Business Intelligence Performance Optimization

Ángel N. Sierra
Master in Computer Science
Dr. Juan Ramirez
Electrical & Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— What happens when you are the Business Intelligence (BI) developer and you need to find a solution for an Analysis Service Job execution that takes more than 20 hours to process over 1.3 terra-bytes of data? This project will presents an atypical scenario for a business solution. In a typical scenario many introductory books, papers or tutorials presents a random company that uses a web site to sell their products such as "Adventure Works". Adventure works is a company invented by Microsoft where they present a database that they use to teach many concepts, including Business Intelligence. In this project we have a Business Intelligence solution already deployed in a production environment, but the database administrators are reporting performance issues running the data processing to maintain this solution up to date. I will be examining this solution to assess where exactly the problem is and provide a new solution that is capable to improve processing performance.*

Key Terms *— Analysis Services, Business Intelligence, Cube, Data Warehouse, Optimization, Performance*

PROBLEM STATEMENT

A BI Solution developed by an outsource company is affecting production performance when processing the CUBE created in Microsoft SQL Server Analysis Services (SSAS). Due to this situation, the Software Development Group for the company "XYZ" has to review the design and provide new solutions, which minimize the impact that the processing is having in the production environment.

MICROSOFT SQL SERVER – ANALYSIS SERVICES (SSAS) ARCHITECTURE

SSAS is a service that is provided by Microsoft to perform analysis and calculations such as aggregation and data mining.

Typically, companies that have large sales are interested in this type of solutions, as it allows them to have a better understanding of the sales at different levels. For example, the North America Sales region manager could connect to the SSAS database with Excel and generate pivot tables that will allow everyone to see how many bicycles his company sold in the last quarter. The supply chain manager could connect to the same SSAS database to check how many parts were used in the production of certain bicycle during the last month.

In this project I will not be developing a business intelligence (BI) solution from scratch. I will be reviewing a solution that was created and implemented by another developer in a production environment. I will also design a new solution that would improve SSAS performance.

To achieve this goal I will be using the SSAS architecture presented in figure 1 as a guide to allocate the source of the performance problem.

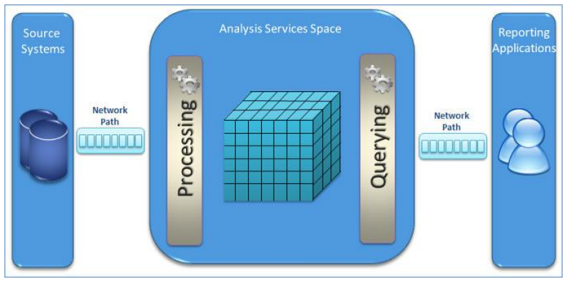

Figure 1
Microsoft SQL Server – Analysis Services Architecture

Source Systems

The source system presented in figure 1 refers to the data source; the location where you are obtaining the information you want to analyze. Generally, the source could be another database engine such as Oracle, Teradata, or DB2. It can also be a flat file generated by a specific program.

In this project the data source is a Microsoft SQL Server database engine for which we have no documentation available such as an Entity Relationship Diagram (ERD).

The entity relationship diagram is a graphical representation of the tables that were created in a database. This diagram can present information such as the attributes inside the table and primary keys. Figure 2 shows an example entity relationship diagram created with Microsoft SQL Server Management Studio (SSMS).

The tools available in SSMS allowed me to generate an ERD. Inspection of this diagram showed the following key information:

- The database contains one group of fifty one tables related.
- The database contains one group of four tables related.
- The database contains five groups of three tables related.
- The database contains three groups of two tables related.
- The database contains seventy-eight tables unrelated.

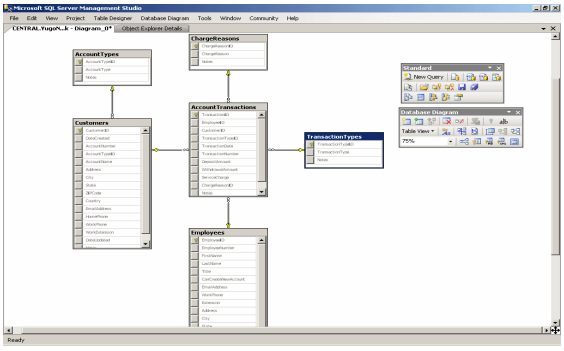

Figure 2
Sample Database Diagram in Microsoft SSMS

The great amount of tables present in the current database, gives a great idea of the current database structure and it can lead to the common assumption that it is using too many *joins* to query the necessary information for the business intelligence solution. To continue with the research of the current data source, I believe it is healthy to check the amount of data each table has. It can give us a great idea of how much data could probably be processed by Analysis Services. This information could be obtained by executing in SSMS the script provided below. The script retrieves the size statistics for each table inside the database being queried.

```
USE [MyDBName]
GO
SET NOCOUNT ON DBCC UPDATEUSAGE(0)
EXEC sp_spaceused
CREATE TABLE #tempTableSize (
[name] NVARCHAR(128)
, [rows] CHAR(11)
, reserved VARCHAR(18)
, data VARCHAR(18)
, index_size VARCHAR(18)
, unused VARCHAR(18))
INSERT #tempTableSize EXEC
sp_msForEachTable
    'EXEC sp_spaceused ''?'''
    SELECT * FROM   #tempTableSize
SELECT SUM(CAST([rows] AS int)) AS
[rows]FROM #tempTableSize
    DROP TABLE #tempTableSize;
```

The script presented above will generate a temporary table in *tempdb* that will allow us to see metadata such as number of rows and size for each object. In this particular case we are obtaining the information for one database object and 154 tables object.

This script uses two store procedures to obtain the desired statistics, *sp_spaceused* and *sp_msForEachTable*. The first time *sp_spaceused* is called alone, just before the CREATE clause, at this point it is used to obtain the overall size statistics of the database. The second time it is used in conjunction with sp_msForEachTable, this time it will fill the temporary table with size information for each table inside the database.

After executing this script against the source database I was able to observe the following:

- The biggest table is composed of 1,818,925,691 rows, approximately 1,060 GB.
- The second biggest table is composed of 376,730 rows, approximately 0.08 GB.
- About twenty-five tables were empty and thirty-five tables had less than ten rows.

Information like this allows us to be more curious about where the 1,060 GB table is being used and why do we have so many empty tables. Since this database is being used to store transactions coming from different sources, it could suggest that the original intention was to create some sort of hybrid database. Some tables may be used for data warehousing purposes while other tables would be used for online transaction processing and data warehousing.

Analysis Services Space

The next area in the architecture is the Analysis Services Space and as presented in the architecture, it is divided in two engines and a common element in business intelligence, known as the cube.

In a very generalized and quick overview, the first engine is the processing unit. Its functionality is to retrieve the desired information from the data source and perform the required aggregations to populate the cube designed by the BI developer.

The Cube is the logical unit that stores the desired measurements to be analyzed and the actual level for which you would like to know its total values.

The second engine is the querying unit and it is in charge of returning the answer to the query requested by its users.

Since this solution is already created, the best approach is to create a new project in Business Intelligence Development Studio (BIDS) if you are using SQL Server 2008/ 2008 R2, or SQL Server Data Tools (SSDT) if you are using SQL Server 2012 or greater.

BIDS or SSDT is a tool provided by Microsoft to develop and deploy your BI solutions to the desired environment. This developer tool is essential to analyze the actual cube design by reviewing different objects such as the data source object, data source view, cube measurements and cube dimensions.

The *Data Source Object* is created to provide information such as Data Source type, IP address, credentials, etc. It all depends on the data source(s) type being used for the BI solution and security access required to connect to the source and retrieve data.

Since we can have multiple data sources and not every table in a database is needed, Microsoft designed an object that would allow us to define a "new relational database" containing only those tables that we need for our BI solution. This object is known by Microsoft as the Data Source View and in many books is called "the star schema" or "snowflake diagram". Inside this object we can create new tables through queries, known as *named query*, and add new attributes to a table known as *calculated names*.

The named queries allow us to create a temporary table that will be used to define measurements and/or dimensions. The named calculations are functions that we can create to add attributes that do not exist in a specific table. For example, the full name of a person by concatenating the first and last name.

In a BI solution optimization, in which the major concern is at the processing area, I would strongly suggest to avoid using named queries, named calculations and view tables. Especially if the tables included in the joints contains a huge amount of data. This tables with huge amount of data will cost a long time to process because the named query has to review each row to perform a specific task such as read, write, or calculate. In this particular area, I have decided to review each named query of the data source view to observe the rows and execution time returned after executing them directly in SSMS. These values will allow us to observe the impact each query could have in the database engine. Table 1 shows the results of the review made to the named queries and tables views found to be used in the data source view definition.

Table 1
Named Query and View Tables Execution Results

Named Query or View Table	Rows Amount	Execution Time
Named Query 1	81	00:00:02
Named Query 2	1,408,289	00:00:54
View Table 1	692,785	00:00:13
View Table 2	19,103	01:15:19
Named Query 3	N/A	1:51:52

In the table presented above I have changed the name of the actual named queries and view tables for security reasons. However, the results can show the great difference that a complex query could cause during the reading execution.

The major difference between Named Query 3 and the other tables and views is that this query is using more than eight joints to obtain all the desired information. Also, View Table 2 and Named Query 3 are using the biggest table in the data source (over 1,000 GB size) as part of their query.

Have in mind that the test is being performed in an environment that lacks the hardware needed to simulate production environment in terms of speed and disk space. Limitations in disk space were the main reason for Named Query 3 not being able to complete execution as requested.

The *Cube Measurement* object is defined by the table(s) that have the facts about the "sales" or element you are studying. This object looks for those attributes defined by the developer and performs the necessary aggregations for the cube definition.

The *Cube Dimension* object is defined by what exactly the customer would like to know about the measurement (e.g., Sales). For example, if the customer would like to know how many mountain bicycles were sold by the company in a monthly basis, the sale amount would represent the measurement aggregation while the time element (e.g., monthly) represents the dimension.

In this particular case, the developer has to define the table that is considered as the dimension and specify the attributes that are required by the customer to be present. In this object we can also define the hierarchy in which you would like the information to be presented. In reality the hierarchy that is defined in the dimensions will allow the customer quickly change between dimensions (e.g., Weekly, Monthly, Quarterly and Annually).

Up until now all I have done is analyze the definition of the cube and highlight those areas of concerns that could potentially lead to the solution of the problem.

At this point you may wonder about the actual performance issue and how am I going to study the issue. Where the answer would be Microsoft SQL Profiler.

MICROSOFT SQL PROFILER

Microsoft SQL Profiler is a tool provided with SQL Server and it is used to connect to the SQL Server Database Engine or SQL Server Analysis Services and capture the metadata corresponding to the opened session. Information such as *EventClass*, *TextData*, *Connection ID*, *NT UserName*, *Duration*, among others as presented in Figure 3.

As a unit testing scenario, I have imported the cube definition into BIDS and deployed the same to my local machine. Since the cube was going to be deployed for the first time the processing of the information has to be a *full processing* type, this means that we are forced to execute a processing of the entire data source instead of processing the missing portion from the last time it was performed. To observe the actual deployment effects on the analysis services, I created a new event trace with SQL profiler to track the events that internally occurs when the cube is deployed.

In this test, the cube was not able to deploy due to disk space limitations. However, the SQL Profiler presented the following areas of concerns:

- More than one measurement processing failed due to missing key attribute while reading result of the executed SQL performed by the measurement partition.
- Many simultaneous processing were observed in the profiler results.

- Many events had a start time but they had no end time.
- Many of the data was being inserted as unknown due to empty or NULL values.

To minimize these concerns I used and redefined a SQL Server Integration Services package.

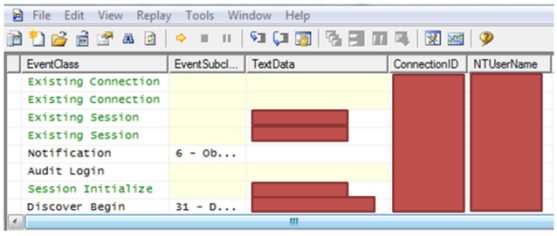

Figure 3
Microsoft SQL Profiler Trace Sample

MICROSOFT SQL SERVER – INTEGRATION SERVICES (SSIS)

SSIS is a service that allows SQL users, administrators and developers to design workflows that will allow them to complete a specific job.

BIDS or SSDT is the tool used to define these packages and its jobs can vary from data extraction, transformations and data loading among others.

In this case we will be using control flow task known as "Analysis Services Processing Task" to process first the dimension objects and later the measurements objects.

This process could take quite some time depending on the objects definitions. In this case we have one specific named query, Named Query 3 from table 1, which represents the major concern.

Execution of this package confirmed the concerns with the definitions of some of the data source view. Among those, the table containing the biggest amount of data and named calculations, this table is being used as a measurement. It consumed the whole temporary disk space available and took over 6.5 hours to execute prior disk space error notification.

Table 2
SSIS Analysis Services Processing Task Execution Time

Object Updated	Object Type	Process Type	Duration
Table 1	Dimension	Process Full	0:00:23
Table 2	Dimension	Process Full	0:00:05
Table 3	Dimension	Process Full	0:00:04
Table 4	Dimension	Process Full	0:00:04
NamedQuery2	Dimension	Process Full	0:03:53
Table 5	Dimension	Process Full	0:00:07
Table 6	Dimension	Process Full	0:00:06
Table 7	Dimension	Process Full	0:00:07
Table 8	Dimension	Process Full	0:01:38
NamedQuery3	Dimension	Process Full	0:40:35
Table 9	Fact	Process Full	0:00:48
Table 10	Fact	Process Full	0:00:06
Table 11	Fact	Process Full	*6:30:00

*Last object updated did not complete the task due to disk space limitation

SUMMARY OF CONCERNS

After performing the various tests I can confirm that the area of concern is in the processing data stage. The root cause in this case is the design definition of the data source view and the missing concept of data warehousing. Another area of concern is the hardware, a big solution like this will need more memory and disk space as time goes by and more transactions are added in a daily basis. For this particular solution, the amount of transaction that is added in a single day is around 2,000,000 records or rows.

POSSIBLE SOLUTIONS

The best solution to fix the current optimization concern is the design and creation of a data warehouse with extract-transform-load package that will transfer data from the OLTP database to the data warehouse. Using this data warehouse will optimize the execution time to process the data for each dimension and measurement. It will also simplify the monitoring and execution of each update as defined by the business whether it is daily, weekly, or monthly.

However, data warehousing could represent an issue for the company in terms of budget and time. This solution will require an investment for new equipment, resources and time. BI developers will need to be able to emulate an exact copy of the current environment in order to complete the design and test the new environment prior to deploy as the new business solution for the customer.

The second option is a solution that will allow the business to keep current BI solution design. But, close monitoring will be needed to ensure the job is executed in a daily basis. One day that it is not executed, it will need to process the entire data from the beginning resulting in a long execution time. This approach will only need the generation of a SSIS package and query definition to obtain all data corresponding to the previous date.

I suggest implementing the second solution as a temporary solution, but consider the first option as the goal to provide continues BI solution with a lot less impact to the processing data stage.

TEST LAB ENVIRONMENT

To optimize performance of a Business Intelligence solution, Microsoft recommends to divide your solution in three different servers with different disk/spindles for SQL Server and a different machine for developing tools such as BIDS or SSDT. Figure 4 presents the ideal infrastructure in which each main service of the SQL Server is installed in a different server with its own disk and the developer tool is installed in a laptop or PC.

Even though this is the best approach to organize your SQL Server infrastructure we are limited to another environment set. However, to be able to quantify the performance improvement offered by the new solution I have tested both solutions in the same environment. The test environment is presented in figure 5.

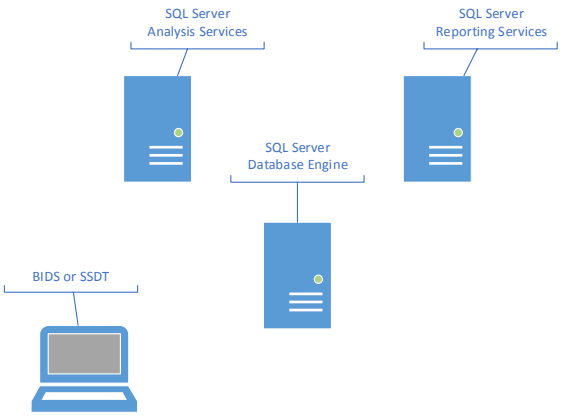

Figure 4
Microsoft SQL Server Recommended Infrastructure Configuration

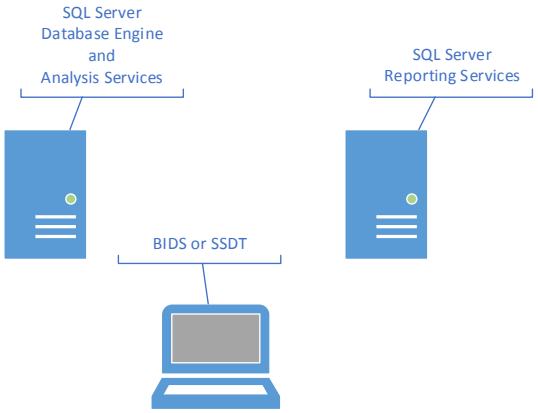

Figure 5
BI Solution Test Infrastructure

TEST RESULTS

Due to hardware limitations we will not be able to construct a test environment to design and test the implementation of a new data warehouse with its respective extract-transform-load workflow definition. But we will be able to design and test the temporary solution that is expected to solve the processing data performance issue.

Table 3
SSIS Analysis Services Processing Task Execution Time for New Solution

Task	Start	End	Duration
SSAS Processing Dimensions	11:29:19am	11:33:52am	0:04:32
SSAS Processing Measurements	11:33:52am	12:37:19pm	1:03:26

Table 4
SSIS Analysis Services Processing Task Execution Time Differences

BI Solution	Processing	Duration
Old BI Solution	Full	*12:00:00
New BI Solution	Incremental with View Tables	1:07:58

*Since we were not able to complete task for full process in the test environment we took the execution time for the full processing from the production environment logs.

As we observe from the table 4, the greatest benefit of this solution is that it significantly improves the overall processing time. The new solution only takes 9% of the time it takes in the production environment.

The only problem is that the solution requires close monitoring of the processing. If the solution does not process the new data or any data in the data source have to be changed. The solution will need to process the data completely in order to present the correct results in different reports or pivot tables generated with Excel.

CONCLUSION

Sometimes there are business constraints that will limit the design of a business solution. In this particular case I was able to observe that the BI solution currently presented in the production environment doesn't follow the best practices in regards to the design of a data warehouse.

However, we can still fix the current solution by designing a data warehouse and implementing indexes that will allow us to optimize the whole application. The only problem would be the cost involved in the solution. The current cost could be greater than the cost it would have be if it was designed to scale from the inception of the solution as it will now require twice the hardware to avoid downtime.

The cost estimated to provide the new fixes to the BI solution would be another area of investigation, but I believe that we can minimize the expenses by using services such as Windows Azure, This services allow us to design and test the solution prior to deploy the same in our own private cloud.

REFERENCES

[1] *SQL Server - Lesson 13: Relationships and Data Integrity*, 2007 - 2012. Retrieved November 1, 2013, from functionx: http://www.functionx.com/sqlserver2005/Lesson13.htm

[2] Ali, A. (2012, 2 17). *SSAS - Best Practices and Performance Optimization - Part 1 of 4*. Retrieved on November 1, 2013, from mssqltips: http://www.mssqltips.com/sqlservertip/2565/ssas--best-practices-and-performance-optimization--part-1-of-4/.

[3] Atkinson, P., & Vieira, R., *Beginning Microsoft SQL Server 2012 Programming*, Indianapolis: John Wiley & Sons, Inc., 2012.

[4] *Define Named Calculations in Data Source View (Analysis Services)*, (n.d.). Retrieved on November 1, 2013, from TechNet: http://msdn.microsoft.com/en-us/library/ms174859.aspx.

[5] *Design Database Diagrams*, (n.d.). Retrieved on November 1, 2013, from TechNet: http://msdn.microsoft.com/en-us/library/ms171971.aspx.

[6] Harinath, S., Pihlgren, R., Guang-Yeu Lee, D., Sirmon, J., & Bruckner, R. M., *Microsoft SQL Server 2012 Analysis Services with MDX and DAX*, Indianapolis: John Wiley & Sons, Inc., 2012.

[7] Jorgensen, A., Wort, S., LoForte, R., & Knight, B., *Microsoft SQL Server 2012 Administration*, Indianapolis: John Wiley & Sons, Inc., 2012.

[8] Larsen, G. A. (2004, 11 30). *SQL Server Undocumented Stored Procedures sp_MSforeachtable and sp_MSforeachdb*, Retrieved on November 1, 2013, from Database Journal: http://www.databasejournal.com/features/mssql/article.php/3441031/SQL-Server-Undocumented-Stored-Procedures-spMSforeachtable-and-spMSforeachdb.htm.

[9] *Query Fundamentals*, (n.d.). Retrieved February 4, 2014, from TechNet: http://technet.microsoft.com/en-us/library/ms190659%28v=sql.105%29.aspx.

[10] *sp_spaceused (Transact-SQL)*, (n.d.). Retrieved November 1, 2013, from TechNet: http://technet.microsoft.com/en-us/library/ms188776.aspx.

[11] *Using Data Sources and Data Source Views*, (n.d.). Retrieved November 1, 2013, from TechNet: http://technet.microsoft.com/en-us/library/cc505861.aspx.