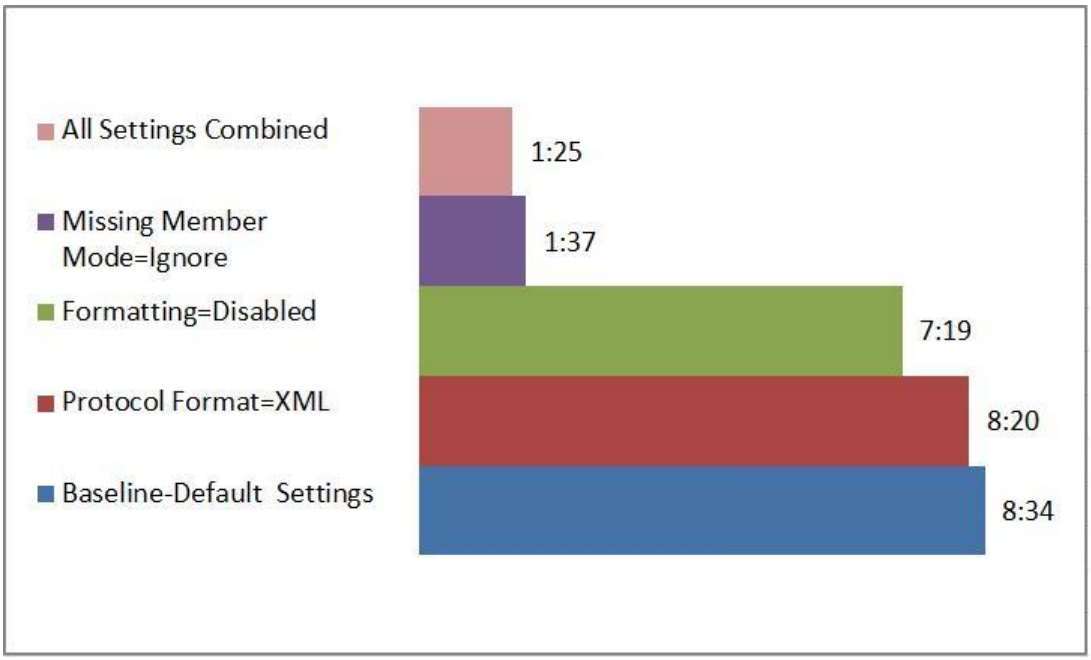
The following graph displays the performance of a standard MSIT report by a typical user (restricted access to only a slice of the cube) executed in my home in Redmond, Washington, United States over a RAS connection to the Analysis Services cube in Redmond, Washington, United States. This report took 7 minutes and 35 seconds to execute using the default settings and 1 minute and 10 seconds with all three amended settings. For this type of report, the use of these settings increased performance by approximately 650 percent. Notice that the ignoring of missing members had the most significant impact on performance in our environment, with the change in protocol encoding format and the elimination of formatting showing significant performance benefits.



Report Performance over RAS from home to Redmond, Washington, United States (in minutes)

Dublin-Redmond

The following graph displays the performance of a standard MSIT report by a typical user (restricted access to only a slice of the cube) executed in Dublin, Ireland over a wired connection to the Analysis Services cube in Redmond, Washington, United States. This report took 8 minutes and 34 seconds to execute using the default settings and 1 minute and 25 seconds with all three amended settings. For this type of report, the use of these settings increased performance by approximately 600 percent. Notice that the ignoring of missing members had the most significant impact on performance in our environment, with the elimination of formatting showing significant performance benefits. The change in protocol encoding format showed only a modest impact on performance.



Report Performance Dublin, Ireland to Redmond, Washington, United States (in minutes)

Summary

Through the use of the three connection settings discussed in this technical note, we were able to increase the performance of complex Excel reports containing large numbers of cube formulas when querying Analysis Services. The performance increase ranged from 250 percent to 650 percent depending on the performance characteristics of the underlying network connections. While these reports still execute more slowly over slow connections, their performance is much more acceptable to remote users than before these settings were incorporated into our MSIT report environment.

Note: For future investigation, we plan to investigate utilizing Excel Services to further improve the performance for our remote users.

Reintroducing Usage-Based Optimization in SQL Server 2008 Analysis Services

Introduction

From the early days of Microsoft® SQL Server® Analysis Services, one of the query performance optimization techniques was for developers to use the usage-based optimization feature. Developers would create their OLAP cube and cube partitions and then set the aggregations for the cube at a low percentage. It was common to see developers set the aggregations for their partitions at 0% - 5%. After the OLAP database was deployed to the production servers, there would be a beta test period for users to work with the cube and to store the components of the queries executed against the cube. Afterwards developers could build aggregations based on what users had queried, typically optimizing for the slowest queries. After this was done, these aggregations could significantly improve the performance of these initially slow queries. With the release of SQL Server 2008 Analysis Services, an improved usage-based optimization algorithm has been included that enhances Analysis Services' ability to create better aggregations.

Customer Databases

To write this technical note, we performed some extensive testing on various customer databases to provide the lessons learned below. To do this, the query and database criteria were that the queries resulted in a lot of storage engine-heavy queries (as opposed to formula engine-heavy queries). This could be identified by viewing the Analysis Services profiler trace and noting that there were a lot of Event Class 12\Event SubClass 22 events (Query SubCube Verbose\Non Cache data). Specifically, we wanted queries that were not able to use the cache and needed to hit the file system.

Upon the application of usage-based optimization, we verified that aggregations were being used by verifying the Event Class 60 (Get Data from Aggregation) events in the Analysis Services profiler trace in SQL Server Profiler. For more information about the Analysis Services events, see Query Events in SQL Server Books Online. As well, you can find a translation of the events within the *[Analysis Services Root Folder]\OLAP\bin\Resources\1033\tracedefinition100.xml* file. For more information about how to use the Profiler to identify query performance bottlenecks, see Identifying and Resolving MDX Query Performance Bottlenecks in SQL Server 2005 Analysis Services.

Recommendations

If usage-based optimization is used correctly, you can consistently reduce the length of long-running queries by using usage-based optimization to create smart aggregations, as shown in Figure 1.

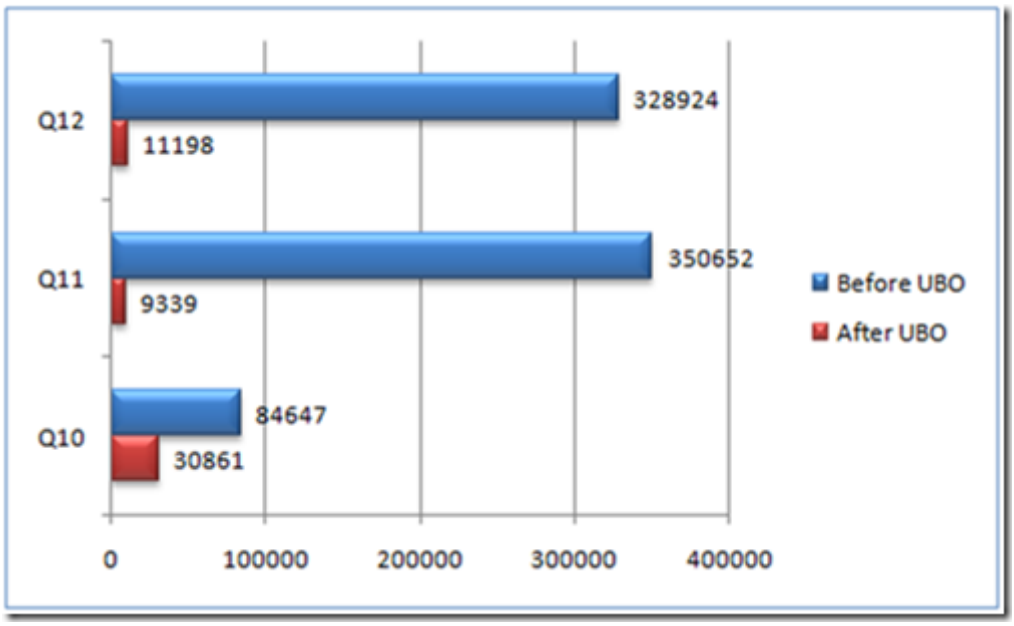


Figure 1: Using usage-based optimization to improve query performance

We learned the following key lessons for using the usage-based optimization feature of SQL Server 2008 Analysis Services:

- Usage-based optimization should only be applied to long-running queries only. Application of usage-based optimization to any and all queries may have the inadvertent effect of reducing query improvement against your long-running queries and increasing processing costs as well.
- Sometimes handmade aggregations will be more optimal in the situation where you want to improve the query performance of a specific query (as opposed to a set of queries). As a general rule, you will still want to start with usage-based optimization to see whether you can get optimal query performance first. Then try applying your handmade aggregations.
- Do not blindly apply and replace existing aggregations with usage-based optimization, because while you may improve one set of queries, you may slow down other queries.
- When using usage-based optimization, you should set the aggregation level to 100% (which is now the default). You should lower it only if processing time is unacceptable or aggregation size increases above the threshold (for example, 1.5 times the size of the facts).
- Test, test, test – make sure you test your queries with your original aggregations, new usage-based optimization aggregations, handmade aggregations, and/or no aggregations. It is important to do this to better understand the query characteristics of your cube.

For more information about the results and tests performed, please read-ahead – if you're not interested – that's it! :-)

Results

As noted above, we executed a large number of queries on various customer databases. Table 1 contains twelve queries that are representative of the larger customer query tests where we compared the query times before and after we applied usage-based optimization. (Times are in milliseconds.)

QueryID	No aggregations	Before usage-based optimization	After usage-based optimization	% difference	Factor (UBO is x times faster)
Q1		32,391	229	99%	141
Q2		37,377	256	99%	146
Q3		337	36,532	-10,740%	-108
Q4				30%	1.4
Q5	267,582	276,071	361,255	-31%	-1.3
Q6		4,679	3,870	17%	1.2
Q7		933	983	-5%	-1.1
Q8		2,627	2,746	-5%	-1.1
Q9		3,759	2917	22%	1.3
Q10		84,647	30,861	64%	2.7
Q11		350,652	9,339	97%	37.5
Q12		328,924	11,198	97%	29.4

All of the tests (including the values in Table 1) had the cache and file system cleared before query execution to ensure that the query results were not obtained from the cache.

Discussion

As can be seen in Figure 2, queries Q1 and Q2 show a two-order of magnitude improvement in query performance. But, in the case of query Q3, there is a two-order of magnitude performance degradation in query performance. This occurred because Q3 was already very fast(337 milliseconds) due to an existing set of aggregations. When the usage-based optimization aggregations were applied, they replaced the existing aggregations instead of adding to them, This replacement caused the performance degradation with Q3.

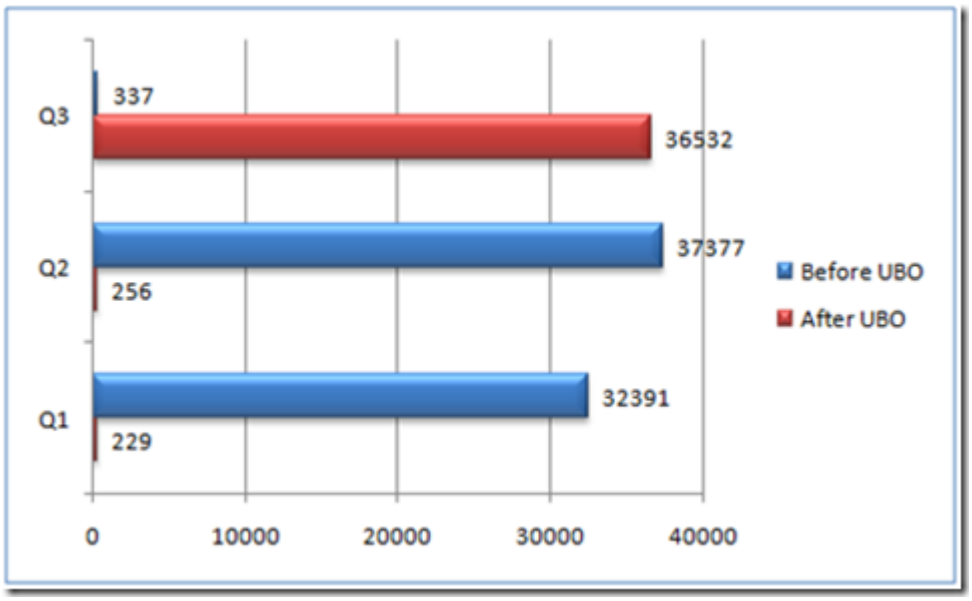


Figure 2: First set of usage-based optimization query tests

When we reapplied the aggregations such that both the usage-based optimization aggregations and the original aggregations were applied, all three queries had return times in the hundreds of milliseconds, that is, less than one second.

***Lesson Learned**: While this is not really specific to usage-based optimization aggretations, remember to be careful and ensure that any new aggregations that you create with usage-based optimization do not override existing aggregations that you may want to keep.*

Figure 3 breaks down query Q5, which shows the same query execution times when no aggregations, handmade aggregations (Before UBO), and usage-based optimization aggregations (After UBO). As seen here, the application of usage-based optimization may sometimes result in the creation of aggregations that may slow down query performance.

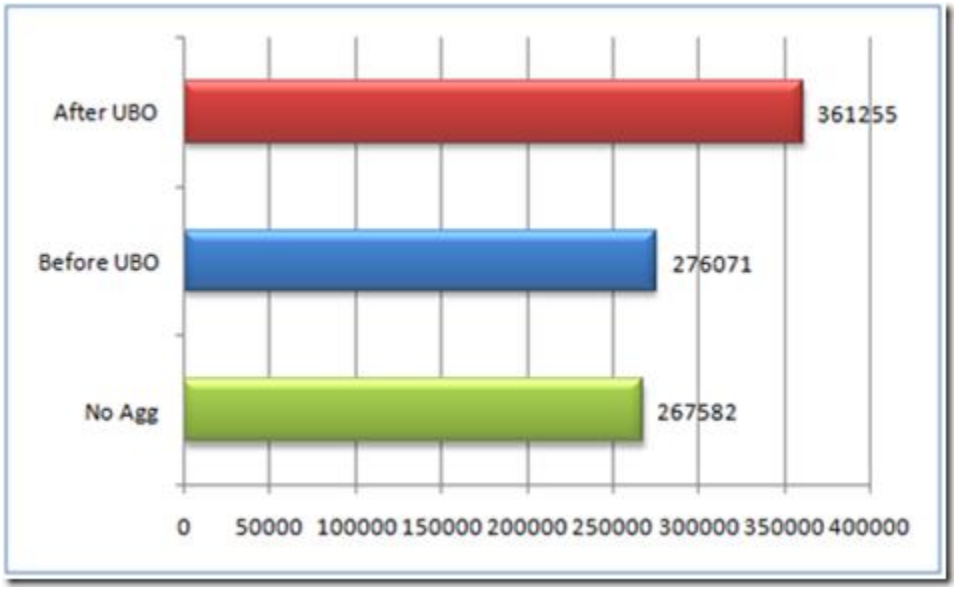


Figure 3: Application of usage-based optimization may be slower

There are times when the creation of any aggregations may slow down query performance. There are any number of reasons that could cause this, ranging from cube design to the way the MDX statements were written. But as you can see in Figure 3, when aggregations were removed from the cube and the query was re-executed, query performance actually improved. As noted in the Table 2, even though the number of subcube requests to the storage engine were the same, the duration is significantly higher for usage-based optimization.

Event	SubEvent	Events	Duration		
			Usage-Based Optimization	No Aggregations	Handmade Aggregations
Query Subcube	Cache data	1329	2,960	464	512
Query Subcube	Non-cache data	204	1,514	1,223	1,480

Table 2: Usage-based optimization has longer duration to query process the same number of events

***Lesson Learned**: Make sure to test all of your queries with your original aggregations (if you had any) and without any aggregations at all. There are times when the Analysis Services MOLAP engine can scan through the file system faster than aggregations can be gone through. This can be especially apparent in the case where the size of the aggregations is larger than the data itself.*

As well, the "Before UBO" query in Figure 3 was against a cube where handmade aggregations were. If we discount the "No Agg" query, it becomes apparent that there are times where handmade aggregations can be faster. This is because when you use the Aggregation Manager to design your own aggregations, you can optimize for that specific query. Note that you can also use usage-based optimization to design aggregations for a single query. But the

advantage of usage-based optimization is that it can create and combine aggregations for all of your select queries (for example,replace aggregations 5, 17, and 33 with a new one that covers them all).

***Lesson Learned**: To improve the performance of individual queries, sometimes you can get faster query performance by creating handmade aggregations because you can optimize for that specific query.*

As can be seen in Figure 4, all of these queries here had relatively fast query times – that is, all of them were under 5s query time. In some cases, usage-based optimiation had improved performance (Q9, Q6) while in some other cases, there was a negligible performance degradation (Q7, Q8) that may or may not have anything to do with the use of aggregations.

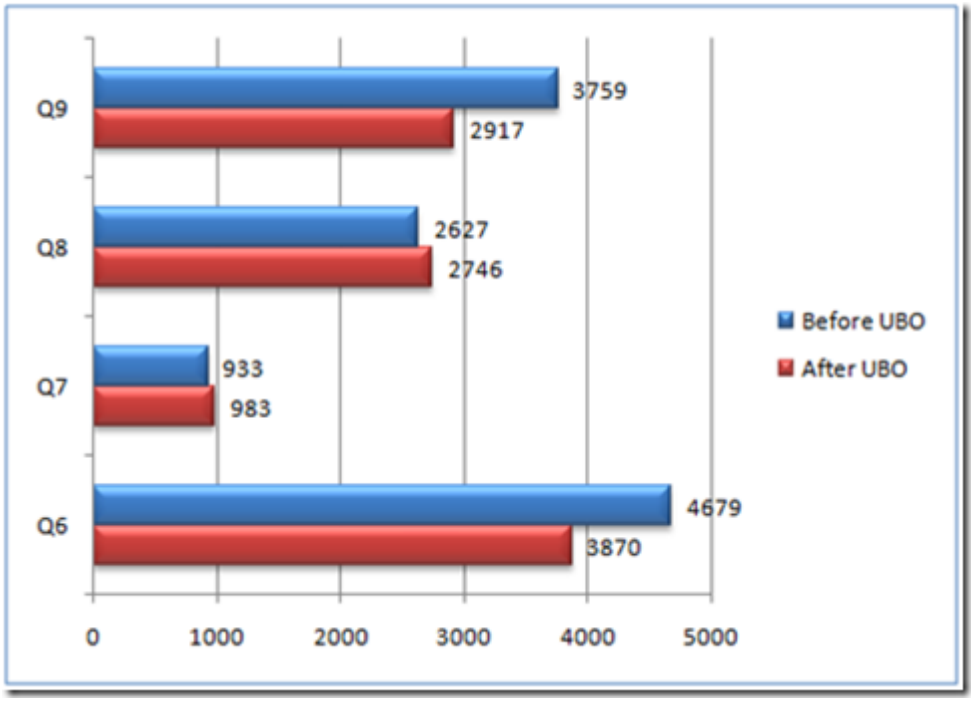


Figure 4: Usage-based optimization query comparison for fast queries

Nevertheless, the key lesson learned here is that the application of usage-based optimization aggregations may have very little effect when you apply them to fast queries.

***Lesson Learned**: Make sure that when you apply usage-based optimization, you apply it to your long-running queries only. Usage-based optimiation is a complex algorithm that creates aggregations based on the queries that you select. If you end up selecting many queries that are already fast, the impact of these aggregations will only be slight and may even reduce the impact of your long-running queries.*

After we take into account the above lessons learned, we can consistently reduce the length of long-running queries by using usage-based optimization to create smart aggregations, as shown in Figure 5.

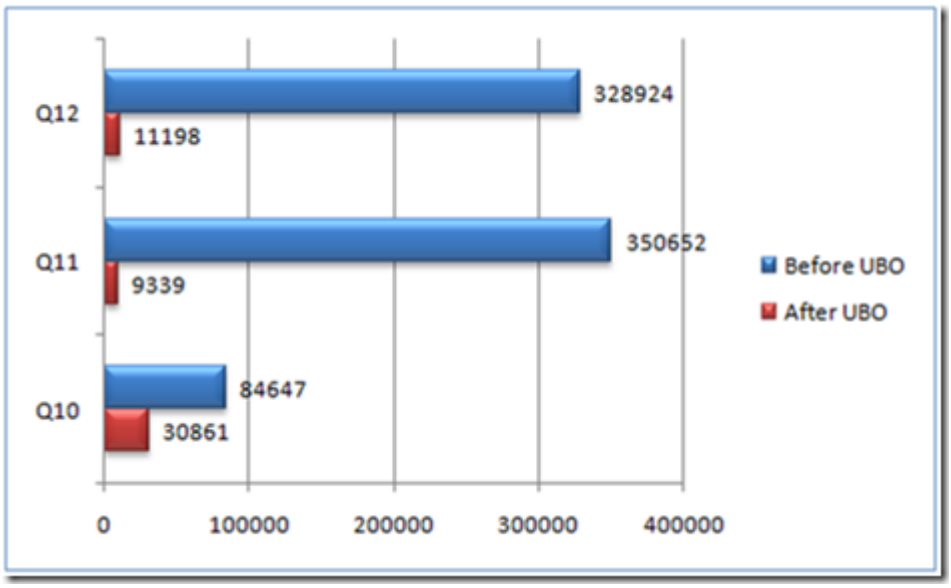


Figure 5: Using usage-based optimization to improve query performance

Conclusion

A summary of the recommendations to optimize your use of usage-based optimization can be found in the Recommendation section earlier in this technical note. Ultimately, these tests and lessons learned provided us with some valuable lessons learned and affirmation of this feature.

Sections 3: Analysis Services Scaleout

Analysis Services Load Balancing Solution

ASLB is a custom Analysis Services load balancing solution that consists of a load balancing Web service backed by a SQL Server load balancing metadata database to load balance MDX queries across multiple query servers. The metadata database contains information about each Analysis Services query server to which queries can be redirected by the Web service.

Load Balancing Web Services	Location of the Analysis Services load balancing (ASLB) Web application. Internet Information Services (IIS) is required on these servers. These servers can use Network Load Balancing (NLB) for failover. For more information about using Network Load Balancing, see Network Load Balancing Deployment Guide.
Load Balancing metadata db	Location of ASLB database. The ASLB database can be mirrored for failover (optional, but recommended for availability). For more information about mirroring, see Database Mirroring Best Practices and Performance Considerations.
AS servers with IIS and MSMDPump	Location of Microsoft® SQL Server® Analysis Services databases. IIS and MSMDPump.DLL must be installed on each of these servers.

Important: For more information, including a discussion of the problems that are solved by ASLB that are not solved by other software and hardware load balancing solutions, see the Microsoft SQL Server 2008 Analysis Services Consolidation Best Practices article.

To Get Started: Download the setup instructions and solution files at:
http://sqlsrvanalysissrvcs.codeplex.com/SourceControl/changeset/changes/81732

SSAS Monitoring Scripts For Management Data Warehouse

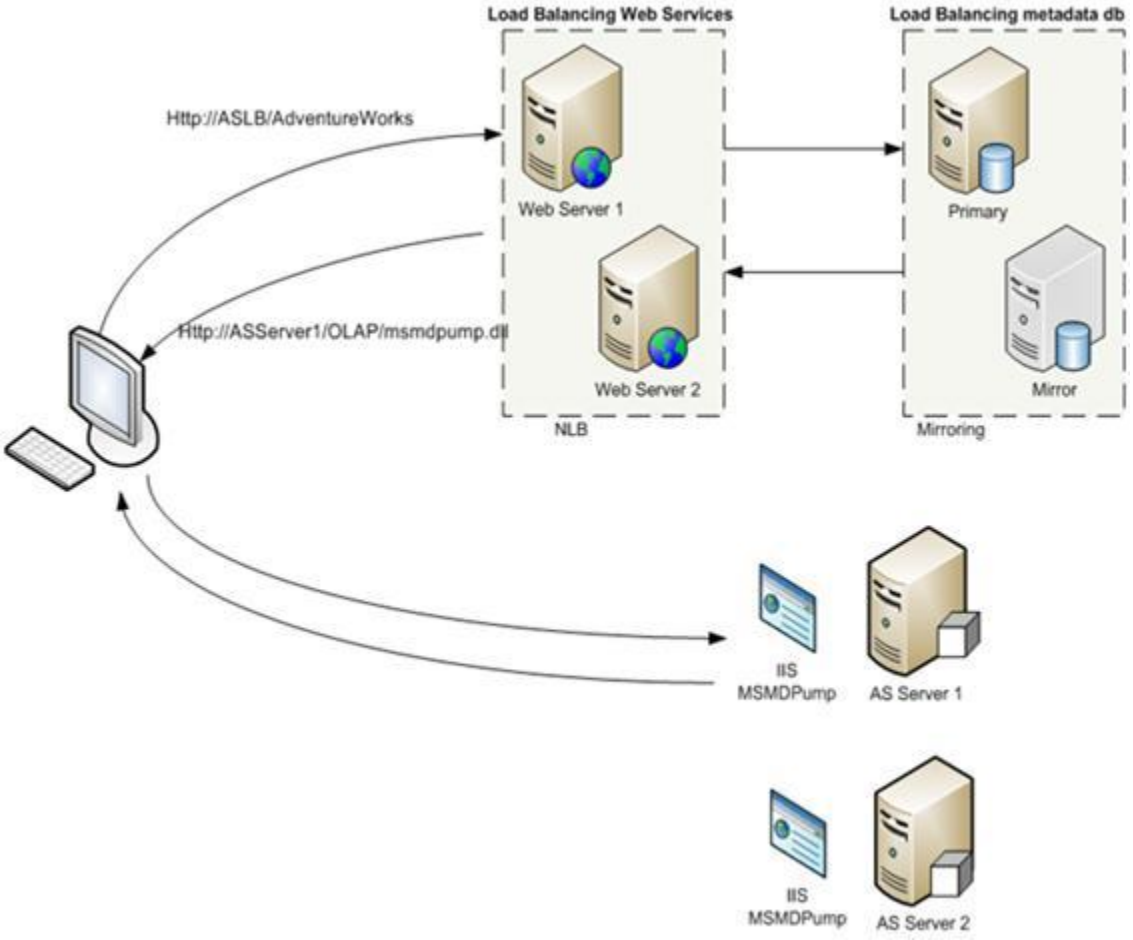
SSAS Monitoring Scripts For Management Data Warehouse at http://www.codeplex.com/SQLSrvAnalysisSrvcs

This project on Codeplex includes scripts for creating a custom collection set in Management Data Warehouse for collecting performance counter data specific to Analysis Services operations. These scripts were extracted from the A Solution for Collecting Analysis Services Performance Data for Performance Analysis project in the same codeplex release.

These scripts were updated on 3/6/2009 to fix bug related to named instances - namely, Performance Monitor counters for Analysis Services named instances must be entirely in capital letters in order to work in the Management Data Warehouse performance counter data collector.

ASLB Toolkit

The ASLB toolkit is a custom Analysis Services load balancing solution that consists of a load balancing Web service backed by a SQL Server load balancing metadata database to load balance MDX queries across multiple query servers. The metadata database contains information about each Analysis Services query server to which queries can be redirected by the Web service.



Load Balancing Web Services	Location of the Analysis Services load balancing (ASLB) Web application. Internet Information Services (IIS) is required on these servers. These servers can use Network Load Balancing (NLB) for failover. For more information about using Network Load Balancing, see Network Load Balancing Deployment Guide.
Load Balancing metadata db	Location of ASLB database. The ASLB database can be mirrored for failover (optional, but recommended for availability). For more information about mirroring, see Database Mirroring Best Practices and Performance Considerations.
AS servers with IIS and MSMDPump	Location of Microsoft® SQL Server® Analysis Services databases. IIS and MSMDPump.DLL must be installed on each of these servers.

Important: For more information, including a discussion of the problems that are solved by ASLB that are not solved by other software and hardware load balancing solutions, see the Microsoft SQL Server 2008 Analysis Services Consolidation Best Practices paper on the SQLCAT web site.

To Get Started: Download the setup instructions and solution files at:
http://sqlsrvanalysissrvcs.codeplex.com/SourceControl/changeset/changes/81732

Section 4: Real World Scenarios

Edgenet Realizes the Power of Dynamic IT and Self-Service BI

Author: Carl Rabeler

Contributor: Steve Hord

Reviewers: Robert Bruckner, Kevin Cox

Dynamic IT and Microsoft Business Intelligence

The latest Microsoft® business intelligence tools and applications enable IT departments to operate with more agility and to operate more efficiently by optimizing people, processes, and technology. This article is a case study showing how the IT department at Edgenet uses the self-service business intelligence capabilities in Microsoft SharePoint Server® 2010 and Microsoft SQL Server® 2008 R2 to improve their agility and efficiency while delivering analytics to customers in an Internet-facing deployment. Furthermore, this case study illustrates some of the advantages for IT departments through the use of SQL Server Reporting Services in SharePoint mode compared with native mode.

Overview of Edgenet

Edgenet provides applications and services that help companies in multiple industries, regardless of size, sell more products through the use of data services, selling solutions, and Internet marketing. Typically retailers and distributors drive sales with Edgenet's online and in-store applications. Suppliers can specify products in applications and also offer their products to millions of online consumers through product placements and product listings.Edgenet provides analytics as part of its services that enable its customers to better understand how their customers interact and behave with the products that they sell. The Edgenet analytics consist primarily of Reporting Services reports that each customer accesses on the Internet.

Previous Environment

Edgenet utilized Reporting Services in SQL Server 2005 and SQL Server 2008 running in native mode for its analytics before they upgraded to SharePoint Server 2010 and SQL Server 2008 R2. Previously, for each customer, Edgenet configured a new Reporting Services instance and a set of standardized reports that were based on standard templates. This set of standardized reports consisted of 10 -20 reports, with up to 50 customized reports depending on the level of analytics the customer requested. Edgenet used separate instances of Reporting Services for each customer for security isolation; this ensured that each company's reports were completely isolated. Each customer URL was unique and connected them to their own Reporting Services environment.However, the Edgenet IT and business departments identified the IT department as the bottleneck in delivering customer analytics. They identified the following causes for this fact:

- **Provisioning**: The provisioning of analytics for a new customer required approximately two weeks from the initial request from the business department until the analytical environment for each new customer was provisioned. This two-week period included the elapsed time to get

a time allocation from the IT department, the actual time spent by the IT department in provisioning the analytical environment, and reciprocation with the business department to ensure the provisioned reports were precisely what the business department wanted for each new customer. The actual time spent by the IT department consisted of approximately 1 – 2 days provisioning a new Reporting Services instance and then an additional 2 – 3 days to customize and deploy the required set of standardized and customized reports for the new customer. The IT department found that it was increasingly difficult to fit the typical 3 – 5 new customer provisioning days into their normal schedule of other tasks required of the IT department.

- **Customizing Standardized Reports**: The customization of standardized reports by the IT department for existing customers was only permitted on a prioritized basis, due to the many other tasks required of the IT department. Many business department requests simply could not be addressed due to the IT department's limited time resources. The reason that many requests for customization could not be addressed is that while the customization of a report might take only two hours of hands-on time, it might to take two weeks for the customized report to be completed – if it was approved at all – due to constraints due to other obligations of the IT department.
- **New Analytics**: New types of analytics for existing customers were not even considered due to the IT department's lack of capacity.

New Environment

To specifically address these concerns, the Edgenet IT department chose to upgrade their business intelligence environment to SQL Server 2008 R2 Reporting Services and to run Reporting Services in SharePoint mode with SharePoint Server 2010. This new environment resolved the first two bottlenecks discussed previously and enabled the consideration of new types of analytics because the IT department now has the time and capacity to investigate additional technologies.These technologies, as implemented by the Edgenet IT department, enabled the IT department to eliminate themselves as the bottleneck in delivering customer analytics in the following ways:

- **Provisioning**: When the Edgenet IT department switched from Reporting Services in native mode to Reporting Services in SharePoint mode, the security isolation business requirement for each customer was realized by using a separate SharePoint Server 2010 site collection for all customers. With this new environment, the IT department can use Windows® PowerShell™ for SharePoint and WMI scripts to configure security groups in Active Directory® Domain Services (AD DS) and create the base site collection in just a few minutes. After the IT department completes these steps, the business user skins, or applies a theme to the portal and configures site security using Active Directory groups for a new customer in 30 – 45 minutes. Customizing and deploying the set of reports for each new customer still requires approximately 2 – 3 days, but it is now done by a business user instead of the IT department. This work-shift is made possible by the self-service business intelligence features in SQL Server 2008 R2 Reporting Services and the ease of use of Report Builder 3.0.
- **Customizing Standardized Reports**: The Edgenet IT department is now able to offload significant portions of the customization of standardized reports to its business users, simply by using the new self-service reporting capabilities in SQL Server 2008 R2 Reporting Services. The new features used are shared datasets and report parts, consumed by Report Builder 3.0:

- A shared dataset provides a way to share a query to help provide a consistent set of data for multiple reports. The dataset query can include dataset parameters. You can configure a shared dataset to cache query results for specific parameter combinations on first use, or you can specify a schedule. You can use shared dataset caching combined with report caching and report data feeds to help manage access to a data source. For more information, see Managing Shared Datasets (http://go.microsoft.com/fwlink/?LinkId=203907&clcid=0x409)
- A report part is a table, chart, or other report item from a report, such as a template or master report that is published for reuse in other reports to support collaborative authoring in Report Builder 3.0. For more information, see Report Parts in Report Designer (SSRS) (http://go.microsoft.com/fwlink/?LinkId=200095&clcid=0x409)

The customizing of standard reports by business users was previously bottlenecked by the IT department allocating the time to perform the following tasks:

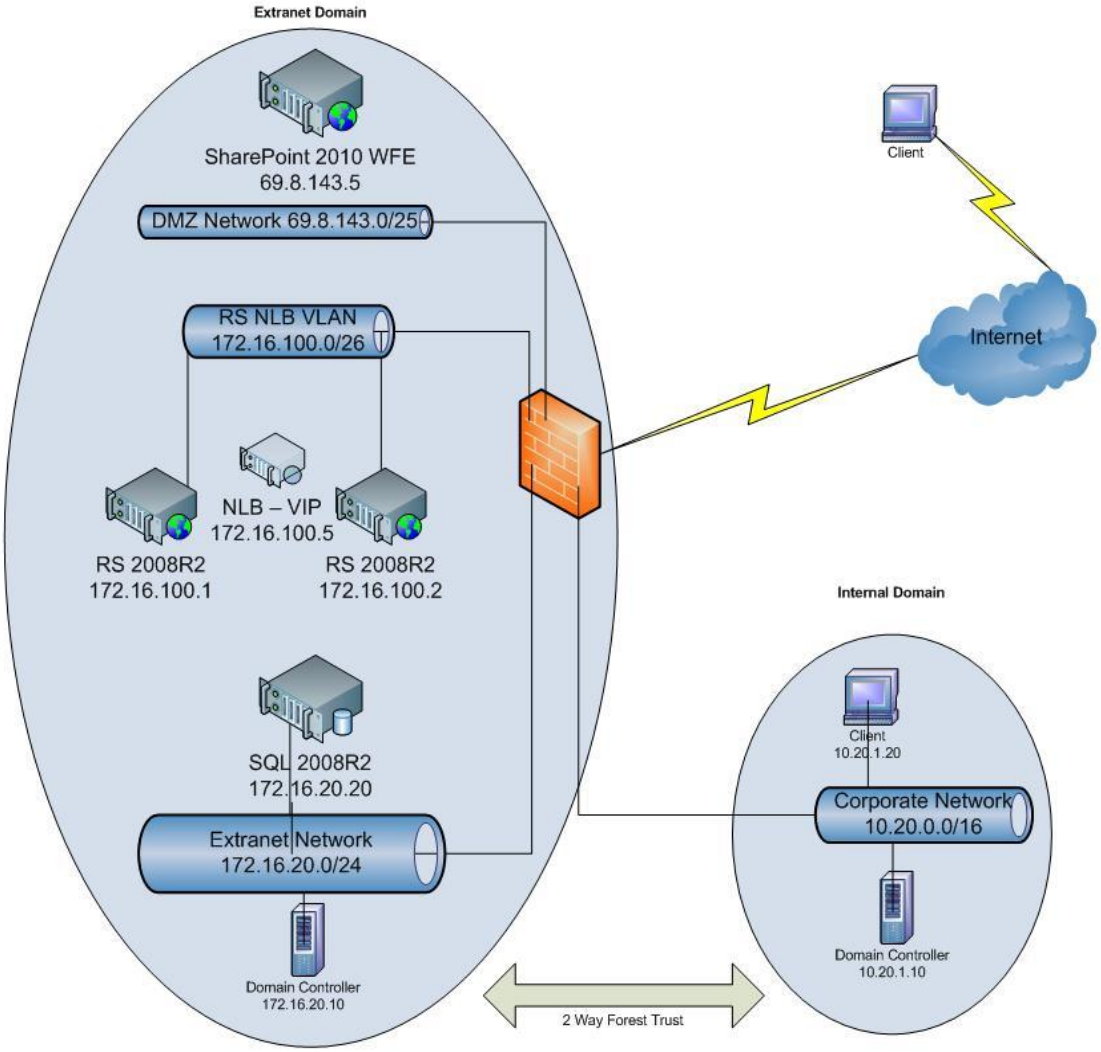
- Generate the Transact-SQL query for reports.
- Build the actual report (including the reciprocation with the business department). The IT department spent the majority of this time building the actual report (including the reciprocation with the business user and the customer). In this new environment, the IT department converted the standard reports into reports with shared datasets and report parts, and it now creates new shared datasets whenever existing queries do not meet the needs for customizing either existing reports or generating new reports. The business users create reports from these shared datasets. Early in the evaluation process of this new environment, the business department brought an example of a new report that required a new Transact-SQL query that needed to be written by a DBA with intimate knowledge of the source schema. The business department was surprised to see that the IT department was able to generate this new query in under 15 minutes. After the new query was generated and published as a shared dataset, the business department was immediately able to use Report Builder to generate the new report. Because the business department can now do some of the work by using report templates, report parts, and shared datasets, the IT department's role no longer creates a bottleneck.

- **New Analytics**: By reducing the time required to provision a new user and offloading much of the customization of reports and the SharePoint site to business users, the IT department is able to consider the implementation of new types of analytics, including PerformancePoint dashboards and PowerPivot workbooks.

New Topology

The following diagram shows the physical topology of the current, new Edgenet environment. (In the diagram, RS stands for Reporting Services, SQL 2008R2 stands for SQL Server 2008 R2, and DMZ stands

for perimeter network, also known as demilitarized zone and screened subnet.)



Extranet Domain Configuration in the New Environment

In the new Edgenet environment, there is an extranet domain for external users and an internal domain for Edgenet users. There is a two-way forest trust between the domains that enables basic Kerberos delegation between the domains, rather than simply a one-way trust. While basic Kerberos delegation is not required for Edgenet's configuration of Reporting Services in their SharePoint 2010 farm (security is discussed in greater detail later in this document), it is required for other applications in their environment.

Note: Constrained Kerberos delegation does not work across domain boundaries.

To facilitate secure operation of the SharePoint Server 2010 farm for each customer's SharePoint site, the IT department configures the following Active Directory groups in the extranet domain:

- **[Customer] Owner group**: The user account for the internal coordinator from the internal domain from the business department for the customer is placed in the Owner group in the extranet domain.
- **[Customer] Members group**: The user account for the internal contributors and report developers from the business department for the customer are placed in the Members group in the extranet domain.
- **[Customer] Visitors group**: The consumers of the reports from the customer are given accounts in the extranet domain and placed in the Visitors group.

SharePoint Server 2010 Configuration in the New Environment

In the new Edgenet environment, there are currently three server computers in the SharePoint Server farm within the perimeter network, plus the SQL Server computer hosting the SharePoint content databases and the Reporting Service catalogs, which is behind the firewall. One server in the farm hosts the SharePoint Server Web front-end and the central administration site. The other two servers in the farm are in a software Network Load Balancing (NLB) Reporting Services cluster (RS NBL VLAN). This RS NLB cluster is in its own VLAN to limit flooding on the network. If the SharePoint Server Web front-end becomes a resource bottleneck, an additional Web front-end can be configured and placed into its own NLB cluster on its own VLAN. For more information about NLB configuration, VLANs, and flooding, see Preparing the Network for NLB 2008 (http://go.microsoft.com/fwlink/?LinkId=203908&clcid=0x409) and Network Load Balancing (http://go.microsoft.com/fwlink/?LinkId=203909&clcid=0x409).

Important: Placing multiple software NLB clusters on the same VLAN causes massive network traffic flooding and takes down the network segment.

When a new site for a customer needs to be set up, the IT department creates the new Web application, creates a new site collection (using the Blank Site template), and then activates the Reporting Services features for the new site. Additionally, the Reporting Services service account is granted permissions on the Web application content database (this step is required because different application pool identities are used). The following commands are run from the SharePoint 2010 Management Shell:

- `$w = Get-SPWebApplication -Identity http://<site address>`
- `$w.GrantAccessToProcessIdentity("<domain>\<service account>")`

For more information, see Configuring Kerberos Authentication for Microsoft SharePoint 2010 Products (http://go.microsoft.com/fwlink/?LinkId=203910&clcid=0x409).The new site is then skinned with a custom look and feel for that customer. The default SharePoint security groups are utilized as follows:

- **Owners**: The extranet Owners group is added to this SharePoint group.
- **Members**: The extranet Members group is added to this SharePoint group.

- **Visitors**: The extranet Visitors group is added to this SharePoint group. By default, members of this group can open or download any content type uploaded to the document library.

To complete the security setup for the SharePoint site, the IT department performs the following additional steps:

1. It adds the Reporting Services content types to the document library. Reporting Services provides predefined content types that are used to manage shared data source (.rsds) files, report models (.smdl), and Report Builder report definition (.rdl) files. By default, these content types are only enabled automatically to the Business Intelligence Center site template. Adding a Report Builder Report, Report Model, and Report Data Source content type to a library enables the **New** command so that you can create new documents of that type. For more information, see How to: Add Report Server Content Types to a Library (Reporting Services in SharePoint Integrated Mode) (http://go.microsoft.com/fwlink/?LinkId=203911&clcid=0x409).
2. It modifies the permissions associated with the Visitor role for all document libraries containing Reporting Services artifacts. By default, the Visitors role uses the Read permission level. This permission level allows users to view pages and list items, as well as download documents. To protect the Edgenet intellectual property within their published Reporting Services artifacts, the permission to download is removed. To accomplish this, the Open Item permission must be removed from the Visitors role. Edgenet initially tried to just modify the Read permission level, but this prevented the cascading of their custom style sheets on the site. So, instead of modifying the Read permission level, a copy of the Read permission level is created. The Open Item permission is removed from the copy of the Read permission level and then saved as a new permission level. This new permission level is used for the Visitor role for all document libraries containing Reporting Services artifacts, rather than the original Read permission level, and the inheritance of permissions from the parent site is removed. This second step disables the downloading of the content types, for which the document library is configured, which includes Reporting Services reports, models, and data sources based on the first step.
3. It adds custom Reporting Services content types to the document library. The predefined Reporting Services content types discussed in step one do not include shared data sets and report parts. As a result, in order to disable the downloading by extranet users of these Reporting Services artifacts, which contain Edgenet intellectual property, Edgenet created custom content types. For more information, see Content type and workflow planning (http://go.microsoft.com/fwlink/?LinkId=95966&clcid=0x409).

Responsibilities of the IT Department and the Business Department in the New Environment with Respect to Report Development

In the new environment, the responsibilities of the Edgenet IT department and the business department have changed:

- **IT Department**: The IT department now assists the business department in developing and publishing report templates as full reports based on shared datasets and publishing their components as report parts for consumption in other reports. The IT department also now develops and publishes shared datasets, upon request by business users, for the development of new reports. In addition, if a report is extremely complicated, the IT department steps in to aid the business user.**Note**: Initially, the IT department was also involved in teaching business users

how to work with the report templates, report parts, and shared datasets. This involvement is diminishing over time.

- **Business Department**: The business department is now responsible for developing and publishing the following:○ Report templates for customization for new customers○ Standardized reports for new customers based on the published report templates○ Customized reports for new and existing users based on the published report parts and existing as well as new shared datasets

Summary

The new self-service capabilities in Microsoft SharePoint Server 2010 and SQL Server 2008 R2 enabled the Edgenet IT department to eliminate themselves as the bottleneck in the delivery of customer analytics and to empower their business department to better serve their customers.

Section 5: Reporting Services

Reporting Services Scale-Out Deployment Best Practices

Introduction

This technical note is part of the Reporting Services Scale Out Architecture which provides general guidance on how to set up, implement, and optimize an enterprise scale-out architecture for your Microsoft® SQL Server® Reporting Services environment. This note provides guidance for both SQL Server 2005 and 2008 Reporting Services. The focus of this technical note is Scale-Out Deployment Best Practices – the Reporting Services Windows®/Web Services servers that provide your reports.

Architecture

Figure 1 shows a typical scale-out Reporting Services environment; as noted by the red box, the focus of this technical note is scale-out deployment.

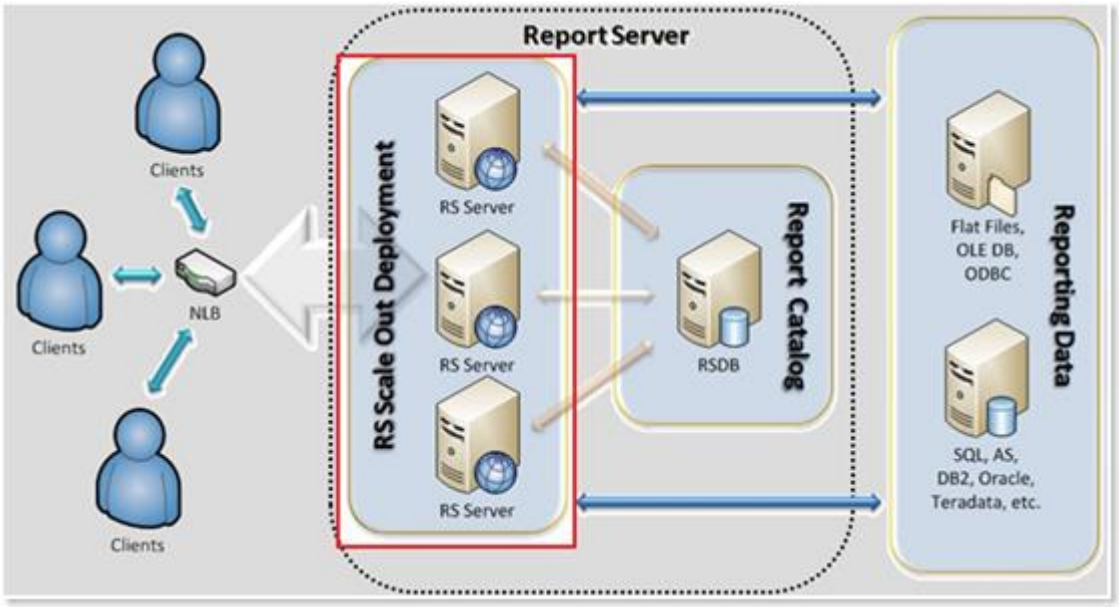


Figure 1: Reporting Services scale-out architecture

Report Catalog Sizing

As noted in the previous technical note of this series, Report Server Catalog Best Practices, the report server catalogs are an integral part of your Reporting Services solution. When you begin the process of building your Reporting Services scale-out deployment, you will need to appropriately scale your report server catalogs to handle the

additional workload. In addition to the concepts discussed in the aforementioned technical note, a key consideration is Report Catalog sizing; the two report server catalogs (**RSDB** and **RSTempDB**) have different sizing requirements.

Report Server (RSDB) Database Size

The size of this database typically varies by the number of reports published and the number of history snapshots associated with each report. A general rule of thumb is that a moderate-sized report definition will take about 100-200 KB of disk space, which is larger than the actual size of the RDL. This is because Reporting Services will persist both the RDL and the compiled binary representation to improve report execution performance. In the case of history snapshot sizes, this is a function of the amount of data that is in the report (i.e., the larger your report and datasets, the larger your snapshot). A general rule of thumb is that Reporting Services has an approximate 5:1 compression ratio; therefore if your report has 10 MB of data, then the snapshot will be about 2 MB in size. Note that reports that do not perform aggregations will be slightly larger than reports that do.

Report Server TempDB (RSTempDB) Database Size

The size of this database varies depending on the number of users who are concurrently using the report servers. Each live report execution generates a report snapshot that is persisted in the **RSTempDB** for the duration of the user's session. Because of the high degree of variability of the number of users who are accessing your report servers at any one time, your sizing estimate should be based on the maximum number of concurrent users that access your Reporting Services environment. A general rule of thumb here is that you will typically see a maximum of 10-20% concurrency of your user base.

For example, if you have 1,000 users, then you can have up to 200 concurrent users. If most of these users are accessing your 10-MB report, then you will need to have at least 400 MB of storage when taking compression into account. Of course, as users are no longer querying for reports and/or report sessions are timing out, the space will be reclaimed and made available for new users. But at the same time, to size properly, you will want to make your calculation based on the maximum number of concurrent users.

Enable File System Snapshots for SQL Server 2005 Reporting Services

Please note that this is applicable for SQL Server 2005 Reporting Services, not SQL Server 2008 SQL Server 2008 Reporting Services. As noted in Scaling Up Reporting Services 2008 vs. Reporting Services 2005: Lessons Learned, this feature did not provide the scale of performance benefit (more on this later) for SQL Server 2008 Reporting Services.

Recall that the report server uses snapshot data stored in **RSTempDB** to render reports. Therefore **RSTempDB** gets filled up fast, because there are many transactions performed to keep report consistency (e.g., if I get the first page of a report, we need a snapshot of the second page that is taken from the same data as the first). Because of this concern for consistency, it is then important for Reporting Services to cache all of this information. To reduce the number of transactions being placed on your **RSTempDB**, use the File System (FS) snapshots feature instead. With it enabled, snapshot data persists on the local Reporting Services server. Therefore, this reduces the network traffic from

the Reporting Services server to the catalog, which also reduces the number and size of transactions within the catalog. For more information about this feature, see the Planning for Scalability and Performance with Reporting Services white paper.

For Reporting Services to efficiently access this snapshot data, it is spread over a number of chunks – logical divisions of smaller size. During regular operations, the report server may need to access the chunks multiple times to satisfy user requests. For example, a particular chunk may have been removed from server memory, so the report server needs to fetch it again from **RSTempDB**. By default, the server must query **RSTempDB** each time it needs to get a snapshot chunk. As user load increases due to report complexity or concurrency, the number of transactions to **RSTempDB** also increases. This can cause performance degradation on **RSTempDB**. To mitigate this, Reporting Services provides configuration options to create file system chunks (i.e., file system snapshots), which act as a cache for the snapshot chunks on the node that created the chunk. This reduction in load on **RSTempDB** allows your Report Catalog to scale better.

Now that the report data is stored on the file system, it is a requirement to use a load balancing solution to affinitize user sessions to their report server node (more on this later). The slight drawback to this feature is that if you were to lose a reporting server, you would also lose the file system data cache that corresponds to that reporting server. However, the metadata is still available in **RSDB** and the original data is in the reporting data. Your users would simply connect to a different report server and make a new request for this report, experiencing slower initial query performance before the file system snapshots kick in.

Therefore, for large-scale SQL Server 2005 Reporting Services deployments, we recommend you turn on the file system storage of snapshot chunks. To do that, you must set **WebServiceUseFileShareStorage** and **WindowsServiceUseFileShareStorage** to **True**. You can control the location of the files using the **FileShareStorageLocation** configuration property. These settings are available in the file rsreportserver.config located under the report server installation folder.

```
<Add Key="WebServiceUseFileShareStorage" Value="true" />
<WindowsServiceUseFileShareStorage>True</WindowsServiceUseFileShareStorage>
```

Note that if you enable this feature, it will require more disk space than your standard Internet Information Services (IIS) configuration, because of the large amount of data that will be written onto disk. (For more information, see the Report Catalog sizing information in the previous section.)

Why Not File System Snapshots for SQL Server 2008 Reporting Services?

There are a number of changes and optimizations to SQL Server 2008 Reporting Services, including better utilization of memory and of the file system. Enabling file system snapshots in SQL Server 2005 Reporting Services was an advantage because unless this feature was enabled, a majority of the transactions required hitting the **RSTempDB**. But SQL Server 2008 Reporting Services caches a lot of this data into memory. Meanwhile the data is continually persisted to the Report Catalogs so that the local file system acts as a write-through cache, regardless of whether file system snapshots are enabled. This was done to ensure stability in case of incorrect load balancing affinity or gaps (e.g., secondary output streams such as CSS style sheets in HTML, report images retrieved in HTML) in report snapshots.

The design in SQL Server 2005 Reporting Services was that it generally would not materialize the original datasets in the report snapshot, except for particular cases. This was possible since the entire report was calculated in SQL Server 2005 Reporting Services when the first page was requested, and the report contents were persisted in a output-format independent way. On the other hand, SQL Server 2008 Reporting Services does not precalculate everything on the initial request – therefore, it has to always serialize all datasets so that it can calculate report contents as needed later. There are still certain parts in SQL Server 2008 Reporting Services that once calculated (e.g., group/sort/filter/aggregates) are persisted and not recalculated. However, many parts of a report (e.g., text box values, style values) are calculated on demand instead of precalculated and stored.

Another reason file system snapshots do not have as profound impact for SQL Server 2008 Reporting Services is that when reports are executed, the Reporting Services on-demand engine will initially retrieve all of the data for your report and write this data into the **RSTempDB**, to achieve a consistent data snapshot. It will then do the calculations and groups required for the first page. But subsequent calls are more efficient and have much less impact than SQL Server 2005 Reporting Services (e.g., page 2 of the report) because the on-demand engine will only access a subset of the data instead of requesting for the entire dataset. Note that the data is materialized in the initial request (e.g., render first page) when all dataset requirements are executed. Later session requests (e.g., navigate to second page, export to Microsoft Excel®) will use the data already materialized in the session snapshot.

This observation was also noted in the Scaling Up Reporting Services 2008 vs. Reporting Services 2005: Lessons Learned technical note. Saying this, you may still want to test the efficacy of this feature within your environment, because there could be variables that allow this feature to help improve performance – but it certainly doesn't always have benefits as it did for SQL Server 2005 Reporting Services.

Turn on Cache Execution

As you may have noticed, so far we have a recurring theme concerning the effective use of memory and desire to have minimal impact on the file system. After all, a key component for scalability is the ability to reduce I/O. Of course, CPU and memory are extremely important for scalability, but often it is the disk I/O that is slowing down your performance. A powerful feature to reduce I/O is to enable the cache execution feature of your reports. By default, when you deploy your Reporting Services reports, it will be a live execution report – i.e., upon report execution, Reporting Services will query the snapshot or your data source. At a minimum, this means that for every report execution some I/O occurs to provide your users with data. By using cache execution, Reporting Services caches a temporary copy of this report into memory that expires after a set number of minutes. Report executions during this time period request data from memory instead of disk, thus reducing I/O. Even if you set your cache timeout for your report to every five minutes, you will still see an 80% reduction in I/O for your report.

To configure cache execution, after you deploy your reports to your report server, you will need to configure each report (i.e., there is no global setting) to cache a temporary copy of the report. Note that you will want to set your reports to use stored credentials instead of Windows integrated security, in order for the cache execution to work effectively. Do not forget to set up your Reporting Services keys to ensure encryption of the stored credentials.

Load Balance Your Network

It is very important to load balance your many client connections to your Reporting Services servers. We recommend the use of a hardware network load balancer (NLB) that allows your client connections to balance across your Reporting Services servers, as noted in Figure 2.

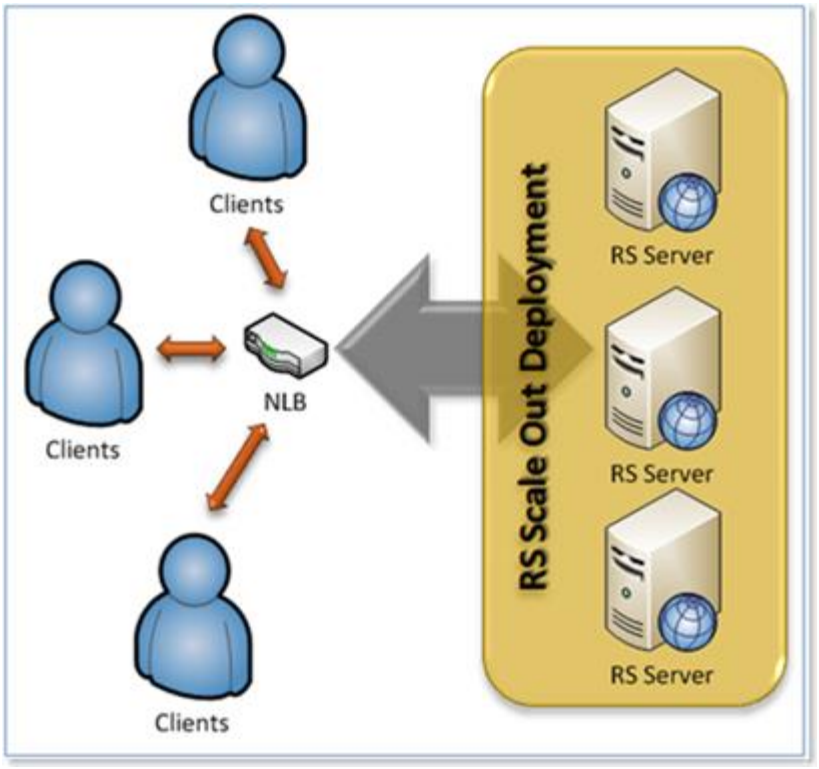


Figure 2: Load balancing your network

We also recommend that you use cookie persistence (refer to your NLB guide to set this up) to preserve the client-to-Reporting Services server connection – i.e., to affinitize the client to the Reporting Services server. You can consider IP affinity, but this can overload under browser-based or proxy connections. Note that you typically will round-robin the initial connections to minimize the load on your NLB. Subsequent queries will have affinity to allow the clients to make use of the Reporting Services local file cache.

Because your Reporting Services servers are making heavy use of the network, you may want to consider also placing additional NIC cards for your Reporting Services servers. In enterprise Reporting Services environments, an important task will be to profile your network resources so you can determine where your network resource bottleneck is. If you are running into network bottlenecks, a setup to consider if you want to scale up your network load is to have two NIC cards, one for the browser traffic and one for the database traffic.

Isolate Your Workloads

Within your enterprise Reporting Services environment, you will typically have two types of users – one set that assesses scheduled reports and another set that reviews the reports in an interactive manner. A common profile differentiation between these two types of reports is that users accessing interactive reports will execute their reports often in order to drill up, drill down, filter, etc. On the other hand, scheduled reports typically will process against much larger sources of data (which involves a considerable hit on the system to produce the reports; hence scheduling will reduce the load) or have a much larger audience (where the load created by the concurrency of a large number of users to query the same report can be reduced by creating the report beforehand). Whether you are dealing with interactive users or scheduled report users, you have a situation where your users produce two different types of profiles as noted in Figure 3.

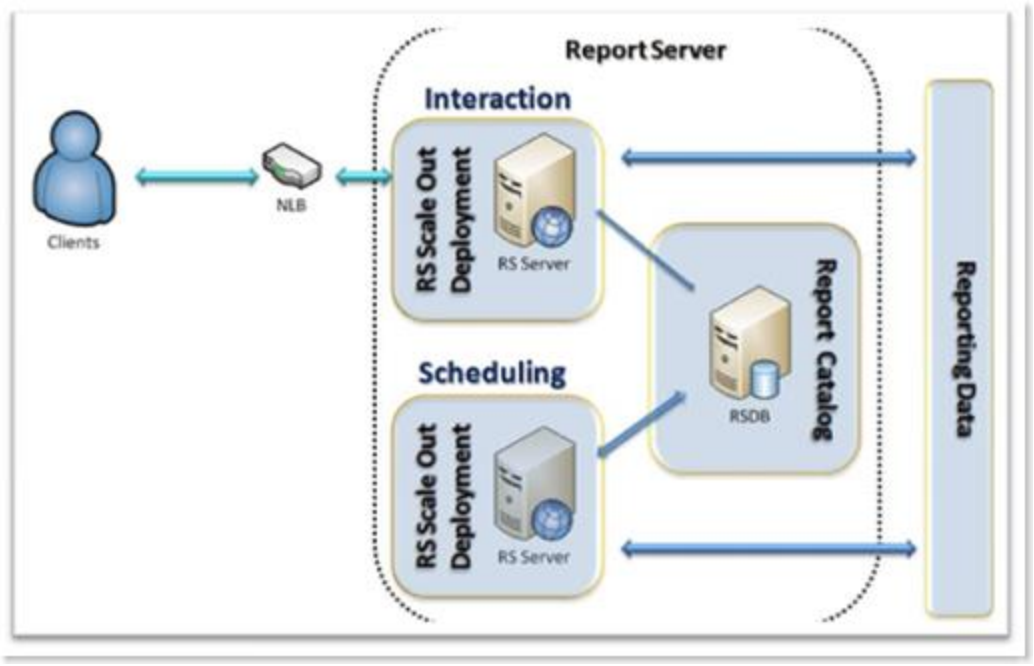


Figure 3: Isolating your workloads

Therefore, it is suggested that within your scale-out deployment, you also isolate your workloads so that you have one set of servers devoted to working with your interactive users (**Interaction**) while another set of servers work on scheduled reports (**Scheduling**). As implied in Figure 3, doing so will provide you with a more predictable workload because of the different resource profile that is used by these different report users. Workload isolation will also allow you to isolate any performance issues associated with these two types of reporting users; that way, interactive users and scheduled report users are not interfering with each other.

Note that ad-hoc report users (i.e., users who use Report Builder) are a different class of interactive users. Because these users can build whatever reports desired, there is a high degree of unpredictability in the amount of resources that this class of users will require. In extreme cases, you may want to consider creating a completely separate deployment for these ad-hoc users so that they do not bottleneck your interactive or scheduled report users.

To isolate your workloads, set your scheduling Reporting Services servers to **Schedule Events and Report Delivery Enabled**, and set your interactive Reporting Services servers to **Web Service and HTTP Access Enabled**. By default both options are enabled, so make sure only one is enabled on each server when you decide workload isolation is the path you will take.

Additional Notes on Workload Isolation

You can also make these workload isolation changes programmatically through WMI using the Reporting Services WMI Provider Library. You will change the **SetServiceState** method; for more information, see the SQL Server Books Online > SetServiceState Method.

To help isolate your reporting bottlenecks for scheduled reports (the **Scheduled Events and Report Delivery**), schedule your reports so they are not being processed in parallel. This way you will be able to isolate performance issues.

Report Data Performance Considerations

With all of these configuration and infrastructure considerations, you will be able to build an optimal scale-out environment for your Reporting Services solution. But while your Reporting Services environment is scaled out, do not forget the impact on your underlying reporting data – i.e., the data source. Here are some of the things you can do to reduce the impact on your report data:

- Limit the dataset size by using query filters.
- If you need to provide large dataset sizes, use alternate mechanisms such as Integration Services to provide an extract that is accessible from the file system, instead of providing a million-page report.
- Limit the number of reports that users can access. This is not just a security concern. Limiting access reduces your maintenance cost by preventing users from creating multiple similar subscriptions and reports.

But even after making the above changes, you may still find yourself placing far too much load on your reporting data. To help resolve these issues, you may want to consider also scaling out your underlying data sources. Figure 4 is an example of a scale-out solution where a set of two Reporting Services servers are assigned to a single Analysis Services server. The underlying reporting data (in this case Analysis Services cubes) is replicated to multiple Analysis Services query servers so that every two Reporting Services servers query only a single Analysis Services instance. Note that to do this you will need to manually create local data source aliases on the reporting servers that will perform this type of server assignment.

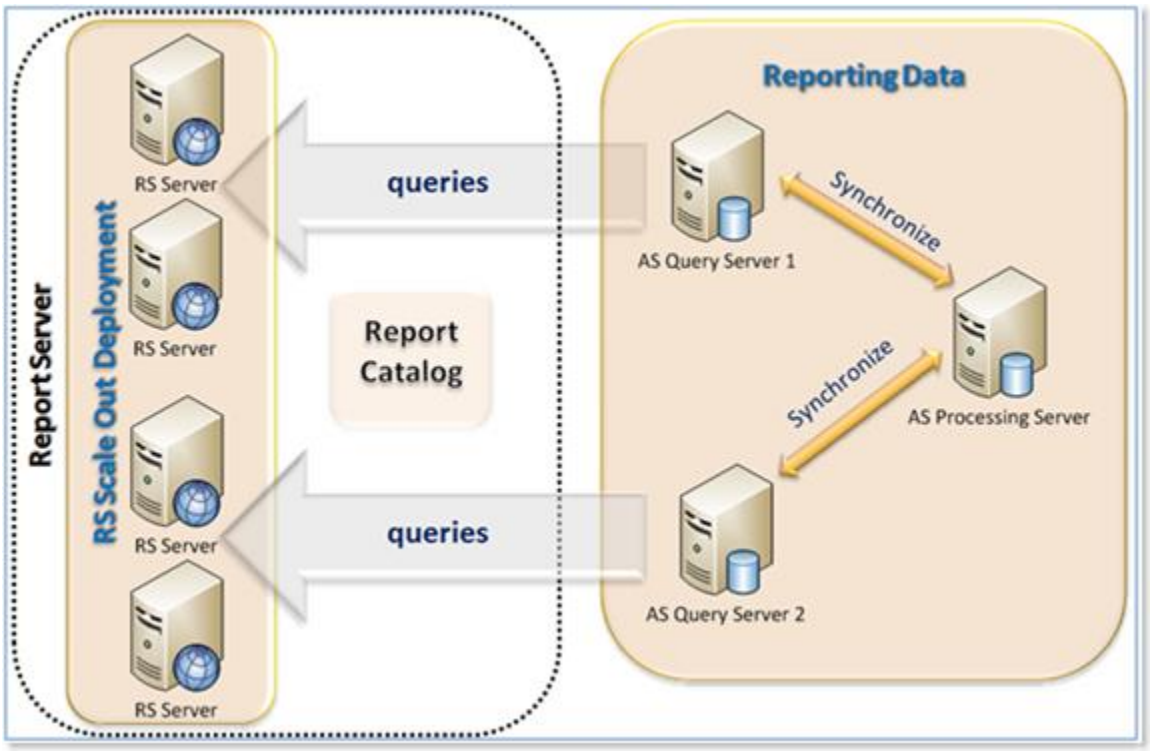


Figure 4: Scaling out Analysis Services

These options are outside the scope of a Reporting Services specific technical note, but here are some resources that you may find helpful:

- Deploying a Scalable Shared Database: This feature allows you to deploy multiple read-only versions of your SQL Server database for the purposes of reporting. Scalable shared databases offer a scale-out solution in which multiple servers hold read-only copies of your SQL Server database that users can query thus distributing the query load across multiple databases or servers.

- SQL Server Replication: Providing High Availability using Database Mirroring: This white paper covers the use or database mirroring to increase availability of the replication stream.
- Database Mirroring and Log Shipping: You can use database mirroring and log shipping together to provide both high availability and disaster recovery.
- SQL Server Replication Features: Choose among transactional, merge, or snapshot replication to provide yourself with query replica databases.
- Scale-Out Querying with Analysis Services: This white paper describes how to replicate a read-only Analysis Services database to multiple servers.
- Scale-Out Querying with Analysis Services Using SAN Snapshots: If you have a SAN, you can make use of SAN snapshots to perform the task of metadata replication instead of physically copying the underlying Analysis Services database.
- Scaling out an Analysis Services Solution: In addition to attaching or detaching an Analysis Services database, you can also attach a read-only copy of your Analysis Services database that places the Analysis Services database metadata locally and refers to a remote copy of the database. That is, one copy of a database has metadata distributed to multiple servers so that the formula calculation load can be scaled out to many servers.

Next Steps…

Now, that you've scaled out your Reporting Services deployment – there are additional features and configurations that may optimize your Reporting Services environment; for more information, see ***SSRS Performance Optimization Configurations [link provided when published]***.

Report Server Catalog Best Practices

Introduction

This technical note is part of the Reporting Services Scale Out Architecture which provides general guidance on how to set up, implement, and optimize an enterprise scale-out architecture for your Microsoft SQL Server Reporting Services (SSRS) environment. This note provides guidance for both SQL Server 2005 and 2008 Reporting Services. The focus of this technical note is the report server catalog—the underlying databases that provide the metadata (parameters, snapshots, history, and so on) that are used by Reporting Services to provide your reports.

Architecture

Figure 1 represents a typical scale-out Reporting Services environment; as indicated by the red box, the focus of this technical note is that of the report server catalog.

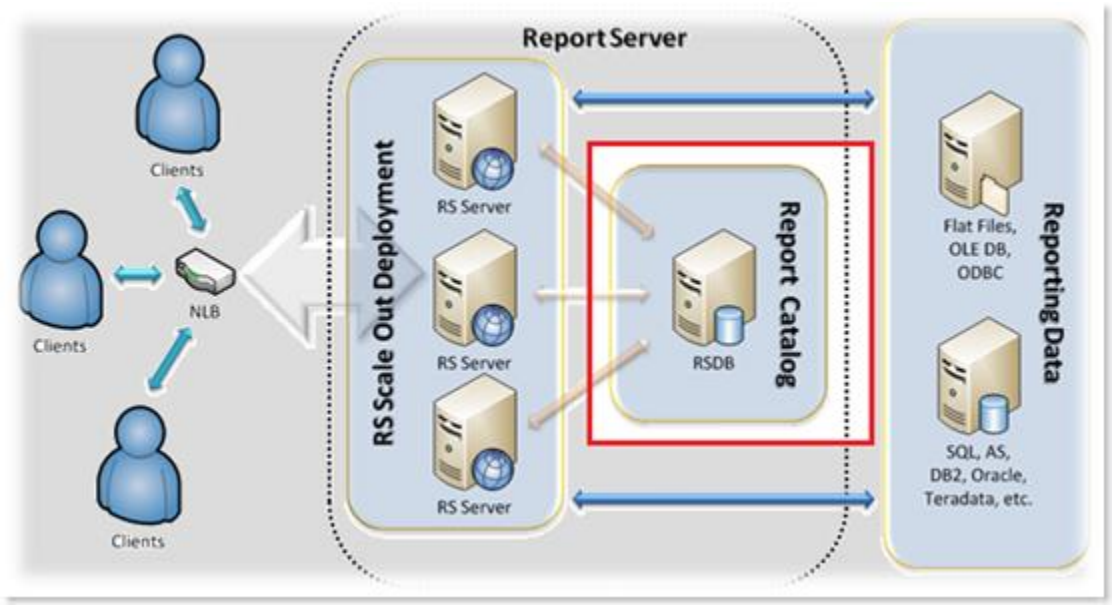


Figure 1: Reporting Services Scale Out Architecture

As you can see, all Reporting Services servers have connections to the report server catalog. As implied by this diagram, this single instance could be a possible bottleneck in the SSRS environment. Because of this, let's focus on what these report server databases do and how to optimize them for your environment.

Report Server Databases

The report server catalog is comprised of two report server databases:

- **ReportServer** (**RSDB)** stores all report metadata including report definitions, report history and snapshots, and scheduling information.

- **ReportServerTempDB** (**RSTempDB)** stores all of the temporary snapshots while reports are running.

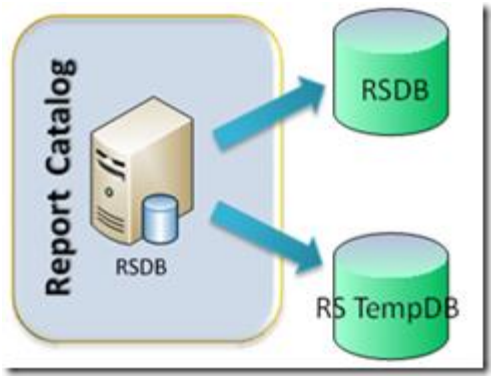


Figure 2: Breakdown of Report Server Catalog

These databases are responsible for containing the metadata that is needed by the Reporting Services engine so that it knows how to request data from the data source (such as a SQL Server database or an Analysis Services database). When a report is executed by a report server, SSRS loads report metadata from **RSDB**. This includes the data source information for the report, the default parameter values, and the report itself. Note that for ad-hoc reports, this data is not loaded from the **RSDB** catalog database because the report comes from the RDL that is executed, published, and bound to the user's session. As well, all user session information and live or cached execution snapshots are stored in the **RSTempDB** database. History and execution snapshots are stored in the **RSDB** database. Subsequent report requests can use the stored state to view the report rather than re-executing the report.

Report execution state is represented in a snapshot. The snapshot contains the data that is retrieved from the report's queries to the data source. SSRS keeps these snapshots so that subsequent requests for the same report can be satisfied by using the snapshot instead of re-executing the report and obtaining the data again from the original data source (thus reducing the load on your data source). By default, these snapshots are stored in the **RSTempDB** database. Therefore, commonly requested reports hit the **RSTempDB** database instead of querying the original data source (provided the snapshot is current). While this reduces the load against your original data source, it does increase the load on your SSRS catalog. The size of your snapshot is directly correlated to the size of the data returned by the queries of your reports. Because of this, it is important to reduce the size of your snapshots by including only the columns (and rows) you need.

In addition, this also implies that you must optimize the SSRS report server catalog to prevent it from being a bottleneck since every single report request hits **RSDB**. For frequently requested reports, many database queries will hit **RSTempDB**. Because these databases are hit for every single report request, the report server catalog has a lot of I/O and transactions in order to share state information across multiple SSRS servers. The **RSTempDB** tables and log will grow quickly because many transactions are performed to maintain report consistency. (For example, if you get the first page of a report, you need a snapshot of the second page that is consistent.) If you have an environment with many concurrent requests (typical of most enterprise Reporting Services environments), there can be a lot of write activity to the **RSTempDB** log.

For SQL Server 2005 Reporting Services, many of the insertions are performed to the **ChunkData**, **SnapshotData**, and **SessionData** tables. But SQL Server 2008 in general does not write to these tables. They exist in the catalog to support the upgrade of pre-2008 catalog. SQL Server 2008 Reporting Services uses a new snapshot storage mechanism that shreds chunks across multiple rows in a table named **Segment**. For SQL Server 2008 Reporting Services, this table generally takes on the majority of transactions of the **RSTempDB** database.

As implied by the above, there are differences in **RSDB** and **RSTempDB** access patterns between versions of SSRS. SQL Server 2008 Reporting Services makes use of **RSTempDB** significantly more than SQL Server 2005 Reporting Services does. This is because we process data incrementally during report execution rather than immediately when the report is executed. To make the report execution reliable, we store the data for the report in **RSTempDB** until it is needed to generate a particular page of the report. Clearly, this increases the number and size of queries executed against the **RSTempDB** database and can lead to bottlenecks.

Therefore, to optimize your report server catalog, we suggest the following best practices.

Use a Dedicated Server

As noted earlier, a heavy transaction load is placed on the SSRS report server catalog. Because of this, the SSRS report server catalog load can interfere with other processes and vice versa. For example, common environment setups include the following:

Same server as SSRS Windows/Web Services

The default one-box setup usually has the SSRS Windows/Web Services and the report server catalog on the same server. While this works great for small environments, in enterprise environments it causes far too much resource (CPU, memory, and disk) contention between the SSRS Windows/Web Services and the report server catalog. As well, when you scale out and add more SSRS servers, you want to have all of SSRS servers point to one dedicated report server catalog to reduce contention.

Same server as your data source relational database (SQL)

Another common approach is to place your SSRS report server catalog on the same server as your SQL Server data source. The issue here is that you will have SQL resource contention (**tempdb**, plan cache, memory, buffer pool, etc.) between your SQL Server data source and your SSRS report server catalog. As you have more concurrent users, you will have a lot of hits to the SSRS report catalog (**RSDB** for report metadata, **RSTempDB** for report snapshots) and transactions against the relational data source. As the load increases, it will become necessary to monitor things like CPU utilization, I/O response times, network resources, and buffer pool counters to ensure that adequate resources are still available. A common method to alleviate these issues is to separate the SSRS report server catalog from your SQL server data source.

As you can see, these two common scenarios create database resource contentions that slow down performance. Because of this, it makes sense to have a dedicated server for your SSRS report server catalog so that you can tune your report server databases separately from your relational data source and not run into SSRS Windows/Web Services and report catalog resource contention.

High-Performance Disk

Because your SSRS report server catalog has a lot of transactions, ultimately there will be a lot of disk I/O so that storage may be your resource contention. Because of this, you want to have a high-performance disk such as a SAN or high-RPM direct-attach storage for your report server catalog. Some more specifics:

- To optimize your disk I/O, see the SQL Server Best Practices white paper Predeployment I/O Best Practices, which provides great information on how storage works and how to use various tools to understand what performance you may be able to obtain for SQL Server from the storage you have.
- Have more smaller sized disks with faster rotation speeds (e.g. >= 15,000 RPMs) rather than fewer larger sized disks with slower rotation speeds. That is, you should size your I/O based on IOS requirements instead of sizing based on capacity.
- Maximize and balance your I/O across all of the available spindles.
- Use separate disks for **RSDB** and **RSTempDB**. The profile for your **RSDB** is a lot of small transactions because it asks for report metadata. Your **RSTempDB** will have a lot of transactions as well, but they will be larger transactions because this database contains the report data. Having separate disks will enable you to tune your disk for the **RSDB** and **RSTempDB** databases separately.
- Pre-grow your SSRS report server catalog databases instead of having SQL Server perform autogrow on these databases. **RSTempDB** can grow very quickly under load and the default autogrow options lead to a lot of file system fragmentation and blocking during the autogrow task.
- For your SSRS server catalog databases, stripe your database files to the number of server cores at a ratio of 0.25 to 1.0 depending on how heavy your workload is. This enables you to minimize database allocation contention and makes it easier to rebalance your database when new LUNs become available.
- If you are using RAID, like many other SQL implementations that are write-intensive, use RAID 10 to get the best performance. Do not use RAID 5 because of the write penalty that may be involved.
- Monitor disk response times to ensure that disk latency is generally lower than 20ms, ideally 10ms, and log latency is no more than 1-5ms. To do this, look for high wait times on PAGEIOLATCH_xx or use **sys.dm_os_virtual_file_stats** to monitor response times specifically on SSRS-related databases.

Move to 64-bit

For starters, if you need to stay with 32-bit and have >3 GB of memory because of the available hardware and OS, remember to use the /3GB and/or the /PAE (for systems with >4 GB of memory) switches (for the OS) and enable AWE in SQL Server so that it can use more than 3 GB of memory. Note that AWE can only be used for data cache. Do not forget that this involves both SQL Server changes (configure advanced options) and changes to the Windows OS boot.ini file.

We suggest moving to 64-bit because much of the hardware available right now is 64-bit and, as of SQL Server 2005, SQL Server itself natively supports 64-bit. With 64-bit, you have a much larger addressable memory space to use—especially if you increase the amount of memory. This means that you can handle larger queries (more data) and handle more connections to the server running SQL Server. Note that this does not result in higher throughput as that is typically bound to CPU. Nevertheless, the ability to handle more connections and larger reports minimize the chance that your report server catalog will be a bottleneck for your system. As well, the ability for 64-bit to scale is much higher than 32-bit and this is the platform of choice for SQL databases going forward.

Backup/Restore

The data in **RSTempDB** is highly volatile—typically one can expect its lifespan to be approximately equal to the **SessionTimeout** value configured for the SSRS server for most reports and viewing and usage time. The default **SessionTimeout** is 10 minutes, which is the report lifetime policy that defines when data can be cleaned up. The **CleanupCycleMinutes** value is the parameter that guides the background cleanup thread. Once the session timeout value is reached, we clean up the temporary snapshot from **tempdb**. We do that every cleanup cycle minutes, or continuously if the previous cleanup didn't complete yet. The actual lifespan varies based on usage patterns but a lifespan longer than one day would be rare. As such, it is not necessary to protect **RSTempDB** data for data recovery purposes.

The data in **RSDB** is long lived—this data should be backed up following the standard guidance provided for SQL Server:

- Backing Up and Restoring Databases in SQL Server
- Optimizing Backup and Restore Performance in SQL Server

As well, do not forget to back up (and restore) the encryption key associated with these databases; you can find more information at Backing Up and Restoring Encryption Keys

Maintain Your SSRS Report Server Catalog Databases

Recall that the SSRS report server catalog databases are SQL Server databases specifically for Reporting Services usage. Therefore, the standard techniques to maintain SQL databases apply to the SSRS report server catalog databases. For example, periodically re-indexing the catalog tables and/or updating the database statistics may improve query performance.

As noted above, you may want to consider configuring the **CleanupCycleMinutes** setting in the RSReportServer.config file. This setting determines how frequently expired session content or unused snapshot data is removed from **RSTempDB**. The default setting is 10 minutes, which is similar to the default session timeout. **RSTempDB** generally stays more compact when using frequent cleanups, but at the cost of increasing the general load on the system. If the size of **RSTempDB** not a major concern and the system has high throughput user loads, you may want to considering slightly increasing the **CleanupCycleMinutes** configuration (such as setting it to 20 minutes).

Discussion

Since the SSRS Windows/Web Services interact with your report server catalog for almost all SSRS queries, it is important that you optimize your SSRS catalog databases so that they are not a point of contention. Standard SQL optimization techniques come into play here since SSRS report server catalogs are instances of SQL Server databases. Following the above suggested methods will make your SSRS environment easier to scale to enterprise data loads.

Recall that because of the concern for report consistency, it is important for Reporting Services to cache all report data. Therefore, to reduce the number of requests that are placed on your report server catalog, you may want to consider using the File System (FS) Snapshots feature, which is discussed in the next technical note of this technical series—S*SRS Scale-Out Deployment Best Practices* [work in progress].

Reporting Services Performance Optimizations

Introduction

This technical note is part of the Reporting Service Scale Out Architecture which provides general guidance on how to set up, implement, and optimize an enterprise scale-out architecture for your Reporting Services environment. This note provides guidance for Reporting Services in both Microsoft® SQL Server® 2005 and SQL Server 2008. The focus of this technical note is to optimize your Reporting Services architecture for better performance and higher report execution throughput and user loads.

Architecture

Figure 1 represents a typical scale-out Reporting Services environment; as noted by the red box, the focus of this technical note is that of **Performance Optimization** of your scale-out deployment.

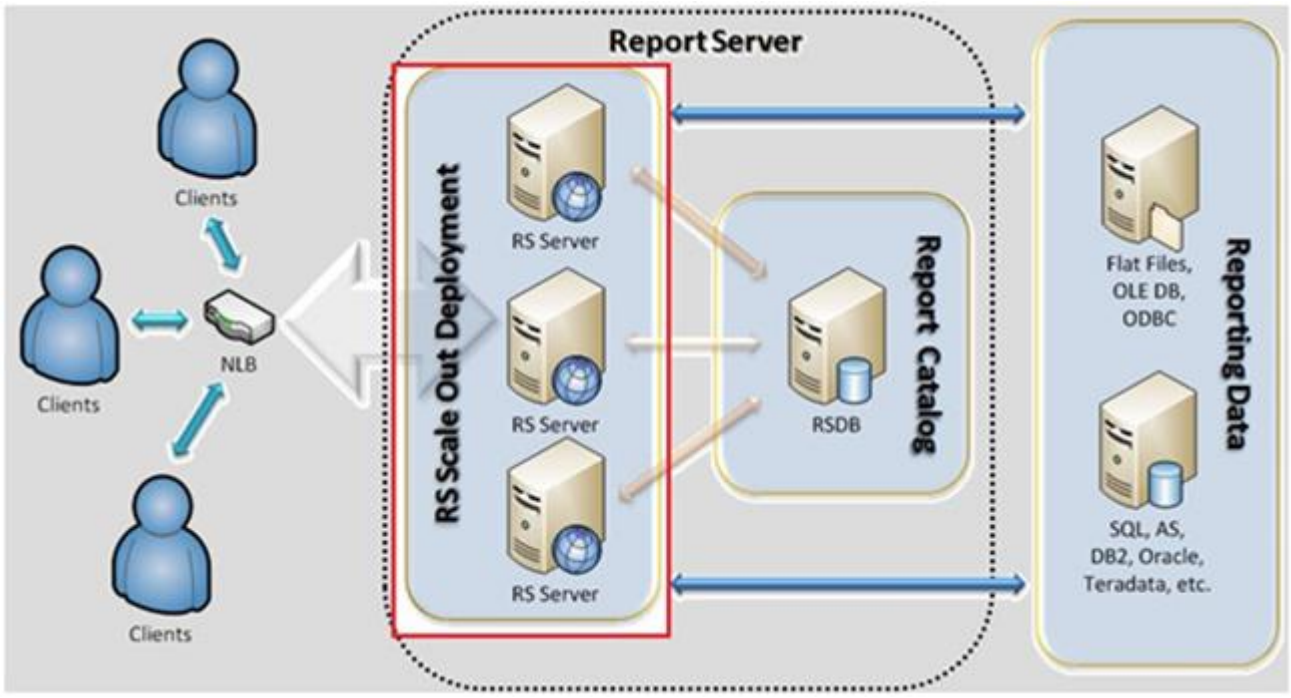


Figure 1: Reporting Services Scale-Out Architecture

Should You Use 64-Bit?

Yes.

Why, you ask? The answer can be divided into two different sections – how 64-bit helps your report server catalog performance and how it helps your report server performance.

How 64-bit Helps Your Report Server Catalog Performance

Remember that your report server catalogs are SQL Server databases, so your standard database techniques for optimizing SQL Server databases come into play here, as noted in the Report Server Catalog Best Practices technical note. Since SQL Server 2005, the database has been written natively for 64-bit and it is able to make use of the additional addressable memory.

How 64-bit Helps Your Report Server Service

For your report servers, the answer is slightly more complicated. As a general statement, most of your Reporting Services reports are memory-intensive, and the additional addressable memory made available by 64-bit will provide you with more scale. Note that for some workloads, 32-bit can execute faster, especially in scenarios where you have many small reports. But if you have more memory made available by 64-bit, you can handle more concurrent report users. Because there is less memory contention, report throughput will be higher; that is, you can allow more users to view and export larger reports. In SQL Server 2005 Reporting Services, each report's data set is placed into memory; the more concurrent reports, the more memory used. If you use 32-bit, you can easily hit the 3 GB ceiling, which can also result in Internet Information Services (IIS) process recycles, leading to report failure.

But as noted in the Reporting Services Scale-Out Deployment Best Practices technical note, SQL Server 2008 Reporting Services is not memory-bound. It is able to effectively use the file system in order to move data structures in and out of memory and the file system if the report server experiences memory pressure. These memory limits are explicitly configurable in SQL Server 2008 Reporting Services via RSReportServer.config, as noted in the *Memory Configurations for SQL Server 2008 Reporting Services* section below. When Reporting Services uses the file system, the reports run more slowly, because it is much more efficient to request data from the memory than from disk. The file system is used only if Reporting Services memory usage gets close to the configured memory limits. If you overload the report server in SQL Server 2008 Reporting Services with a large number of concurrent users and/or very large reports, all of your reports can still complete processing, albeit more slowly, because Reporting Services can hold all of this data without running out of memory space. In enterprise environments, you will eventually run into situations where your servers will need to be able to handle many concurrent users and a large load – the optimization in SQL Server 2008 Reporting Services (in comparison to SQL Server 2005 Reporting Services) is that while the reports may run slower at times, they will complete.

Exceptions

Keep in mind that certain data providers are not available for 64-bit (for example, the Microsoft JET provider or certain third-party providers). In these cases, customers will need to continue using 32-bit for their Reporting Services environment.

Handling a Large Workload

As noted in the previous section, the two main issues concerning enterprise reporting environments are the ability to handle concurrent user load and the ability to handle a large workload (that is, large reports). To help mitigate the

concurrency issue, your solution is to scale out to multiple report servers to handle the user query load, as noted in the Reporting Services Scale-Out Deployment Best Practices technical note.

To get the highest performance when handling large workloads that include user requests for large reports, implement the following recommendations.

Control the Size of Your Reports

You will first want to determine the purpose of these reports and whether a large multipage report is even necessary. If a large report is necessary, how frequently will it be used? If you provide users with smaller summary reports, can you reduce the frequency with which users attempt to access this large multipage report? Large reports have a significant processing load on the report server, the report server catalog, and report data, so it is necessary to evaluate each report on a case-by-case basis.

Some common problems with these large reports are that they contain data fields that are not used in the report or they contain duplicate datasets. Often users retrieve more data than they really need. To significantly reduce the load placed on your Reporting Services environment, create summary reports that use aggregates created at the data source, and include only the necessary columns. If you want to provide data feeds, you can do this asynchronously using more appropriate tools, such as SQL Server Integration Services, to provide the file data feed.

Use Cache Execution

As noted in the Reporting Services Scale-Out Deployment Best Practices, if you have reports that do not need to have live execution, enable the cache execution setting for each of your appropriate reports. This setting causes the report server to cache a temporary copy of those reports in memory.

Configure and Schedule Your Reports

For your large reports, use the **Report Execution Timeouts** setting to control how long a report can execute before it times out. Some reports simply need a long time to run, so timeouts will not help you there, but if reports are based on bad or runaway queries, execution timeouts ensure that resources are not being inappropriately utilized.

If you have large reports that create data processing bottlenecks, you can mitigate resource contention issues by using **Scheduled Snapshots**. Instead of the report data itself, a regularly scheduled report execution snapshot is used to render the report. The scheduled snapshot can be executed during off-peak hours, leaving more resources available for live report users during peak hours.

Deliver Rendered Reports for Nonbrowser Formats

The rendering performance of nonbrowser formats such as PDF and XLS has improved in SQL Server 2008 Reporting Services, as noted in the Scaling Up Reporting Services 2008 vs. Reporting Services 2005: Lessons Learned technical note. Nevertheless, to reduce the load on your SQL Server Reporting Services environment, you can place nonbrowser format reports onto a file share and/or SharePoint® team services, so users can access the file directly instead of continually regenerating the report.

Prepopulate the Report Cache by Using Data-Driven Subscriptions for Parameterized Reports

For your large parameterized reports, you can improve performance by prepopulating the report cache using data-driven subscriptions. Data-driven subscriptions enable easier population of the cache for set combinations of parameter values that are frequently used when the parameterized report is executed. Note that if you choose a set of parameters that are not used, you take on the cost of running the cache with little value in return. Therefore, to identify the more frequent parameter value combinations, analyze the ExecutionLog2 view as explained below. Ultimately, when a user opens a report, the report server can now use a cached copy of the report instead of creating the report on demand. You can schedule and populate the report cache by using data-driven subscriptions. For more information, see Report Caching in Reporting Services.

Back to the Report Catalogs

You can also increase the sizes of your report server catalogs, which allows the databases to store more of the snapshot data. For more information, see Report Server Catalog Best Practices.

Tuning the Web Service

IIS and Http.sys tuning helps get the last incremental performance out of the report server computer. The low-level options allow you to change the length of the HTTP request queue, the duration that connections are kept alive, and so on. For large concurrent reporting loads, it may be necessary to change these settings to allow your server computer to accept enough requests to fully utilize the server resources.

You should consider this only if your servers are at maximum load and you do not see full resource utilization or if you experience connection failures to the Reporting Services process. To do this:

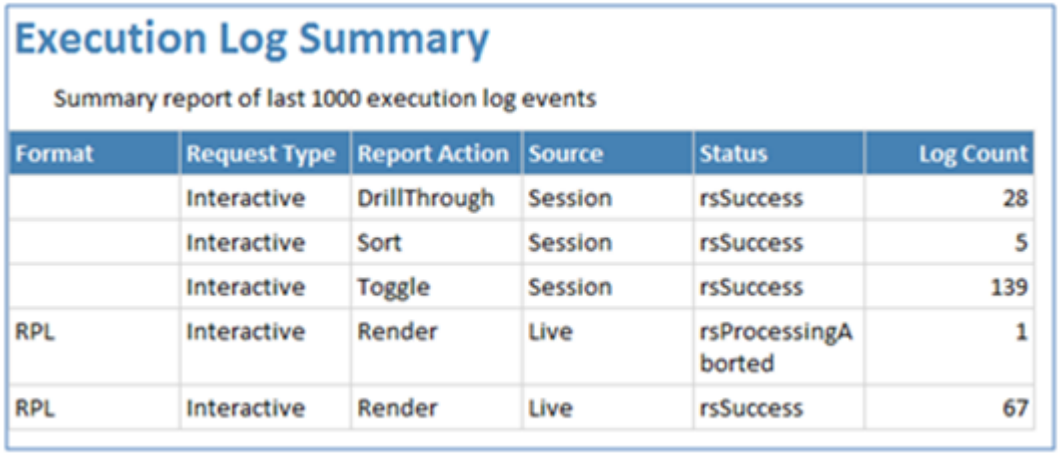
- For SQL Server 2005 Reporting Services, tune IIS.
- For SQL Server 2008 Reporting Services, tune Http.sys within the operating system: Windows® 2003 or Windows 2008.

Monitoring by Using ExecutionLog2

The Reporting Services ExecutionLog2 view is a good starting point from which to analyze your current workloads and understand its dataset size, performance, and complexity characteristics. For more information, see Robert Bruckner's blog, which provides extensive details on the ExecutionLog2 view (http://blogs.msdn.com/robertbruckner/archive/2009/01/05/executionlog2-view.aspx). For more information about query and reporting on report execution log data, see SQL Server Books Online (http://msdn.microsoft.com/en-us/library/ms155836.aspx).

In particular, this view contains a new **AdditionalInfo** column. ExecutionLog2.AdditionalInfo contains information related to the size of memory-pressure responding data structures. One way this information can be useful is to check whether you have reports with high values (10s, or 100s of MBs) – these reports might be candidates for further review, focusing on the design of those reports and the dataset query sizes.

Below are some tips on how to view the ExecutionLog2 view to quickly understand potential performance bottlenecks.

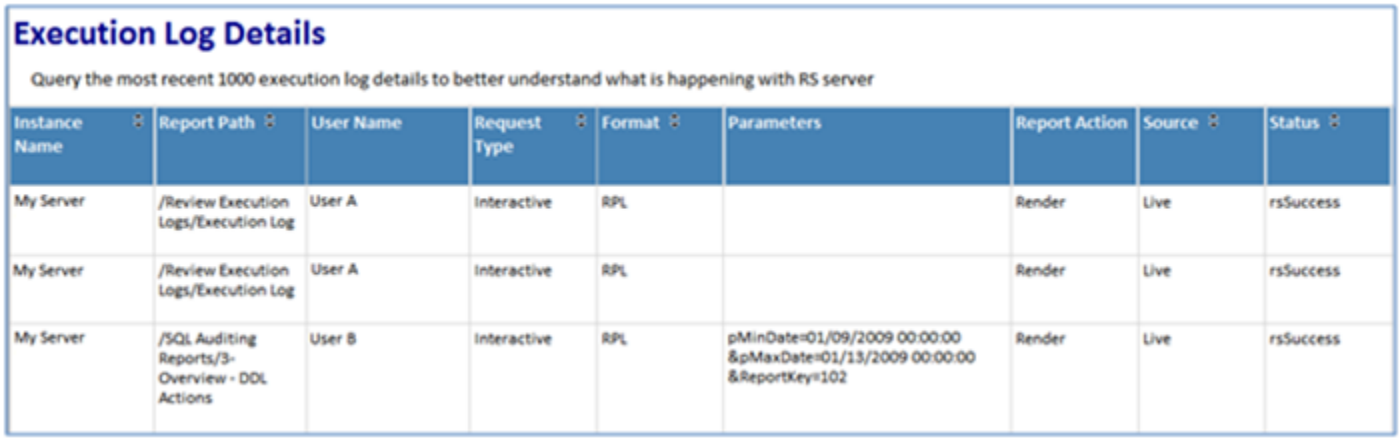


Execution Log Summary

Summary report of last 1000 execution log events

Format	Request Type	Report Action	Source	Status	Log Count
	Interactive	DrillThrough	Session	rsSuccess	28
	Interactive	Sort	Session	rsSuccess	5
	Interactive	Toggle	Session	rsSuccess	139
RPL	Interactive	Render	Live	rsProcessingAborted	1
RPL	Interactive	Render	Live	rsSuccess	67

Figure 2: Review Execution Logs (ExecutionLog2) Summary Report



Execution Log Details

Query the most recent 1000 execution log details to better understand what is happening with RS server

Instance Name	Report Path	User Name	Request Type	Format	Parameters	Report Action	Source	Status
My Server	/Review Execution Logs/Execution Log	User A	Interactive	RPL		Render	Live	rsSuccess
My Server	/Review Execution Logs/Execution Log	User A	Interactive	RPL		Render	Live	rsSuccess
My Server	/SQL Auditing Reports/3-Overview - DDL Actions	User B	Interactive	RPL	pMinDate=01/09/2009 00:00:00 &pMaxDate=01/13/2009 00:00:00 &ReportKey=102	Render	Live	rsSuccess

Figure 3: Review Execution Logs (ExecutionLog2) Details Report

Long-Running?

Sorting by **ElapsedSec** or **RowCount** helps you identify long-running reports. If the value for **TimeDataRetrieval** is high, the data source is your bottleneck, and you may want to optimize. If the there is a high value for **RowCount**, a lot of data is being retrieved and aggregated by Reporting Services – perhaps have your data source do this to reduce the load on your report server.

Subscriptions or Interactive?

Sorting by the **RequestType** field allows you to determine whether you have a lot of subscriptions; you can then determine the bottlenecks and stagger-schedule the reports (that is, schedule the subscription execution times of the reports at different times).

Live Data or Snapshots?

Sorting by the **Source** field allows you to determine whether your reports are typically live data or snapshots. If the reports can be snapshots (for example, yesterday's report), create snapshots so you can avoid query execution, report processing, and report rending.

Load Balanced?

Sorting by the **Instance** field can help you see whether your network load balancer is handling report requests in a balanced fashion. You can also see if some nodes are down or not processing requests.

Discover Report Patterns

Sorting by **ReportPath** and **TimeStart** can help you to find interesting report patterns – for example, an expensive report that takes 5 minutes to run is executed every 10 minutes.

Report Health

You can sort by status to determine if you have a high number of failures that occurred before (for example, incorrect RDL) or after (for example, subscription delivery error) the report is processed. This can also help you identify reports where there is outdated information or settings (for example, expired data source passwords or missing subreports).

In addition, if ScalabilityTime > 0, Reporting Services is in scale mode, which means that it is under heavy memory pressure and will start pushing long-running reports to the file system to ensure enough memory to complete smaller query requests. If this happens frequently, consider trying one of the following:

- Reduce the size of the dataset.
- Simplify the grouping, filtering, or sorting logic in the report to use less memory.
- Add more memory.
- Add more servers to better handle the load.

Data-Driven Subscriptions

Based on all of this information, you can then create your own data-driven subscriptions that can sort, filter, and track your issues. For example, you can create a subscription that alerts you if Errors > 5%.

Memory Configurations for SQL Server 2008 Reporting Services

As alluded to above in the 64-bit section, memory in SQL Server 2008 Reporting Services is used more efficiently at the data structure level. Under intensive workload pressures, it uses a file system cache in order to reduce memory utilization. By being more efficient and writing to disk, it can process more report executions (even large reports) successfully. The administrator uses settings to determine when SQL Server 2008 Reporting Services starts to use the file system cache and how aggressively the cache is used. The administrator should consider configuring memory settings for the SQL Server 2008 Reporting Services to optimally use their computer resources. Fine-tuning these settings can provide an increase in performance under intensive workload over the default configuration.

For more information, see SQL Server 2008 Books Online: Configuring Available Memory for Report Server Applications. The key memory configurations are:

- **WorkingSetMinimum**: This is the minimum amount of memory that Reporting Services makes available in order for it to perform its task; that is, it does not use the file system cache if SQL Server Reporting Services process memory utilization is below this limit. It is configured as a KB value within the RSReportServer.config file. After this threshold is hit, Reporting Services responds to memory pressure by having long-running requests use the file system cache and smaller queries use memory.

For many concurrent users, you can potentially increase this configuration value after it hits peak performance, enabling more process requests to be completed within memory instead of the file system.

- **WorkingSetMaximum**: This is the maximum amount of physical memory Reporting Services will use; it is also configured as a KB value within the RSReportServer.config file. After this threshold is hit and exceeded for a period of time, Reporting Services recycles the app domains to reduce memory utilization. This will ensure that there is enough memory left for the operating system to complete its task. You can increase this value if you want to process more reports concurrently.

- **MemorySafetyMargin** and **MemoryThreshold**: These values define how aggressively the file system cache is used. **MemorySafetyMargin** defines the boundary between low-pressure and medium-pressure scenarios, with a default value of 80%. **MemoryThreshold** defines the boundary between medium-pressure and high-pressure scenarios, with a default value of 90%. Both are configured within the RSReportServer.config file.

In general, if you are constantly hitting memory thresholds, it is recommended that you consider scaling up (adding more memory, altering these configuration values) and then scaling out. You should scale up first because resources are better managed in SQL Server 2008 Reporting Services, as noted above and in Scaling Up Reporting Services 2008 vs. Reporting Services 2005: Lessons Learned.

Memory Configurations for SQL Server 2005 Reporting Services

As noted in the Reporting Services Scale-Out Architecture technical note, you can scale up by adding more memory as well. But the impact may not be as dramatic; one scenario might see a doubling of memory and CPU (to 16 GB RAM and 8 cores) with only 1/3 increase in capacity. Nevertheless, there are things you can still do to scale up SQL Server 2005 Reporting Services prior to scaling out.

As with SQL Server 2008 Reporting Services, you can use memory configurations to address memory threshold issues. There are two primary memory configurations in SQL Server 2005 Reporting Services that we recommend you change if you're constantly hitting memory thresholds:

- **MemoryLimit**: This configuration is similar to **WorkingSetMinimum** in SQL Server 2008. Its default is 60% of physical memory. Increasing the value helps Reporting Services handle more requests. After this threshold is hit, no new requests are accepted.
- **MaximumMemoryLimit**: This configuration is similar to **WorkingSetMaximum** in SQL Server 2008. Its default is 80% of physical memory. But unlike the SQL Server 2008 version, when its threshold is met, it starts aborting processes instead of rejecting new requests.

While we do recommend memory configuration changes if you're constantly hitting memory thresholds, note that changing these values might create or exacerbate other resource contentions.

Conclusion

This concludes our four-part "Building and Deploying Large Scale SQL Server Reporting Services Environments" technical note series. We hope that this set of technical notes will help you design, manage, and maintain your enterprise SQL Server Reporting Services environments.

Reporting Services Scale-Out Architecture

Introduction

This technical note is the introduction to the *Building and Deploying Large Scale SQL Server Reporting Services Environments Technical Note Series,* which provides general guidance on how to set up, implement, and optimize an enterprise scale-out architecture for your SQL Server Reporting Services (SSRS) environment. This note provides guidance for both SQL Server 2005 and 2008 Reporting Services. The focus of this technical note is the Reporting Services scale-out architecture, which is referenced throughout this technical note series.

Architecture

Figure 1 represents a typical scale-out Reporting Services environment.

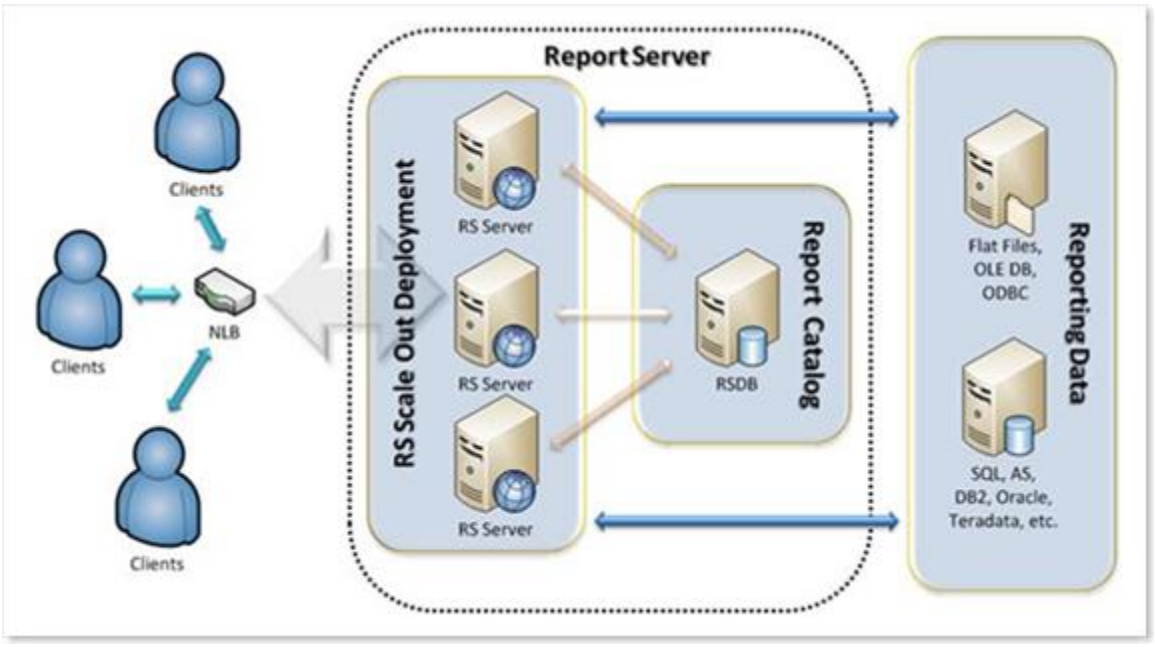


Figure 1*: Reporting Services Scale-Out Architecture*

From this architecture, you will notice the following:

- When your **clients** execute a query to Reporting Services, the query goes through a Network Load Balancer (NLB) to connect to the appropriate Reporting Services server. In order to balance the query load, your NLB can round-robin between the different SSRS servers so that the requests are evenly distributed amongst all the SSRS servers and a single server instance is not overburdened by too many requests.

- The Reporting Services Windows/Web Services (denoted as **RS Server**) handle the client query request and perform the task of submitting the query to the report server catalog and/or the assigned relational data source, as well as resolving the data results back to the client. In enterprise environments, you will often want to scale out to multiple servers to handle many concurrent queries against your Reporting Services servers.

- The RS Server makes a request to the **report catalog** databases initially to obtain the report metadata, which includes report definitions, report history and snapshots, and scheduling information from the **ReportServer** database (**RSDB**). If the report data desired is already available in a SSRS snapshot, the RS Server makes a request for this from the **RSTempDB** database, which stores all of the temporary snapshots while reports are being run. As noted in the above architecture design, in an enterprise environment you have multiple SSRS servers reading metadata information from the same report catalog instance. This is done to keep state information between SSRS servers; when a user requests a report from one SSRS server, if the subsequent query goes to another SSRS server, the state of that report (which report was used, what parameters were used, etc.) is kept so that the transition from one SSRS server to another is transparent to the user requesting the report.

- If the report information is not available in the form of a SSRS snapshot, the SSRS server makes a request to the data source (denoted as **Reporting Data)**. Whether your data source is a flat file or an SQL database, how this data is requested is defined by the parameters in the report catalog to enable the SSRS server to perform the task of requesting this data.

Keys to Success

To ensure a successful enterprise implementation of your Reporting Services environment, please note first that a lot of documentation is available online. Many mistakes in enterprise customer implementations can be avoided by reviewing the following white papers.

- Upgrading Reporting Services (SQL Books Online): This a good reference to understand the issues related to upgrading from SQL Server 2000 Reporting Services to SQL Server 2005 Reporting Services. You can find the documentation for upgrading from SQL Server 2005 Reporting Services to SQL Server 2008 Reporting Services at Upgrade (Reporting Services), though this is pre-release documentation at this time.

- Configuring a Report Server Scale-Out Deployment: Before starting a scale-out deployment for Reporting Services, please be sure to review this document, which includes deployment steps, creating and configuring SSRS instances, and other important configuration steps.

- Planning for Scalability and Performance with Reporting Services: To plan for scalability and optimal performance with Reporting Services, this is a very important white paper for you to review. Before you design your scale-out solution, we definitely advise you to review this paper.

Customer Scenario

Below is an example customer scenario that has been shrunk down to its core components.

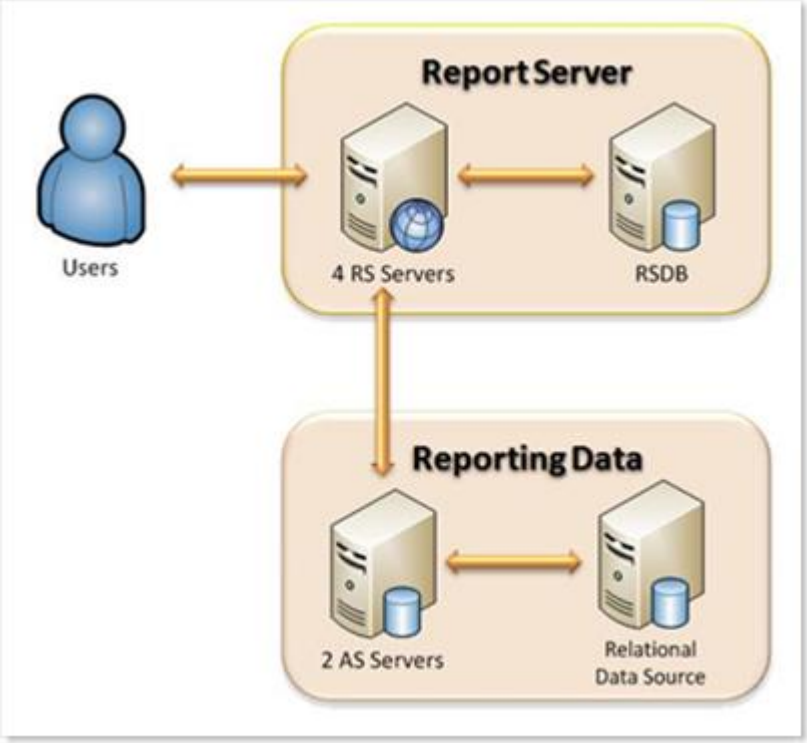


Figure 2: *Customer Scenario*

This customer currently has:

- Four SSRS servers that keep state information by connecting to a separate server that contains both report server catalog databases (**RSDB** and **RSTempDB**).

- The four SSRS servers connect to two SQL Server Analysis Services (SSAS) servers to provide pivot capability in the form of multidimensional reports.

- Their two SSAS servers are connected to a single relational data source; they have two SSAS servers in order to handle the query loads placed on SSAS servers. For more information on scaling out SSAS servers, see:
    - Analysis Services Synchronization Best Practices
    - Scale-Out Querying with Analysis Services
    - Scale-Out Querying with Analysis Services Using SAN Snapshots
    - Sample Robocopy Script to customer synchronize Analysis Services databases

To help handle the high query load, the customer also made use of:

- Use of a dedicated **RSDB** server for the Reporting Services database instances to reduce database blocking issues due to resource contention. For more information, see Report Catalog Best Practices within this technical note series.

- SSRS File System snapshots to make use of the cache; more information on this can be found in *SSRS Scale-Out Deployment Best Practices* [to be published] within this technical note series.

With this setup, they were able to use Visual Studio Test System to verify that the above configuration could handle 1,800 concurrent users with a 10-sec think time (time delay between the execution of each user's reports) and an average transaction time of 35 sec.

Performance Testing

Another key component to ensuring a successful enterprise implementation is the ability to determine the query load prior to actually placing it into production. It is important that you understand your reporting scenarios and the data that needs to be delivered so that you can better plan capacity (hardware, resources, etc.). The above customer executed tests to make sure they could handle their expected maximum workload of 1,800 concurrent users. This allowed them to better plan for hardware capacity—and they realized that they did not need to build out more SSRS servers to handle their workload.

Your report scenarios are defined by user personas and usage patterns. For user personas, you need to understand the type of users who are using your reports, whether it is the director who typically wants high-level summary statistics or the analyst who needs detailed reports. Understanding usage patterns enables you to know if you have users who typically want to review a few sets of reports or want to heavily interact (drill down / up / through) with these reports. Once you understand your typical report scenarios, it is important for the purpose of performance to use actual reports when you test. If you do not have these, your test reports should closely reflect what users will actually be querying. As well, any and all tests that you do should isolate Reporting Services from other systems. This is implied from the above architecture diagrams, but a common issue is that customers run performance tests on shared systems causing both performance degradation and making it difficult to isolate problems.

To run performance tests, a good tool to use is Visual Studio Team System (VSTS)—and a great reference is Using Visual Studio 2005 to Perform Load Testing on a SQL Server 2005 Reporting Services Report Server. As well, please refer to the technical note Scaling Up Reporting Services 2008 vs. Reporting Services 2005: Lessons Learned.

Customer Scenario Testing

Our customer was able to use and define within VSTS for their tests:

- Mean transaction time range: 33-36 sec
- Think time: 10 sec

The configuration of the tests involved the use of a single test controller (where VSTS would execute and control the tests) that used four 32-bit agents, which went against the SSRS servers. The SSRS servers all used the same hardware with 8-GB RAM with four cores.

To define the report query workflow, they used a predefined workload where 75% was simple and 25% was complex using a randomized selection of report parameters. An example of a workflow is:

- Step 1
  - Open list of reports
  - Think time
- Step 2
  - Open report by using default parameters

    - Report renders
    - Think time
- Step 3
    - Change parameter
    - Report renders
    - Think time

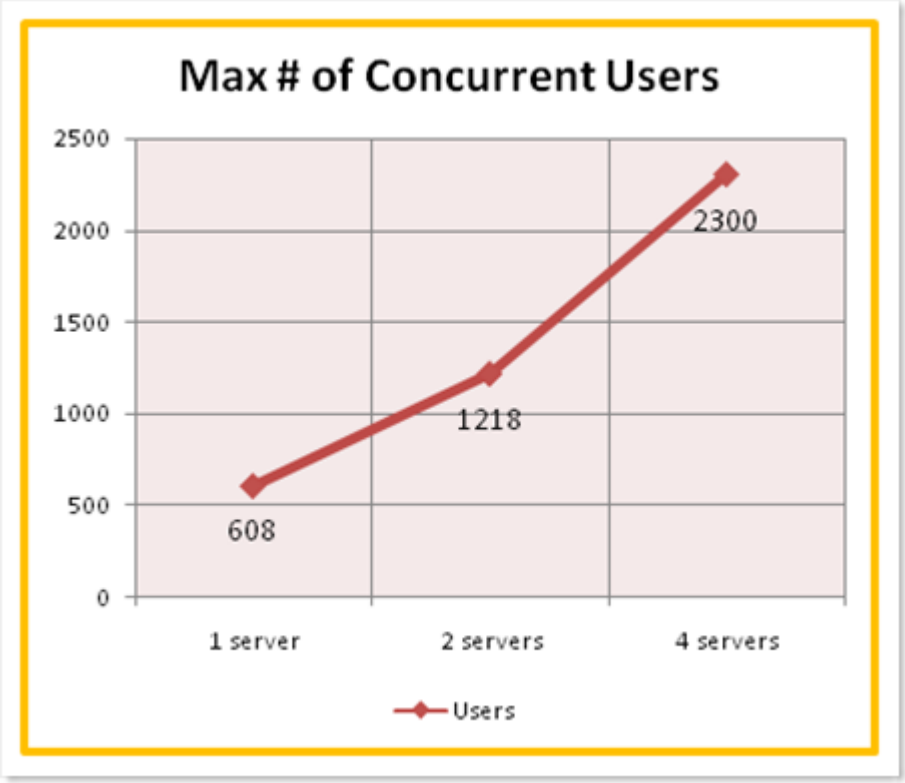


Figure 3: *VSTS workload to determine number of servers to maximum number of concurrent users for sustained time period (>= 15min) on the SSRS server*

For this specific customer's tests (i.e., their workload and their data), you can see from Figure 3 that one SSRS server could handle a maximum of 608 concurrent users for a sustained time period (>=15 minutes). By scaling out the SSRS servers, they also found that they could handle 1,218 concurrent users with two servers and approximately 2,300 concurrent users with four servers.

From this specific test scenario (four servers with this specific workload and tests), they also noted that with >2,000 concurrent users there was a higher rate of failed tests (SSRS queries executed with no results or errors returned). But at 1,800 concurrent users (the expected maximum workload), the rate of failed tests was at an acceptable level. If they wanted to handle a larger number of concurrent users, they would only need to scale out the number of SSRS servers further. Also of interest in their environment was that doubling the memory and CPU of their hardware (increasing to 16-GB RAM and eight cores) resulted in only a 1/3 increase in load capacity. Note, SQL Server 2008 Reporting Services should make better use of resources when adding more cores and memory. Therefore, it made sense for them to

scale out (by increasing the number of servers) instead of scaling up (by adding more memory and CPUs to the servers) to handle higher workloads.

Please note, the results of these tests might not apply to your environment. These test results are specific for this customer scenario according to their workload, hardware, and scenarios. Nevertheless, you can see the importance of doing performance testing within your Reporting Services environment so you can more accurately predict the load it can handle.

Next Notes

As noted in the above section, this is the first technical note of the *Building and Deploying Large Scale SQL Server Reporting Services Environments Technical Note Series*. While this technical note provides the high level architecture, we also provide some additional guidance:

- Report Catalog Best Practices: Provides guidance and best practices on the report server catalogs—the underlying databases that provide metadata (parameters, snapshots, history, etc.) used by Reporting Services to provide your reports.

- Reporting Services Scale-Out Deployment Best Practices: Provides guidance and best practices on deployment details for scaling out your Reporting Services environment including configurations and the use of File System snapshots.

- SSRS Performance Optimization Configurations: Provides guidance and best practices on using specific Reporting Services features and configurations to optimize the performance of your Reporting Services environment.

SQL Server Reporting Services Disaster Recovery Case Study

Introduction

There are many ways to perform disaster recovery with Microsoft SQL Server Reporting Services (SSRS). Based upon customer experience and internal testing, this technical note provides guidance around best practices to design and manage robust end-to-end disaster recovery (DR). This DR method will involve both automatic and manual failover in the form of content switches, SQL Server failover clustering, and database mirroring. This case study focuses on the lessons learned from CareGroup Healthcare System.

Environment Setup

Figure 1 shows a basic disaster recovery scenario. The left side (in green) represents the **primary data center**, while the right side (in orange) represents the **disaster recovery (DR) site**. In general the two sites should be identical or at least very similar to each other so that the DR site will be able to handle the load of the primary data center; all of this should be transparent to the users.

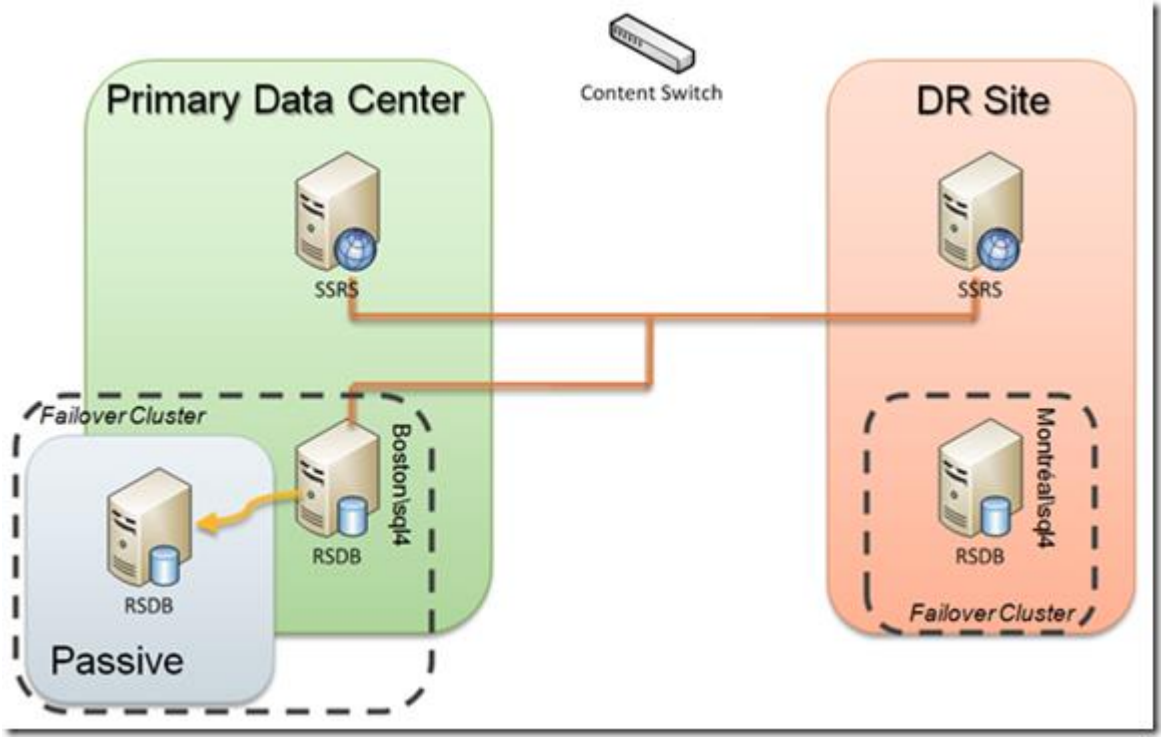


Figure 1: Primary data center and the disaster recovery site

Primary Data Center

The primary data center contains the Reporting Services Web service servers (there is only one for the purpose of this diagram); the report catalogs are placed on a separate server (RSDB). For more information about how to design a scale out Reporting Services environment, see the Reporting Services Scale Out Architecture article. As a general rule, you should set up the report catalogs (RSDB) in a failover cluster, because the report catalogs are central to the Reporting Services environment. For more information about configuring and optimizing report catalogs, see Report Server Catalog Best Practices.

Disaster Recovery Site

The disaster recovery site should closely duplicate the primary data center Reporting Services environment. It should be located in another geographic location such that if the primary data center experiences a resource disruption (for example, in connectivity or power), all traffic can be redirected to the alternate site with minimal disruption to service. In noncritical scenarios, you can utilize fewer resources (for example, less powerful servers, fewer servers) but for mission-critical systems, we highly recommend a server-for-server duplication of resources.

Network Setup

To ensure connectivity from the clients to the primary data center and the disaster recovery site, a common technique is to use a content switch to load-balance traffic within the individual sites as well as between the global sites. In the case of CareGroup Healthcare, a Cisco GSS is used as the content switch. As well, there is direct fiber network connectivity between the primary data center and the disaster recovery site to ensure minimal latencies for any communication between the two centers. If the primary site goes down for any reason, the content switch transparently redirects all client traffic to the disaster recovery set of Reporting Services servers. If the content switch is unavailable, the IP address can be changed at the DNS level. This latter change is a manual switch with a slightly longer network outage, which is due to the DNS cache clearing the old IP address and pointing to the new one.

Database Setup

For the CareGroup Healthcare environment, the primary RSDB instance within the primary data center is named **Boston\sql4** and is a SQL Server cluster using SQL Server 2008 Enterprise, with the content switch pointing to the alias **sql4**. An active/passive failover cluster is used; this allows other active database instances to be located on the server in the passive node. This technique works well if the passive node is not overutilized such that it can handle the Reporting Services workload if the Reporting Services active node goes offline. It may make sense to go to an active/active cluster if there is enough traffic, concurrent users, and/or resources, though the discussion of the tradeoffs between idle machines and high availability scenarios is outside the scope of this technical note. Each of these clustering techniques has unique advantages; ultimately, it is important to ensure that the report catalogs are consistently running with minimal downtime. For more information about how to best configure a SQL Server clustering environment, see SQL Server 2008 Failover Clustering.

To ensure that the RSDB within the disaster recovery site is up-to-date, a common technique is to use asynchronous database mirroring. For the CareGroup Healthcare environment, the mirrored RSDB instance is named **Montréal\sql4** and is a SQL Server cluster using SQL Server 2008 Enterprise. Note that while the domain names are different (Montréal vs. Boston in this case), the SQL Server clusters for the primary data center and the disaster recovery site have the same name (more on this later). As noted in Report Server Catalog Best Practices, the key database to keep in sync is the report server (RSDB) database, because it stores all of the report metadata, including report definitions, report history and snapshots, and scheduling information. All Reporting Services operations must connect to this database to access its metadata to perform their functions. The reason asynchronous database mirroring is commonly chosen is that asynchronous mirroring has minimal or no impact on response time performance. In addition, asynchronous mirroring works well because RSDB metadata is not frequently updated. For more information about database mirroring, see SQL Server Replication: Providing High Availability using Database Mirroring

The following table describes an example endpoint setup for these database mirrors.

Mirror Setup	Boston\sql4 (Primary)	Montréal\sql4 (DR site)
Name	Principal	Partner
Listener Port	50212	50212
Encryption	Yes	Yes
Role	Partner	Partner

Initializing Database Mirror

A relatively easy way to initialize a database mirroring setup is to:

1) Make full and transaction log backups of the Reporting Services databases on the principal server.

2) Copy the backups over to the disaster recovery site, restoring each Reporting Services database in no-recovery mode.

3) Set up the failover partner on the mirror (that is, the DR site) before you set up the failover partner on the principal server.

Failover Scenarios

There are three main disaster recovery scenarios that can occur within this Reporting Services disaster recovery environment:

· Reporting Services servers within the primary data center go offline.

· The active server for the RSDB in the primary data center failover cluster goes offline.

· The entire cluster for the RSDB in the primary data center goes offline.

Please note that this note does not explicitly cover the scenarios concerning the Reporting Source databases. While most of the information here is applicable to these databases (such as data marts, operational data stores, transactional systems, and data warehouses), they have their own disaster recovery requirements and procedures that should be separate from the reporting infrastructure.

Reporting Services Servers within the Primary Data Center Go Offline

As noted in Figure 2, both Reporting Services servers point to the same cluster of databases. In this case, they both point to the active/passive database cluster **Boston\sql4**.

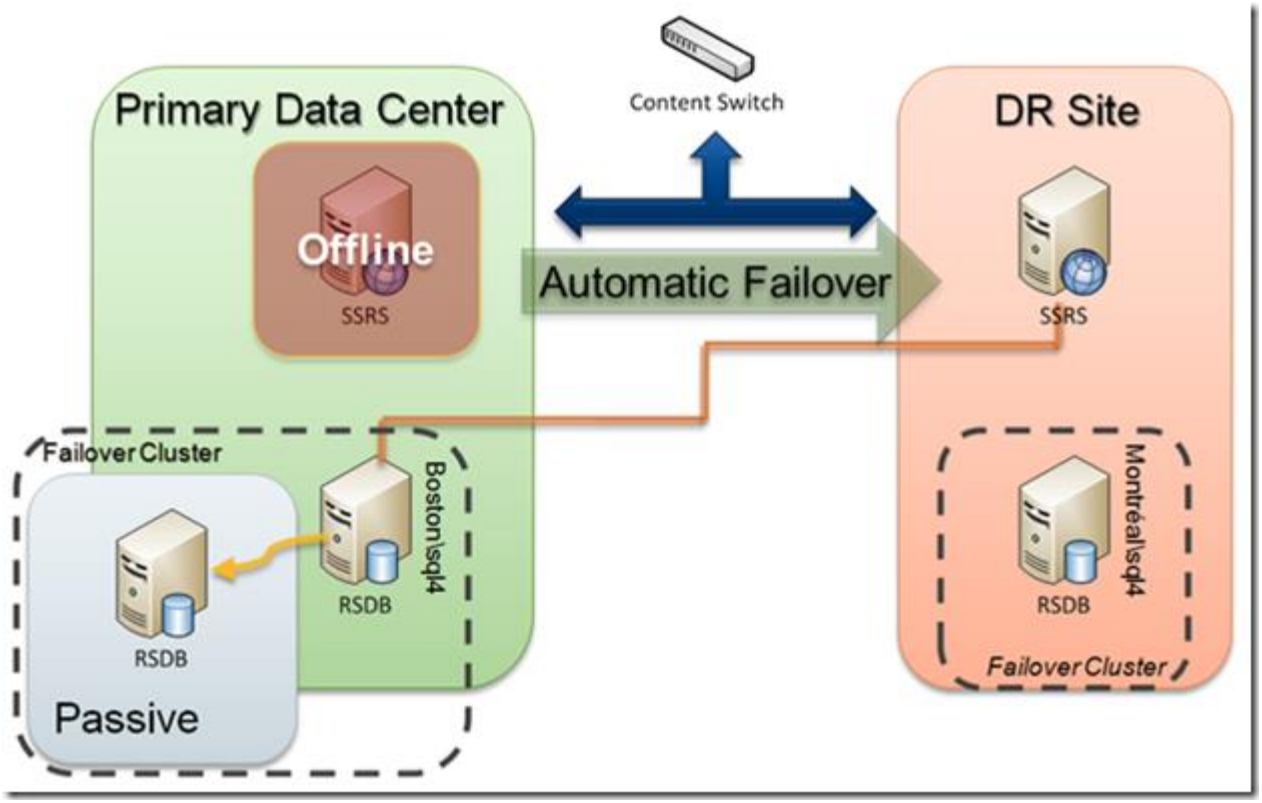


Figure 2: Reporting Services servers within the Primary Data Center go offline

In this scenario where the Reporting Services servers within the primary data center go offline, the hardware content switch will detect a network connectivity issue and **automatically fail over** to the disaster recovery site. By default, the Reporting Services farm connects to the report catalog in the primary data center because it has the most up-to-date information (provided that the network is fast and responsive). Therefore in most cases, the client connectivity to the Reporting Services environment in the DR site is minimally impacted (clients might need to be closed and restarted to establish a new network connection, or they could receive a broken link/page in the browser and need to refresh or restart their report execution).

Active RSDB Server in the Primary Data Center Failover Cluster Goes Offline

If the active server in the primary data center failover cluster goes offline, the active/passive setup of the SQL Server failover cluster will kick in. The active server will **automatically fail over** to the passive instance of the node with little to no service interruption. The report server may report that it cannot connect to the report server database, but after the passive node comes online, it will automatically re-establish the connection. All report server connectivity will continue as previously performed except that now the passive instance has become the active instance.

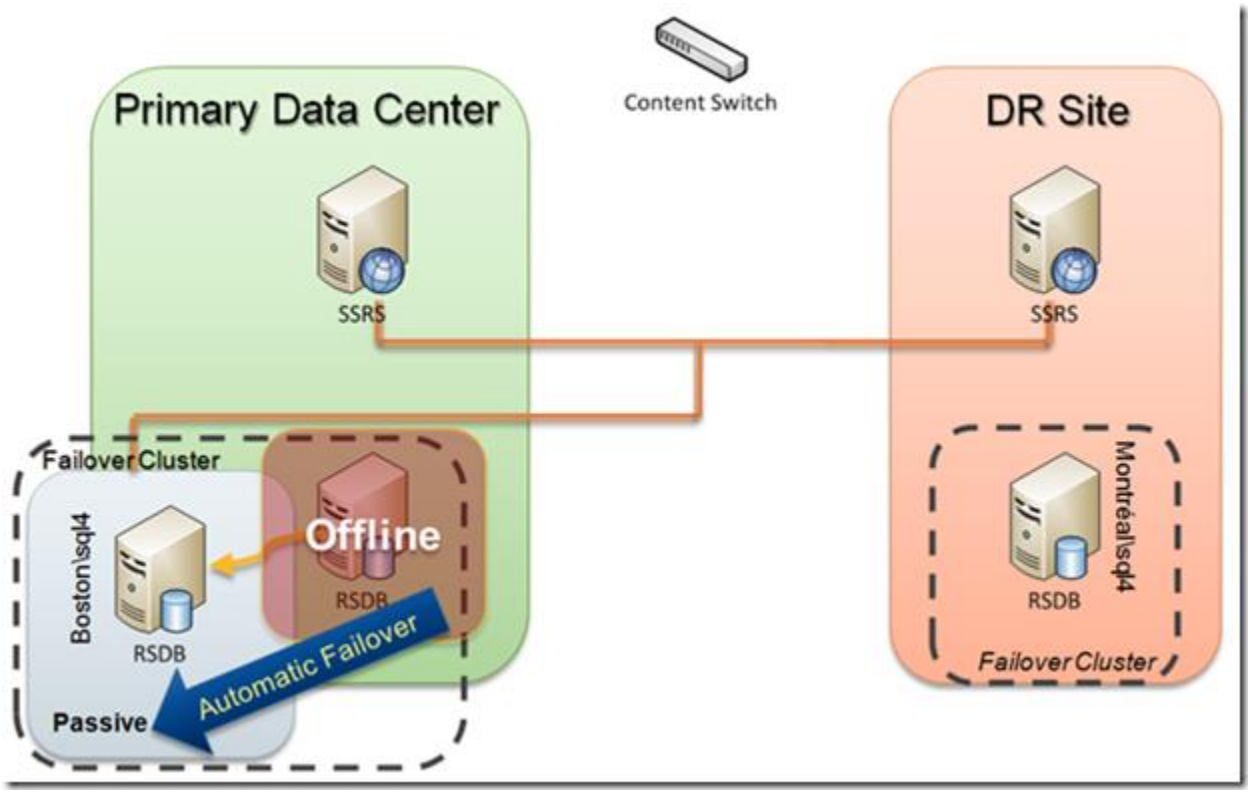


Figure 3: Active cluster for RSDB in the primary data center goes offline

Entire RSDB Cluster in the Primary Data Center Goes Offline

If the primary data center RSDB SQL Server cluster goes offline, the disaster recovery solution is to manually fail over to the RSDB on the disaster recovery site to continue serving up reports. But note that this is a **manual failover** process, which requires manual configuration of the Reporting Services servers in the Reporting Services farm to point to the RSDB instance within the disaster recovery site. It is important to be aware that, with asynchronous mirroring, manual failover by definition forces the service to allow data loss. With Reporting Services, however, the data loss should be minimal because report metadata is not frequently updated and snapshots can be re-created. By configuring the primary data center and disaster recovery site RSDBs with identical instance names (recall that both instances are named **sql4** but that they are in different domains), the manual failover is simplified because the Reporting Services servers will connect to an identically named instance of the RSDB (different domain, but same name).

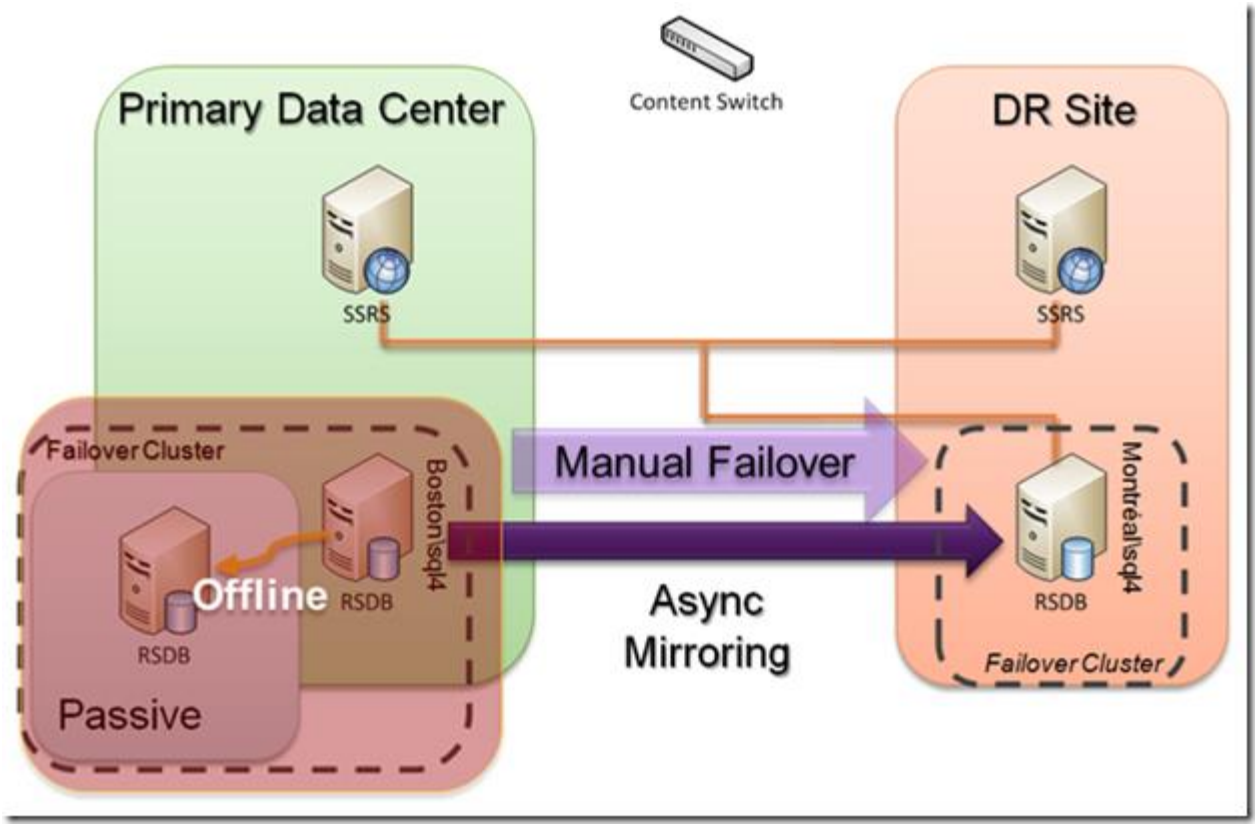


Figure 4: Entire cluster for RSDB in the primary data center goes offline

Primary Data Center Outage Scenarios

If the entire primary data center goes offline for either a planned or unplanned outage, the content switch will automatically fail over to the Reporting Services farm in the disaster recovery site, but you will need to perform a manual failover to ensure that the Reporting Services servers can connect to the copy of the RSDB in the disaster recovery site.

Planned Outages

For planned outages:

1) The content switch will perform the task of suspending the primary IP addresses and activating the DR site IP addresses so that all connections will be redirected to the disaster recovery site.

2) You will need to execute a script on the primary database to manually switch to the mirrored database. After the partner database becomes the primary, the same script can be re-executed to switch back to the original primary database. Here is an example:

```
-- Script to manually fail over to the mirror database

-- Example: switch from Boston\sql4 to Montreal\sql4

USE [master]

GO

-- Database ReportingServer

ALTER DATABASE ReportingServer SET PARTNER FAILOVER

GO
```

Unplanned Outages

For unplanned outages:

1) The content switch will perform the task of suspending the primary IP addresses and activating the DR site IP addresses so that all connections will be redirected to the disaster recovery site.

2) You will need to run a manual failover script to force the service to switch with possible data loss. The script needs to run on the mirrored server in the case of loss of the principal server. Here is an example:

```
-- Script to force database fail over of Boston\sql4 to Montreal\sql4

-- in case of SQL Server instance failure

USE [master]

GO

-- Database ReportingServer
```

```
ALTER DATABASE ReportingServer

SET PARTNER FORCE_SERVICE_ALLOW_DATA_LOSS

GO
```

After this script is run, the mirror server is the principal server and the mirror itself is suspended.

For more information about Reporting Services configuration for the database connection, see Configuring a Report Server Database Connection.

Break/Rebuild the Mirror

After the system goes down, it is undesirable to have the active system push data to the passive system (it cannot anyways). So it is important to break the mirror with the primary data center server and then re-establish it after the primary data center (in this example, Boston) server is back online. Note that the disaster recovery (in this example, Montréal) server is the one that is now active, so you will need to manually transfer the data from the DR site server back to the primary data center server (in this example, Montréal to Boston). To fully break the mirror, execute the following on the primary server:

```
ALTER DATABASE ReportingServer SET PARTNER OFF

RESTORE DATABASE ReportingServer WITH RECOVERY
```

To rebuild the mirror, follow the original steps in the *Initializing Database Mirror* section above.

Detail Logistics

It is important to understand that the Reporting Services report jobs, schedules, and subscriptions will not be mirrored between the RSDB from the primary data center and the disaster recovery site. This occurs because SQL Server Agent jobs are stored in the **msdb** database, which is not mirrored. A common method to resolve this issue is to script all of these Reporting Services jobs in the primary site and then use these scripts to re-create them on the disaster recovery site (with the jobs disabled); activate these jobs only if the system fails over. In the case of Reporting Services database *migration*, it is not necessary to perform this operation, as noted in How to: Migrate a Reporting Services Installation.

Note that logins created on the primary site may not be replicated to the disaster recovery site. To work around this problem, we recommend storing all of these logins into their own database and mirroring this database to the DR site as well. Then create a script to generate the logins within the DR site to re-establish the logins. For more information, see How to transfer logins and passwords between instances of SQL Server.

Conclusion

With these guidelines, you can prepare your Reporting Services environment for disaster recovery scenarios. The key here is that you will need to use a combination of hardware components, network connectivity, SQL Server database mirroring, and architecture to create a disaster recovery infrastructure to support an enterprise SQL Server Reporting Services environment. As noted, this case study has used specific techniques such as active/passive clustering and asynchronous database mirroring to support their disaster recovery requirements. Your situation may be different, but this technical note should provide you with the basic framework for Reporting Services disaster recovery. The hope is that you will never need to use it, but at least now you are ready in case you do.

Scaling Up Reporting Services 2008 vs. Reporting Services 2005: Lessons Learned

Introduction

Microsoft SQL Server 2008 Reporting Services contains many improvements designed to improve performance and capacity. The rearchitecture of the Reporting Services 2008 server as a single server with embedded HTTP capabilities enables Reporting Services 2008 to exercise more control over resource management (thread, memory, and state management) to improve performance and scalability, as well as configurability. The new report engine architecture includes a new on-demand processing model designed to ensure that sufficient memory is always available to execute very large reports and heavy workloads from a large number of concurrent users (by paging and releasing memory in response to memory pressure). Using an early RC0 build (which was not an optimized build), we tested the performance of SQL Server 2008 Reporting Services versus SQL Server 2005 Reporting Services, SP2 (which was optimized) on several scale-up hardware platforms. This technical note discusses our tests and our learnings.

Executive Summary

Reporting Services 2008 was able to respond to 3–4 times the total number of users and their requests on the same hardware without HTTP 503 Service Is Unavailable errors compared with Reporting Services 2005, regardless of the type of renderer. In stark contrast, Reporting Services 2005 generated excessive HTTP 503 Service Is Unavailable errors as the number of users and their requests increased, regardless of the report renderer.

Our tests clearly show that the new memory management architecture of the report server enables Reporting Services 2008 to scale very well, particularly on the new four-processor, quad-core processors. With our test workload, Reporting Services 2008 consistently outperformed SQL Server 2005 with the PDF and XLS renderers on the four-processor, quad-core hardware platform (16 cores) both in terms of response time and in terms of total throughput. Furthermore, with these renderers on this hardware platform, Reporting Services dramatically outperformed other hardware platforms regardless of Reporting Services version, responding to 3–5 times the number of requests than when running on either of the other hardware platforms. As a result, we recommend that you scale up to four-processor, quad-core servers for performance and scale out to a two-node deployment for high availability. Thereafter, as demand for more capacity occurs, add more four-processor, quad-core servers.

Finally, with all renderers and with all hardware platforms using our test workload, the performance bottlenecks were the processor on the front-end server and the disk subsystem on the data source with Reporting Services 2008, whereas the Reporting Services front-end Web service was the performance bottleneck with Reporting Services 2005.

Note: This technical note assumes that you are familiar with the following articles:

· Planning for Scalability and Performance with Reporting Services

· Reporting Services Scale-Out Architecture

Test Environment

We used a standard remote catalog environment for all testing. Figure 1 illustrates the test environment for our scale-up performance tests. For our Reporting Services 2008 tests, SQL Server 2008 components were installed on all servers and for our Reporting Services 2005 tests, SQL Server 2005 components were installed on all servers. Windows Server 2003 was used for all our tests.

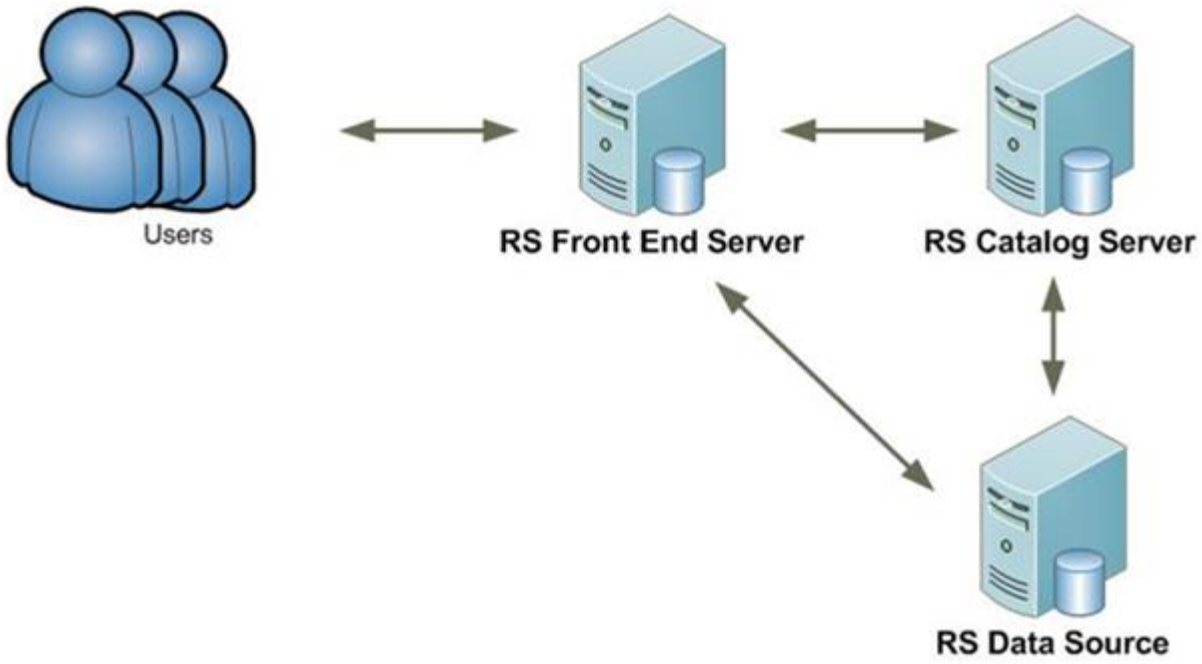


Figure 1: Reporting Services Scale-Up Architecture with Remote Catalog and Remote Data Source

Reporting Services "Front-End" Servers

For the Reporting Services front-end server, we tested Reporting Services 2008 and Reporting Services 2005 on the following three hardware platforms:

· Hardware Configuration #1 (2x2): One HP Proliant BL 460c G1 computer with two x64 dual core processors and 16 GB RAM

· Hardware Configuration #2 (4x2): One Dell 2970 computer with four x64 dual core processors and 16 GB RAM

· Hardware Configuration #3 (4x4): One HP Proliant BL 460c G5 computer with four x64 quad core processors and 16 GB RAM

Reporting Server Catalog Servers

For the Report Server catalog server, we used one HP Proliant BL 460c G1 computer with two x64 dual core processors and 8 GB RAM for all tests. The Report Server catalog server was connected to an EMC Clarion CX3 Model 40 SAN.

Reporting Data Source Server

For the Report Server data source server, we used one HP Proliant BL 460c G5 computer with four x64 quad core processors and 16 GB RAM for all tests. The Report Server data source server was connected to an EMC Clarion CX3 Model 40 SAN.

Important: We used the same configuration for the Report Server catalog server and the Report Server data source server for all tests. Our testing with Reporting Services 2008 revealed that disk queuing was a performance bottleneck and that even closer attention must be paid to disk configuration for optimum performance with Reporting Services 2008 than with Reporting Services 2005. This close attention includes the disk configuration for the source database(s) as well as the **ReportServer** and **ReportServerTempDB** databases. For more information about optimizing storage for best performance, see Storage Top 10 Best Practices and Predeployment I/O Best Practices.

Test Harness

For our performance tests, we used Visual Studio Team System 2008 Test Edition (VSTS) to test a report load based on the Adventure Works sample relational database. For our tests, we created 50 separate reports of varying complexity and size and designed for different user personas, such as Executive, Territory Manager, Product Manager, and Sales Manager. The report load consisted of:

· A mix of execution modes with 95% live and 5% snapshot

· A mix of data region types with 80% table and 20% chart

· A mix of query data sizes with 60% small, 20% medium, and 20% large

All tests were executed using the same mix of these reports. These reports were executed with a think time of 20 seconds between the execution of each report and we ramped up to a specified number of users. While we conducted our tests with different numbers of users, this technical note focuses primarily on our 5,000-user test as the most representative of the differences and similarities between Reporting Server 2008 and Reporting Services 2005 (Ramp Up Test). At the end of this technical note, we discuss the results of a push test to determine how far we can push Reporting Services 2008 compared to SQL Server 2005, and to observe where the bottleneck occurs with each version of Reporting Services (push test). We executed test runs for the following report renderers:

· HTML

· PDF

· XLS

· Mix (approximately 33% of each renderer)

Note: We modified the **MaxActiveReqForOneUser** configuration setting for this test to enable us to test with this number of users.

Test Results

The following sections discuss the results that we observed with each report renderer on each hardware platform.

Ramp Up Test with a Mix of Report Renderers

With a mix of report renderers, Reporting Services 2008 outperformed Reporting Services 2005 on the 4x4 and 4x2 hardware platforms, with the 4x4 hardware platform substantially outperforming the 4x2 and 2x2 hardware platforms regardless of the Reporting Services version. Figures 2, 3 and 4 graphically display the errors/sec, average response time, and total requests counters for these hardware platforms for each version of Reporting Services.

Errors/Sec

When we analyzed the errors/sec counter values from our ramp up test with a mix of report renderers, we discovered that excessive HTTP 503 Service Is Unavailable errors occurred during the Reporting Services 2005 tests on all hardware platforms at approximately 4,500 users (see Figure 2). During the Reporting Services 2008 tests on the three hardware platforms with the mix of report renderers, we observed that no such hard limit was hit. We observed that this same limit was hit with all report renderers during our Reporting Services 2005 tests and was not hit with any of our Reporting Services 2008 tests. Reporting Services 2008 manages memory under heavy load much better than Reporting Services 2005, and it avoids overloading the Reporting Services front-end server. The excessive HTTP 503 errors with Reporting Services 2005 indicate that the heavy load coming from the clients overloaded the Reporting Services front-end server to the point where the server is simply starved for resources, and reveal that memory management under load is clearly a problem with Reporting Services 2005.

Note: The Errors/sec counter value includes HTTP errors (HTTP 500 and HTTP 503 errors). HTTP 503 errors with Reporting Services tell you that the Web service itself is unable to respond (frequently due to insufficient memory). HTTP 500 errors with Reporting Services tell you that the Web service was unable to complete the request for some unknown reason (generally due hardware bottlenecks on the front-end server, the network or the back-end server). During the push test discussion (later in this technical note), we analyze in more detail the bottlenecks we saw with Reporting Services 2008 that resulted in HTTP 500 errors.

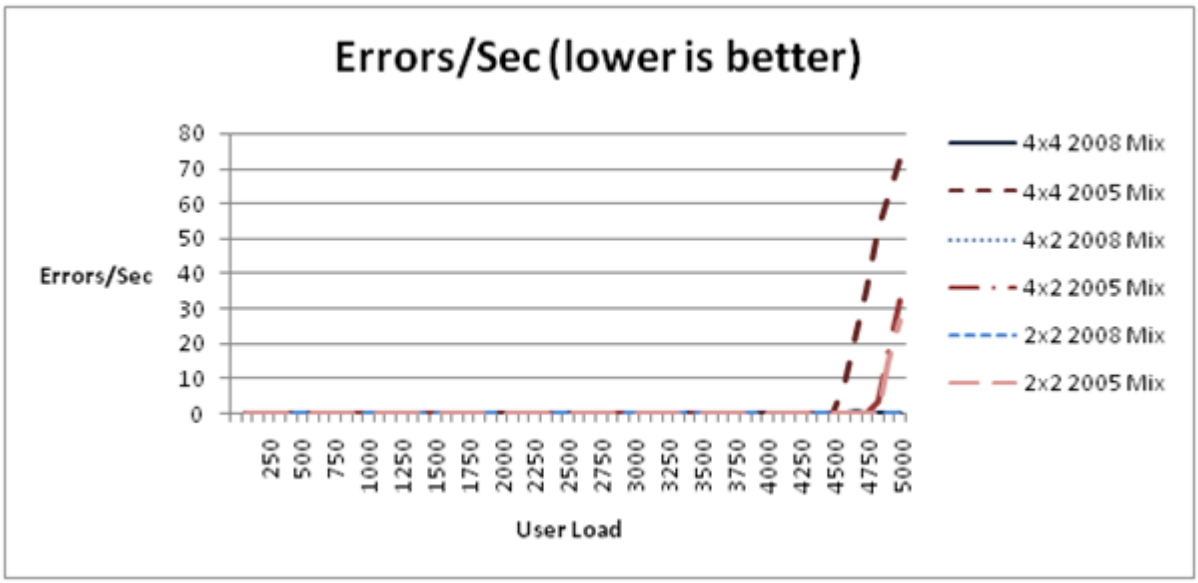


Figure 2: Mix of Renderers—Errors/Sec

Average Response Time

When we analyzed the average response time counter values from our ramp up test with a mix of report renderers, we discovered that the average response time with Reporting Services 2008 was better than with Reporting Services 2005 on the 4x4 and 4x2 hardware platforms (see Figure 3). We discovered no significant difference in the average response time on the 2x2 hardware platform. Furthermore, we discovered that the 4x4 hardware platform substantially outperformed either of the other two hardware platforms with either Reporting Services 2008 or Reporting Services 2005. Finally, we also noticed that the improvement of the average response time of Reporting Services 2008 on the 4x2 hardware platform over Reporting Services 2005 became more significant as the number of users submitting requests increased.

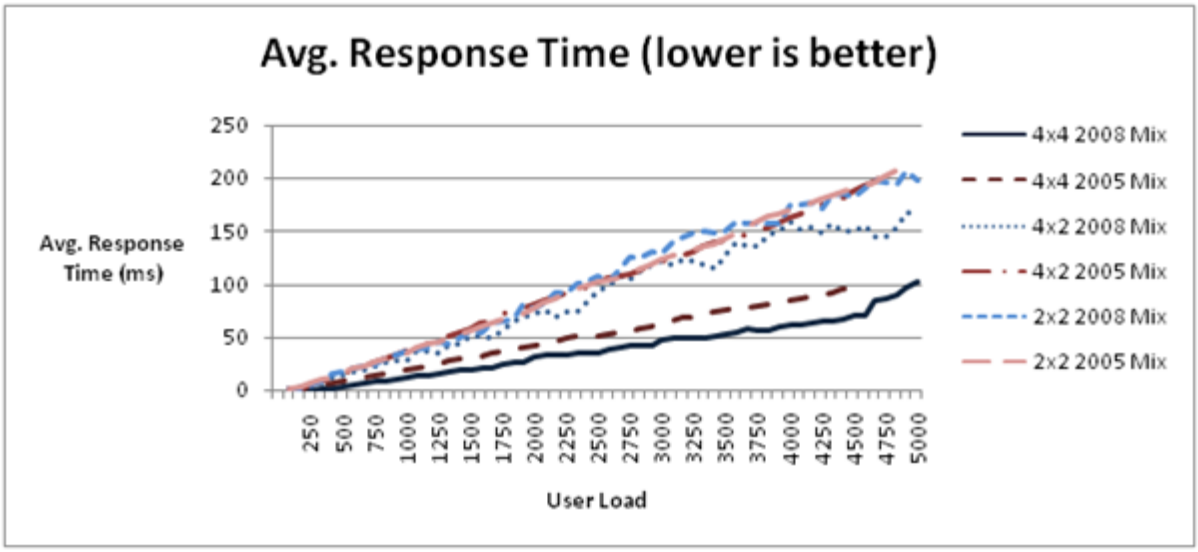


Figure 3: Mix of Renderers—Average Response

Note: The chart lines for the Reporting Services 2005 tests in Figure 3 only include the response times for the user load before the number of errors/sec climbed to the point where they completely skewed the overall response time numbers.

Total Requests

When we analyzed the total requests counter values from our ramp up test with a mix of report renderers, we discovered that not only did Reporting Services 2008 handle a greater number of users submitting reports without generating excessive timeout or service unavailable errors and render those reports with a better average response time, Reporting Services 2008 also sustained a higher throughput than Reporting Services 2005 on the 4x4 and 4x2 hardware platforms (see Figure 4). This increased throughput was most substantial on the 4x4 hardware platform, on which the throughput for the duration of the test was approximately 40% better with Reporting Services 2008 than with Reporting Services 2005. The new memory management capabilities of Reporting Services 2008 enabled the superior processing capability of the 4x4 hardware platform to perform substantially better than Reporting Services 2005 on the same platform.

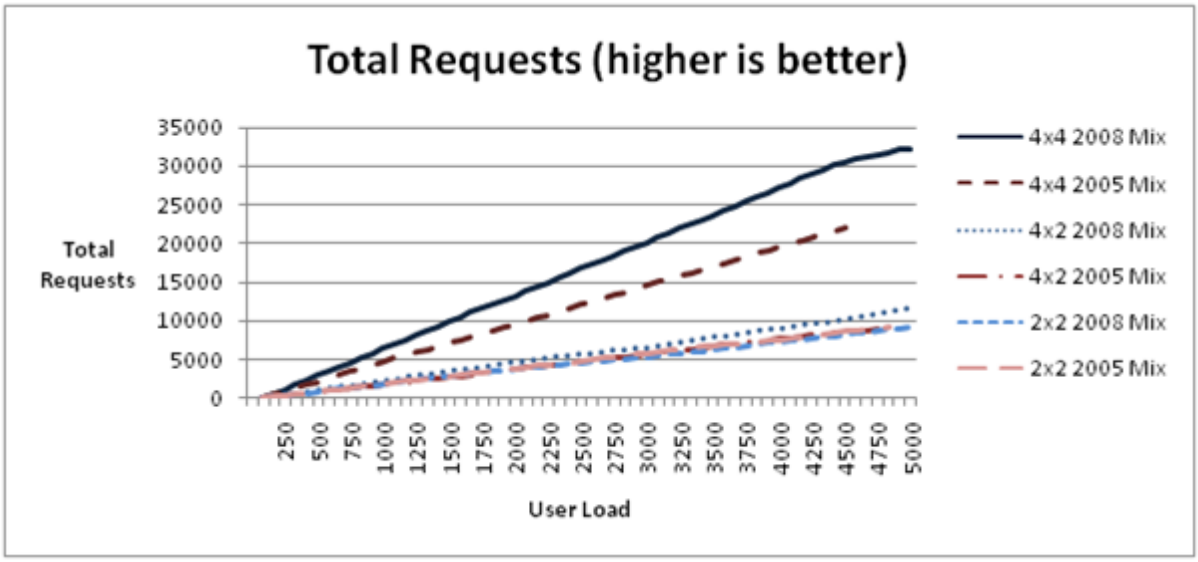


Figure 4: Mix of Renderers—Total Requests

Note: The chart lines for the Reporting Services 2005 tests in Figure 5 only include the total requests for the user load before the number of errors/sec climbed to the point where they completely skewed the overall total request numbers. On all hardware platforms, Reporting Services 2008 kept on responding to users after Reporting Services 2005 was unable to continue due to HTTP 503 Service Is Unavailable errors.

Ramp Up PDF Report Renderer Test

With the PDF report renderer, Reporting Services 2008 outperformed Reporting Services 2005 on the 4x4 and 4x2 hardware platforms, with the 4x4 hardware platform substantially outperforming the 4x2 and 2x2 hardware platforms regardless of the Reporting Services. Figures 5 and 6 graphically display the average response time and total requests counters for each hardware platform for each version of Reporting Services. With the PDF report renderer, until the HTTP 503 error threshold discussed previously was hit with Reporting Services 2005, the processor was the performance bottleneck.

Average Response Time

When we analyzed the average response time counter values from our ramp up test with the PDF report renderer, we discovered that the average response time with Reporting Services 2008 was better than with Reporting Services 2005 on the 4x4 and 4x2 hardware platforms (see Figure 5). Furthermore, we discovered that the 4x4 hardware platform substantially outperformed either of the other two hardware platforms with either Reporting Services 2008 or Reporting Services 2005.

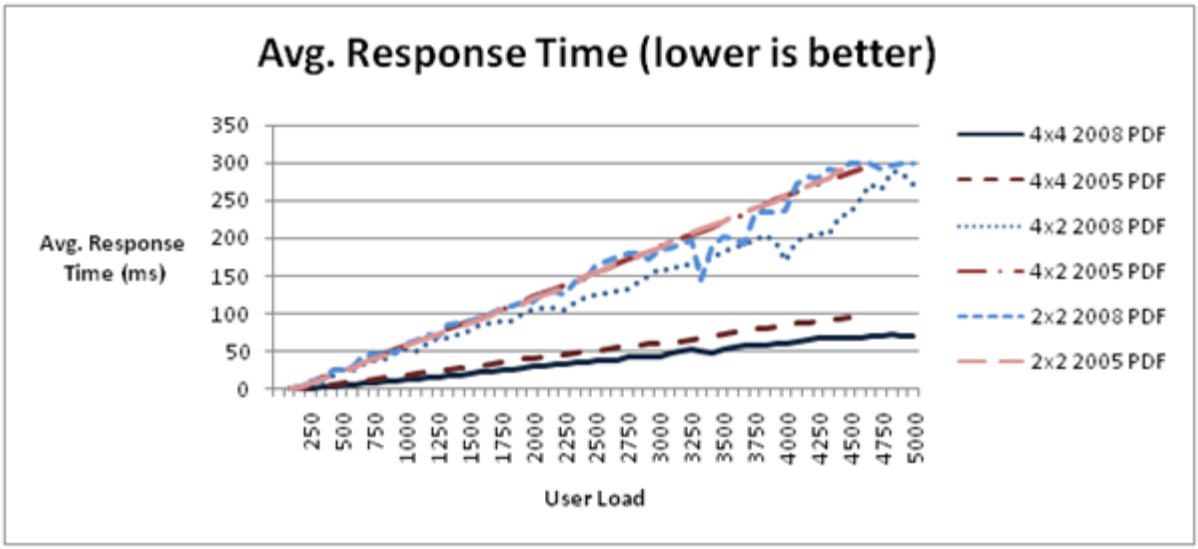


Figure 5: PDF Renderer—Average Response

Note: The chart lines for the Reporting Services 2005 tests in Figure 5 only include the response times for the user load before the number of errors/sec climbed to the point where they completely skewed the overall response time numbers. On all hardware platforms, Reporting Services 2008 kept on responding to users after Reporting Services 2005 was unable to continue due to HTTP 503 Service Is Unavailable errors.

Total Requests

When we analyzed the total requests counter values from our ramp up test with the PDF report renderer, we discovered that Reporting Services 2008 responded to a significantly greater number of requests compared to Reporting Services 2005 over the same amount of time only on the 4x4 hardware platform (see Figure 6).

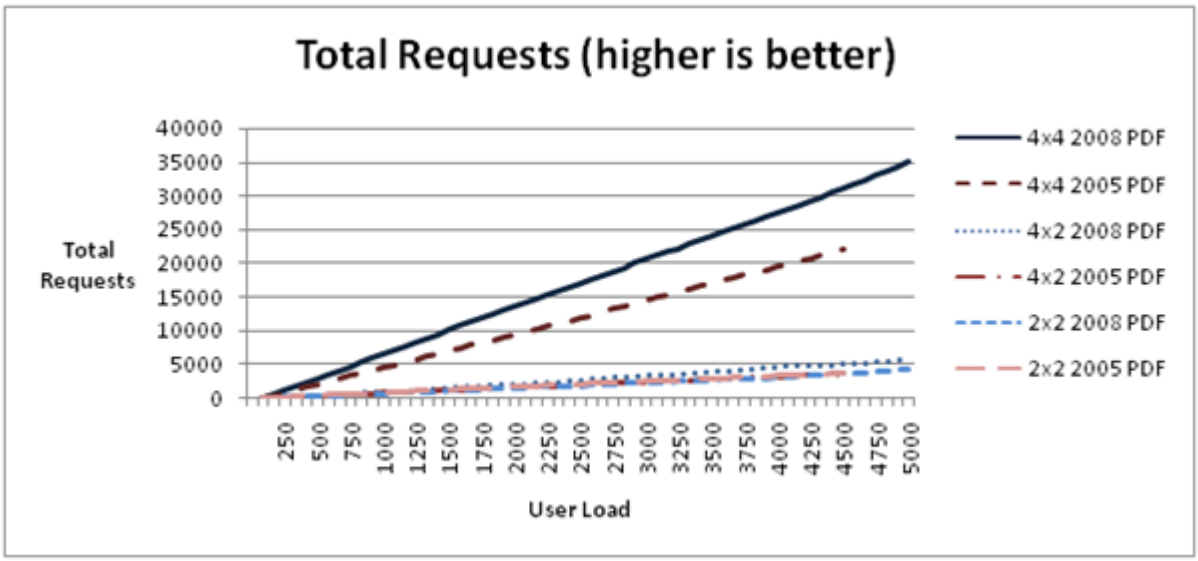


Figure 6: PDF Renderer—Total Requests

Note: The chart lines for the Reporting Services 2005 tests in Figure 6 only include the total requests for the user load before the number of errors/sec climbed to the point where they completely skewed the overall total request numbers. On all hardware platforms, Reporting Services 2008 kept on responding to users after Reporting Services 2005 was unable to continue due to HTTP 503 Service Is Unavailable errors.

Ramp Up XLS Report Renderer Test

With the XLS report renderer, Reporting Services 2008 outperformed Reporting Services 2005 primarily on the 4x4 hardware platform, with the 4x4 hardware platform substantially outperforming the 4x2 and 2x2 hardware platforms regardless of the Reporting Services version. Figures 7 and 8 graphically display the average response time and total requests counters for each hardware platform for each version of Reporting Services. With the XLS report renderer, until the HTTP 503 error threshold discussed previously was hit with Reporting Services 2005, the processor was the performance bottleneck.

Average Response Time

When we analyzed the average response time counter values from our ramp up test with the XLS renderer, we discovered that the average response time with Reporting Services 2008 was better than with Reporting Services 2005 on the 4x4 and 2x2 hardware platforms (see Figure 7). Furthermore, we discovered that the 4x4 hardware platform substantially outperformed either of the other two hardware platforms with either Reporting Services 2008 or Reporting Services 2005. Finally, we also noticed that (contrary to our tests with the mix of renderers and with the PDF report renderer) the average response time of Reporting Services 2008 and Reporting Services 2005 on the 2x2 hardware platform was somewhat better than on the 4x2 hardware platform, with Reporting Services 2005 outperforming Reporting Services 2008 on this platform. This performance difference was not expected and we did not have time to investigate precisely why this performance difference was observed.

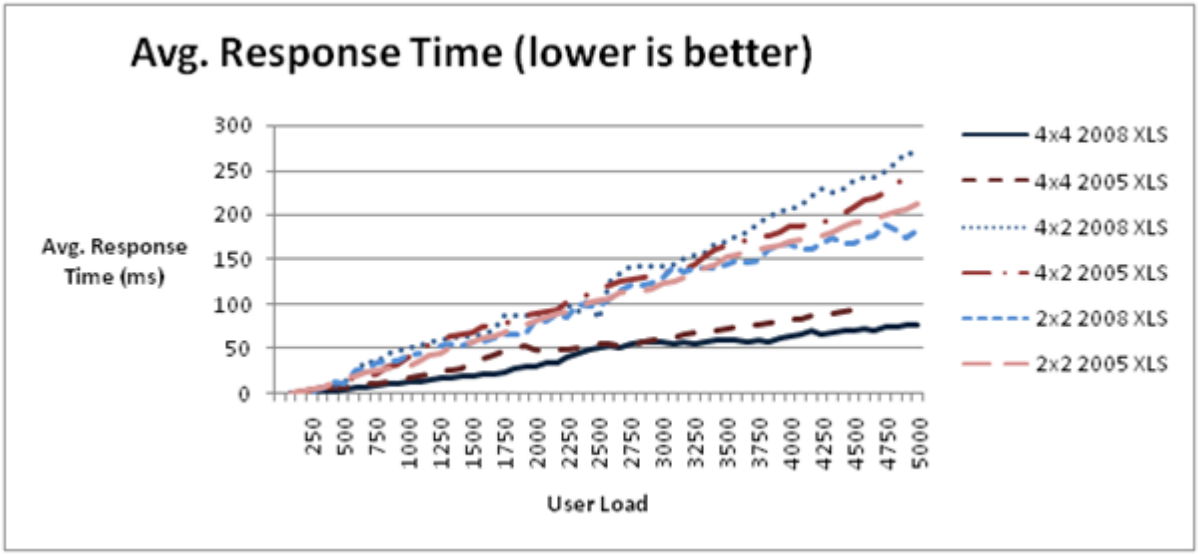


Figure 7: XLS Renderer—Average Response

Note: The chart lines for the Reporting Services 2005 tests in Figure 7 only include the response times for the user load before the number of errors/sec climbed to the point where they completely skewed the overall response time numbers. On the 4x4 hardware platform, Reporting Services 2008 kept on responding to users after Reporting

Services 2005 was unable to continue due to HTTP 503 Service Is Unavailable errors. Notice that the HTTP 503 error threshold with Reporting Services 2005 was only hit on the 4x4 hardware platform; on the other hardware platforms using the XLS report renderer with Reporting Services 2005, the processor was the performance bottleneck.

Total Requests

When we analyzed the total requests counter values from our ramp up test with the XLS renderer, we discovered that Reporting Services 2008 responded to a significantly greater number of requests compared to Reporting Services 2005 over the same amount of time only on the 4x4 hardware platform (see Figure 8). Finally, we noticed that the total requests of Reporting Services 2008 and Reporting Services 2005 on the 2x2 hardware platform were slightly higher than on the 4x2 hardware platform.

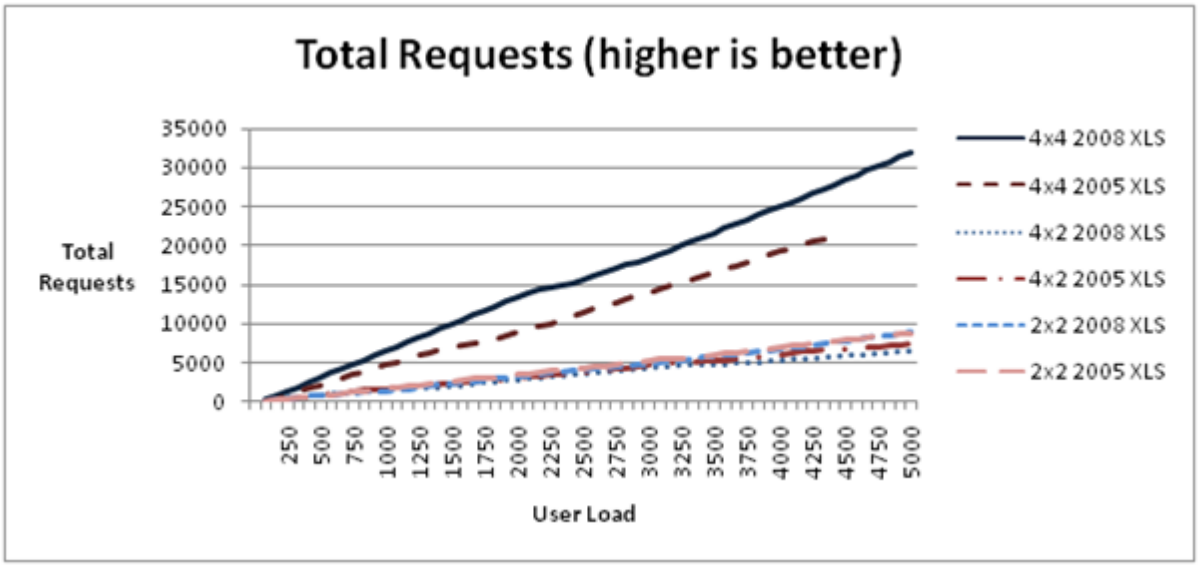


Figure 8: XLS Renderer—Total Requests

Note: The chart lines for the Reporting Services 2005 tests in Figure 8 only include the total requests for the user load before the number of errors/sec climbed to the point where they completely skewed the overall total request numbers. On the 4x4 hardware platform, Reporting Services 2008 kept on responding to users after Reporting Services 2005 was unable to continue due to HTTP 503 Service Is Unavailable errors. Notice that the HTTP 503 error threshold with Reporting Services 2005 was hit only on the 4x4 hardware platform; on the other hardware platforms using the XLS report renderer with Reporting Services 2005, the processor was the performance bottleneck.

Ramp Up HTML Renderer Test

With the HTML report renderer, Reporting Services 2008 did not significantly outperform Reporting Services 2005 on any of the hardware platforms. Furthermore, the 4x4 hardware platform did not substantially outperform either the 4x2 or the 2x2 hardware platforms regardless of the Reporting Services version. Figures 9 and 10 graphically display the average response time and total requests counters for each hardware platform for each version of Reporting Services.

Note: It should be noted again here that our analysis of the performance counters on the Report Server catalog server as well as on the reporting data source server indicated that these tiers were a performance bottleneck with Reporting Services 2008 and were not a performance bottleneck with Reporting Services 2005. Based on our workload and our hardware/software configuration for these tests, Reporting Services 2008 could have performed even better than it did with more tuning and/or with more hardware capability on these tiers than Reporting Services 2005. We do not know, with more capacity on the back-end servers, whether the HTML renderer on Reporting Services 2008 would have outperformed the HTML rendered on Reporting Services 2005. Further testing in this area is needed.

Average Response Time

When we analyzed the average response time counter values from our ramp up test with the HTML renderer, we discovered that Reporting Services 2008 on the 4x2 and the 2x2 hardware platforms had a better average response time than Reporting Services 2005 on the same hardware platform. However, Reporting Services 2008 on the 4x4 hardware platform suffered from degradation in response times as the number of users and the workload increased (see Figure 9). We are not sure why this degradation occurred and the root cause requires more analysis and testing.

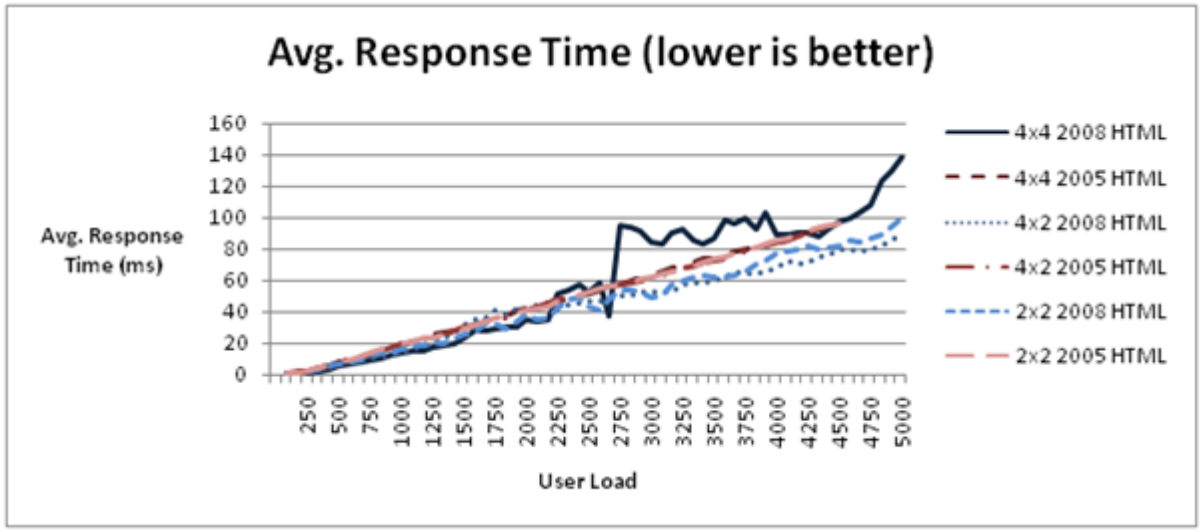


Figure 9: HTML Renderer—Average Response

Note: The chart lines for the Reporting Services 2005 tests in Figure 9 only include the response times for the user load before the number of errors/sec climbed to the point where they completely skewed the overall response time numbers. On all hardware platforms, Reporting Services 2008 kept on responding to users after Reporting Services 2005 was unable to continue due to HTTP 503 Service Is Unavailable errors.

Total Requests

When we analyzed the total requests counter values from our ramp up test with the HTML renderer, we discovered that Reporting Services 2008 did not render a significantly higher number of reports in the same amount of time compared to Reporting Services 2005 (see Figure 10) until after the HTTP 503 error threshold was reached.

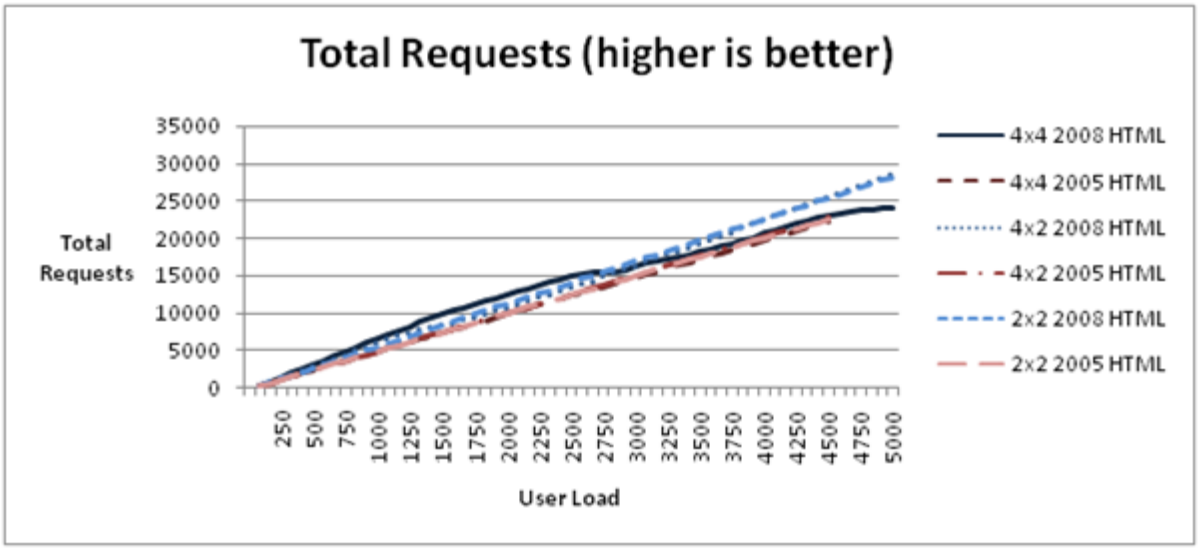


Figure 10: HTML Renderer—Total Requests

Note: The chart lines for the Reporting Services 2005 tests in Figure 10 only include the total requests for the user load before the number of errors/sec climbed to the point where they completely skewed the overall total request numbers. On all hardware platforms, Reporting Services 2008 kept on responding to users after Reporting Services 2005 was unable to continue due to HTTP 503 Service Is Unavailable errors.

Push Test

We also performed some limited testing to test how far we could push the 4x4 hardware platform without significant errors. To accomplish this, we modified our VSTS test to eliminate the 20-second think time. We tested with the mix of report renders and then pushed the 4x4 hardware platform for 25 minutes, ramping up to 10,000 users over the first 10 minutes and then continuing at that number of users for the duration of the push test.

Errors/Sec

When we analyzed the errors/sec counter values from our push test with a mix of report renderers, we discovered that excessive HTTP 503 Service Is Unavailable errors occurred during the Reporting Services 2005 tests on all hardware platforms at 9 minutes into our test at approximately 9,000 users (see Figure 11). During the Reporting Services 2008 push test on the 4x4 hardware platform, we observed that no such hard limit was hit.

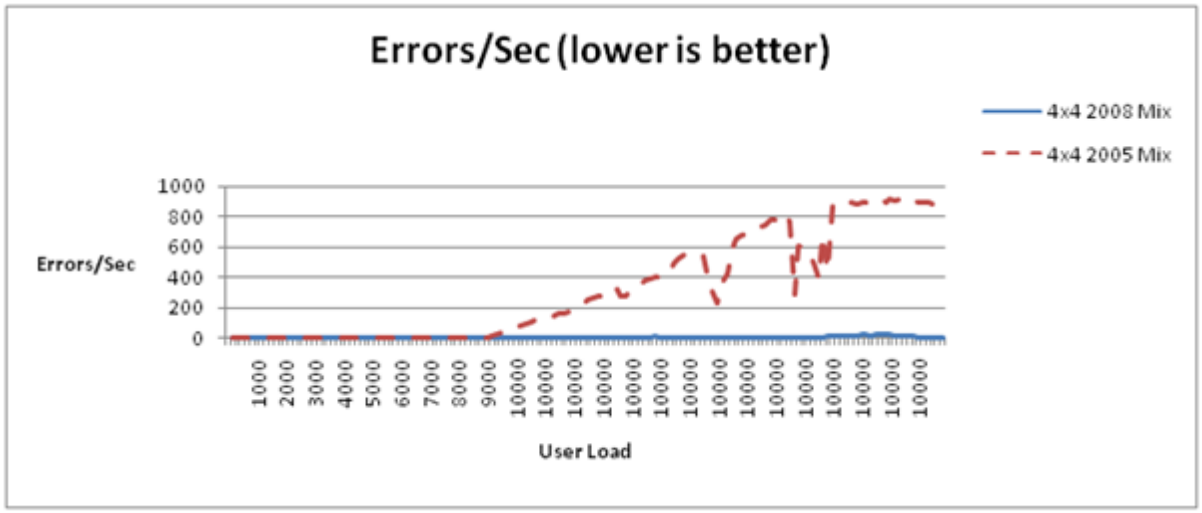


Figure 11: Push Test with Mix of Report Renderers—Errors/Sec

Http Errors vs. Request Time-Out Errors

To better understand the hardware limits we hit with Reporting Services 2008 compared to Reporting Services 2005, we compared the HTTP 503 errors we saw with Reporting Services 2005 with the Reporting Services time-out errors that we saw with Reporting Services 2008. We discovered that while the failure rate for reports with Reporting Services 2005 was directly correlated with HTTP 503 errors, the failure rate for reports with Reporting Services 2008 was directly correlated with time-out errors. These time-out errors, combined with our analysis of the disk queuing on the data source, indicate that the data source was unable to keep up with the request for data from the front-end servers.

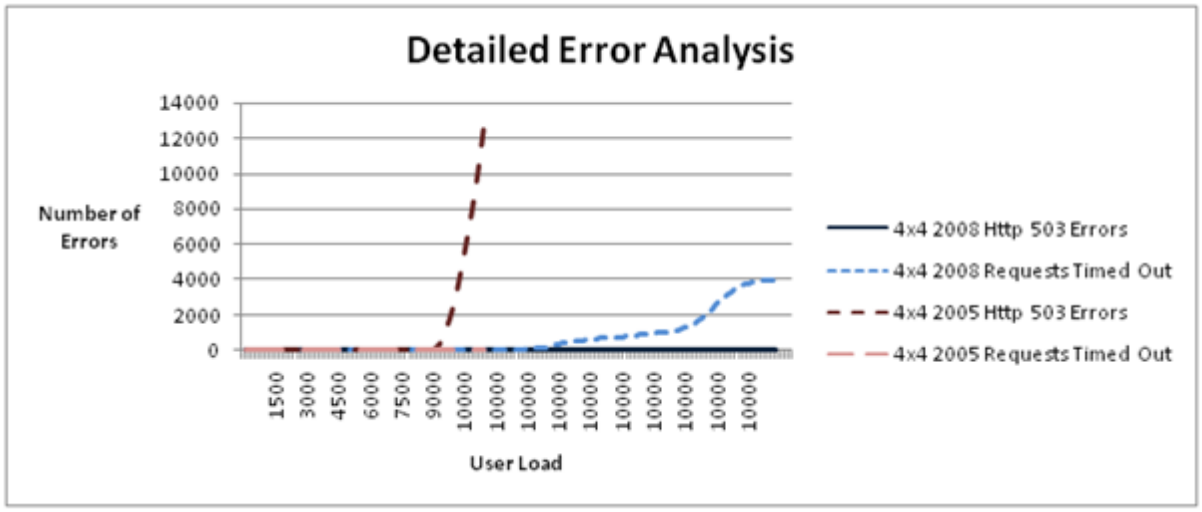


Figure 12: Push Test – Detailed Error Analysis

Note: The chart lines for the Reporting Services 2005 tests in Figure 12 only include the errors for the user load before the number of errors/sec climbed to the point where they completely skewed the overall numbers.

Average Response Time

When we analyzed the average response time and total requests counter values from our push test with a mix of report renderers (see Figure 13 and Figure 14), we discovered that while the average response time with Reporting Services 2005 was better than with Reporting Services 2008 until Reporting Services 2005 was completely overwhelmed and unable to keep responding (at about 9,000 users), Reporting Services 2008 just kept on rendering reports for a much greater throughput. Based on the disk queuing counters that we observed on the Reporting Services data source, we believe that significantly greater response times could have been achieved with Reporting Services 2008 with more hardware capacity.

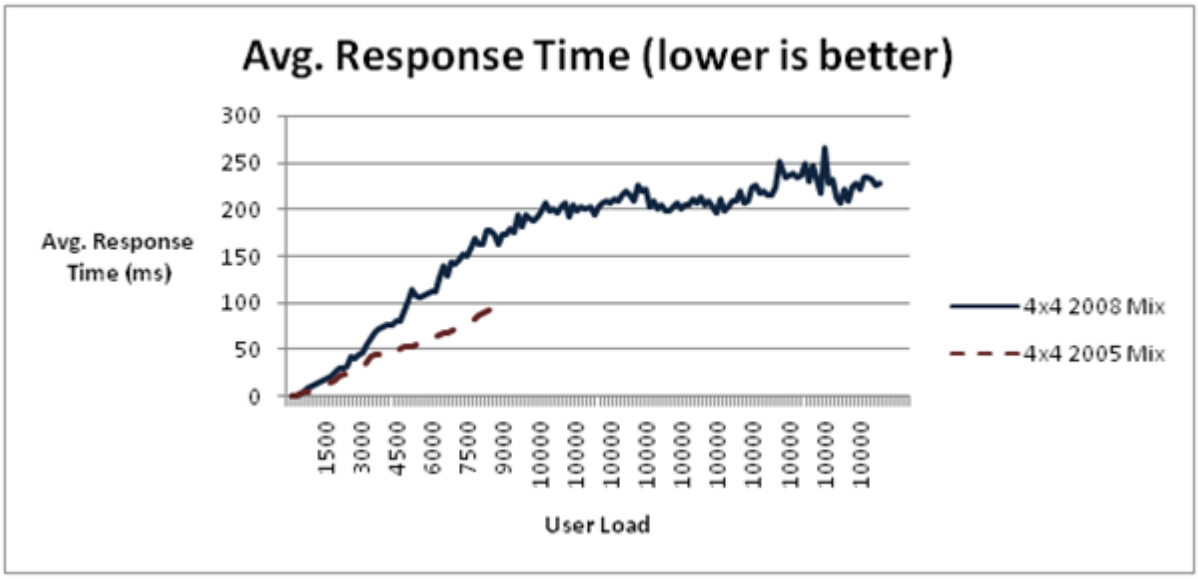


Figure 13: Push Test with Mix of Report Renderers—Average Response Time

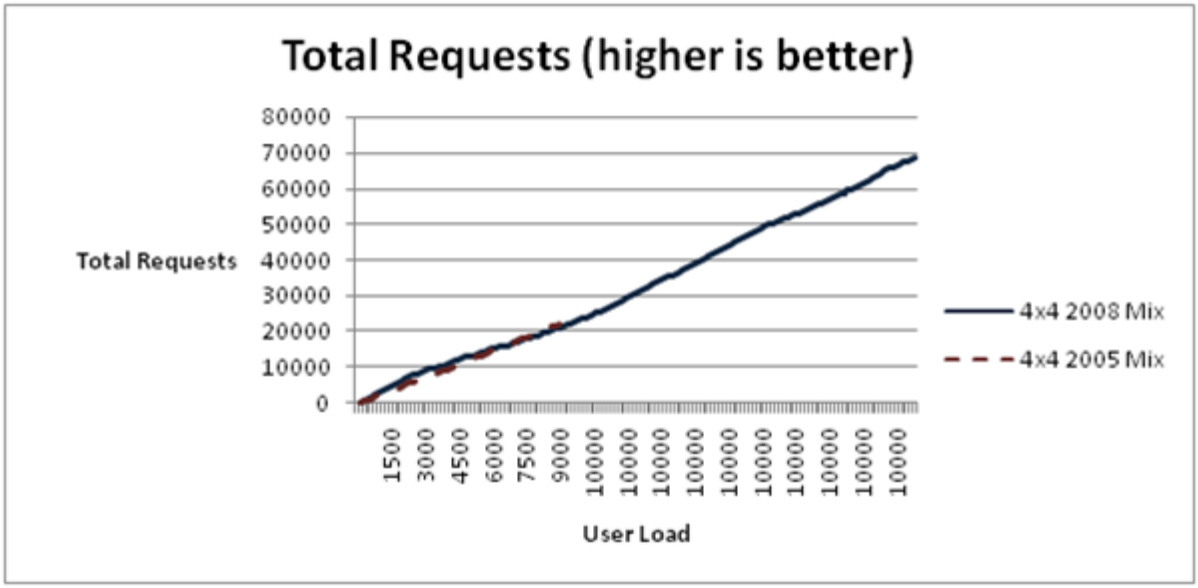


Figure 14: Push Test with Mix of Report Renderers—Total Requests

Miscellaneous

During our testing, we also observed the following:

1. The Reporting Services 2005 best practice of using the file system for extreme loads was observed to have a negative impact on results and should be avoided in Reporting Services 2008. For more information on this setting in Reporting Services 2005, see Planning for Scalability and Performance with Reporting Services.

2. During our initial optimization of the default Reporting Services 2008 and Reporting Services 2005 configurations, the only change that we made was to preallocate space to the **ReportServerTempDB** database. This change had a significant beneficial impact on performance. We did not attempt to optimize the performance of this database by placing this database on a very fast drive or by spreading this database across multiple drives. From our analysis of our tests, we believe that the disk configuration of the **ReportServer** and **ReportServerTempDB** databases is even more important in Reporting Services 2008 than in Reporting Services 2005. Based on our tests, we believe that every Reporting Services 2008 installation or upgrade should pay close attention to the disk configuration of these databases. For additional information about optimizing disk configuration for Reporting Services 2005, see Planning for Scalability and Performance with Reporting Services. While the advice in this white paper is targeted to Reporting Services 2005, most or all of the advice is applicable to Reporting Services 2008. More testing is probably advised in this area.

3. At the end of our tests, we made some preliminary tests to determine if Windows Server 2008 would provide some benefit over Windows Server 2003. Among the improvements in Windows Server 2008 are improvements in memory management and in the TCP/IP stack. While there was some improvement of Reporting Services 2008 on Windows 2008 versus Reporting Services 2008 on Windows 2003, the data that we collected was inconclusive. We expect that the rendering of many larger reports may well benefit from the memory performance improvements in Windows Server 2008. In addition, subsequent to the build that we used for our tests, there is an improvement in Reporting Services 2008 related to the rendering of images (PDF, EMF/Print, TIFF) when Reporting Services 2008 is running on Windows Server 2008. Based on other testing, this has been shown to result in about a 5.7% performance benefit for image rendering. More testing is required to determine the benefit of Windows Server 2008 in conjunction with Reporting Services 2008.

Reporting Services Performance in SharePoint Integrated Mode in SQL Server 2008 R2

Overview

Integrating Microsoft SQL Server Reporting Services (SSRS) with your Microsoft SharePoint environment provides many benefits to your enterprise, including specifically shared storage, shared security, and same site access for all business documents. For more information about the benefits of Reporting Services integration with SharePoint, see Edgenet Realizes the Power of Dynamic IT and Self-Service BI and Overview of Reporting Services and SharePoint Technology Integration. SharePoint integration does, however, introduce some performance overhead. Naturally, the performance of Reporting Services reports in SharePoint integrated mode is an area of concern for customers. Since the initial release of software that integrated Reporting Services with SharePoint approximately five years ago, Microsoft has improved report performance (for both native and SharePoint integrated mode) through service packs and with new releases of SQL Server, SharePoint, and Windows. After the release of SQL Server 2008 R2 and SharePoint 2010, the SQL CAT and Reporting Services teams collaborated to measure the performance of Reporting Services reports in SharePoint integrated mode and in native mode to more fully understand and document the current performance landscape. We performed concurrent throughput and individual report performance tests against Reporting Services in SQL Server 2008 compared with Reporting Services in SQL Server 2008 R2, specifically focused on the performance delta between native mode and SharePoint mode.

Executive Summary

Reporting Services in SQL Server 2008 R2 and SharePoint 2010 (with Windows Server 2008 R2) delivers significant performance improvements compared with SQL Server 2008 and SharePoint 2007 (with Windows Server 2008) in both native and SharePoint integrated mode. These performance improvements include improvements in overall throughput (operations per second) and in individual report performance. There is a measurable amount of additional overhead associated with SharePoint integrated mode that impacts report performance (between 250 milliseconds and 500 milliseconds of additional overhead to view the first page of any report). This performance difference is most evident when users request sub-second reports; in this case, the SharePoint overhead is a substantial percentage of the overall execution time. When users request multiple-second reports, the performance difference is a small fraction of the total time required to render the first page, and, when they exporting to Microsoft Excel, SharePoint integrated mode in SQL Server 2008 R2 Reporting Services is equal to or faster than native mode with SQL Server 2008 Reporting Services.

Furthermore, the performance of individual reports viewed using the ASP.NET Ajax-enabled Report Viewer web part in SharePoint 2010 degrades significantly when there are multiple viewers on a single web page, the viewer generates very large HTML, and the web page is viewed using older browsers. This is an area that we are working very hard to resolve, both through Cumulative Updates and in future releases (it is a multi-faceted issue). For more information about fixes in Cumulative Update 3 related to this issue, see It takes a long time to open a SQL Server 2008 R2 Reporting Services report that contains many parameter values in Internet Explorer.

Important: The additional overhead associated with SharePoint integrated mode will vary depending on server hardware, network infrastructure, and other factors, such as report design, all-in-one vs distributed architecture, SharePoint response time, anti-virus software, etc. Due to this variety of factors, the amount of additional overhead that will be experienced in any specific environment cannot predicted or guaranteed.

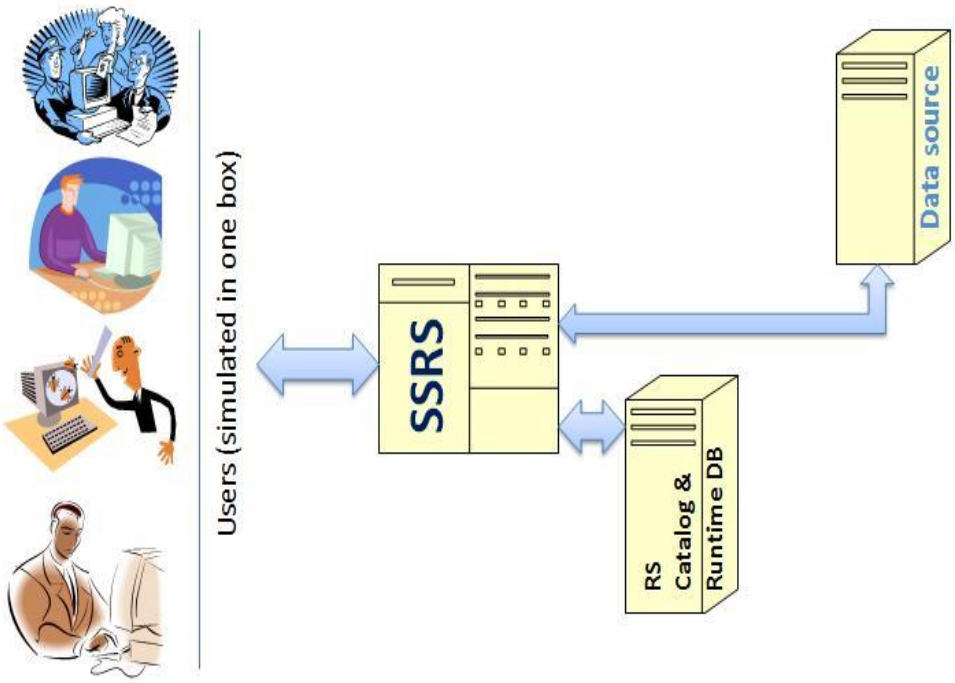
Test Topology

We utilized the following topologies to test Reporting Services in SQL Server 2008 (with Service Pack 1).

Note: For Reporting Services in SQL Server 2008, we utilized Windows Server 2008. For Reporting Services in SQL Server 2008 R2, we utilized Windows Server 2008 R2.

Native Mode

The following diagram illustrates our environment for testing native mode performance.



In this topology, each report request from our simulated user requires a minimum of four network hops to fulfill the report request, with two additional network hops required for the reports that were exported:

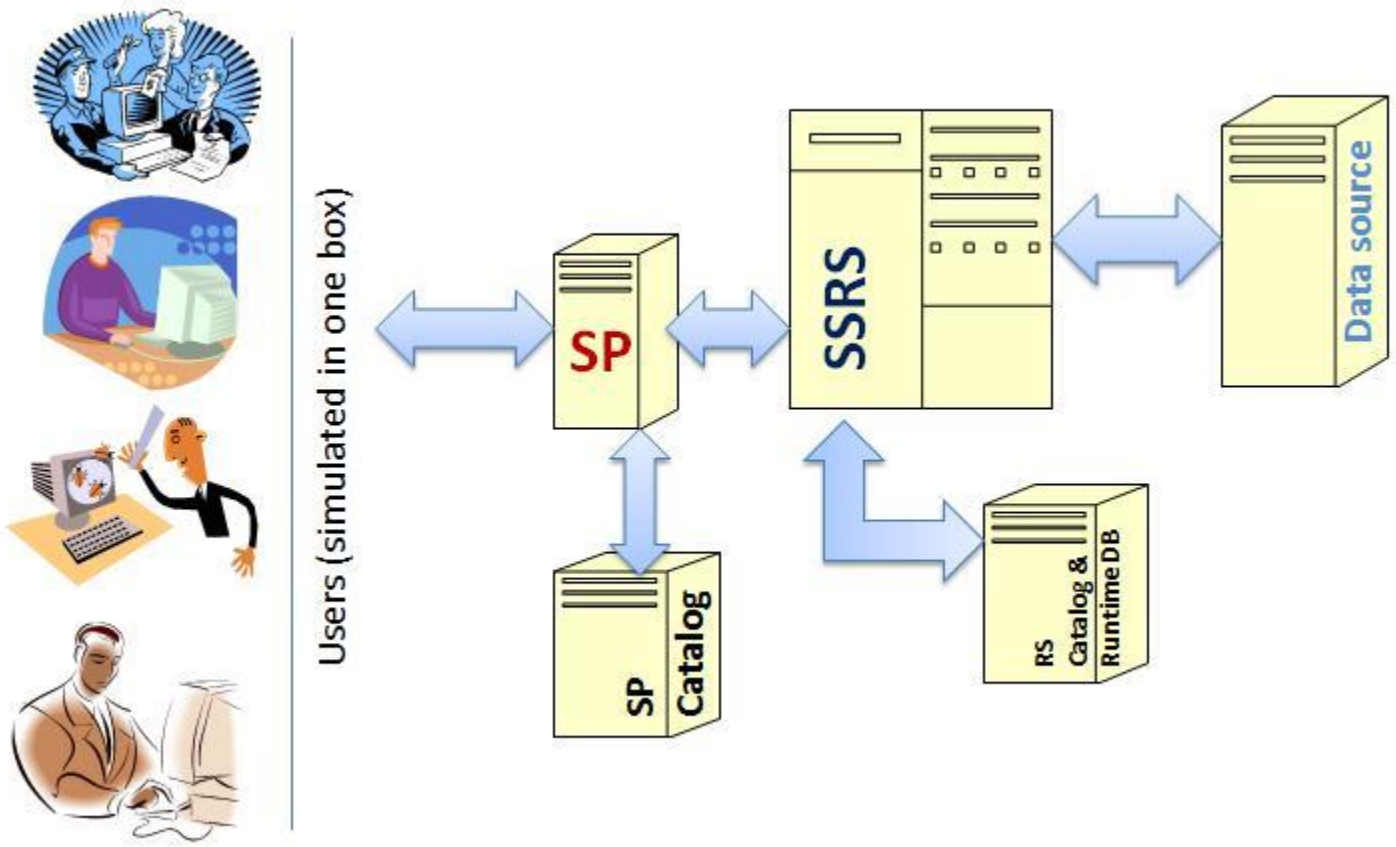
- 1. Report request from user to Reporting Services engine
- 2. Reporting Services engine retrieval of report definition from RS catalog server
- 3. Reporting Services engine query request to data source
- 4. Return of the rendered report from the Reporting Services engine to the user

- 5. [If applicable] Request from user to Reporting Services engine to render and export in a specified format
- 6. [If applicable] Return of the report in the desired export format from the Reporting Services engine to the user

Important: All report requests are handled directly by the Reporting Services engine.

SharePoint Mode

The following diagram illustrates our environment for testing SharePoint integrated mode.



(SP stands for SharePoint.) In this topology, each report request from our simulated user requires a minimum of seven network hops to fulfill the report request, with four additional network hops required for the reports that were exported:

- 1. Report request from user to the SharePoint web service
- 2. SharePoint engine retrieval of the report definition from the SharePoint catalog (the master copy)
- 3. SharePoint web service report request to Reporting Services engine

- 4. Reporting Services engine query to the RS catalog server to verify that the report definition exists in the RS catalog and that it is same as the master copy
- 5. Reporting Services engine query request to data source
- 6. Reporting Services forward of rendered report to the SharePoint web service
- 7. Return of the report from the SharePoint web service to the user
- 8. [If applicable] Request from user to the SharePoint web service to render and export in a specified format
- 9. [If applicable] Return of the report in the desired export format from the Reporting Services engine to the SharePoint web service
- 10. [If applicable] Return of the report in the desired export format from the SharePoint web service to the user

Important: The report requests are handled initially by the SharePoint web service and then by the Reporting Services engine. There is basic overhead of communication with the SharePoint object model for doing basic work such as getting the item and checking access. There is also overhead associated with additional network hops, with the amount of this overhead dependent upon your SharePoint topology and performance capabilities of your network.

Test Workload

We utilized 53 different reports that exercised a wide range of product features and typical report design practices, including:

- Tables
- Matrixes
- Charts
- Rich text
- Recursive aggregates
- Running value aggregates
- Images
- Recursive hierarchies
- Filtered data sets
- Complex expressions
- Recursive drilldowns
- Subreports
- Show / hide
- Gauges
- Calculated fields
- Conditional formatting

These reports consisted of mix of large reports (roughly 250,000 rows) and small reports (roughly hundreds to thousands of rows).

Important: This workload, with its mix of product features of report sizes, was specifically designed to place a heavy load on the system; it does not necessarily simulate a typical real-world workload. This workload included only product features supported by SQL Server 2008 Reporting Services and SQL Server 2008 R2 Reporting Services.

Concurrent Throughput Performance

The goal of this test was to measure Reporting Services performance from the point of view of the IT department and total workload. The following sections describe the concurrent throughput performance workload and test results.

Concurrent Throughput Performance Workload

We designed a concurrent throughput performance workload based on the test workload. Approximately one-third of the workload was rendered from snapshots and two-thirds of the workload was rendered from live data. Using a custom test harness to randomize the report executions, the workload had the following characteristics:

- Approximately 50 percent of the reports were rendered to HTML from live data.
- Approximately 20 percent of the reports were rendered and then exported to PDF from live data.
- Approximately 20 percent of the reports were rendered and then exported to PDF from snapshots.
- The remaining 10 percent of the reports were rendered and then exported to CSV, Excel, XML, and Microsoft Word formats equally from live data and from snapshots.

Important: This concurrent throughput performance workload does not necessarily simulate a typical real-world workload, with its mix of report types, report sizes, mix of live versus snapshot data, and mix of export formats.

Note: The live reports used the rs:ClearSession URL parameter to render reports from live data rather than cache.

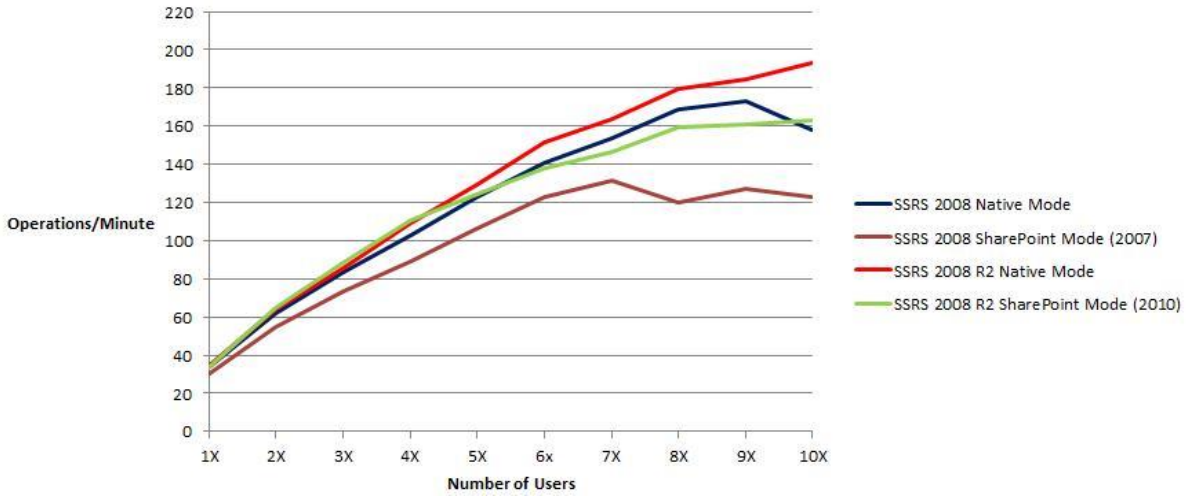
We tested different concurrency levels (*n*). The number of users (the factor of *n*) in your environment depends, of course, on your server hardware, network speed, data sources, and the actual reports.

Concurrent Throughput Performance Test Results

Reporting Services in SQL Server 2008 R2 delivers significant concurrent throughput performance improvements, in both native mode (with Windows Server 2008 R2) and SharePoint mode (with Windows Server 2008 R2 and SharePoint 2010). Using our specific workload, we observed the following:

- **SharePoint Mode**: Throughput improvements occur with Reporting Services in SQL Server 2008R2 in SharePoint integrated mode at all concurrency levels, beginning at a 12 percent increase in concurrent operations at 1x concurrent users and increasing to a 33 percent increase in throughput at 10x concurrent users when compared with Reporting Services in SQL Server 2008.
- **Native Mode**: Throughput improvements appear gradually with Reporting Services in SQL Server 2008 R2 in native mode as concurrency increases, with a 22 percent increase in concurrent operations at 10x concurrent users when compared with Reporting Services in SQL Server 2008.

The test results are displayed in the following graph.



(SSRS 2008 stands for SQL Server 2008 Reporting Services, and SSRS 2008 R2 stands for SQL Server 2008 R2 Reporting Services.) We found the following observations most interesting:

- Whereas SQL Server 2008 Reporting Services in SharePoint integrated mode performed fewer operations per minute than SQL Server 2008 Reporting Services in native mode even at low concurrency, SQL Server 2008 R2 Reporting Services in SharePoint mode performed approximately the same number of operations per minute as SQL Server 2008 R2 Reporting Services in native mode until the system reached higher levels of concurrency.
- SQL Server 2008 R2 Reporting Services in SharePoint mode performed more operations per minute than SQL Server 2008 Reporting Services in native mode until the system reached higher levels of concurrency.
- At the highest levels of concurrency that we utilized in this test, all of the workloads began to level out in performance. Our investigation showed us that our bottlenecks at high concurrency were in the data source and the network (our goal in this test was not to tune the data source or the network).

Important: The actual performance improvements that you experience with your actual workloads will vary depending upon your workload and your environment. These performance numbers are not based on real workloads, but rather on a workload that was designed to test the entire range of the product's features and capabilities.

Individual Report Performance

The goal of this test was to measure Reporting Services performance from the point of view of the user executing the report. The following sections describe the individual report performance workload and test results.

Individual Report Performance Workload

We executed each of the reports in the test workload serially using Microsoft Internet Explorer 8, executing against Reporting Services in both native mode and SharePoint integrated mode. We measured:

- Time to load the initial viewer frame
- Time to load the first page
- Time to load the remaining pages

We also executed each of the reports in the test workload using Internet Explorer running against a single dashboard page in SharePoint containing multiple Ajax Report viewers.

Individual Report Performance Test Results

Reporting Services in SQL Server 2008 R2 delivers significant individual report performance improvements, in both native mode (with Windows Server 2008 R2) and SharePoint mode (with Windows Server 2008 R2 and SharePoint 2010).

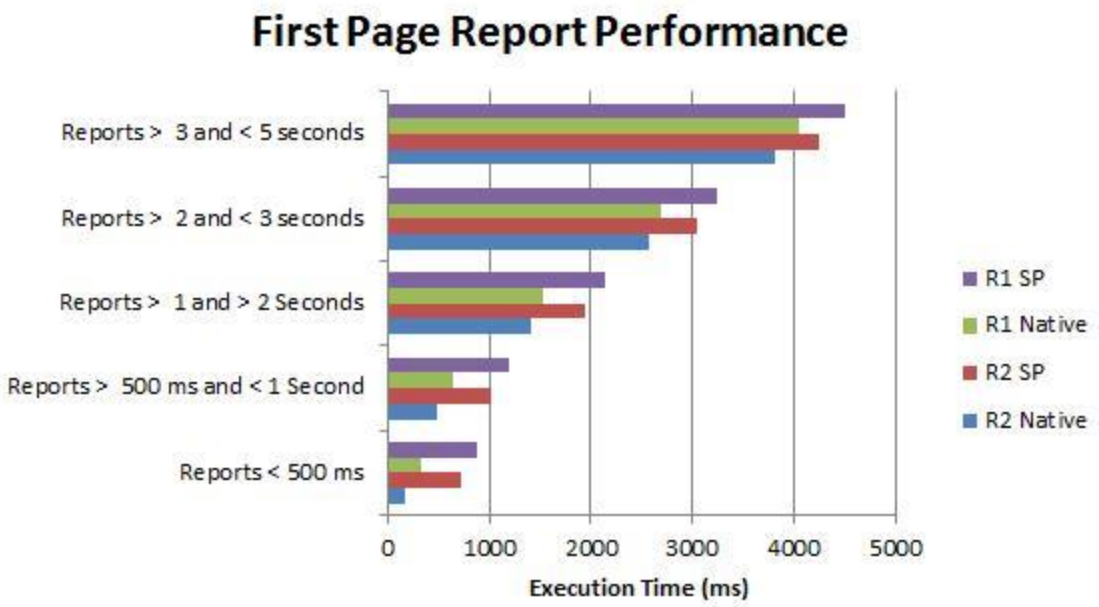
Using our specific workload, we observed that individual report performance improvements occur with Reporting Services in SQL Server 2008 R2 in both SharePoint mode and native mode. The performance improvement ranged from no improvement for some reports to improvements above 30 percent for a handful of reports.

The test results are displayed in the following graphs, with the results grouped based on the report execution time for reports in native mode in SQL Server 2008 Reporting Services.

Important: The actual performance improvements that you experience with your actual workloads will vary depending upon your workload and your environment. These performance numbers are not based on real workloads, but rather on a workload that was designed to test the entire range of the product's features and capabilities.

Rendering the First Page of a Report

The following chart shows the average time required to return the first page of the requested report to Internet Explorer. We focused initially on the time to return the first page of a report because if the first page of a report renders too slowly, the user experience is not satisfactory.

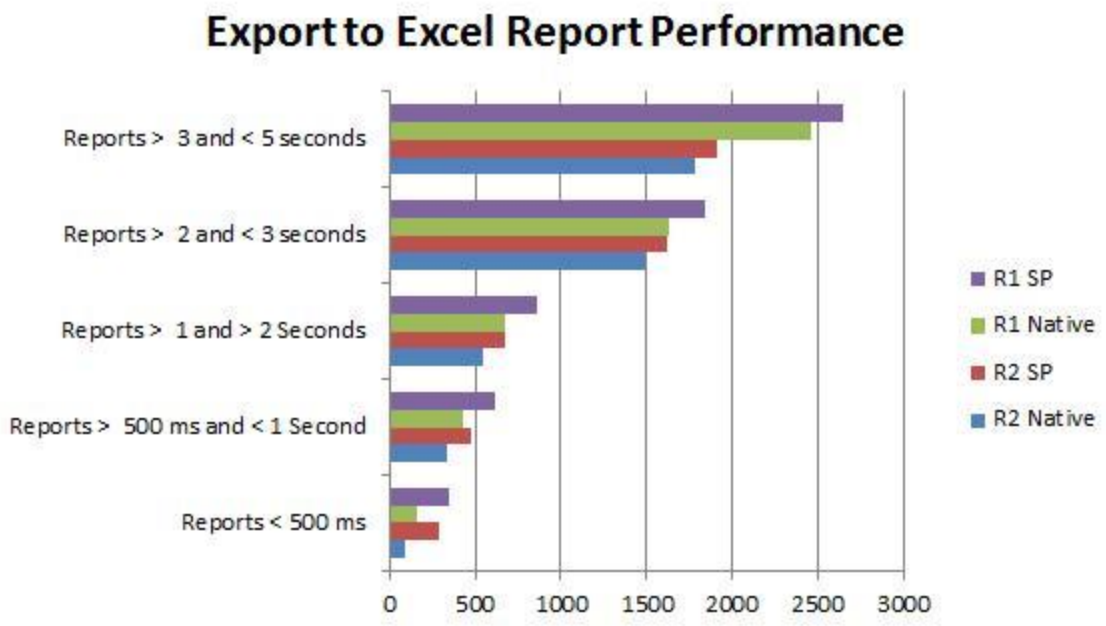


Notice that Reporting Services in SQL Server 2008 R2 is faster than in SQL Server 2008 for each mode. Notice also that native mode is faster than SharePoint integrated mode, but that the impact of that difference depends on the length of time to render the first page. From the individual user perspective, users will experience the performance delta between SharePoint integrated mode and native mode primarily with reports that take less than a few seconds to return the first page because in these cases the SharePoint overhead is responsible for the bulk of the execution time to render the first page.

Note: We observed that the overhead for SharePoint to establish the connection from the web front end to Reporting Services and return results was approximately 400 milliseconds for each report, regardless of size of the report rendered.

Exporting a Report to Excel

The following chart shows the average time required to export the requested report to Excel for reports of various execution time ranges. We focused on export to Excel as the most common export format, and this provides a good measure of the time to render, paginate, and format the entire report.



(R1 Native and R2 Native stand for SQL Server 2008 Reporting Services and SQL Server 2008 R2 Reporting Services in native mode, and R2 SP stands for SQL Server 2008 R2 in SharePoint integrated mode.) Notice again that Reporting Services in SQL Server 2008 R2 is faster that in SQL Server 2008 for each mode. Notice also that native mode is faster than SharePoint integrated mode for the export phase for the sub-second reports, but that SharePoint integrated mode for Reporting Services in SQL Server 2008 R2 is actually faster that native mode for Reporting Services in SQL Server 2008 for reports running longer than two seconds.

Multiple AJAX Viewers

Viewing the reports in a multiple viewers in a single dashboard in SharePoint revealed that Internet Explorer (as well as other browsers) had performance issues with large reports generating excessively large HTML in some circumstances. While this does not result in a functional problem, some browsers are not able to handle the size of the HTML well. As a result, we are seeing varying degrees of performance degradation on different browsers. This performance problem seems to be particularly noticeable on pages with a larger number of valid values in a parameter, a large report page, or multiple viewers (or web parts) on a single page. In the meantime, we recommend using the latest version of your browser, because the problem appears to be more significant on older browsers, and ensuring that you are using the newest Cumulative Update. For more information about fixes in Cumulative Update 3 related to this issue, see It takes a long time to open a SQL Server 2008 R2 Reporting Services report that contains many parameter values in Internet Explorer.

Conclusion

Reporting Services in SQL Server 2008 R2 and SharePoint 2010 (with Windows Server 2008 R2) delivers significant performance improvements compared with SQL Server 2008 and SharePoint 2007 (with Windows Server 2008) in both

native and SharePoint integrated mode. These performance improvements include improvements in overall throughput (operations per second) and in individual report performance. There is a measurable amount of additional overhead associated with SharePoint integrated mode that impacts report performance (between 250 milliseconds and 500 milliseconds of additional overhead to view the first page of any report). This performance difference is most evident when users request sub-second reports; in this case, the SharePoint overhead is a substantial percentage of the overall execution time. When users request multiple-second reports, the performance difference is a small fraction of the total time required to render the first page, and, when exporting to Excel, SharePoint integrated mode in SQL Server 2008 R2 Reporting Services is equal to or faster than native mode with SQL Server 2008 Reporting Services.

Deploying a Business Intelligence Solution Using SharePoint 2007, SQL Server 2008 Reporting Services, and PerformancePoint Monitoring Server 2007 with Kerberos

This technical note describes how we designed and implemented a business intelligence solution that utilized a server farm containing Microsoft® Office SharePoint® Server 2007, Microsoft Office PerformancePoint® Server 2007 Monitoring Server, and Microsoft SQL Server® 2008 Reporting Services in SharePoint mode, all running on Windows Server® 2008 R2 and with all servers and applications configured for Kerberos authentication. In this technical note, we discuss the design requirements for this business intelligence solution, its logical architecture, the challenges we faced in architecting and implementing this solution, and our resolutions to these challenges.

Note: The business intelligence solution discussed in this technical note was configured and tested in the SQL CAT customer lab and is currently being deployed (substantially as discussed in this article) at a customer site.

Design Requirements

The design requirements for this business intelligence solution are:

- Deploy the solution into a Windows Server 2008 R2 domain.Deploy all server applications in this solution onto servers running Windows Server 2008 R2.

- Deploy a server farm running Office SharePoint Server 2007 (SharePoint), which includes the SharePoint Web front-end (WFE) and the application server, on one server and the SQL Server (SQL Server) database on another server.

- Configuring the SharePoint services for later scale-out to multiple SharePoint WFE front-end servers and multiple SharePoint application servers

- Deploy SQL Server 2008 Reporting Services (Reporting Services) on a server, integrate Reporting Services with SharePoint, and deploy its databases to a shared metadata database server

- Deploy Microsoft Office PerformancePoint Server 2007 Monitoring Server (PerformancePoint) on the SharePoint application server, deploying its Monitoring System database to the shared metadata database server, and configuring PerformancePoint for later scale-out to a dedicated application server.

- Utilize SQL Server and SQL Server Analysis Services (Analysis Services) data sources (the data source servers) for the Reporting Services reports and PerformancePoint dashboards.

- Design reports using Report Builder 2.0 and PPS Dashboard Designer.

- Execute Reporting Services reports and view PPS dashboards utilizing Windows® Internet Explorer® 7 or Internet Explorer 8.

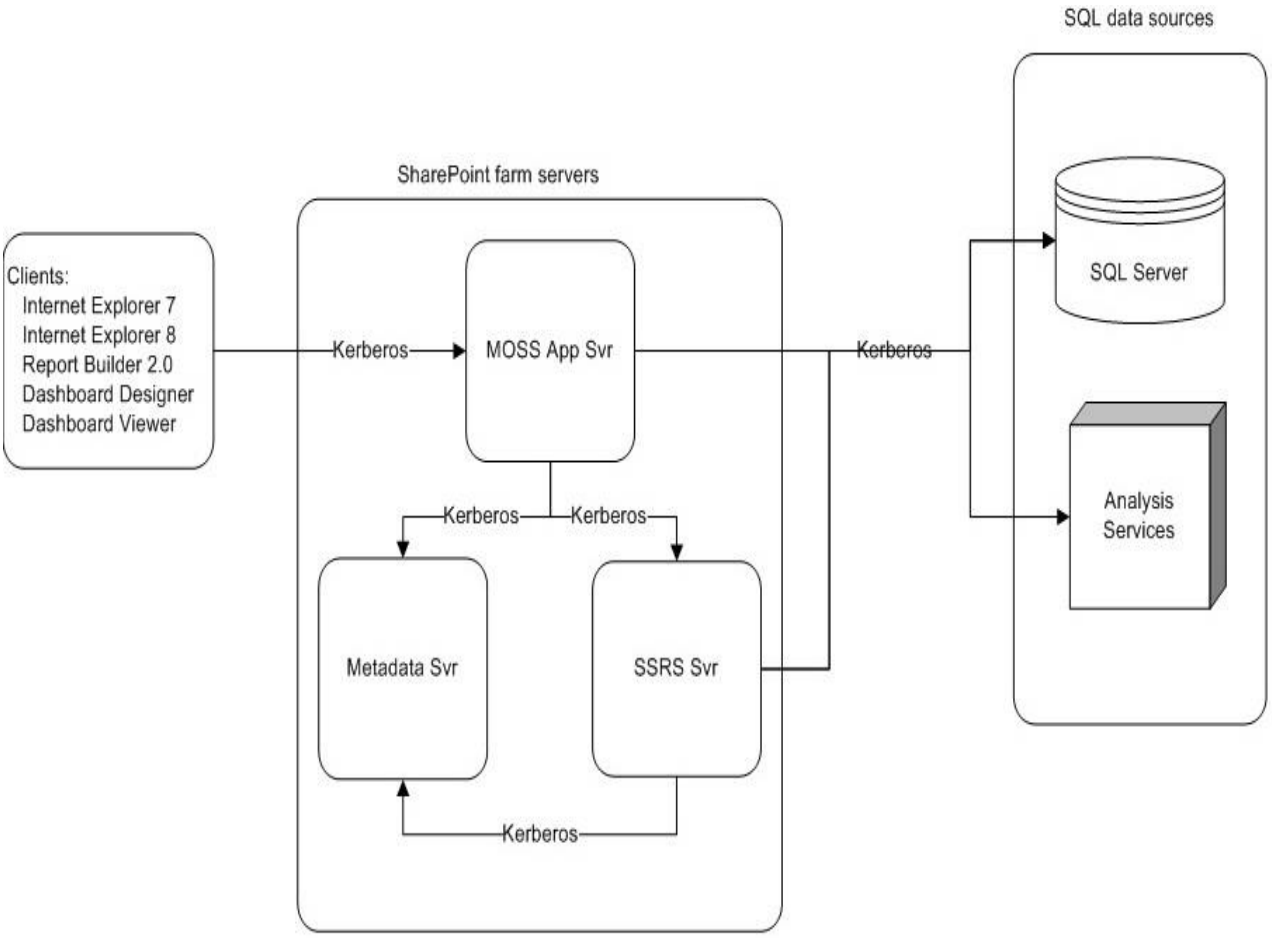
- Configure the MOSS application server to authenticate business users retrieving business intelligence data into their Reporting Services reports and PPS dashboards from the the data source servers using the business user's authentication credentials.

Solution Architecture

The following two diagrams represent the solution architectures for the initial deployment and for eventual scale-out. The discussion in the remainder of this document describes the challenges we faced and our resolutions in implementing this solution architecture.

Initial Deployment Architecture

In this initial deployment architecture, the SharePoint WFE and application server along with the PerformancePoint Server are located on a single application server (called the MOSS App Svr in the following diagram), with a single dedicated server on which Reporting Services is installed in SharePoint mode (called the SSRS Svr in the following diagram), and with a single server on which SQL Server is installed that contains all of the metadata for SharePoint, PerformancePoint and Reporting Services (called the Metadata Svr in the following diagram).

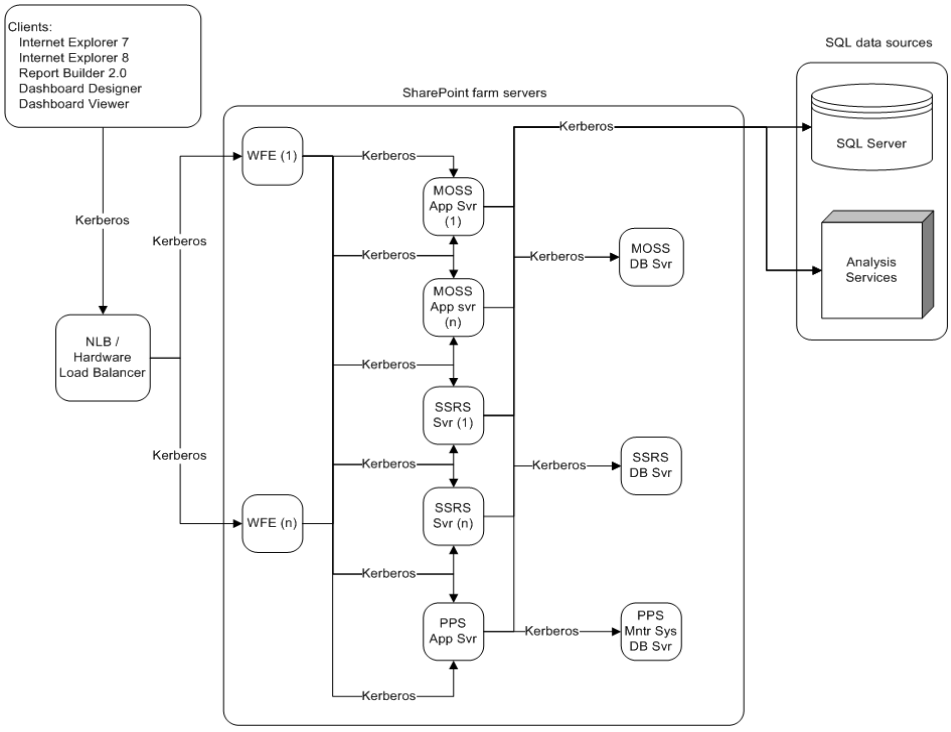


Full Scale-Out Deployment Architecture

In this full scale-out deployment architecture, the components in the previous diagram are scaled out to the following components:

- Multiple SharePoint WFE servers (called WFE (1) through WFE (n) in the following diagram).

- Multiple SharePoint application servers (called MOSS App Svr (1) through MOSS App Srv (n) in the following diagram).
- A dedicated SharePoint metadata server (called the MOSS DB Svr in the following diagram).
- A dedicated PerformancePoint Server application server (called the PPS App Svr in the following diagram).
- A dedicated PerformancePoint Server metadata server (called the PPS Mntr Sys DB Svr in the following diagram).
- Multiple Reporting Services application servers (called SSRS Svr (1) through SSRS Svr (n) in the following diagram).
- A dedicated Reporting Services metadata server (called the SSRS DB Svr in the following diagram).



Challenges Encountered and Resolutions

In designing and deploying this solution, we encountered a number of installation challenges. Some were unique to Kerberos, and others were related to software installation. These challenges and our resolutions are discussed next.

Note: This technical note assumes a basic understanding of Kerberos; a full discussion of Kerberos architecture and step-by-step configuration procedures is beyond the scope of this article. For more information about Kerberos architecture, see Microsoft TechNet. It contains has many articles, including the following one: Kerberos. For step-by-step guidance, see the articles that are mentioned in context in

this technical note. We include our comments on lessons learned from our use of these articles. There is no single article for this combination of components.

Designing and Configuring for Kerberos Authentication

To enable the middle-tier applications (MOSS, PPS, and SSRS) on the Application server to seamlessly authenticate users in accessing report and dashboard data on the data source servers, Kerberos is required. If it is properly configured, the client computer obtains the Kerberos ticket for the user from the Kerberos Key Distribution Center (KDC) based on properly registered Service Principal Names (SPNs), and then it presents this ticket to the SharePoint server farm when it establishes a connection.

Note: The KDC is a network service that supplies session tickets and temporary session keys to users and computers within an Active Directory® domain. The KDC runs on each domain controller as part of Active Directory Domain Services.

The application servers in the SharePoint server farm, if they are properly configured, impersonate this user when then retrieve the requested data from the appropriate SQL data server (SQL Server or Analysis Services). This is the classic double-hop scenario, which requires Kerberos to securely impersonate a user credentials when connecting to a service on another computer. In order for the Internet Information Services (IIS) service or the Reporting Services service to impersonate the user during connection to the appropriate SQL Data server, the service account under which the IIS service or the Reporting Services service is running must have permission to impersonate, and the computers involved must be authorized for delegation.

Note: For more information about troubleshooting with respect to Kerberos and delegation, see Kerberos authentication and troubleshooting delegation issues.

On the application server in our solution, there are multiple IIS Web applications within MOSS (Central Administration, a portal site, a My Site, and an SSP Administration site) as well as the PPS web application and the SharePoint WFE. These Web applications utilize different identities via dedicated application pools, and these Web applications listen on both default and nondefault ports (that is, ports other than TCP port 80 and SSL port 443). In addition, the WFE forwards the SSRS requests to the SSRS server. In order for impersonation of the user's Kerberos credentials to work properly, the identity for each Web application must have an SPN registered. Furthermore, the appropriate SPN must be utilized when the client connects to a particular IIS Web application. In addition, the SSRS service on the SSRS server along with the Analysis Services and SQL Server services on the SQL Data servers must each have properly registered SPNs. To implement Kerberos authentication for this environment, we utilized the approach described in the following Microsoft TechNet article: Configure Kerberos authentication (Office SharePoint Server), supplemented by the Excel® Services documentation in the following Knowledge Base article: How to configure SharePoint Server 2007 and Excel Services for Kerberos authentication and PPS-specific information in the following Microsoft TechNet article: Configure Monitoring Server for Kerberos delegation.

Note: One of things we really like about the Configure Kerberos authentication (Office SharePoint Server) TechNet article is its step-by-step approach to verifying that Kerberos is configured and working properly. However, you should be aware that the Event ID in the Event log for a successful login using Kerberos has changed with Windows Server 2008 and Windows Vista®. It is now Event ID 4624 (see Knowledge Base article 947226); it was Event ID 540 in Windows Server 2003 (as documented in the article).

Kerberos Challenge 1:

As pointed out in the Configure Kerberos authentication (Office SharePoint Server) article, Internet Explorer does not, by default, include port numbers in its Kerberos ticket requests. If port numbers are not included, impersonation is attempted using the Kerberos ticket associated with the application pool identity for the Web application listening on port 80 (or port 443 if SSL is involved). Because in our solution architecture we have Web applications listening on both nondefault and default port numbers, this default behavior does not result in the appropriate Kerberos ticket being retrieved for Web applications listening on nondefault ports – resulting in authentication errors.

To enable Internet Explorer to include port numbers in every request, you can change a registry setting on every client. For more information about this behavior and registry key, see Knowledge Base article 908209. While this Knowledge Base article refers only to Internet Explorer 6.0, the behavior and the registry key changes also apply to Internet Explorer 7 and 8. For the customer environment for which we were designing this solution, modifying the registry on each client (even by means of a global policy) was not acceptable. Without the port numbers in the Kerberos ticket request, only one SPN can be registered for each host name.

Our challenge was to design a solution that enabled an SPN for each Web application in IIS such that a unique Kerberos ticket is retrieved for each Web application on the application server if the application is listening on a nondefault port.

Kerberos Resolution 1:

To enable a unique SPN and Kerberos ticket for each IIS Web application listening on a nondefault port, we did the following:

- Defined unique host names in DNS for each IIS Web application on the Application server.
- Configured each Web application with its own application pool identity (service account).
- Registered four SPN variations for each unique host name using the service account identity for that Web application.

To illustrate, we began by creating host names in DNS for the SharePoint portal site (hostname), for Central Administration (hostnameca), and PerformancePoint (hostnamepps). Then we registered SPNs for the short host name as well as the fully qualified domain name (FQDN) (hostname.mydomain.com), registering these names both with and without the port number.

Note: We registered SPNs both with and without port numbers to ensure that our Kerberos configuration worked successfully whether Internet Explorer submitted the port numbers or not.

<u>hostname</u>

- setspn -A HTTP/hostname mydomain\mossportalpool
- setspn -A HTTP/hostname:80 mydomain\mossportalpool
- setspn -A HTTP/hostname.mydomain.com mydomain\mossportalpool
- setspn -A HTTP/hostname.mydomain.com:80 mydomain\mossportalpool

<u>hostnameca</u>

- setspn -A HTTP/hostnameca mydomain\mossfarmadmin

- setspn -A HTTP/hostnameca:11111 mydomain\mossfarmadmin
- setspn -A HTTP/hostnameca.mydomain.com mydomain\mossfarmadmin
- setspn -A HTTP/hostnameca.mydomain.com:11111 mydomain\mossfarmadmin

hostnamepps

- setspn -A HTTP/hostnamepps mydomain\ppsapppool
- setspn -A HTTP/hostnamepps:40000 mydomain\ppsapppool
- setspn -A HTTP/hostnamepps.mydomain.com mydomain\ppsapppool
- setspn -A HTTP/hostnamepps.mydomain.com:40000 mydomain\ppsapppool

This approach resolved the Internet Explorer issue with connecting to a Web application running on a nondefault port. With this approach, when Internet Explorer connects to a virtual hostname for the appropriate Web application (rather than the physical machine name), DNS refers the user to the application server, and it uses the SPN registered for the virtual hostname. As a result, the appropriate Kerberos ticket is retrieved (rather than the Kerberos ticket for the application listening on port 80). Using our example, when Internet Explorer connects to hostnameca or hostnameca:11111, the SPN for the mossfarmadmin is used, and if Internet Explorer connects to hostnamepps or hostnamepps:40000, the SPN for ppsapppool is used.

Note: We also registered SPNs for the service account for the SQL Server relational engine, Analysis Services, and Reporting Services services. Only two SPNs were registered for these services because connectivity to these services is not made by Internet Explorer; instead, the middle-tier services on the application servers connect directly to these services.

Kerberos Challenge 2:

Our solution to Kerberos Challenge 1 required that we configure IIS to utilize different application pool identities for each Web application: for example, hostname, hostnameca, and hostnamepps each used a different application pool identity. In IIS 7.5 with Windows Server 2008 R2 (as well as with IIS 7.0 in Windows Server 2008), IIS utilizes kernel mode authentication by default. Kernel mode authentication runs under the machine account of the IIS server. For more information, see Security Authentication <authentication> and Advanced Settings Dialog Box – Windows Authentication Feature. This change to kernel mode authentication is generally a good thing, because performance is improved and the configuration for Kerberos is simplified in many scenarios; authentication is done under the LocalSystem account, regardless of the application pool identity. However, in a Web farm environment (such as in our scenario), we needed to use the application pool for authentication to ensure the correct Kerberos ticket would be utilized for the appropriate Web service when it was scaled out to multiple application and front-end servers.

Kerberos Resolution 2:

There are two ways to resolve this problem:

- Set the useAppPoolCredentials attribute in the system.webServer/security/authentication/Windows-Authentication configuration section of the ApplicationHost.config file to “True”.

- Disable Kernel Mode authentication.

We chose the second option. As mentioned previously in this technical note, we used this TechNet article: Configure Monitoring Server for Kerberos delegation to help us configure PPS. This article specifically recommends that we configure kernel mode authentication to use application pool credentials with IIS 7.0 and Windows Server 2008. As a result, this was our initial approach to resolving this issue. This solution worked well with Internet Explorer 7. However, with Internet Explorer 8, we encountered an additional challenge: The first time a user connects to a Web application in MOSS that is configured with Kerberos and that uses an application pool identity, the user is repeatedly prompted for authentication when navigating for the first time through nonroot portions of the SharePoint site. We discovered that if we disabled kernel mode authentication, we did not see this behavior.

After discussing the situation with the developers on the SharePoint and Internet Explorer teams, we discovered that we had run into a recently discovered incompatibility with kernel mode authentication when using an application pool identity. The incompatibility was traced back to performance enhancements in Internet Explorer 8 that impact SharePoint 2007. We ran both functional and performance tests in our environment using Internet Explorer 7 and Internet Explorer 8, with kernel mode authentication disabled for all of the Web applications that used dedicated application pool identities, and we determined that this configuration functioned properly and performed well. As a result, our recommended solution is to disable kernel mode authentication with SharePoint 2007 to seamlessly support a mix of Internet browsers.

Kerberos Challenge 3:

The resolutions to our first two Kerberos challenges performed as expected when the SharePoint farm consisted of a single SharePoint server (with either a local or a remote SQL Server database). However, we encountered an issue with this approach when we scaled out to multiple front-end Web servers and multiple application servers. When we scaled out, we changed the name of our Web applications for Network Load Balancing (NLB) to a virtual name, which NLB then directed to a specific physical server. For example, we changed the name for our SharePoint portal site from hostname to kerbportal.mydomain.net and changed our SPNs accordingly. We followed the steps in Update a Web application URL and IIS bindings (Office SharePoint Server) to update the Web application URL and IIS bindings. When we did so, we were unable to view our deployed Reporting Services reports and PerformancePoint dashboards. We received errors stating that the published PPS dashboards and SSRS reports were not able to find the underlying dashboard and report locations. For example, reports that were deployed to http://hostname/ReportServer were still trying to connect to that location, but the newly configured location for the report server was http://kerbportal.mydomain.com/ReportServer. However, the deployed report was not configured to use this new URL path.

Note: For more information about this problem, see section 8.4 of Reporting Services SharePoint Integration Troubleshooting.

Kerberos Resolution 3:

The solution was quite straightforward after we identified the problem. Namely, we needed to configure virtual host names in DNS that would eventually be utilized in NLB, but until it was time to scale out, we pointed these virtual host names in DNS to the single SharePoint server.

So, for example, for the SharePoint portal site, create a DNS entry for a virtual host called (for example) kerbportal that points to the SharePoint server. Later, you can update this entry in DNS to point to the NLB server. Then, make sure to specify kerbportal.mydomain.com in the Host Header field when you create the portal Web site in the Central Administration application in SharePoint. Similarly, configure virtual names for your My Site and SSP Administration sites. Following this approach, adjust the SPNs

that you register to utilize the virtual host names that you will use when you later scale out. If you use this approach, you will not have to reconfigure your SPNs when you later scale out, and you will not have to redeploy your Reporting Services reports or your PPS dashboards.

For more information, see Plan for host-named site collections (Office SharePoint Server).

Installing Software Components

During the installation of the various software components, we also encountered a number of installation challenges. While none of these challenges were showstoppers, we present these resolutions to ensure that you can easily resolve these issues if you encounter them. The challenges we faced and how we resolved them are discussed in the following sections.

Installing the SQL Server Reporting Services Add-in on the SharePoint Server on Windows Server 2008 R2

When we installed the SQL Server Reporting Services Add-in (rsSharepoint.msi) on the SharePoint server, we followed the steps listed in SQL Server Books Online: How to: Install or Uninstall the Reporting Services Add-in. When we attempted to install the Reporting Services Add-in (rsSharepoint.msi) on the SharePoint server on Windows Server 2008 R2, the Reporting Services Add-in did not complete successfully, and the installer rolled back the attempted installation. When we reviewed the Application log in Event Viewer, we discovered numerous “Insufficient SQL database permission” errors. We reran the Reporting Services Add-in from a command prompt with elevated permissions, but we received the same errors. To resolve this issue, we followed the “How to Install Reporting Services Add-in in Files-Only Mode” section in How to: Install or Uninstall the Reporting Services Add-in. These steps enabled us to successfully install the Reporting Services Add-in. We verified the successful installation of the Reporting Services Add-in by reviewing the RS_SP_<N>.log file in C:\Users\<login>\AppData\Local\Temp.

Configuring SQL Server Reporting Services in SharePoint Mode for Windows Authentication with User Access Control Enabled

When we configured Reporting Services integration in SharePoint, we followed the steps listed in SQL Server Books Online: How to: Configure Report Server Integration in SharePoint Central Administration. When we configured report server integration settings in Central Administration, we were unable to select Windows Authentication to specify that connections be made using Windows integrated authentication. While this initially seemed to indicate a Kerberos problem, we checked and double-checked our Kerberos configuration and found no errors. After some research on the Internet using Bing, we found the following article Troubleshooting Server and Database Connection Problems that suggested that User Account Control (UAC) might be the problem – although the problem described in this article was not exactly the same as the one we were experiencing. We closed the Central Administration site in Internet Explorer and reconnected with elevated permissions, and then we were successfully able to select Windows Authentication for our SharePoint integration with Reporting Services.

Installing Prerequisites for PerformancePoint Monitoring Server on the SharePoint Server with SQL Server 2008

When we installed the prerequisites for PerformancePoint Monitoring Server on the SharePoint Server with SQL Server 2008, we followed the steps listed in the following TechNet article: Install PerformancePoint Monitoring Server with SQL Server 2008. However, we were unable to install the

SQLServer2005_XMO component (XMO 9.0), which is a prerequisite for a fully-functional PPS installation. After some research on the Internet using Bing, we discovered Knowledge Base article 970496, which explained that there was a known incompatibility between the SQL Server Reporting Services Add-in and XMO 9.0. This bug prevents the installation of PerformancePoint Planning Server (not Monitoring Server) and the importing of key performance indicators (KPIs) from SQL Server when using Monitoring Server. However, further research revealed that this bug was fixed in both the Cumulative Update Package 5 for SQL Server 2005 Service Pack 3 and the Cumulative Update Package 15 for SQL Server 2005 Service Pack 2. Details of the fix can be found in Knowledge Base article 972694. By using the newest version of XMO 9.0, we were able to install the SQL Server 2005 Management Objects Collection and subsequently import KPIs from SQL Server.

Use the Application Pool for Root PPS Site and Subsites

When we installed PerformancePoint Monitoring Server, we followed the steps listed in the following TechNet article: Configure Monitoring Server for Kerberos delegation. As discussed previously, we configured a unique application pool and identity for the PPSMonitoringPreview and PPMonitoringWebService subsites, as well as a unique virtual host name and port number for the PPS Web application. The virtual host name enables us to use the appropriate Kerberos ticket for the PPS service account. However, when we used Internet Explorer to connect to the PPSMonitoring site (HTTP://ppsmonitoring.mydomain.com:40000), we were unable to connect to either the WebService or the Preview subsites. Because we were reasonably confident of our Kerberos configuration at this point, we immediately looked at the identity for the application pool for the root site as well as these two subsites. We discovered that while we had configured the application pool identity for each of the subsites to the PPS service account, the root site was not similarly configured – instead, it was configured to use the NetworkService account. As soon as we changed the identity of the application pool for the root site to the PPS service account, PPS worked as expected. Upon rereading the TechNet article, we noted that this step was not clearly stated.

SharePoint with Report Builder 1.0, Not Report Builder 2.0

During our testing of this solution, we created some reports using Report Builder 2.0 to display the user identity when connecting to our SQL Data servers in order to verify that Kerberos was configured and working properly. When we attempted to open one of these reports using the Report Builder ClickOnce application, we received the following error: “System.IO.StreamReader: The Report Element was not found.”

After a quick bit of research on the Internet using Bing, we learned that by default, the ClickOnce application opens Report Builder 1.0 rather than Report Builder 2.0. The solution is to install the add-in, Microsoft® SQL Server® 2008 Report Builder 2.0 ClickOnce for SharePoint in order to make Report Builder 2.0 ClickOnce available from a SharePoint site.

Note: This add-in is separate from the ClickOnce version of Report Builder 2.0 in SQL Server 2008 PCU1.

You then need to configure the default ClickOnce application in Report Manager and in SharePoint Central Administration to open Report Builder 2.0 rather than Report Builder 1.0. The following ReadMe and SQL Server Books Online articles explain the problem and solution:

- Microsoft SQL Server 2008 Report Builder 2.0 ClickOnce for SharePoint Readme
- How to: Set Report Builder 2.0 as the Default ClickOnce Report Builder Version

Summary

With proper planning, you can successfully you can successfully deploy a server farm using Microsoft Office SharePoint Server 2007, Microsoft SQL Server 2008 Reporting Services, and Microsoft Office PerformancePoint Server 2007 Monitoring Server, and you can configure Kerberos authentication for user authentication throughout the server farm. The use of virtual host names for each Web application with a unique application pool identity enables you to specify unique SPNs for each of these Web applications such that Internet Explorer clients can seamlessly connect to the resources within the server farm using their own authentication credentials.

Section 6: SQL Server Integration Services

Increasing Throughput of Pipelines by Splitting Synchronous Transformations into Multiple Tasks

Introduction

The Derived Column transformation is one of the most common data flow items used in Microsoft® SQL Server® Integration Services packages. Typically, existing column values are overwritten by an expression or a new column is added to the data flow using the Derived Column transformation.

Consider the situation where you need to change multiple columns in the pipeline. Because the Derived Column task is a synchronous task, splitting the work done into multiple Derived Column tasks will not add a new execution tree to the execution plan of the Integration Services package. This means that you would expect that using a single Derived Columns task to change multiple columns to have the same speed as using multiple Derived Column transformations to change the columns.

However, our testing has shown that if you want to change more than one column in a data flow by using a Derived Column task, there is a performance benefit to splitting that transformation into multiple Derived Column tasks. In this note we will show that doing many smaller pieces of work in multiple Derived Column tasks is significantly faster than doing all the work in a more complex work in a single Derived Column task.

Test Setup

We performed the tests using SQL Server 2008 R2 RC0 running on a 96-core Unisys ES7000 Server. To eliminate other variables, we used simple package design that reads from a flat file source, calculates data in one or more Derived Column tasks, and then sends the result to a Row Count task transformation.

Before we ran the Derived Column transformation, we needed a baseline of the throughput of reading a file without transformations. To measure this, we used the data source and a row count.

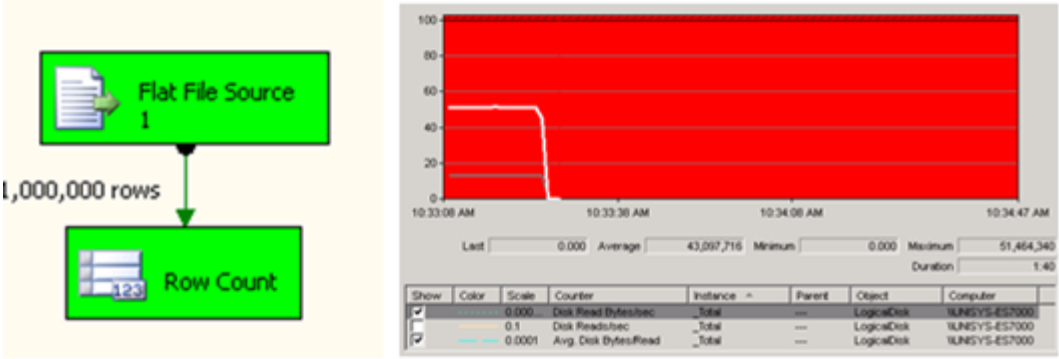


Fig. 1: Baseline without transformations

The input file contained 1 million rows with 96 columns and has a size of 1 GB. It took 23 seconds, with a throughput of 50 MB per second, to read the data into a row count.

The Integration Services flat file reader retrieved data with 128 KB Read IOPS, which was completed within 1 millisecond. The average CPU utilization of the the DTEXEC process was less than 5 percent. There was room to spare for our transformations, and there was no I/O bottleneck.

Next, we established the baseline for the Derived Column task. Initially, we tested with 96 transformations in a single Derived Column task. The package we ran looks like this.

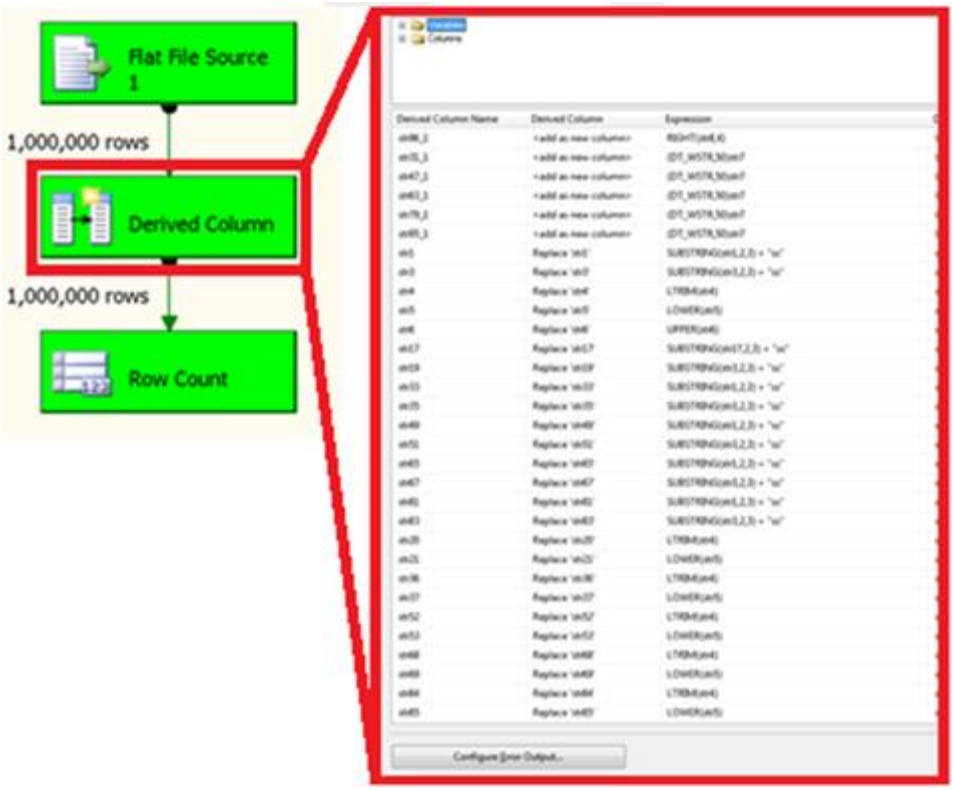


Fig. 2: One derived column task

The run time was 56 seconds, and throughput was 20 MB per second.

We then proceeded to split the transformations into more Derived Column tasks. For example, we ran a package like this.

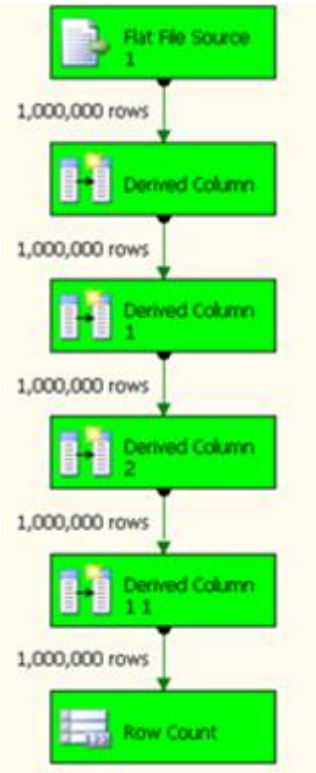


Fig. 3: Derived Column task package

Each Derived Column task included 24 tranformations, or a quarter of the original 96 transformations. The run time was 24 seconds, and throughput was 47 MB per second.The following table and graph show the tests we did with varying numbers of Derived Column tasks used to perform the 96 transformations.

	Row count only	**1 DC**	**2 DC**	**4 DC**	**8 DC**	**16 DC**	**32 DC**	**64 DC**	**96 DC**
Duration (seconds)	23	56	32	24	24	25	25	26	35
CPU Util% (avg)	5	0.25	0.354	0.289	0.315	0.263	0.204	0.223	11.84
DTEXEC.exe process CPU Util% (avg)	5	140	263	349	349	354	432	561	540
Effective throughput approx. (Read Bytes /sec)	50	20	36	47	47	47	47	32	32

Buffer Memory (bytes max)	41.940K	73.396K	73.396K	41.490K	41.940K	41.940K	52.426K	62.911K	73.396K
# Buffers in use (max)	5	8	10	10	10	10	13	16	17

** **DC** stands for derived column*

*** The tests were performed on a 96-core Unisys ES7000/7600R server*

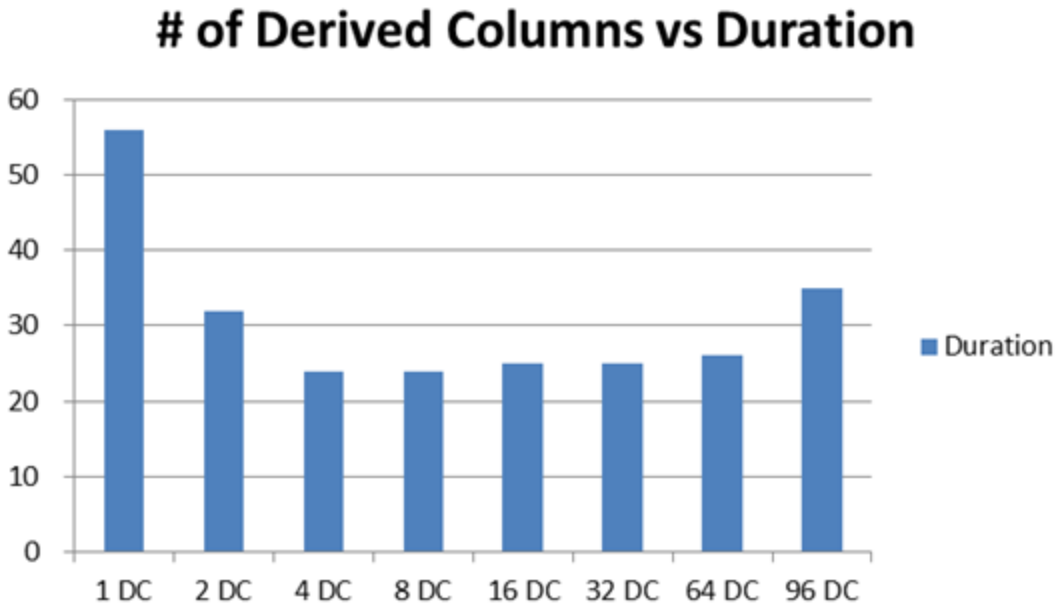


Fig. 4: Measuring derived columns (DC) and duration

Why This Works

Even inside a single execution tree in Integration Services you can increase parallelism. As long as source throughput keeps up, Integration Services will spawn multiple buffers (up to five for sources, more for asynchronous transformations) in a single execution tree.

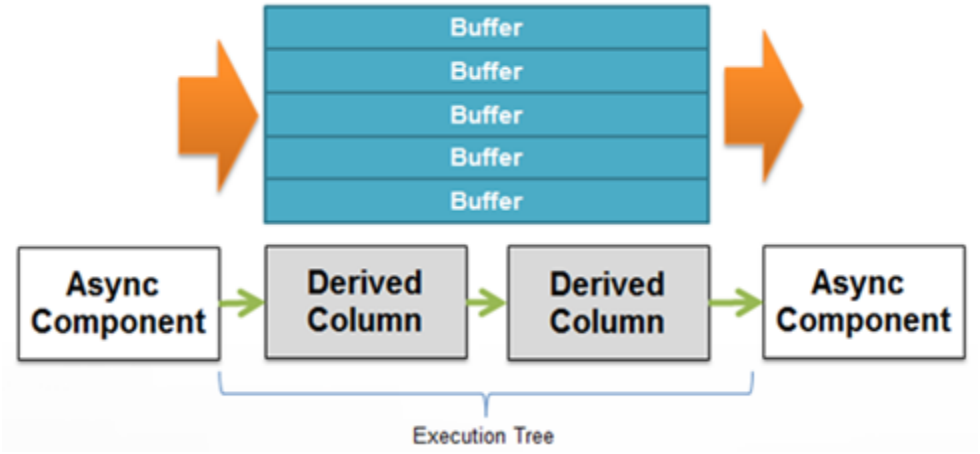


Fig. 5: Multiple buffers within an execution tree

The Derived Column task performs each column transformation serially inside a single task on the currently active buffer. However, when you split the column transformations into multiple Derived Column tasks, each individual column transformation can now be fully parallelized across multiple buffers. As you will notice from the test results, there is a sweet spot around four or five buffers where throughput reaches the peak. Because the data source cannot provide more than five buffers concurrently, five is also the maximum amount of added parallelism we can gain in this scenario by splitting into multiple Derived Column tasks. If you have a scenario with multiple sources, this sweet spot may be different in your case.

Conclusion

Synchronous tasks do NOT add a new execution tree to the execution plan of the Integration Services package – and hence, you would not expect synchronous tasks chained together to increase throughput of a pipeline. However, even inside one synchronous execution tree, you can increase parallelism by splitting the transformation into multiple task steps.

If you have a Derived Column task that contains complex operations, splitting the work in the Derived Column task into multiple tasks can improve overall throughput performance.

Moving Large Amounts of Data Between Oracle and SQL Server: Findings and Observations

Case

A large online gaming company tested a number of methods to quickly and efficiently move data between an Oracle 9i/UNIX system and SQL Server 2005. Their goal was to move 200-GB of daily user activity data within two hours. The data was to move without the use of transformations from Oracle's OLTP system to the SQL Server Operational Data Store (ODS) database. An ODS database is used in data warehouse architecture to hold an exact and intact copy of the source data, representing the only true data and history. The ODS data would be used later by SSIS packages to transform and load data into a staging database, a DW database, and an OLAP cube for reporting purposes.

Note This Technical Note does not describe a suggested method to fully migrate data from Oracle to SQL Server in all cases. Depending on the application, third-party tools and wizards may be a better fit for your migration.

Process

The company started by using a 64-bit server with 32 and 64 cores, testing both x64 and ia64. They installed SQL Server, Analysis Services and SSIS on that server. Since it was a 64-bit platform, the Microsoft OLE DB Provider for Oracle was ruled out because it does not yet support 64-bit environments. The project team decided to use Oracle's 10.2.0.2.2 version provider. Here are the initial tests and findings:

Solution 1: The first solution they tried was using the SQL Server Integration Services (SSIS) OLE DB Source task to move the Oracle 200-GB data into SQL Server. However, the speed was very slow and they could not meet their goal of two hours so they stopped the test. In this scenario, they used the 64-bit server and the 64-bit provider from Oracle and the projected time of completion was about 18 hours.

Solution 2: The next solution was to use a separate 32-bit server (8 cores, 20-GB RAM) for SSIS only and 32-bit Oracle's OLEDB Provider for Oracle. They downloaded the Oracle data into a binary file (using the network drive on the 64-bit server) by using the SSIS Raw File destination task. From there they uploaded the data into SQL Server staging tables, and then converted it into SQL ODS tables by using the SSIS Raw File source task.

Result: 2 hours 47 min elapsed.

Issue : Large staging file and staging DB (not ODS) size
Even though the size of the Oracle data was 200 GB, the binary file size was over 600 GB. Then, the 600 GB of data had to be uploaded into SQL Server tables without data conversion. After that, it was necessary to convert it into SQL Server target ODS

table format. The process required at least 1.4 terabytes of additional disk space for the binary file and staging tables.

Note Less overall time would have been required if they had used Oracle SQL*Plus commands to spool the data into a file. That way, they would not have had to use the slow provider to extract into a raw file. They did not have time to test this scenario but it is worth mentioning.

Solution 3: Next they tried using DataDirect Technologies' Oracle provider with wire protocol on the x64 server (it seems DataDirect does not have an ia64 provider). The achieved speed results were very impressive. In a comparison test, they loaded 10,000 records from Oracle to SQL Server. In that test, the DataDirect provider took 2 seconds versus 22 seconds when using the Oracle provider. However, the overall test failed because of conversion errors.

> **Issue:** They tried to retrieve the Oracle data by using the SSIS OLE component with DataDirect's provider (x64) but there was a code page conversion error for the SQL Server code page CP949 (Korean character set). They tried to use Unicode conversion by using an output column (in SSIS), but this did not work either. They plan to open a support case with DataDirect. It seems there are some special Korean characters that do not get mapped properly in CP949 and the driver fails.

Solution 4: The company finally decided to use the SSIS OLE DB source and target components to move data from Oracle to SQL Server directly through an 8-core 32-bit server using Oracle's32-bit OLEDB provider (ver. 10.2.0.2.2) and Korean_90_Bin collation (with the SQL Server ODS database). When Korean_90_bin is used, they do not have to convert Korean data explicitly in Oracle, thus reducing response time.

> **Result:** The response time was 2 hr 33m 32 sec (without the need of an additional 1.4 terabytes of disk space).

Findings

- Oracle OLEDB Provider performance appears to be better optimized on 32-bit compared to x64 or IA64. (Oracle driver speed: x86 > x64 > IA64). So, the company had to use 32-bit middle server to meet the target time.
- It seems that the 32-bit middle server's spec is important to the data move time. Based on this test result, if they used a 32-bit server with more than 8 cores, it is possible they could reduce the time even more. But, based on the tests, when they went from 8 cores to 16 cores, the difference was small. The total time went from 2h33min to 2h11min.
- To achieve even better performance, they used the Fast Load option under the Data Access mode of the OLE DB Destination task. They decided not to use the SQL Server destination, even though it is supposed to be faster, because it requires that SSIS and SQL Server be on the same server. By using the OLE DB Destination, they have the option to deploy the packages on a separate SSIS server.
- Regarding the Oracle to SQL ODS step (using a direct load via OLEDB and an x86 server), they found that performance on a 16 dual-core x64 is better than on a 32 dual-core x64. (The gap is 10min.) The x64 server hosts SQL Server and the ODS database.

Detailed Test Results

Following are the test results on an HP superdome IA64 with 16- and 32-way plus a Unisys ES7000 16-way and 32-way x64 based on EMC and Hitachi storage. The SQL Database DW size is 47 GB and the

Analysis Services database with 20 cubes is 19 GB. It appears that a Unisys x64 32-way EMC may be the right option for their requirements.

Unisys x64

Case	Step	Server	Elapsed Time	CPU	
				DW	Middle
case1	Oracle ODS -> Staging ODS (With SSISOLE)	32way - HDS	2:53:34	AVG 7% MAX 11%	AVG 81% MAX 99%
		16way - HDS	2:52:02	AVG 13% MAX 27%	AVG 82% MAX 99%
		32way - EMC	2:43:35	AVG 7% MAX 11%	AVG 86% MAX 100%
		16way - EMC	2:33:32	AVG 15% MAX 27%	AVG 92% MAX 100%
case2	ETL (Fact, Mart)	32way - HDS	1:35:30	AVG 23% MAX 44%	
		16way - HDS	1:42:57	AVG 36% MAX 76%	
		32way - EMC	1:34:13	AVG 28% MAX 50%	
		16way - EMC	1:40:34	AVG 37% MAX 74%	
case3	Cube build	32way - HDS	0:53:40	AVG 13% MAX 42%	
		16way - HDS	0:56:03	AVG 27% MAX 72%	
		32way - EMC	0:52:29	AVG 16% MAX 44%	
		16way - EMC	0:55:13	AVG 27% MAX 72%	

HP IA64

Case	Step	Server	Elapsed Time	CPU	
				DW	Middle
case1	Oracle ODS -> Staging ODS (with Raw files)	32way - HDS	2:53:00	AVG : 13% MAX : 44%	AVG : 71% MAX : 99%
		16way - HDS	2:47:07	AVG : 25% MAX : 87%	AVG : 75% MAX : 99%
		32way - EMC	2:51:58	AVG : 16% MAX : 42%	AVG : 77% MAX : 100%
		16way - EMC	2:52:38	AVG : 27% MAX : 85%	AVG : 73% MAX : 99%

case2	ETL (Fact, Mart)	32way - HDS	1:52:14	AVG : 18% MAX : 45%	
		16way - HDS	2:05:24	AVG : 30% MAX : 82%	
		32way - EMC	1:55:19	AVG : 20% MAX : 50%	
		16way - EMC	1:57:37	AVG : 34% MAX : 81%	
case3	Cube build	32way - HDS	1:03:07	AVG : 13% MAX : 41%	
		16way - HDS	1:00:29	AVG : 25% MAX : 84%	
		32way - EMC	1:01:01	AVG : 13% MAX : 38%	
		16way - EMC	59:01	AVG : 25% MAX : 96%	

Conclusion

Based on these results for the specific requirements of this company it appears the best solution is Solution 4. The company plans to get the following servers to maximize price/performance ratio:

- A 4 dual-core 32-bit server with 20-GB of RAM using Oracle's latest 32-bit OLE DB Provider as a dedicated SSIS server
- An 8 dual-core x64 server with 32-GB of RAM and EMC SAN to host SQL Server RBMS and Analysis Services

SSIS Package For Collecting SSAS DMV Data

SSIS Package For Collecting SSAS DMV Data at http://www.codeplex.com/SQLSrvAnalysisSrvcs

This project on Codeplex includes an SSIS package along with its configuration file to collect Analysis Services Dynamic Management View data by means of a SQL Agent job and store the collected data in tables in a database called DMV. These scripts were extracted from the A Solution for Collecting Analysis Services Performance Data for Performance Analysis project in the same codeplex release and are intended to demonstrate, in a stand-alone fashion, how to periodically query your Analysis Services instance using DMVs to determine the current state of your server - collecting connection, session and command information.

The “Balanced Data Distributor” for SSIS

There is a new transform component available for SQL Server Integration Services. It’s called the Balanced Data Distributor (BDD) and the download is available here. The BDD provides an easy way to amp up your usage of multi-processor and multi-core servers by introducing parallelism in the data flow of an SSIS package.

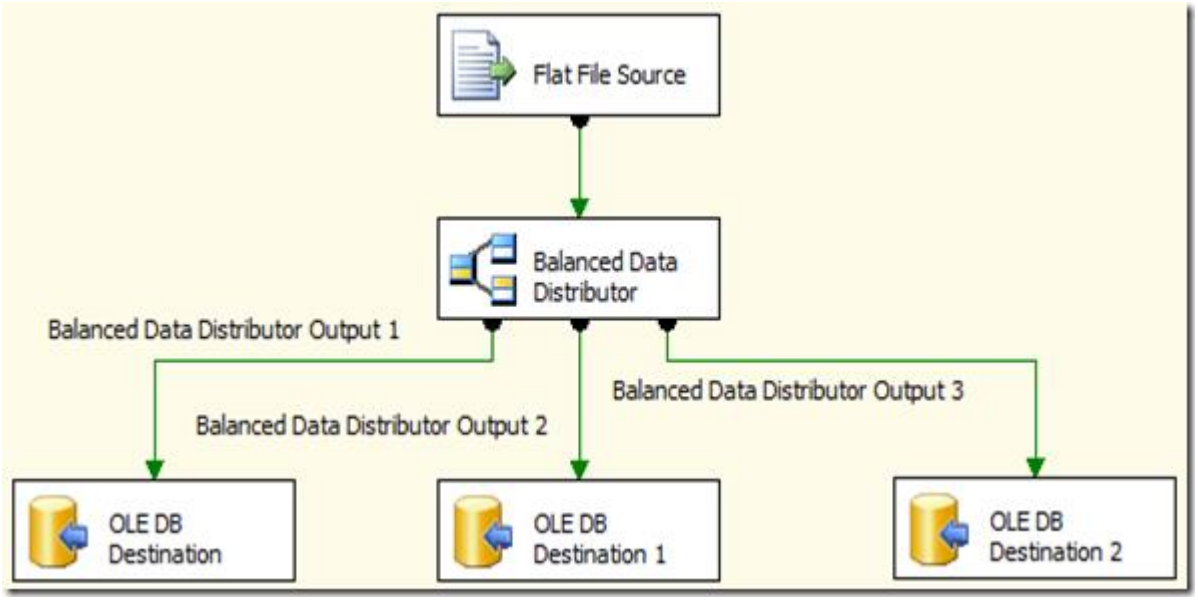
Functionality of the BDD

The functionality of the BDD is very simple: It takes its input data and routes it in equal proportions to its outputs, however many there are. If you have four outputs, roughly ¼ of the input rows will go to each output. Instead of routing individual rows, the BDD operates on buffers of data, so it’s very efficient.

Some of you will already be noticing that there is no transformational value in the BDD, and no control over which data rows go to which output. You may be wondering, what the heck is the value of that?

The value of the BDD comes from the way modern servers work: Parallelism. When there are independent segments of an SSIS data flow, SSIS can distribute the work over multiple threads. BDD provides an easy way to create independent segments.

This diagram gives a trivial example:



If you run this data flow on a laptop, there probably won’t be any speed advantage, and there may even be a speed cost. But suppose you run this on a server with multiple cores and many disk spindles supporting the destination database. Then there might be a substantial speed advantage to using this data flow.

When to use the BDD

Using the BDD requires an understanding of the hardware you will be running on, the performance of your data flow and the nature of the data involved. Therefore it won't be for everyone, but for those who are willing to think through these things there can be significant benefits. Here is my summary description of when to use BDD:

1. There is a large amount of data coming in.

2. The data can be read faster than the rest of the data flow can process it, either because there is significant transformation work to do or because the destination is the bottleneck. If the destination is the bottleneck, it must be parallelizable.

3. There is no ordering dependency in the data rows. For example if the data needs to stay sorted, don't go and split it up using BDD.

Relieving bottlenecks in the SSIS Data Flow

Let's talk about bottlenecks, since changing bottlenecks is what BDD is all about. A bottleneck is whatever limits the performance of the system. In general there are three places that could be the bottleneck in an SSIS data flow: The source, the transformations, or the destination.

Bottlenecks in the Source

If the limiting factor is the rate at which data can be read from the source, then the BDD is not going to help. It would be better to look for ways to parallelize right from the source.

Bottlenecks in the Transformations

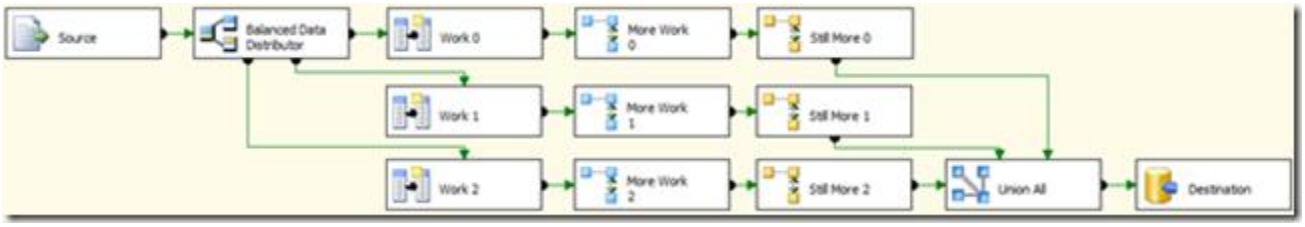
If the limiting factor is the transformation work being done in the data flow, BDD can help. Imagine that there are some lookups, derived columns, fuzzy lookups and so on: These could easily be the components limiting performance. Make two or four or eight copies of the transformations, and split the data over them using the BDD. Let the processing run in parallel. If there are several transformations in the data flow, put as much as you can after the BDD, to get more things running in parallel.

Bottlenecks in the Destination

If the limiting factor is the destination, BDD might be able to help - you need to determine whether the destination can be run in parallel. You might be surprised at some times when it can. One example is when loading data into a simple heap (table with no indexes) in SQL Server. With the database properly distributed over a number of disks, it is quite possible to load in parallel with good performance. When working on the ETL World Record a while ago, we used a heap for a side experiment and found that loading 56 streams concurrently into a single heap was almost as fast as loading 56 streams into 56 independent tables. Many sites already drop or disable their indexes during data loading, so this could be more of a freebie than you would expect. More recently we saw a benefit from parallel loading into SQL Server Parallel Data Warehouse (PDW). PDW is an architecture designed for parallelism!

When the destination does not support parallel loading

A final case to consider is when the limiting factor is the transformation work being done in the data flow but the destination cannot receive data in parallel for some reason. In this case, consider using BDD to parallelize the transforms followed by a Union All to rejoin the data into a single flow; then a single destination can be used. Here is an illustration:



Best practice – balanced!

One final note: Whatever you put behind the BDD, be sure the same work is being done on all paths. It doesn't make logical sense to have the paths be different, and from a performance point of view, you want them all to be the same speed. Remember, the "B" in BDD stands for "Balanced".

Conclusion

Someday maybe SSIS will be able to do the work of the BDD automatically, but for now you have an easy way to amp up your usage of multi-processor and multi-core servers by introducing parallelism in the data flow of an SSIS package.

Section 7: SQL Top 10

Top 10 Performance and Productivity Reasons to Use SQL Server 2008 for Your Business Intelligence Solutions

Microsoft SQL Server 2008 provides new functionality not found in previous versions and numerous performance and usability enhancements that are specifically designed for business intelligence (BI) solutions. These features are designed to improve performance, increase developer productivity, and enrich the end-user experience. The intent of this article is to provide a list, with some details, of the top performance reasons to use SQL Server 2008 for your new business intelligence solutions and to upgrade to SQL Server 2008 for your existing business intelligence solutions. The reasons begin with the performance enhancements that do not require modifications to your existing business intelligence solutions and then proceed to technology.

Important: For information about upgrading your existing Business Intelligence solutions to SQL Server 2008, see the SQL Server 2008 Upgrade Technical Reference Guide.

1. Scalability and Resource Management Enhancements in Reporting Services

With SQL Server 2005 Reporting Services, the reporting engine has limited ability to exercise control over resource management. As a result, very large reports and heavy workloads by a large number of concurrent users can result in out-of-memory conditions. In addition, with SQL Server 2005 Reporting Services, long reports can take a long time to render, because every page is processed before the first page is returned to the user.

With SQL Server 2008 Reporting Services, the Report Engine has been redesigned to enable Reporting Services to control resources (threads, memory and state) to ensure that sufficient memory is always available to execute very large reports and heavy workloads from a large number of concurrent users (by paging and releasing memory in response to memory pressure). This is accomplished by:

- Embedding HTTP capabilities into the Report Engine and removing the dependency of Reporting Services on Internet Information Services (IIS). As a result, configuring Reporting Services is much easier because it is isolated from other Web applications.
- Calculating report items on demand.
- Rendering only the pages requested by users (unless the total page count is referenced within the report), which improves first page response time.

In working with SQL Server 2008 Reporting Services in our lab, SQL Server 2008 Reporting Services was able to respond to 3–4 times the total number of users and their requests on the same hardware without HTTP 503 Service Is Unavailable errors compared with SQL Server 2005 Reporting Services, regardless of the type of renderer. In stark contrast, SQL Server 2005 Reporting Services generated excessive HTTP 503 Service Is Unavailable errors as the number of users and their requests increased, regardless of the report renderer.

Our tests clearly demonstrated that the new resource management architecture of the report server enables SQL Server 2008 Reporting Services to scale very well, particularly on the new four-processor, quad-core processors. With our test workload, SQL Server 2008 Reporting Services consistently outperformed SQL Server 2005 with the PDF and XLS renderers on the four-processor, quad-core hardware platform (16 cores) both in terms of response time and in terms of total throughput. As a result, we recommend with SQL Server 2008 Reporting Services that you scale up to four-processor, quad-core servers for performance and scale out to a two-node deployment for high availability. Thereafter, as demand for more capacity occurs, add more four-processor, quad-core servers. With SQL Server 2005

Reporting Services, we recommend that you scale out above four processors due to resource bottlenecks. Our tests clearly demonstrate that SQL Server 2008 Reporting Services utilizes your hardware resources more efficiently than SQL Server 2005 Reporting Services, particularly with the new four-processor, quad-core servers. For more information, see Scaling Up Reporting Services 2008 vs. Reporting Services 2005: Lessons Learned.

2. Subspace Computations in Analysis Services

The SQL Server Analysis Services formula engine devises a query execution plan to retrieve data requested by a query. Each node of the query execution plan utilizes one of two types of evaluation modes: a subspace computation mode or a cell-by-cell evaluation mode. If the subspace computation mode is used, the Analysis Services formula engine operates only on cells with data that contribute to the result. If the cell-by-cell evaluation mode is used, the Analysis Services formula engine operates on each cell that theoretically could contribute to the result, regardless of whether it contains data. In a sparse cube, which is typical, retrieving data using the cell-by-cell evaluation mode is inefficient for calculation-intensive queries, because many null cells (which do not contribute to the result) are operated on.

In SQL Server 2005, before Service Pack 2 (SP2), the query execution plans devised by the Analysis Services formula engine primarily contained nodes that utilized the cell-by-cell evaluation mode. In SP2 of SQL Server 2005, a small number of MDX functions support the subspace computation mode. In SQL Server 2008 Analysis Services, the vast majority of MDX functions use the subspace computation mode. In working with our customers with SQL Server 2008 Analysis Services, we have observed tremendous performance gains with the following:

- Sparsely populated cubes
- Cubes with complex calculations

These performance gains with SQL Server 2008 Analysis Services are achieved without modification of your business intelligence solution. In addition, you can frequently further increase the performance of cubes with complex calculations by performing additional tuning of your MDX calculations in the cube. For more information, see Performance Improvements in MDX in SQL Server 2008 Analysis Services and the SQL Server 2008 Analysis Services Performance Guide.

3. Pipeline Parallelism in Integration Services

In SQL Server 2005 Integration Services, each execution tree in a DataFlow task within a package is generally assigned a single worker thread, and, under certain conditions, each execution tree actually shares a thread with other execution trees. This approach has the following benefits:

- All thread scheduling is done during the pre-execution phase, rather than consuming time during the execution of the package.
- The buffer's data stays in cache because a thread runs a full buffer over a complete execution tree.

However, this approach has the following downsides that frequently outweigh the preceding benefits, particularly with long or branched execution trees:

- The scheduling of threads is not always optimal, because the relative amount of work for each execution tree is not known before execution.
- Not all cores are utilized on a multiprocessor computer if there are fewer execution trees than processor cores. The lack of utilization of all cores is particularly an issue if an execution tree has many synchronous

components, because in this scenario all of the synchronous components share a single thread, and they can become processor bound on a single core. This same issue occurs if you split the data into multiple paths by using Multicast, because Multicast is a synchronous component. As a result, even though it visually appears that you are parallelizing your package, the paths are executed serially, because the output paths belong to the same execution tree and only a single thread is allocated to the execution tree.

Note: Inserting an asynchronous transformation in the data flow to increase parallelism by creating a second execution tree (a performance design trick on high-performance multiprocessor computers) has performance overhead, because all data must be copied for the new execution tree.

In SQL Server 2008 Integration Services, pipeline parallelism has been enhanced with architecture improvements to the data flow engine. Now worker threads are dynamically assigned to individual components from a common threadpool. As a result, the data flow engine automatically utilizes all cores on multicore computers without the need to resort to design tricks. The increased pipeline parallelism in SQL Server 2008 Integration Services from threadpooling increases the speed at which packages can execute, and it makes better use of your existing resources, provided there are no other resource bottlenecks (such as I/O or memory). In working with our customers with SQL Server 2008 Integration Services, we learned that packages with long chains of synchronous transformations and most packages running on multiprocessor computers saw significant performance increases.

Note: For more information about SQL Server 2008 Integration Services from the SQL Server Integration Services team, see the SSIS Team Blog. For additional performance information, see ETL World Record and Top 10 SQL Server Integration Services Best Practices.

4. Scalability and Availability Enhancements in Analysis Services

In SQL Server 2005, backups of large Analysis Services databases have limited scalability, and Analysis Services supports only a single location for all Analysis Services databases within an instance of Analysis Services, although measure group partitions within a database can be distributed across multiple drives to distribute the I/O. Furthermore, after processing a single database for an Analysis Services instance on a processing server, you cannot detach the processed database and then attach it to a query server for increased scalability and availability. In addition, in SQL Server 2005 Analysis Services, attaching read-only copy of an Analysis Services database to multiple servers for increased scalability and availability is not natively supported; this capability is supported only through the use the SAN snapshot technologies (see Scale-Out Querying with Analysis Services Using SAN Snapshots).

In SQL Server 2008 Analysis Services, these issues are resolved with the following scalability and availability enhancements:

- You can back up an Analysis Services database of any size. The performance of Analysis Services database backups scales linearly, and this performance is comparable to file-based backups.
- You can deploy each database within an Analysis Services instance to its own storage location, enabling you to place each Analysis Service database within an instance on separate drives.
- You can detach an entire database from an Analysis Services instance, copy or move it to a new location, and then attach it to another Analysis Services instance. This feature increases scalability and performance by enabling you to process new data into an Analysis Services database on a processing server while users query the current version of the Analysis Services database on a query server. After processing is complete, you can detach the processed database from a processing server and then attach it to the query server. Furthermore, if you utilize two Analysis Services instances on the query server and use a load balancer to

distribute queries to the appropriate instance, you can direct new user connections to the newly attached database without affecting availability, because currently connected users can complete their currently running queries against the previously attached database. This feature also enables you to detach a database from an Analysis Services instance for archiving or backup purposes.

- You can attach an Analysis Services database in read-only mode to multiple Analysis Services instances for increased scalability and availability. The read-only database can be stored on a SAN and mounted to multiple instances, or it can be shared via a file system share and mounted to multiple instances.

Note: With large databases in which history partitions do not change, you will want to use high-speed SAN copy capabilities to clone the detached database or use Robocopy. For information about using Robocopy, see Sample Robocopy Script to customer synchronize Analysis Services databases.

The following diagram illustrates the use of these new features with a single processing server attached to a database in read/write mode and four query servers attached to a single read-only database. A load balancer distributes queries among the four query servers.

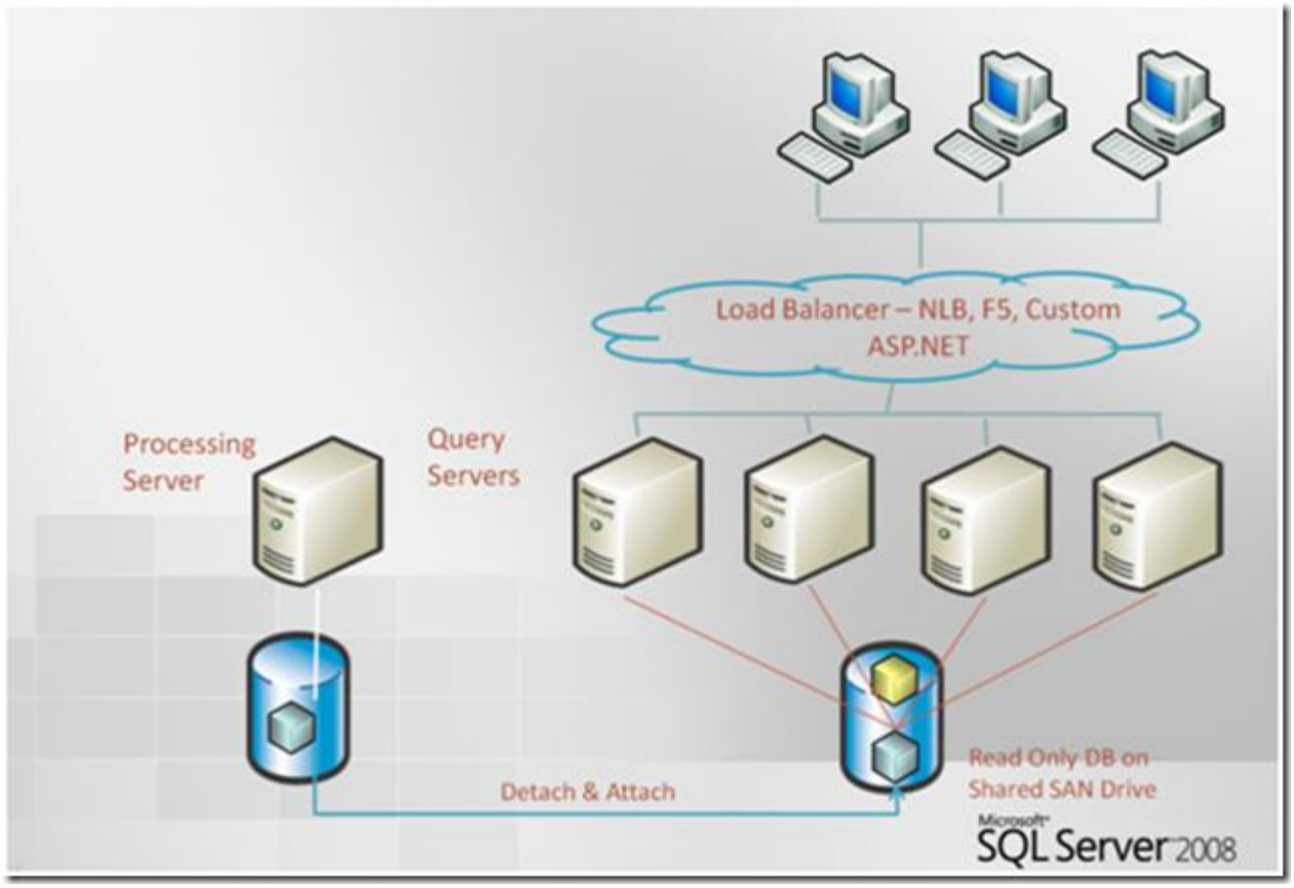


Figure 1: Query Scale-Out using Read-only Databases

5. Optimized Design Experience in Analysis Services

You design SQL Server Analysis Services solutions in Business Intelligence Development Studio (BIDS). In SQL Server 2005, when you design your Analysis Services solution in Business Intelligence Development Studio, it is too easy to

design a solution that does not follow best practices performance guidelines and as a result have performance issues. For example, the failure to design appropriate attribute relationships is a significant design and performance issue, but SQL Server 2005 Business Intelligence Development Studio does not notify you if you fail to create such relationships. In addition, with SQL Server 2005 Analysis Services, you frequently have to manually design aggregations to achieve optimum performance.

In SQL Server 2008 Analysis Services, the Business Intelligence Development Studio development environment includes the following improvements to promote best practices and help developers design high-performing solutions:

- Best practice alerts called AMO warnings have been added to the AMO object model. These warning messages surface in Business Intelligence Development Studio to alert developers when their designs are not consistent with any of over 40 best practices. These warnings are integrated into real-time designer checks through the use of blue squiggly alert lines, and they providea nonintrusive (and dismissible) way for developers to detect potential problems with their design long before deployment. You can also use this new capability to review your existing SQL Server 2005 Analysis Services cubes for compliance with design best practices guidelines, either by opening your Analysis Services database in the SQL Server 2008 version of Business Intelligence Development Studio or by running the newest version of the SQL Server 2005 Best Practices Analyzer (which incorporates these same AMO warnings). You can also access these AMO warnings from PowerShell or a custom client application.
- A new attribute relationship designer has been added for viewing and editing attribute relationships, with built-in validations to help in creating optimal dimension designs.
- The dimension editor has been streamlined with a single dimension wizard, better detection and classification of dimension attributes and member properties, and the automatic detection of parent-child relationships.
- A new aggregation designer has been added within Business Intelligence Development Studio with a new algorithm for creating new aggregations. The aggregation designer is optimized to work with usage driven aggregations. You can now easily look at your existing aggregations, add to those aggregations, or remove and replace them. Intelligent support is provided to help with merging existing and new aggregation designs. For more information about the new aggregation designer and using it with usage-driven aggregations, see Reintroducing Usage-Based Optimization in SQL Server 2008 Analysis Services.

6. New and Enhanced Authoring Capabilities and Visualizations in Reporting Services

In SQL Server 2005 Reporting Services, you can create powerful and functional charts and graphs that enable you to hyperlink or drill down to other reports and Web sites. You can also integrate these reports and graphs into dashboards. However, making these reports and graphs pop and sizzle is sometimes difficult and time-consuming.

SQL Server 2008 Reporting Services offers many authoring enhancements that increase developer productivity. In addition, full-featured versions of the Dundas chart and gauge controls are now included, giving you the ability to create beautifully rendered charts and gauges.

Authoring Enhancements

In addition, with SQL Server 2008 Reporting Services, there are many additional authoring enhancements. These include:

- A new data representation format called Tablix. By using Tablix,you can now author reports with any structure, with the best of table merged with the best of matrix.
- Enhanced formatting capabilities. You can now create text boxes with mixed styles, paragraphs, and hyperlinks, and you can use expressions to implement inline formatting.
- Improved design experience. The report designer in Microsoft Visual Studio is redesigned to simplify the design experience.
- The Report Builder 2.0 authoring environment. This stand-alone authoring environment provides the intuitive and familiar Microsoft Office look.

Chart Enhancements

Chart enhancements include scale breaks, annotations, custom color palettes, merged charts, and multiple axes. New chart types include the following types of charts:

- Stepped Line
- Spline Area
- Pie Callout
- Polar
- Radar
- Gantt
- Range Line/Column
- Funnel
- Pyramid
- Histogram
- Box

Gauges

The new gauges include the following gauge types:

- Circular
- Linear
- Angular
- Thermometer
- Numeric Indicators
- State Indicators

For more information, see Reporting Services and What's New (Reporting Services).

7. Data Compression in the Relational Engine

The disk I/O subsystem is one of the most common bottlenecks for many relational data warehouse implementations. Additional disks are frequently added to reduce read/write latencies. This can be expensive, especially on high-performing storage systems. At the same time, the need for storage space continues to increase due to rapid growth of the data, as does the cost of managing databases (backup, restore, transfer, and so on).

In SQL Server 2008, data compression addresses some of these problems. With data compression, you can selectively compress any table, table partition, or index, resulting in a smaller on-disk footprint, smaller working-set memory size, and reduced I/O. Working with customers, we have seen that enabling data compression has resulted in a 50-80% saving in disk space in many environments. This savings in disk space directly translates into less money being required for high speed disk storage. Relational data warehouses with I/O bottlenecks and without processor saturation may also see an increase in overall performance.

SQL Server supports two types of data compression: row compression, which compresses the individual columns of a table, and page compression, which compresses data pages using row, prefix, and dictionary compression. The compression results are highly dependent on the data types and data contained in the database; however, in general we've observed that using row compression results in lower processor overhead but saves less space. Page compression, on the other hand, results in higher processor overhead, but it results in much larger space savings. Page compression is a superset of row compression, implying that an object or partition of an object that is compressed using page compression also has row compression applied to it. Compressed pages remain compressed in memory until rows or columns on the pages are accessed.

Both row and page compression can be applied to a table or index in an online mode without interruption to the application availability, but this compression can take a long time. Partitions of a partitioned table cannot be compressed or uncompressed online, however. In our testing we found that using a hybrid approach where only the largest few tables were compressed resulted in the best overall performance – this approach saved significant disk space but had a minimal impact on performance. Because additional disk space is required during the compression process, we found that it is best begin by compressing the smallest of the objects that you choose to compress. By doing so, you utilize the least disk space during the compression process and avoid running out of disk space during the compression process.

To determine how compressing an object will affect its size, you can use the **sp_estimate_data_compression_savings** system stored procedure. Database compression is supported only in the SQL Server 2008 Enterprise and Developer editions. It is fully controlled at the database level, and it does not require any application change.

8. Resource Governor in the Relational Engine

Maintaining a consistent level of service by preventing runaway queries and guaranteeing resources for mission-critical workloads has been a challenge for the SQL Server relational engine. In the past there was easy way to guarantee a certain amount of resources to a set of queries and to prioritize access to SQL Server relational engine resources – all queries had equal access to all the available relational engine resources.

For example:

- A single large data warehouse query can be granted as much as 25% of workspace memory; three concurrent large queries of this type will block any additional large queries.
- Specific workloads benefit from reduced parallelism. During partition processing, SQL Server Analysis Services may issue a large number of concurrent queries, and for best performance, we recommend that you configure your Analysis Services processing jobs to issue one query per core on the relational data warehouse computer. However, if the relational engine parallelizes each of these processing queries, threads may thrash. One solution is to run each processing query using MAXDOP=1. However, before SQL Server 2008, this was a server-wide setting that affected all queries, because you could not add the MAXDOP query

hint to Analysis Services processing queries. Using MAXDOP=1 for all queries negatively affected queries that derived substantial benefit from parallelization, forcing you to choose which type of query to optimize.

In SQL Server 2008, Resource Governor addresses these issues by enabling users to differentiate workloads, allocating resources as they are requested, controlling parallelism for particular workloads (for example, by user name or computer name), and substantially increasing concurrency with resource intensive queries. The Resource Governor limits can easily be reconfigured in real time with minimal impact on currently executing workloads. The allocation of the workload to a resource pool is configurable at the connection level, and the process is completely transparent to applications. This new functionality is particularly useful on systems with large memory and many cores.

The diagram below depicts the resource allocation process. In this scenario, three workload pools (the Admin workload, the OLTP workload, and the Report workload) are configured, and the OLTP workload pool is assigned a high priority. In parallel, two resource pools (the Admin pool and the Application pool) are configured with specific memory and processor (CPU) limits as shown. Finally, the Admin workload is assigned to the Admin pool and the OLTP and Report workloads are assigned to the Application pool.

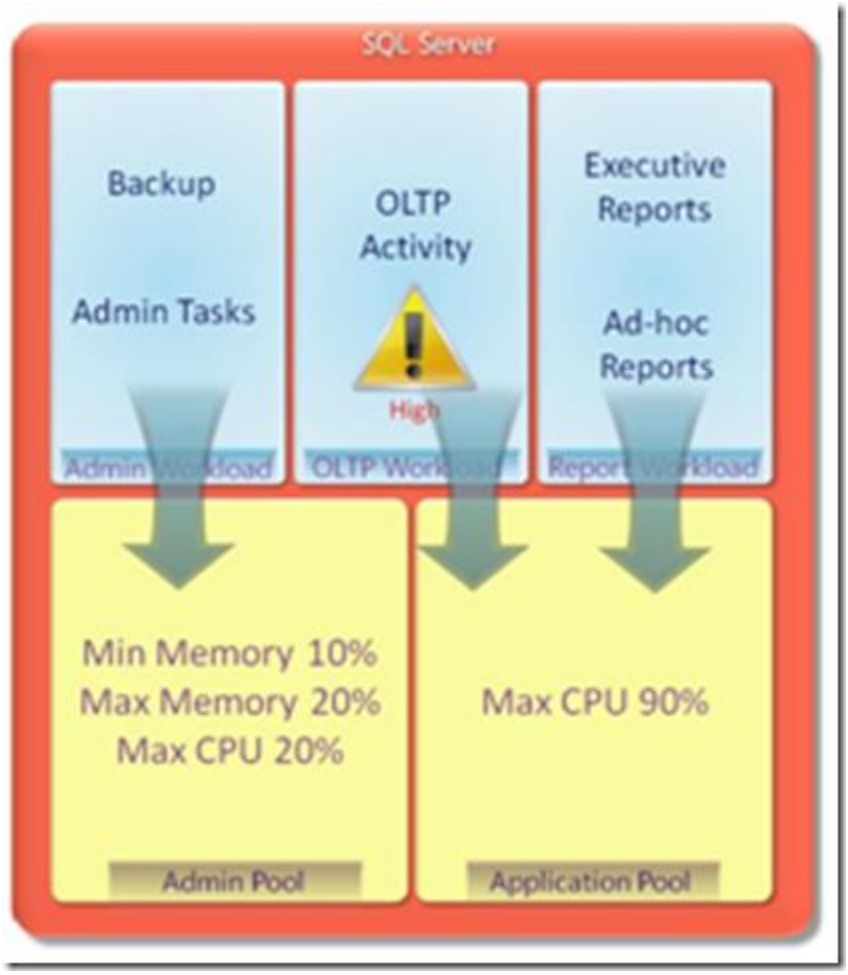


Figure 2: Using Resource Governor to Allocate Resources

Below are some other points you need to consider when you use Resource Governor:

- ·Because Resource Governor relies on login credentials, the host name, or the application name as a resource pool identifier, ISV applications that use a single login to connect multiple application users to SQL Server cannot use Resource Governor without reworking the application. This rework would require the application to utilize one of the resource identifiers from within the application differentiate workloads.
- Database level object grouping, in which resource governing is done based on the database objects being referenced, is not supported in the current version.
- Resource Governor allows resource management within a single SQL Server instance only. For multiple instances, consider using Windows System Resource Manager.
- Only processor and memory resources can be configured – I/O resources cannot be controlled.
- Dynamically switching workloads between resource pools after a connection is made is not possible in the current version.
- Resource Governor is only supported in SQL Server 2008 Enterprise and Developer editions and can only be used for the SQL Server Database Engine; resource usage by SQL Server Analysis Services, SQL Server Integration Services, and SQL Server Reporting Services cannot be controlled by Resource Governor. For these other SQL Server services, consider using Windows System Resource Manager.

9. Minimally Logged Operations in the Relational Engine

While bulk load operations have substantial performance benefits over row-by-row operations, the logging of these bulk load operations (typically inserts) in high-volume data loading scenarios is frequently the cause of disk I/O bottlenecks. Unlike fully logged operations, which use the transaction log to keep track of every changed row, minimally logged operations keep track of extent and page allocations and metadata changes only. Because much less information is tracked in the transaction log, a minimally logged operation is often faster than a fully logged operation if logging is the bottleneck. Furthermore, because fewer writes go the transaction log, a much smaller log file with a lighter I/O requirement becomes viable. Minimally logged operations are available only if your database is in bulk-logged or simple recovery mode. For more information, see Operations That Can Be Minimally Logged. Note that performing a bulk operation in a bulk-logged database has impact on the backup strategy for the database. Additionally, the rollback of a minimally logged operation is fully logged, which can result in the rollback of a minimally logged operation taking significantly longer than the originally logged operation. For more information about the implications, see Backup Under the Bulk-Logged Recovery Model .

SQL Server 2008 introduces INSERT…SELECT as a new way to perform minimally logged insert operations, allowing Transact-SQL based INSERT statements to be minimally logged under certain circumstances:

- Heaps – An INSERT statement that takes its rows from a SELECT operation and inserts them into a heap is minimally logged if a WITH (TABLOCK) hint is used on the destination table.
- Clustered index – Whether an INSERT… SELECT operation into clustered indexes is minimally logged depends on the state of trace flag 610. Not every row inserted in a clustered index with trace flag 610 is minimally logged. If the bulk load operation causes a new page to be allocated within an extent, all of the rows sequentially filling that new page are minimally logged. Rows inserted into pages that are allocated before the bulk load operation occurs are still fully logged, as are rows that are moved as a result of page splits during the load. This means that for some tables, you may still get some fully logged inserts. If trace flag 610 causes minimal logging to occur, you should generally see a performance improvement. But as always with trace flags, make sure you test for your specific environment and workload. Consider these two examples:

Example 1: You have a table clustered in a key that contains even integer key values 0-16. The table has four leaf pages, the pages are not full, and they can hold two more rows on each page. You bulk load eight new rows, with uneven key values 1-15. The new rows fit in the existing pages. The illustration below shows how this table will look before and after the load operation.

Figure 3: A fully logged insert under trace flag 610

In this example, no new pages are allocated and trace flag 610 will not give you any minimal logging.

Example 2: Consider an alternative scenario: The table initially now has two pages, both full, containing the key values 0-7. You bulk load rows with key values 8-16.

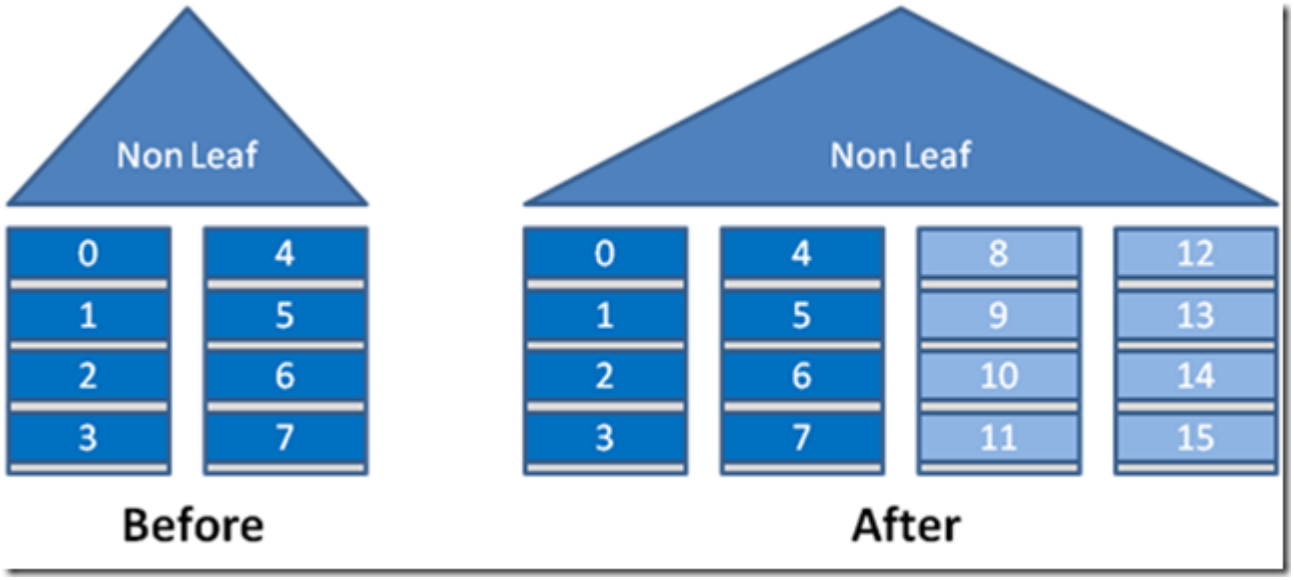


Figure 4: A minimally logged insert under trace flag 610

In this example, the pages holding key values 8-15 (in light blue above) will be minimally logged with trace flag 610.

For more information on using this new functionality to improve data loading performance, see The Data Loading Performance Guide.

10. Other Great Features

SQL Server 2008 introduces the following additional features that improve the performance of business intelligence solutions.

Star Query Enhancements

Most data warehouse queries are designed to follow a star schema and can process hundreds of millions of rows in a single query. By default, the query optimizer detects queries against star schemas and creates efficient query plans for them. One method the optimizer can use to generate an efficient plan is to use bitmap filtering. A bitmap filter uses a compact representation of a set of values from a table in one part of the operator tree to filter rows from a second table in another part of the tree. Essentially, the filter performs a semi-join reduction; that is, only the rows in the second table that qualify for the join to the first table are processed. In SQL Server 2008, bitmap filtering can be introduced in the query plan after optimization, as in SQL Server 2005, or introduced dynamically by the query optimizer during query plan generation. If the filter is introduced dynamically, it is referred to as an optimized bitmap filter. Optimized bitmap filtering can significantly improve the performance of data warehouse queries that use star schemas by removing nonqualifying rows from the fact table early in the query plan. Without optimized bitmap filtering, all rows in the fact table are processed through some part of the operator tree before the join operation with the dimension tables removes the nonqualifying rows. If optimized bitmap filtering is applied, the nonqualifying rows in the fact table are eliminated immediately. Optimized bitmap filtering is available only in the Enterprise, Developer, and Evaluation editions of SQL Server. For more information, see Optimizing Data Warehouse Query Performance Through Bitmap Filtering.

Partitioned Table Parallelism

SQL Server 2008 improves query processing performance on partitioned tables for many parallel plans, changes the way parallel and serial plans are represented, and enhances the partitioning information provided in both compile-time and run-time execution plans. SQL Server 2005 uses a single thread per partition parallel query execution strategy, but SQL Server 2008 can allocate multiple threads to a single partition. This improvement is of particular importance to data warehouse environments; fact tables are often candidates for partitioning, because they typically contain a few columns with a very large number of records. For more information, see Query Processing Enhancements on Partitioned Tables and Indexes.

The MERGE Statement

In SQL Server 2008, you can perform multiple data manipulation language (DML) operations in a single statement by using the MERGE statement. For example, you may need to synchronize two tables by inserting, updating, or deleting rows in one table based on differences found in the other table. Typically, this is done by executing a stored procedure or batch that contains individual INSERT, UPDATE, and DELETE statements. However, this means that the data in both the source and target tables is evaluated and processed multiple times—at least once for each statement. By using the MERGE statement, you can replace the individual DML statements with a single statement. This can improve query performance because the operations are performed within a single statement, minimizing the number of times the data in the source and target tables are processed. However, performance gains depend on

having correct indexes, joins, and other considerations in place. For more information, see Optimizing MERGE Statement Performance.

Change Data Capture

In SQL Server 2008, this new feature provides an easy way to capture changes to data in a set of database tables so these changes can be transferred to a second system such as a data warehouse. Change data capture is designed to capture insert, update, and delete activity applied to SQL Server tables, and to make the details of the changes available in an easily consumed relational format. The change tables used by change data capture contain columns that mirror the column structure of a tracked source table, along with the metadata needed to understand the changes that have occurred. Change data capture is available only in the Enterprise, Developer, and Evaluation editions of SQL Server. For more information, see Tuning the Performance of Change Data Capture in SQL Server 2008.

ExecutionLog2

In SQL Server 2008, the Reporting Services database stores additional information that enables you to analyze report execution log data to help answer questions about performance, such as:

- Which reports might be good for caching?
- How many reports were returned from cache vs. live execution vs. execution snapshot?
- What was the most popular report for a specified time period?
- What are my worst-performing reports?
- Which reports were run by the SQL Server 2008 Reporting Services engine and which reports were run by the SQL Server 2005 Reporting Services engine?
- Which reports were under memory pressure?

This detailed performance information appears in the ExecutionLog2 view. For more information, see ExecutionLog2 View – Analyzing and Optimizing Reports.

Conclusion

SQL Server 2008 is a significant release that delivers new functionality not found in previous versions and numerous performance and usability enhancements that are specifically designed for business intelligence solutions. For a full list of features and detailed descriptions, see SQL Server Books Online and the SQL Server Web site.

Top 10 SQL Server Integration Services Best Practices

How many of you have heard the myth that Microsoft® SQL Server® Integration Services (SSIS) does not scale? The first question we would ask in return is: "Does your system need to scale beyond 4.5 million sales transaction rows per second?" SQL Server Integration Services is a high performance Extract-Transform-Load (ETL) platform that scales to the most extreme environments. And as documented in SSIS ETL world record performance, SQL Server Integration Services can process at the scale of 4.5 million sales transaction rows per second.

SSIS is an in-memory pipeline, so ensure that all transformations occur in memory.

The purpose of having Integration Services within SQL Server features is to provide a flexible, robust pipeline that can efficiently perform row-by-row calculations and parse data all in memory.

While the extract and load phases of the pipeline will touch disk (read and write respectively), the transformation itself should process in memory. If transformations spill to disk (for example with large sort operations), you will see a big performance degradation. Construct your packages to partition and filter data so that all transformations fit in memory.

A great way to check if your packages are staying within memory is to review the SSIS performance counter Buffers spooled, which has an initial value of 0; above 0 is an indication that the engine has started swapping to disk. For more information, please refer to Something about SSIS Performance Counters.

Plan for capacity by understanding resource utilization.

SQL Server Integration Services is designed to process large amounts of data row by row in memory with high speed. Because of this, it is important to understand resource utilization, i.e., the CPU, memory, I/O, and network utilization of your packages.

CPU Bound

Seek to understand how much CPU is being used by Integration Services and how much CPU is being used overall by SQL Server while Integration Services is running. This latter point is especially important if you have SQL Server and SSIS on the same box, because if there is a resource contention between these two, it is SQL Server that will typically win – resulting in disk spilling from Integration Services, which slows transformation speed.

The perfmon counter that is of primary interest to you is **Process / % Processor Time (Total)**. Measure this counter for both **sqlservr.exe** and **dtexec.exe**. If SSIS is not able to drive close to 100% CPU load, this may be indicative of:

- *Application contention:* For example, SQL Server is taking on more processor resources, making them unavailable to SSIS.
- *Hardware contention:* A common scenario is that you have suboptimal disk I/O or not enough memory to handle the amount of data being processed.
- *Design limitation:* The design of your SSIS package is not making use of parallelism, and/or the

package uses too many single-threaded tasks.

Network Bound
SSIS moves data as fast as your network is able to handle it. Because of this, it is important to understand your network topology and ensure that the path between your source and target have both low latency and high throughput.

The following Network perfmon counters can help you tune your topology:

- **Network Interface / Current Bandwidth**: This counter provides an estimate of current bandwidth.
- **Network Interface / Bytes Total / sec**: The rate at which bytes are sent and received over each network adapter.
- **Network Interface / Transfers/sec**: Tells how many network transfers per second are occurring. If it is approaching 40,000 IOPs, then get another NIC card and use teaming between the NIC cards.

These counters enable you to analyze how close you are to the maximum bandwidth of the system. Understanding this will allow you to plan capacity appropriately whether by using gigabit network adapters, increasing the number of NIC cards per server, or creating separate network addresses specifically for ETL traffic.

I/O Bound
If you ensure that Integration Services is minimally writing to disk, SSIS will only hit the disk when it reads from the source and writes to the target. But if your I/O is slow, reading and especially writing can create a bottleneck.

Because tuning I/O is outside the scope of this technical note, please refer to Predeployment I/O Best Practices. Remember that an I/O system is not only specified by its size ("I need 10 TB") – but also by its sustainable speed ("I want 20,000 IOPs").

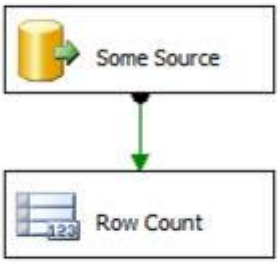
Memory bound
A very important question that you need to answer when using Integration Services is: "How much memory does my package use?"

The key counters for Integration Services and SQL Server are:

- **Process / Private Bytes (DTEXEC.exe)** – The amount of memory currently in use by Integration Services. This memory cannot be shared with other processes.
- **Process / Working Set (DTEXEC.exe)** – The total amount of allocated memory by Integration Services.
- **SQL Server: Memory Manager / Total Server Memory**: The total amount of memory allocated by SQL Server. Because SQL Server has another way to allocate memory using the AWE API, this counter is the best indicator of total memory used by SQL Server. To understand SQL Server memory allocations better, refer to Slava Ok's Weblog.
- **Memory / Page Reads / sec** – Represents to total memory pressure on the system. If this consistently goes above 500, the system is under memory pressure.

Baseline source system extract speed.

Understand your source system and how fast you extract from it. After all, Integration Services cannot be tuned beyond the speed of your source – i.e., you cannot transform data faster than you can read it. Measure the speed of the source system by creating a very simple package reading data from your source with the a destination of "Row Count":



Execute the package from the command line (DTEXEC) and measure the time it took for it to complete its task. Use the Integration Services log output to get an accurate calculation of the time. You want to calculate rows per second:

Rows / sec = Row Count / Time$_{\text{Data Flow}}$

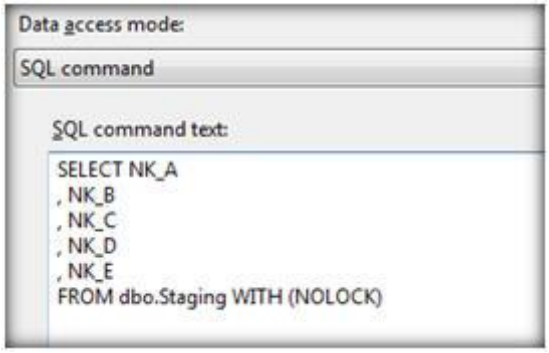
Based on this value, you now know the maximum number of rows per second you can read from the source – this is also the roof on how fast you can transform your data. To increase this Rows / sec calculation, you can do the following:

- *Improve drivers and driver configurations*: Make sure you are using the most up-to-date driver configurations for your network, data source, and disk I/O. Often the default network drivers on your server are not configured optimally for the network stack, which results in performance degradations when there are a high number of throughput requests. Note that for 64-bit systems, at design time you may be loading 32-bit drivers; ensure that at run time you are using 64-bit drivers.

- *Start multiple connections*: To overcome limitations of drivers, you can try to start multiple connections to your data source. As long as the source can handle many concurrent connections, you may see an increase in throughput if you start several extracts at once. If concurrency is causing locking or blocking issues, consider partitioning the source and having your packages read from different partitions to more evenly distribute the load.

- *Use multiple NIC cards*: If the network is your bottleneck and you've already ensured that you're using gigabit network cards and routers, then a potential solution is to use multiple NIC cards per server. Note that you will have to be careful when you configure multiple NIC environments; otherwise you will have network conflicts.

Optimize the SQL data source, lookup transformations, and destination.



When you execute SQL statements within Integration Services (as noted in the above **Data access mode** dialog box), whether to read a source, to perform a look transformation, or to change tables, some standard optimizations significantly help performance:

- Use the NOLOCK or TABLOCK hints to remove locking overhead.
- To optimize memory usage, SELECT only the columns you actually need. If you SELECT all columns from a table (e.g., SELECT * FROM) you will needlessly use memory and bandwidth to store and retrieve columns that do not get used. .

- If possible, perform your datetime conversions at your source or target databases, as it is more expensive to perform within Integration Services..

- In SQL Server 2008 Integration Services, there is a new feature of the shared lookup cache. When using parallel pipelines (see points #8 and #10 below), it provides a high-speed, shared cache. .

- If Integration Services and SQL Server run on the same server, use the SQL Server destination instead of the OLE DB destination to improve performance..

- Commit size 0 is fastest on heap bulk targets, because only one transaction is committed. If you cannot use 0, use the highest possible value of commit size to reduce the overhead of multiple-batch writing. Commit size = 0 is a bad idea if inserting into a Btree – because all incoming rows must be sorted at once into the target Btree—and if your memory is limited, you are likely to spill. Batchsize = 0 is ideal for inserting into a heap. For an indexed destination, I recommend testing between 100,000 and 1,000,000 as batch size.

- Use a commit size of <5000 to avoid lock escalation when inserting; note that in SQL Server 2008 you can now enable/disable lock escalation at the object level, but use this wisely.

- Heap inserts are typically faster than using a clustered index. This means that you may want to drop indexes and rebuild if you are changing a large part of the destination table; you will want to test your inserts both by keeping indexes in place and by dropping all indexes and rebuilding to validate..

- Use partitions and partition SWITCH command; i.e., load a work table that contains a single partition and SWITCH it in to the main table after you build the indexes and put the constraints on..

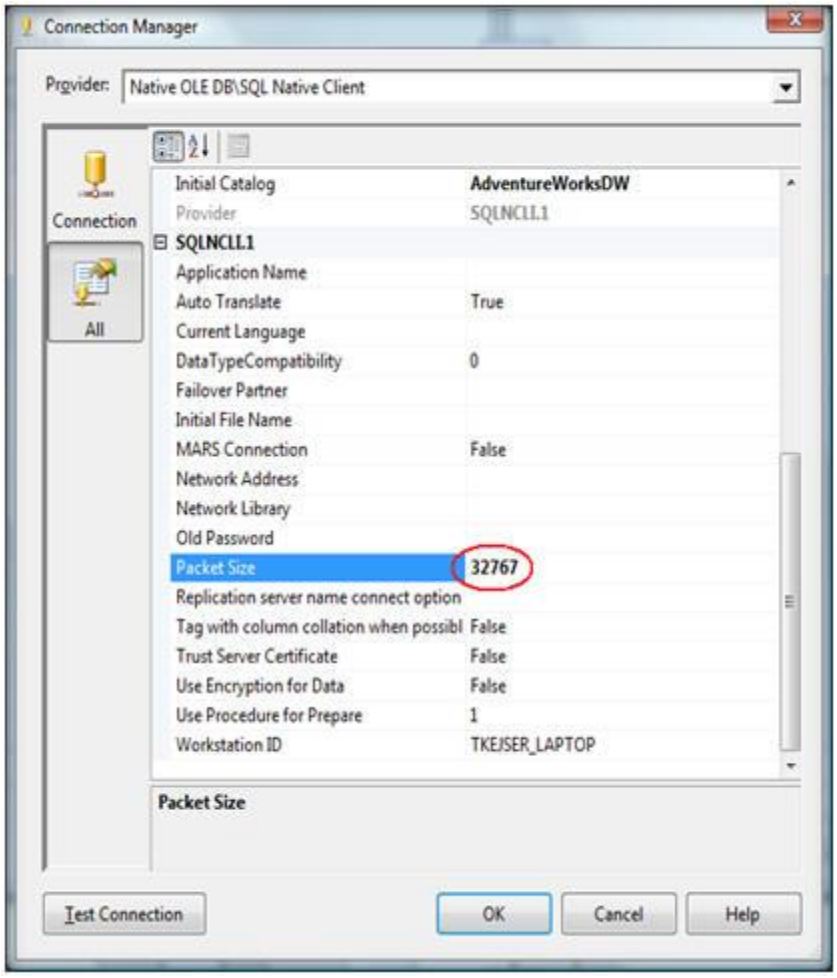
- Another great reference from the SQL Performance team is Getting Optimal Performance with Integration Services Lookups.

Tune your network.

A key network property is the packet size of your connection. By default this value is set to 4,096 bytes. This means a new network package must be assemble for every 4 KB of data. As noted in SqlConnection.PacketSize Property in the .NET Framework Class Library, increasing the packet size will improve performance because fewer network read and write operations are required to transfer a large data set.
If your system is transactional in nature, with many small data size read/writes, lowering the value will improve performance.

Since Integration Services is all about moving large amounts of data, you want to minimize the network overhead. This means that the value 32K (32767) is the fastest option. While it is possible to configure the network packet size on a server level using sp_configure, you should not do this. The database administrator may have reasons to use a different server setting than 32K. Instead, override the server settings in the connection manager as illustrated below.

5



Another network tuning technique is to use network affinity at the operating system level. At high throughputs, you can sometimes improve performance this way.

For the network itself, you may want to work with your network specialists to enable jumbo frames to increase the default payload of 1,500 bytes to 9,000 bytes. By enabling jumbo frames, you will further decrease the amount of network operation required to move large data sets.

Use data types – yes, back to data types! –wisely.

Of all the points on this top 10 list, this is perhaps the most obvious. Yet, it is such an important point that it needs to be made separately. Follow these guidelines:

- Make data types as narrow as possible so you will allocate less memory for your transformation.
- Do not perform excessive casting of data types – it will only degrade performance. Match your data types to the source or destination and explicitly specify the necessary data type casting..
- Watch precision issues when using the **money, float**, and **decimal** types. Also, be aware the **money** is faster than **decimal**, and **money** has fewer precision considerations than **float**.

Change the design.

There are some things that Integration Services does well – and other tasks where using another tool is more efficient. Your tool choice should be based on what is most efficient and on a true understanding of the problem. To help with that choice, consider the following points:

- Do not sort within Integration Services unless it is absolutely necessary. In order to perform a sort, Integration Services allocates the memory space of the entire data set that needs to be transformed. If possible, presort the data before it goes into the pipeline. If you must sort data, try your best to sort only small data sets in the pipeline. Instead of using Integration Services for sorting, use an SQL statement with ORDER BY to sort large data sets in the database – mark the output as sorted by changing the Integration Services pipeline metadata on the data source.

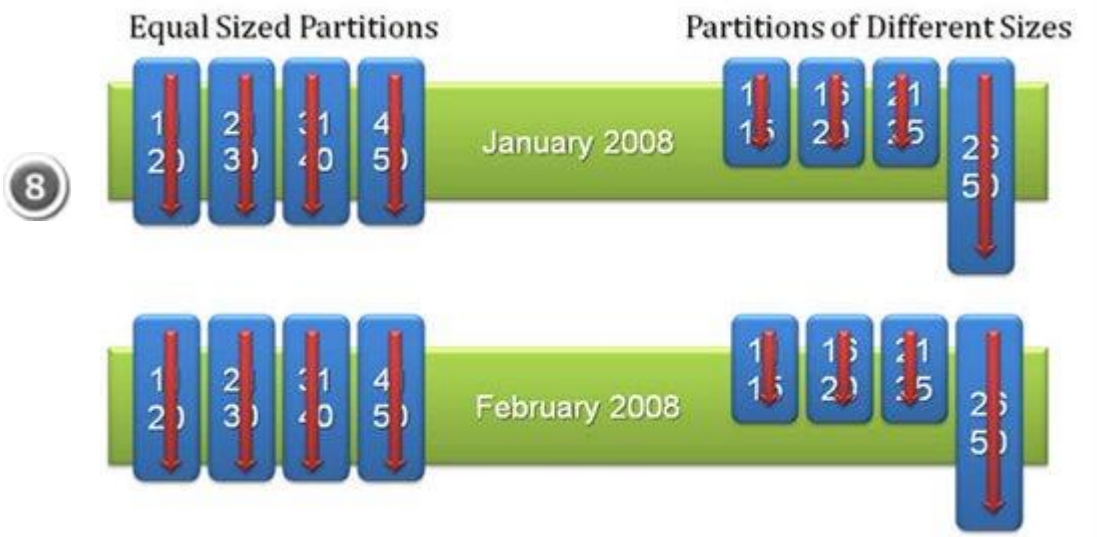
- There are times where using Transact-SQL will be faster than processing the data in SSIS. As a general rule, any and all set-based operations will perform faster in Transact-SQL because the problem can be transformed into a relational (domain and tuple) algebra formulation that SQL Server is optimized to resolve. Also, the SQL Server optimizer will automatically apply high parallelism and memory management to the set-based operation – an operation you may have to perform yourself if you are using Integration Services. Typical set-based operations include:
    - Set-based UPDATE statements - which are far more efficient than row-by-row OLE DB calls.
    - Aggregation calculations such as GROUP BY and SUM. These are typically also calculated faster using Transact-SQL instead of in-memory calculations by a pipeline.
- Delta detection is the technique where you change existing rows in the target table instead of reloading the table. To perform delta detection, you can use a change detection mechanism such as the new SQL Server 2008 Change Data Capture (CDC) functionality. If such functionality is not

available, you need to do the delta detection by comparing the source input with the target table. This can be a very costly operation requiring the maintenance of special indexes and checksums just for this purpose. Often, it is fastest to just reload the target table. A rule of thumb is that if the target table has changed by >10%, it is often faster to simply reload than to perform the logic of delta detection.

Partition the problem.

One of the main tenets of scalable computing is to partition problems into smaller, more manageable chunks. This allows you to more easily handle the size of the problem and make use of running parallel processes in order to solve the problem faster.

For ETL designs, you will want to partition your source data into smaller chunks of equal size. This latter point is important because if you have chunks of different sizes, you will end up waiting for one process to complete its task. For example, looking at the graph below, you will notice that for the four processes executed on partitions of equal size, the four processes will finish processing January 2008 at the same time and then together continue to process February 2008. But for the partitions of different sizes, the first three processes will finish processing but wait for the fourth process, which is taking a much longer time. The total run time will be dominated by the largest chunk.



To create ranges of equal-sized partitions, use time period and/or dimensions (such as geography) as your mechanism to partition. If your primary key is an incremental value such as an IDENTITY or another increasing value, you can use a modulo function. If you do not have any good partition columns, create a hash of the value of the rows and partition based on the hash value. For more information on hashing and partitioning, refer to the Analysis Services Distinct Count Optimization white paper; while the paper is about distinct count within Analysis Services, the technique of hash partitioning is treated in depth too.

Some other partitioning tips:

- Use partitioning on your target table. This way you will be able to run multiple versions of the same

package, in parallel, that insert data into different partitions of the same table. When using partitioning, the SWITCH statement is your friend. It not only increases parallel load speeds, but also allows you to efficiently transfer data. Please refer to the SQL Server Books Online article Transferring Data Efficiently by Using Partition Switching for more information.

- As implied above, you should design your package to take a parameter specifying which partition it should work on. This way, you can have multiple executions of the same package, all with different parameter and partition values, so you can take advantage of parallelism to complete the task faster.

- From the command line, you can run multiple executions by using the "START" command. A quick code example of running multiple robocopy statements in parallel can be found within the Sample Robocopy Script to custom synchronize Analysis Services databases technical note.

Minimize logged operations.

When you insert data into your target SQL Server database, use minimally logged operations if possible. When data is inserted into the database in fully logged mode, the log will grow quickly because each row entering the table also goes into the log.

Therefore, when designing Integration Services packages, consider the following:

- Try to perform your data flows in bulk mode instead of row by row. By doing this in bulk mode, you will minimize the number of entries that are added to the log file. This reduction will improve the underlying disk I/O for other inserts and will minimize the bottleneck created by writing to the log.

- If you need to perform delete operations, organize your data in a way so that you can TRUNCATE the table instead of running a DELETE. The latter will place an entry for each row deleted into the log. But the former will simply remove all of the data in the table with a small log entry representing the fact that the TRUNCATE occurred. In contrast with popular belief, a TRUNCATE statement can participate in a transaction.

- Use the SWITCH statement and partitioning. If partitions need to be moved around, you can use the SWITCH statement (to switch in a new partition or switch out the oldest partition), which is a minimally logged statement.

- Be careful when using DML statements; if you mix in DML statements within your INSERT statements, minimum logging is suppressed.

Schedule and distribute it correctly.

After your problem has been chunked into manageable sizes, you must consider where and when these chunks should be executed. The goal is to avoid one long running task dominating the total time of the ETL flow.

A good way to handle execution is to create a priority queue for your package and then execute multiple instances of the same package (with different partition parameter values). The queue can simply be a SQL

Server table. Each package should include a simple loop in the control flow:

1. Pick a relevant chunk from the queue:
    1. "Relevant" means that is has not already been processed and that all chunks it depends on have already run.
    2. If no item is returned from the queue, exit the package.
2. Perform the work required on the chunk.
3. Mark the chunk as "done" in the queue.
4. Return to the start of loop.

Picking an item from the queue and marking it as "done" (step 1 and 3 above) can be implemented as stored procedure, for example.

The queue acts as a central control and coordination mechanism, determining the order of execution and ensuring that no two packages work on the same chunk of data. Once you have the queue in place, you can simply start multiple copies of DTEXEC to increase parallelism.

Analysis Services Query Performance Top 10 Best Practices

Proper cube design, efficient multidimensional expressions (MDX), and sufficient hardware resources are critical to optimal performance of MDX queries issued against a SQL Server 2005 Analysis Services instance. This article lists the ten most common best practices that the Microsoft SQL Server development team recommends with respect to optimizing Analysis Services query performance. For additional discussions about Analysis Services best practices related to query performance, see The Analysis Services Performance Guide and OLAP Design Best Practices for Analysis Services 2005.

Optimize cube and measure group design

- Define cascading attribute relationships (for example Day > Month > Quarter > Year) and define user hierarchies of related attributes (called *natural hierarchies*) within each dimension as appropriate for your data. Attributes participating in natural hierarchies are materialized on disk in hierarchy stores and are automatically considered to be aggregation candidates. User hierarchies are not considered to be natural hierarchies unless the attributes comprising the levels are related through cascading attribute relationships. With SQL Server 2005 Service Pack 2 (SP2), a warning appears in Business Intelligence Development Studio with each user hierarchy that is not defined as a natural hierarchy.

- Remove redundant relationships between attributes to assist the query execution engine in generating the appropriate query plan. Attributes need to have either a direct or an indirect relationship to the key attribute, not both.

- Keep cube space as small as possible by only including measure groups that are needed.

- Place measures that are queried together in the same measure group. A query that retrieves measures from multiple measure groups requires multiple storage engine operations. Consider placing large sets of measures that are not queried together into separate measure groups to optimize cache usage, but do not explode the number of measure groups.

- Minimize the use of large parent-child hierarchies. In parent-child hierarchies, aggregations are created only for the key attribute and the top attribute (for example, the All attribute) unless it is disabled. As a result, queries returning cells at intermediate levels are calculated at query time and can be slow for large parent-child dimensions. If you are in a design scenario with a large parent-child hierarchy (more than 250,000 members), you may want to consider altering the source schema to reorganize part or all of the hierarchy into a user hierarchy with a fixed number of levels.

- Optimize many-to-many dimension performance, if used. When you query the data measure group by the many-to-many dimension, a run-time "join" is performed between the data measure group and the intermediate measure group using the granularity attributes of each dimension that the measure groups have in common. Where possible, reduce the size of the intermediate fact table underlying the intermediate measure group. To optimize the run-time join, review the aggregation design for the intermediate measure group to verify that aggregations include attributes from the many-to-many dimension.

To understand how to optimize dimensions to increase query performance, refer to the articles SQL Server 2005 Analysis Services Performance Guide and OLAP Design Best Practices for Analysis Services 2005. For

assistance in analyzing your design for compliance with best practices, see the February 2007 Community Technology Preview (CTP) release of the SQL Server 2005 Best Practices Analyzer (the final version should be released soon).

Define effective aggregations

- Define aggregations to reduce the number of records that the storage engine needs to scan from disk to satisfy a query. If SQL Server Profiler traces indicate that most user queries that are not resolved from cache are resolved by partition reads rather than aggregation reads, consider using the Aggregation Manager sample application to design custom aggregations. This sample is available on CodePlex at http://www.codeplex.com/MSFTASProdSamples and a version of this sample that has been updated by the community is available on CodePlex at http://www.codeplex.com/bidshelper.

- Avoid designing an excessive number of aggregations. Excessive aggregations reduce processing performance and may reduce query performance. While the optimum number of aggregations varies, the SQL Server Best Practices team's experience has been that the optimum number is in the tens, not hundreds or thousands in almost all cases.

- Enable the Analysis Services query log to capture user query patterns and use this query log when designing aggregations. For more information, see Configuring the Analysis Services Query Log.

To understand how to design aggregations to increase query performance, refer to the articles SQL Server 2005 Analysis Services Performance Guide and OLAP Design Best Practices for Analysis Services 2005.

Use partitions

- Define partitions to enable Analysis Services to query less data to resolve a query when it cannot be resolved from the data cache or from aggregations. Also define the partitions to increase parallelism when resolving queries.

- For optimum performance, partition data in a manner that matches common queries. A very common choice for partitions is to select an element of time such as day, month, quarter, year or some combination of time elements. Avoid partitioning in a way that requires most queries to be resolved from many partitions.

- In most cases, partitions should contain fewer than 20 million records size and each measure group should contain fewer than 2,000 total partitions. Also, avoid defining partitions containing fewer than two million records. Too many partitions causes a slowdown in metadata operations, and too few partitions can result in missed opportunities for parallelism.

- Define a separate ROLAP partition for real-time data and place real-time ROLAP partition in its own measure group.

To understand how to design partitions to increase query performance, refer to the SQL Server 2005 Analysis Services Performance Guide, the Microsoft SQL Server Customer Advisory Team blog, and OLAP Design Best Practices for Analysis Services 2005.

Write efficient MDX

- Remove empty tuples from your result set to reduce the time spent by the query execution engine serializing the result set.

- Avoid run-time checks in an MDX calculation that result in a slow execution path. If you use the Case and IF functions to perform condition checks which must be resolved many times during query resolution, you will have a slow execution path. Rewrite these queries using the SCOPE function to quickly reduce the calculation space to which an MDX calculation refers. For more information, see Budget Variance - A study of MDX optimizations: evaluation modes and NON_EMPTY_BEHAVIOR, Comparing Levels in MDX and CONDITION vs. SCOPE in cell calculations, and Multiselect friendly MDX calculations.

- Use Non_Empty_Behavior where possible to enable the query execution engine to use bulk evaluation mode. However, if you use Non_Empty_Behavior incorrectly, you will return incorrect results. For more information, see Budget Variance - A study of MDX optimizations: evaluation modes and NON_EMPTY_BEHAVIOR and Averages, ratios, division by zero and NON_EMPTY_BEHAVIOR.

- Use EXISTS rather than filtering on member properties to avoid a slow execution path. Use the NonEmpty and Exists functions to enable the query execution engine to use bulk evaluation mode.
- Perform string manipulations within Analysis Services stored procedures using server-side ADOMD.NET rather than with string manipulation functions such as StrToMember and StrToSet.

- Rather than using the LookupCube function, use multiple measure groups in the same cube wherever possible.

- Rewrite MDX queries containing arbitrary shapes to reduce excessive subqueries where possible. An arbitrary shaped set is a set of members that cannot be resolved as a crossjoin of sets with a single hierarchality. For example, the set {(Gender.Male, Customer.USA), (Gender.Female, Customer.Canada)} is an arbitrary set. You can frequently use the Descendants function to resolve arbitrary shapes by using a smaller number of subqueries than queries that return the same result that are written using other functions.

- Rewrite MDX queries that result in excessive prefetching where possible. *Prefetching* is a term used to describe cases where the query execution engine requests more information from the storage engine than is required to resolve the query at hand for reasons of perceived efficiency. Generally, prefetching is the most efficient data retrieval choice. However, occasionally it is not. In some cases you may be able to eliminate excessive prefetching by rewriting queries with a subselect in the FROM clause rather than a set in the WHERE clause. When you cannot eliminate excessive prefetching, you may need to disable prefetching and warm the cache using the Create Cache statement. For more information, see How to Warm up the Analysis Services data cache using Create Cache statement.

- Filter a set before using it in a crossjoin to reduce the cube space before performing the crossjoin.

Use the query engine cache efficiently

- Ensure that the Analysis Services computer has sufficient memory to store query results in memory for re-use in resolving subsequent queries. To monitor, use the MSAS 2005: Memory/Cleaner Memory Shrinkable DB and the MSAS 2005: Cache/Evictions/sec Performance Monitor counters.

- Define calculations in the MDX script. Calculations in the MDX script have a global scope that enables the cache related to these queries to be shared across sessions for the same set of security permissions. However, calculated members defined using Create Member and With Member within user queries do not have global scope and the cache related to these queries cannot be shared across sessions.

- Warm the cache by executing a set of predefined queries using the tool of your choice. You can also use a Create Cache statement for this purpose. For more information on using the Create Cache statement, see How to Warm up the Analysis Services data cache using Create Cache statement. For information on how to use SQL Server 2005 Integration Services to warm the cache, see Build Your Own Analysis Services Cache-Warmer in Integration Services.

- Rewrite MDX queries containing arbitrary shapes to optimize caching. For example, in some cases you can rewrite queries that require non-cached disk access such that they can be resolved entirely from cache by using a subselect in the FROM clause rather than a WHERE clause. In other cases, a WHERE clause may be a better choice.

Ensure flexible aggregations are available to answer queries.

- Note that incrementally updating a dimension using ProcessUpdate on a dimension drops all flexible aggregations affected by updates and deletes and, by default, does not re-create them until the next full process.

- Ensure that aggregations are re-created by processing affected objects, configuring lazy processing, performing ProcessIndexes on affected partitions, or performing full processing on affected partitions.

To understand how to ensure flexible aggregations are not dropped, refer to the SQL Server 2005 Analysis Services Performance Guide.

Tune memory usage

- Increase the size of the paging files on the Analysis Services server or add additional memory to prevent out–of-memory errors when the amount of virtual memory allocated exceeds the amount of physical memory on the Analysis Services server.

- Use Microsoft Windows Advanced Server® or Datacenter Server with SQL Server 2005 Enterprise Edition (or SQL Server 2005 Developer Edition) when you are using SQL Server 2005 (32-bit) to enable Analysis Services to address up to 3 GB of memory. To enable Analysis Services to address more than 2 GB of physical memory with either of these editions, use the /3GB switch in the boot.ini file. If you set the /3GB switch in the boot.ini file, the server should have at least 4 GB of memory to ensure that the Windows operating system also has sufficient memory for system services.

- Reduce the value for the Memory/LowMemoryLimit property below 75 percent when running

multiple instances of Analysis Services or when running other applications on the same computer.

- Reduce the value for the Memory/TotalMemoryLimit property below 80percent when running multiple instances of Analysis Services or when running other applications on the same computer.
- Keep a gap between the Memory/LowMemoryLimit property and the Memory/TotalMemoryLimit property – 20 percent is frequently used.
- When query thrashing is detected in a multi-user environment, contact Microsoft Support for assistance in modifying the MemoryHeapType.
- When running on non-uniform memory access (NUMA) architecture and VirtualAlloc takes a very long time to return or appears to stop responding, upgrade to SQL Server 2005 SP2 and contact Microsoft Support for assistance with appropriate settings for pre-allocating NUMA memory.

To understand when to consider changing default memory use, refer to the SQL Server 2005 Analysis Services Performance Guide and Microsoft SQL Server Customer Advisory Team blog.

Tune processor usage

- To increase parallelism during querying for servers with multiple processors, consider modifying the Threadpool\Query\MaxThreads and Threadpool\Process\MaxThreads options to be a number that depends on the number of server processors.
- A general recommendation is to set the Threadpool\Query\MaxThreads to a value of less than or equal to two times the number of processors on the server. For example, if you have an eight-processor server, the general guideline is to set this value to no more than 16. In practical terms, increasing the Threadpool\Query\MaxThreads option will not significantly increase the performance of a given query. Rather, the benefit of increasing this property is that you can increase the number of queries that can be serviced concurrently.
- A general recommendation is to set the Threadpool\Process\MaxThreads option to a value of less than or equal to 10 times the number of processors on the server. This property controls the number of threads used by the storage engine during querying operations as well as during processing operations. For example, if you have an eight-processor server, the general guideline is setting this value to no more than 80. Note that even though the default value is 64, if you have fewer than eight processors on a given server, you do not need to reduce the default value to throttle parallel operations.
- While modifying the Threadpool\Process\MaxThreads and Threadpool\Query\MaxThreads properties can increase parallelism during querying, you must also consider the additional impact of the CoordinatorExecutionMode option. For example, if you have a four-processor server and you accept the default CoordinatorExecutionMode setting of -4, a total of 16 jobs can be executed at one time across all server operations. So if 10 queries are executed in parallel and require a total of 20 jobs, only 16 jobs can launch at a given time (assuming that no processing operations are being performed at that time). When the job threshold has been reached, subsequent jobs wait in a queue until a new job can be created. Therefore, if the number of jobs is the bottleneck to the operation, increasing the thread counts may not necessarily improve overall performance.

Scale up where possible

- Use a 64-bit architecture for all large systems.

- Add memory and processor resources and upgrade the disk I/O subsystem, to alleviate query performance bottlenecks on a single system.

- Avoid linking dimensions or measure groups across servers and avoid remote partitions whenever possible because these solutions do not perform optimally.

Scale out when you can no longer scale up

- If your performance bottleneck is processor utilization on a single system as a result of a multi-user query workload, you can increase query performance by using a cluster of Analysis Services servers to service query requests. Requests can be load balanced across two Analysis Services servers, or across a larger number of Analysis Services servers to support a large number of concurrent users (this is called a server farm). Load-balancing clusters generally scale linearly.

- When using a cluster of Analysis Services servers to increase query performance, perform processing on a single processing server and then synchronize the processing and the query servers using the XMLA Synchronize statement, copy the database directory using Robocopy or some other file copy utility, or use the high-speed copy facility of SAN storage solutions.