A Comparison of OLAP and Coherence Large Scale Aggregations

Kimberley Tate

Lili Zhang

Xinyue Zhong

January 2015

A Comparison of OLAP and Coherence Large Scale Aggregations

A Major Qualifying Project

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree in Bachelor of Science

By

_____

Kimberley Tate, Actuarial Mathematics

_____

Lili Zhang, Industrial Engineering and Mathematical Sciences

_____

Xinyue Zhong, Electrical & Computer Engineering

Date:

January 16, 2015

Sponsoring Organization:

BNP Paribas

Project Advisors:

_____	_____
Jon Abraham	Xinming Huang
_____	_____
Arthur Gerstenfeld	Kevin Sweeney

Table of Contents

List of Figures

List of Tables

Abstract

The goal of the project is to develop a generic mechanism for cache-based aggregation of data. This will allow the end user to have a summary view with multiple pivot points. To achieve this goal, the following steps will be taken:

1. Converting the database to files
2. Loading the files into caches
3. Aggregating the data in the caches

Coherence and aggregators will be used to complete these tasks. To write the code for these processes, Java, C# and SQL are the programming languages that will be used.

Acknowledgements

We would like to take this opportunity to thank the individuals that aided in the completion of this project.

First, we would like to thank Worcester Polytechnic Institute and the Interdisciplinary and Global Studies Division for making the necessary arrangements for us to come to New York. They have made this opportunity possible. We would also like to thank our advisors: Jon Abraham, Arthur Gerstenfeld, Xinming Huang and Kevin Sweeney. They provided us with guidance and encouragement during our time in New York.

Secondly, we would like to thank BNP Paribas for providing us with a place to work every day and the resources needed to construct our project. Special thanks to Kunal Changela, Anton Zlotskiy, Matthew Palmer and Andrew Clark for overseeing our project and acting as our academic advisors during our time at BNP.

Executive Summary

History of BNP Paribas

The BNP Paribas Group played an important role in European economic history. Over time, several banks that supported the economic development in their country, were brought together. During the time that France was going through an economic and political crisis, Comptoir National d'Escompte de Paris (the C.N.E.P.), the Comptoir National d'Escompte de Mulhouse, the two ancestors of BNP, and the modern bank were all introduced. To revive lending, public authorities and some small retailers created establishments that provided the public with discounts. The ancestors of BNP helped to develop international trade by opening branches in China, India, Egypt and Australia, to help companies trade with these countries. Later, France was defeated by Prussia and to finance the loan for the liberation, several European bankers decided to set up Paribas, which was then called the Banque de Paris et des Pays-Bas. The bank also led large international operations, including the first major railway in China in 1899. This was co-financed by SociétéGénérale de Belgique which joined the BNP Paribas Group in 2009 under the name of Fortis. In 1966, The Compagnie Bancaire joined Paribas. The public authorities decided to merge The Banque Nationale pour le Commerce et L'Industrie (the B.N.C.I) and C.N.E.P. to form the largest bank in France: Banque Nationale de Paris (BNP). From that moment on, BNP supported the massive changes in society. When women become emancipated and young people liberated, they were allowed to possess a bank account. BNP opened hundreds of local branches and began offering bank cards from 1967. In the '80s, BNP introduced new communication technologies that made it possible for customers to consult their bank accounts via the French Minitel. Between 1993 and 2003, BNP Paribas became a European leader with an international dimension through large mergers and acquisitions; particularly in the US with the accelerated growth of Bank of the West, in Italy with BNL, in Turkey with TEB, in Belgium with Fortis, and in Luxemburg with BGL. All these banks, most of them with a history of over a hundred years, have accompanied the economic development in their country. BNP

Paribas therefore became a major global group, while, at the same time, preserving its values. For 200 years, BNP Paribas has remained a responsible bank, true to its original role of financing the real economy and serving all its clients, wherever they are in the world.

Project Summary

Below is a diagram of the technology roadmap of the risk presentation system. The section highlighted in red is the portion of the system that the cache-based system will replace. Currently, there are processes that are taking information from the database and saving it in a cube. This allows the users to run queries on the cube to retrieve data. These processes can be very slow and take up a lot of space. The goal of the project is to develop a generic mechanism for cache-based aggregation. In essence, the cube will be replaced by a cluster of caches which should be more efficient and use less server space. To ensure that the proposed process works as expected, a comparison of OLAP and Coherence large scale aggregations will be done to compare the difference in update rates, number of nodes and number of clients.

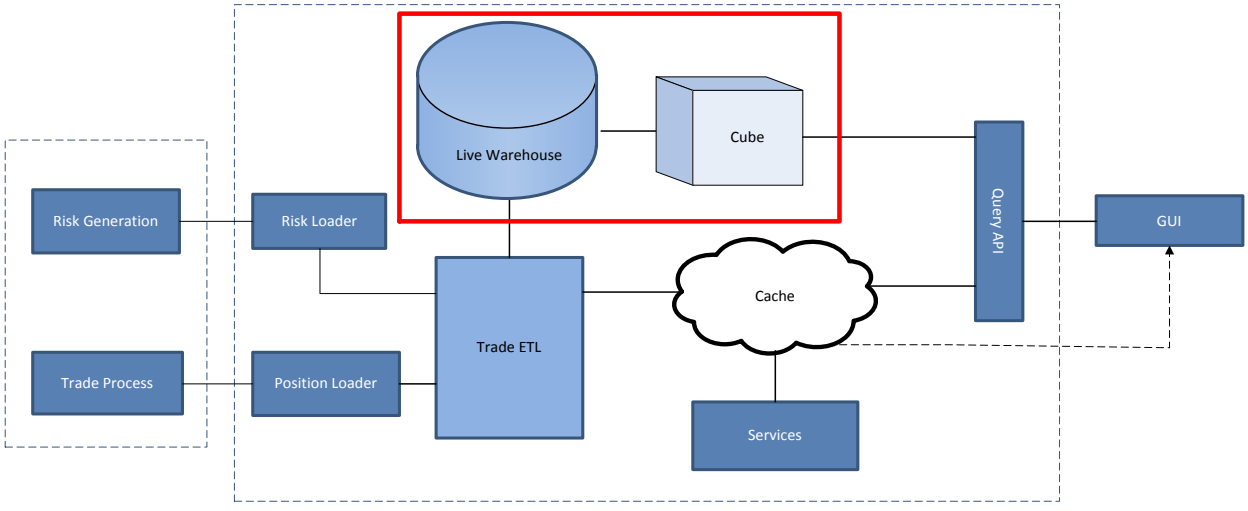


Figure 1: Technology Roadmap (Street, 2012)

Chapter 1: Introduction

1.1 Goal

Currently, to get useful information from the database, BNP Paribas is using OLAP cubes. An OLAP Cube is a multidimensional database that is optimized for data warehouse and online analytical processing (OLAP) applications. This can take up a lot of server space. The motivation to succeed is to help the company save money by getting rid of the cubes and developing a mechanism that will perform the same task faster and use less space. The goal of the project is to develop generic mechanisms for cache-based aggregation of data to replace an OLAP database. Later, a comparison between the OLAP database and the cache-based aggregation will be done by running different performance tests focusing on update rates, number of nodes and number of clients.

1.2 Current Process

Currently, BNP is performing the aggregation of data from cache using C# processes. These processes utilize OLAP cubes and can be moved into Coherence aggregators.

A variety of processes load data from archive files and servers into the database. These processes can be slow. A message is then sent to an OLAP cube to load the data from the database into the OLAP cube. This is done separately for each data set. The OLAP cube runs a query in the database which joins the trade and risk data, using various identification numbers, then does a simple aggregation and returns the data to the cube. This particular query is fairly slow. Once the data is in the cube, it is made visible to the Graphical User Interface (GUI). Users then run queries against the OLAP cube to get their start-of-day risk information.

The slowest part of the current process is loading the data into the cube; queries on the cube are fast. There are two OLAP cubes being used in this process; one for querying and another for loading then

they swap. This is done because the users are not allowed to query the cube which data in being loaded to it. So to ensure that the user always has access to the data, the data is being loaded twice. This process is shown in the diagrams below. The section highlighted in red is the portion of the system that the cache-based system will replace.

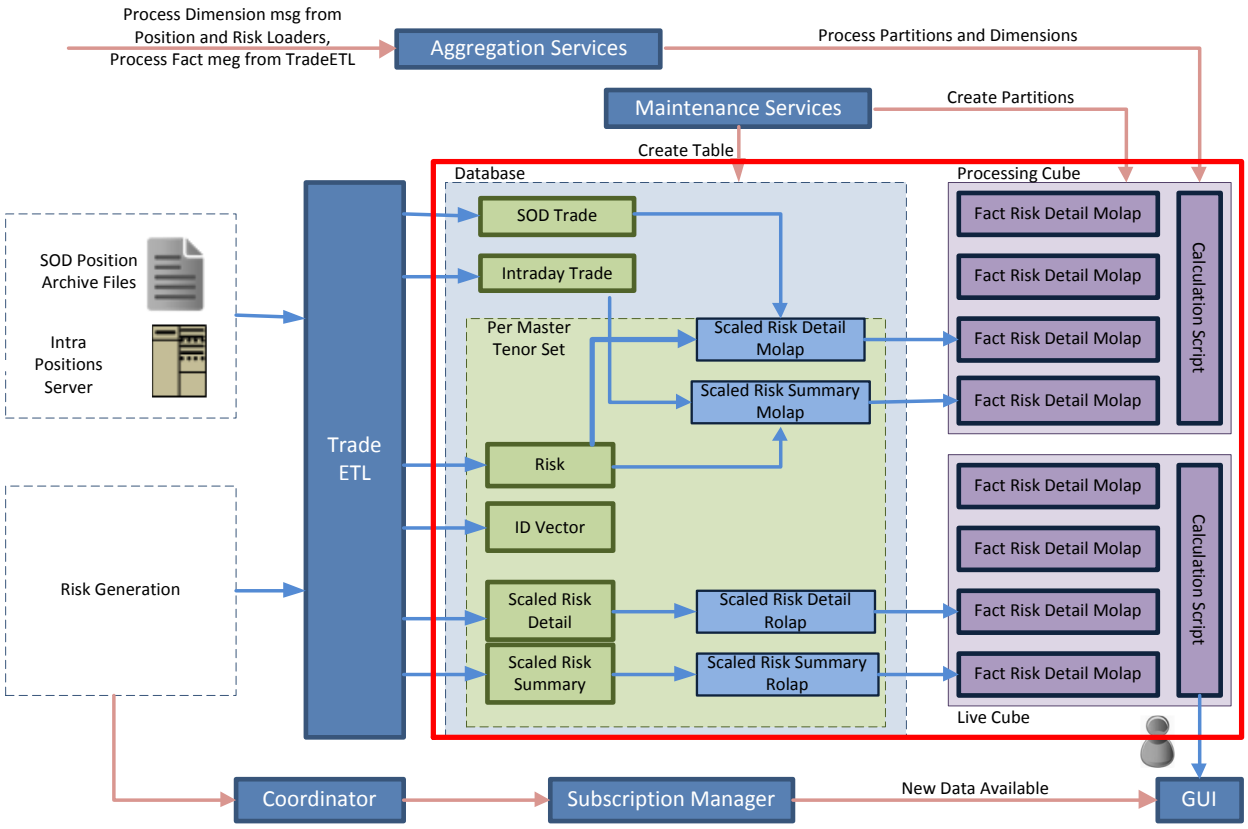


Figure 2: Risk Presentation System (Street, 2012)

1.3 Proposed Process

The main purpose of this project is to get rid of the cubes. To accomplish the goal the following steps will be taken:

1. Convert the databases in to files
2. Convert the files into a cluster of caches

3. Use aggregators to produce aggregated data from the cluster

After these have been accomplished, the will be an assessment of the difference in processing speeds, update rates, number of clients and number of nodes in the cluster. This will help to create a range of the amount of data that can be processed using the generic mechanism. One important characteristic of the current process that will need to be transferred the cache-based environment is the fact that the information provided is in real-time. By default, Coherence aggregators do not operate in real-time. This means that listeners will need to be added to the proposed process in order to maintain this feature. Listeners tell the system to look out for specified activity and update the information as needed.

Chapter 2: Literature Review

2.1 OLAP Cubes

An OLAP Cube is a multidimensional database that is optimized for data warehouse and online analytical processing (OLAP) applications. (*Rouse*, 2012)Before OLAP cubes were introduced, the user would have to structure a request that would aggregate the data in the databases before getting useful information. This process, depending on the size of the database and the data requested, could take minutes, even hours to complete. To solve this problem, OLAP cubes were made to save aggregated data to answer popular queries. (Williams, *2004*)Although it stores data like a traditional database does, an OLAP cube is structured very differently. This is a method of storing data in a multidimensional form. One of the most popular uses of these cubes can be visualized in the form of pivot tables, where rows, columns and filters represent attributes and values in cells are appropriate aggregates of measures (data). (*Pandre*, *2014*)

The end user will access information on the cube using a Data Visualization tool. The user may change the data's orientations and define analytical calculations. Some common operations include slice and dice, drill down, roll up and pivot. These will be discussed next.

2.1.1 Cube Operations

Slice and Dice

A slice is a subset of a multi-dimensional array corresponding to a single value for one or more members of the dimensions not in the subset. (*Pandre*, 2014)

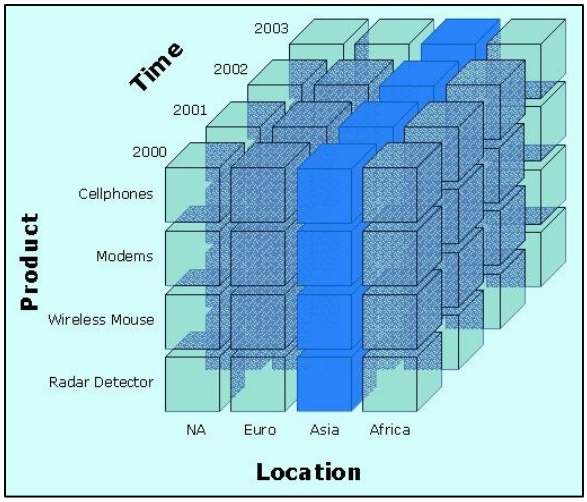


Figure 3: Slice Operation on OLAP Cube (Pandre, 2014)

A dice operation is a slice on more than two dimensions of a cube.

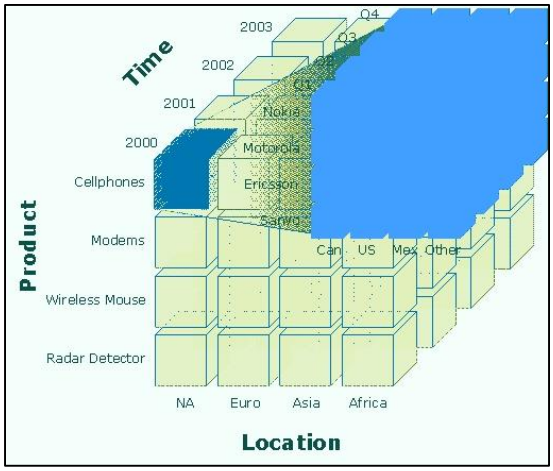


Figure 4: Dice Operation on OLAP Cube (Pandre, 2014)

Drill Down/Up

This operation allows the user to navigate among levels of data rang9ing from the most summarized (up) to the most detailed (down).

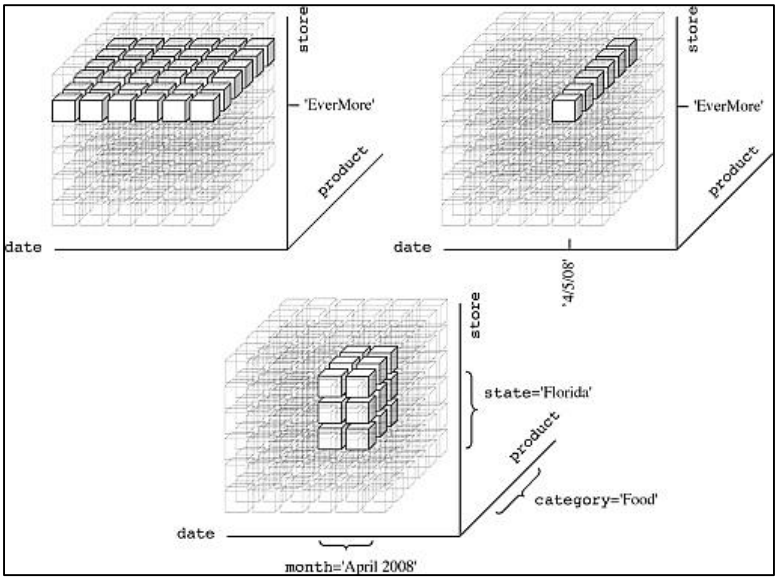


Figure 5: Drill Down/Up Operation on OLAP Cube (Pandre, 2014)

Roll-Up

This involves computing all the data relationships for one or more dimensions. This may be done by defining a computational relationship or formula.

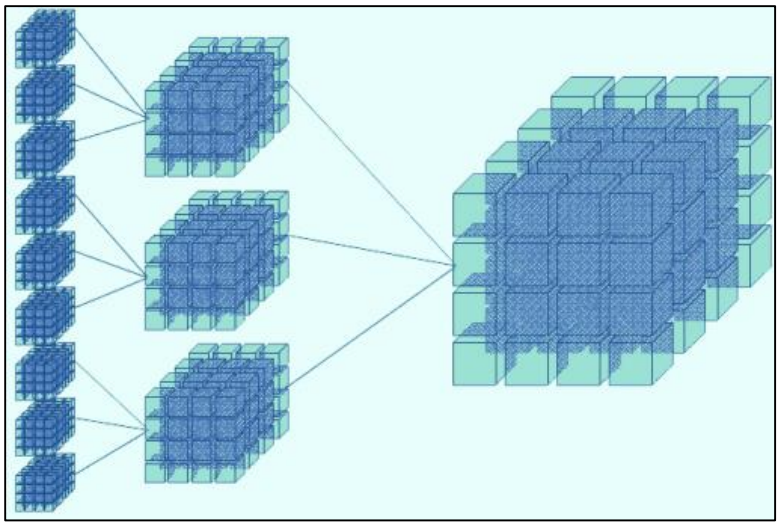


Figure 6: Roll-Up Operation on OLAP Cube (Pandre, 2014)

Pivot

This is also known as the rotate operation because it rotates the data in order to provide an alternative presentation.

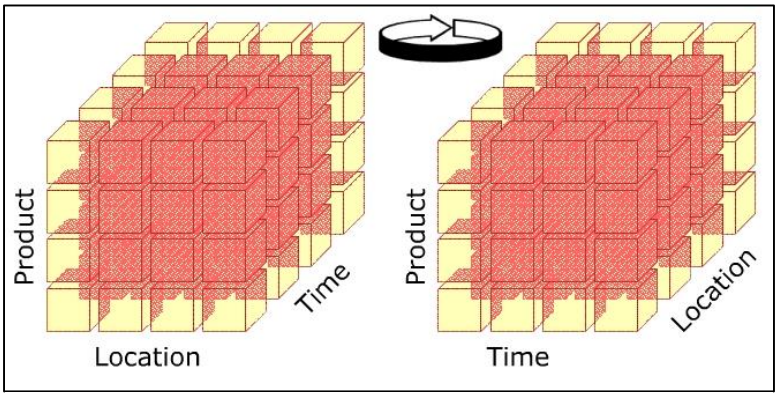


Figure 7: Pivot Operation on OLAP Cube (Pandre, 2014)

2.1.2 Types of OLAP Cubes

OLAP Systems are distinguish by a letter in front of OLAP that represents the type of OLAP Technology this is implemented. The major OLAP technologies are ROLAP, MOLAP and HOLAP and the other types will be discussed in this report are WOLAP, DOLAP, Mobile OLAP and SOLAP. There are other OLAP systems that are not as popular as the ones described in this paper such as Database OLAP (DOLAP), Remote OLAP (ROLAP), Local OLAP (LOLAP) and Real-Time OLAP (RTOLAP). *(OLAP.com Team)* In the current system at BNP, ROLAP and MOLAP are being utilized.

Relational OLAP (ROLAP)

ROLAP works mainly with data that is stored in a database that consists of files that have related information. This allows the user to perform a function similar to the 'slice and dice' operation using an SQL *'where'* clause. This system is considered the fastest growing OLAP technology style. An advantage of this system is the fact that it can handle large amounts of data. The data size limitation in the ROLAP system is dependent on the size limitation on the database. This system also enhances the functionalities of a relational database. A disadvantage of this system is that performance can be slow if the data size is large. All the reports are essentially SQL queries which take time to run. It may be

difficult to run complex calculations on the system because the functionalities are limited to what SQL can do. This disadvantage is being counteracted by ROLAP sellers who develop tools to execute complex functions on the system as well as give the users the ability to define their own functions.*(IT Toolbox Popular Q&A Team, 2014) (OLAP.com Team)*

Multidimensional OLAP (MOLAP)

This system is the most popular OLAP system. It differs from ROLAP in that it is stored in a multidimensional cube versus in a relationship database. Data is structured according to the client's report requirements with the calculations pre-generated on the cubes. This pre-generation of calculations aids in the ability to do complex calculations and return results quickly. These cubes are built for fast, flexible data-modeling and calculations. Since these models incorporate advanced array-processing techniques and algorithms, they can store data efficiently and process calculations much faster than ROLAP. A drawback of this system is the limitation on the amount of data that it can handle. All the calculations are performed when the cube is built so it is not possible to store a large amount of data in the cube. In short, only summary-level information can be stored in the cube. Therefore, relevant data must be transferred from the relational systems. This can potentially be a redundant recreation of data. Also, MOLAP systems are proprietary. They usually require additional human and capital resources. *(IT Toolbox Popular Q&A Team, 2014) (OLAP.com Team)*

Hybrid OLAP (HOLAP)

This technology is a combination of ROLAP and MOLAP. When handling summary-level information, HOLAP leverages cube technology for faster performance. When detailed information is requested, HOLAP can go directly to the relational database that the cube is derived from. These systems allow for larger quantities of detailed data to be stored in relational tables while aggregations are stored in pre-

calculated cubes. It has better scalability, quick data processing and flexibility in accessing data sources. *(IT Toolbox Popular Q&A Team, 2014) (OLAP.com Team)*

Web OLAP (WOLAP)

Web OLAP, also as Web-enabled OLAP, is an OLAP system that is accessible via a web browser. The three components that make up the systems are a client, a middleware and a database server. The main advantages are the inexpensive nature of the system, the enhanced accessibility and the ease of installation, configuration and deployment process. However, it fails in comparison to other traditional OLAP systems where functionality, visual appeal and performance are concerned. *(OLAP.com Team)*

Desktop OLAP (DOLAP)

With this system the user is able to download a portion of the database and work with it locally. DOLAP is easier to deploy but lacks a great deal of the functionality that the other OLAP systems have. *(OLAP.com Team)*

Mobile OLAP

This system allows the user to access OLAP data and applications remotely using their mobile devices. *(OLAP.com Team)*

Spatial OLAP (SOLAP)

This was introduced to combine capabilities of Geographic Information Systems (GIS) and OLAP. This allows the users to manage both spatial and non-spatial data. This includes data that is not only in alphanumeric form but also in images and vectors. This system makes exploration of data easy and quick. *(OLAP.com Team)*

2.2 Cache Clusters

2.2.1 What is a Cache Cluster?

A cache cluster is a collection of one or more cache nodes, all of which run an instance of supported cache engine software. When setting up a cache cluster, the number of cache nodes can be specified and the properties for each cache node can be controlled by a cache parameter group. All the nodes within the cluster will be of the same node type, and have the same parameter and security settings. Every cluster requires a cluster identifier. This is a name given to the cluster by the programmer. This allows users to identify the cluster in commands so it must be unique.

2.2.2 How Cache Clusters Work

A cache cluster uses one configuration that is shared among all the caches in the cluster. This configuration contains general information such as security, session information, and caching rules, which is the same for all the caches in the cluster. The configuration also contains cache-specific information such as capacity, administration and other ports, resource limits, and log files, for each cache. As caches are added to the cluster, the cache-specific information is updated.

When a cache fails, the failure can be detected by other caches in the cluster which will automatically take over ownership of the data saved in the failed cache. When a cache is removed, the caches do the same thing if it were to fail, that is, they take over the ownership of the data of the removed cache and the configuration information updates.

Oracle AS Web Cache uses the relative capacity of each cache to automatically distribute ownership of data among the caches in a cluster. It is also able to save the data that is on-demand. This way the data can be retrieved faster and with a lessened chance of error. This results in improved overall performance.

If there is a request for data that is not in one of the caches, the request is sent to the web server. When there is a request for data that is saved in one of the caches, the next operation depends on whether or not the data is saved in the cache that received the request.

For example, if the content is saved in cache_Y, cache_Y returns the content to the client.

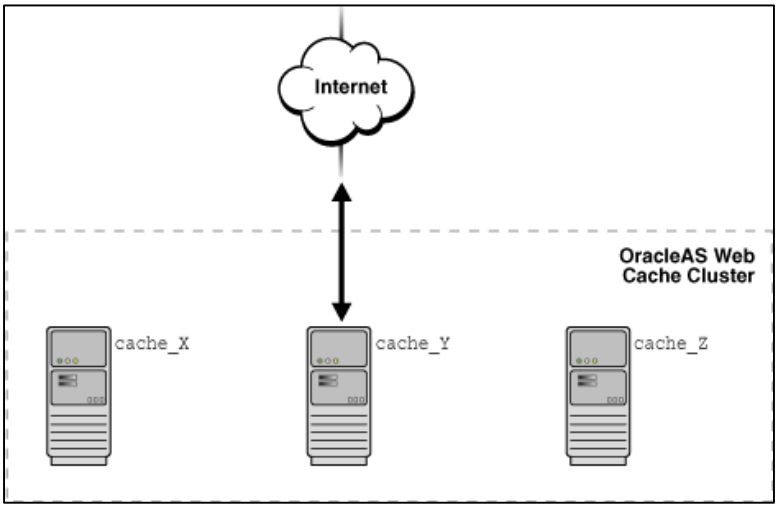


Figure 8: Request for Cached Data (Oracle, 2006)

If cache_Y is not the owner of the requested content, cache_Y will send the request to the owner. For the sake of demonstration, the owner will be cache_Z. If the content is indeed in cache_Z, the content will be sent to cache_Y and saved as on-demand content.

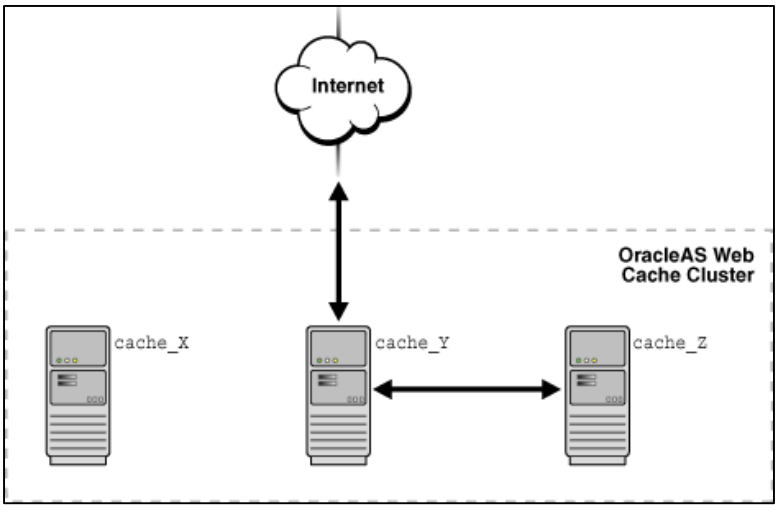


Figure 9: Sending Data between Caches (Oracle, 2006)

If the requested content is not in cache_Z, then cache_Z will send a request to the application web server which will send the requested content to the owner cache. At this point the owner cache, cache_Z, will be able to send the data to the cache that requested it, cache_Y.

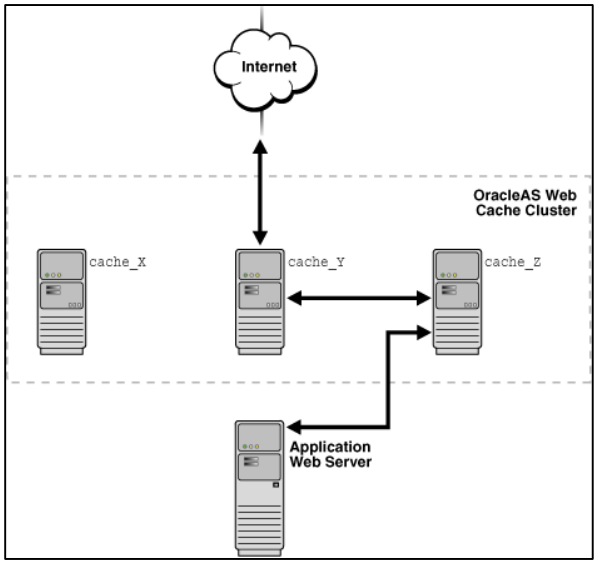


Figure 10: Requesting Data from Application Web Server (Oracle, 2006)

2.2.3 Advantages of Cache Clusters

Firstly, cache clusters provide high availability. Should a cache fail, because of the communication between the caches in a cluster, it can be easily detected and the task may be directed to the most available cache.

Secondly, since the data is shared amongst multiple caches, more content can be stored and more connections can be supported, therefore making the system scalable. In addition, since multiple caches are being utilized, there are more CPUs contributing to the processing power, allowing for multiple requests to be executed in parallel.

Thirdly, the load on the application web server is reduced. Since popular objects are stored in more than one cache, if one fails there is a great chance that the requested data is stored in one of the surviving caches. As a result, fewer requests are sent to the application web server when a cache fails. Another

way that the load is reduced on the application web server in a cache cluster is seen when a failed cache returns to operation. The misses will be routed to other caches before being routed to the application web server. If the caches were operating independently, when the failed cache returns to operation, all the misses would be routed to the application web server.

Also, since the cluster is controlled by one set of validation rules, the cached data is more likely to be consistent across all caches in the cluster. This characteristic makes the cluster easier to manage.

2.3 Portable Object Format (POF)

The Portable Object Format, also known as POF, is a binary format that is not specific to any language or platform, that is found to be significantly efficient in both time and space. POF is extensively used in benchmarking suite. A benchmarking suite is running a set of programs that assesses performance in relation to speed and efficiency.

POF serialization is the process of encoding an object into a binary format. A POF Serializer can be used to convert almost any object into a POF value. It is a crucial component when working with Oracle Coherence to move data around the network. Based on sample tests that implemented POF to load data, it was proved that POF serialization takes one-third to one-sixth of the time loading would have taken without POF. Therefore, POF was implemented in the project.

The POF interface is made up of two methods. The following list gives a description and simple example of both methods.

1. PofReader: This method provides the capability of reading a set of non-primitive user types from a POF stream as an ordered sequence of indexed properties.

```
public void readExternal(PofReader reader) {
    riskId = pofreader .readInt(0);
    riskValue = pofreader. readInt(1);
```

```
        riskDate = pofreader.readString(2);
    }
```

2. PofWriter: This method provides the capability of writing a set of non-primitive user types from a POF stream as an ordered sequence of indexed properties.

```
    public void writeExternal(PofWriter writer) {
        pofwriter.writeInt(0,riskId);
        pofwriter.writeInt(1,riskValue);
        pofwriter.writeString(2,riskDate);
    }
```

These two methods will automatically appear after importing *com.tangosol.io.pof.PortableObject* in Java programs. As shown above, each object must be defined with an appropriate data type. The data type may be integer, string, long, double, Boolean, etc. In addition, POF objects are indexed. This means that each object is followed by a numeric reference that is unique to each element of POF stream.

POF Advantages

POF has many advantages, one of which is the fact that it is not language specific. This means it can be used in Java, .NET, and C++, etc., just to name a few of the languages POF can be used in. Another advantage of POF is its efficiency. In a simple test class with an object that consisted of a string, a long (integers between $-(2^{31}-1)$ and $2^{31}-1$), and three integers, the serialization was seven times faster, and the POF value produced was one-sixth the size without POF. It is also versionable, which means that the objects could evolve to have forward and backward compatibility. It also supports the ability to externalize serialization logic. In addition, POF is indexed. This means that it is possible to extract values without deserializing the entire object.

The POF value contains a type identifier and a value. The type identifier is an integer, where numbers less than zero are used for pre-defined types and numbers greater than zero are used for custom user types.

2.4 Aggregators

An aggregator is a process in Coherence that creates relationships between caches. It is used in conjunction with a filter that retrieves a subset of objects from the cache. When a single result is required as the output, for example, the total number of all orders for a particular customer, a good approach would be to calculate partial results on each cache node for the data it manages, and to aggregate those partial results into a single answer before returning it to the client. To do this, Coherence aggregators can be used. Coherence has a number of useful built-in aggregators. The following table shows a list of aggregators and the description of their functionalities.

Aggregator	**Description**
BigDecimalAverage	Calculates the average for a set of numeric values extracted from the cache entries and returns the results as a BigDecimal.
BigDecimalMax	Returns the maximum value, as a BigDecimal, for a set of numeric values extracted from the cache entries
BigDecimalMin	Returns the minimum value, as a BigDecimal, for a set of numeric values extracted from the cache entries.
BigDecimalSum	Calculates the sum for a set of numeric values extracted from the cache entries and returns the results as a BigDecimal.
DoubleAverage	Calculates the average for a set of numeric values extracted from the cache entries and returns the results as a Double.
DoubleMax	Returns the maximum value, as a Double, for a set of numeric values extracted from the cache entries.
DoubleMin	Returns the minimum value, as a Double, for a set of numeric values extracted from the cache entries.
DoubleSum	Calculates the sum for a set of numeric values extracted from the cache entries and returns the results as a Double.
LongMax	Returns the maximum value, as a Long, for a set of numeric values extracted from the cache entries.
LongMin	Returns the minimum value, as a Long, for a set of numeric values extracted from the cache entries.
LongSum	Calculates the sum for a set of numeric values extracted from the cache entries and returns the results as a Long.
ComparableMax	Returns the maximum value for a set of Comparable values extracted from the cache entries.
ComparableMin	Returns the minimum value for a set of Comparable values extracted from the cache entries.
Count	Returns the number of values in an entry set; equivalent to SQL's "select count(*)".
DistinctValues	Returns a set of unique values extracted from the cache entries; equivalent to

	SQL's "select distinct".
CompositeAggregator	Executes a collection of aggregators against the same entry set and returns the list of results, one for each aggregator in the collection.
GroupAggregator	A wrapper aggregator that allows you to split entries in a set based on some criteria and to aggregate each subset separately and independently.

Table 1: Examples of Basic Aggregators

Advantages

When using an aggregator, the amount of data that can be moved across the wire to the aggregator instance itself is limited. This is an advantage because it reduces the network traffic significantly and ensures that the network is used as efficiently as possible. It also allows performing the aggregation in parallel, using full processing power of the Coherence cluster.

2.5 Filters

Often, it is necessary to iterate through objects in a database and filter based on a number of criteria. Filters are a means of calling a subset of data that has a specified characteristic. They are usually used in conjunction with an aggregator. This means that in one instance, it is possible to tell the system to retrieve the subset and execute a specified aggregation on that subset. The three main filters used to complete the project were the *'AndFilter'*, the *'OrFilter'*, and the *'EqualsFilter'*. These will be described below.

The *'AndFilter'* is the filter for the logic 'AND'. This filter takes two conditions and returns the subset of data that met both conditions. The *'OrFilter'* is the filter for the logic 'OR'. This filter takes two conditions and returns the subset of data that met at least one for the conditions. The *'EqualsFilter'* compares two values and returns the subset of data that met the equality condition.

2.6 Multi-Dimensional Expressions

Multi-dimensional eXpressions (MDX) is an extension of SQL that allows the user to query multidimensional data. It is most commonly used for OLAP databases. The language has two analytical capabilities; as an expression language to calculate values and as a query language to retrieve data. Below is an example of a simple MDX query:

```
SELECT
    { [Measures].[Sales Amount],
        [Measures].[Tax Amount] } ON COLUMNS,
    { [Date].[Fiscal].[Fiscal Year].&[2002],
        [Date].[Fiscal].[Fiscal Year].&[2003] } ON ROWS
FROM [Adventure Works]
WHERE ( [Sales Territory].[Southwest] )
```

Figure 11: Example of a Simple MDX Query

The SELECT clause allows the user to the query axes. Query axes are boundaries for the result returned.

In the example above, the query axes are set to Sales Amount and Tax Amount as Measure dimensions and 2002 and 2003 as Date dimensions. This clause will tell the program to return a table with 2 columns; Sales Amounts, Tax Amount, and Date. The table will be populated with objects dated 2002 and 2003. The FROM clause specifies the cube will be queried. In the example above, the Adventure Works cube is being queried. The WHERE clause is optional and allows the user to specify the slicer axes. The slicer axes filter the data that will be returned. In the example above, the slicer axis is Southwest in the Sales Territory dimension. This means that the table returned will only have the objects whole Sales Territory is Southwest. This is the basic format of most MDX queries.

2.7 Risk Management

The system that this project revolves around is used by traders to manage risk. When investing, it is inevitable that there is risk involved. Even depositing money into a bank account is risky, because there is a chance that, due to inflation, the money saved will not be worth as much in the future. Also, money

deposited in a bank account will only earn the interest that the bank is paying. People choose to take risks because it allows them to reach their savings goals faster. Investments such as stocks and bonds are proven to grow more rapidly than saving money in bank accounts. According to *Investopedia*, risk management is "the process of assessing, managing and mitigating losses." The best way to manage the risk associated with investing is to build a diverse portfolio. This way, should the return on some investments be disappointing, the overall portfolio results may be positive. (*FINRA*)

2.7.1 Types of Risk

Systematic Risk

This relates to risk that can affect the overall economy. Below are some of the most common systematic risks.

Interest-Rate Risk

This refers to the risk that the value of an investment will change due to a change in interest rates. An example of this is the relationship between bonds and interest rates. When the interest rates increase, bond issues must increase coupon rates in order to attract investors. This causes the price of existing bonds to decrease because investors will prefer newer bonds that pay the higher rate. Conversely, when interest rates fall, the maturing bonds or bonds that are paid off before maturity must be reinvested at a lower yield. *(FINRA)*

Inflation Risk

This refers to the risk that the prices of goods and services will increase. This will increase the cost of living, therefore lowering individual purchasing power. This risk is closely related to interest-rate risk, as interest rates generally rise with inflation. Therefore, inflation can affect the value of investments. When prices increase, lenders will demand higher interest rates to compensate for the loss of purchasing power. This will cycle back to losing the value of bonds, because, as previously stated, when interest

rates increase then newly issued bonds will be offer higher interest rates making them more favorable to investors. (*FINRA*)

Currency Risk

This risk is related to the change in the exchange rates between currencies. If money needs to be converted to make an investment, then any change in the exchange rate can increase or reduce the return on the investment. This risk can be managed by diversifying a portfolio; ensuring that only a portion of the portfolio is international investments. *(FINRA)*

Liquidity Risk

This risk refers to "the lack of marketability of an investment that cannot be bought or sold quickly enough to prevent or minimize loss." (*Investopedia*) This risk is usually higher in over-the-counter markets and small-capitalization stocks. Foreign markets are also subject to this risk. Depending on the size of the foreign market, the number of companies listed and the hours of trading, the ability to buy or sell a foreign investment may be limited.

Sociopolitical Risk

This refers to the effect instability and unrest, such as terrorist attacks and wars, in regions has on investment markets. Whether these events are actual or anticipated, they impact investor attitudes toward the market cause stock prices to fluctuate. In addition, changes in politics can trigger a change in laws. This may restrict investments with non-citizens or nationalized businesses.

Nonsystematic Risks

These are risks that affect a small portion of the economy. The affected parties may include a small number of companies or investments and this risk is usually associated with investing in a specific product, company or industry sector.

Management Risk

This refers to the effects of bad management decisions or other internal missteps. This risk is difficult to assess because even if the company is thoroughly researched before investing, it may appear to have solid management, but there is no way to know if a competitor will release a superior product or anticipate a financial or personal scandal that may reduce the stock price or bond rating.

Credit Risk

This is also known as default risk. This risk refers to the possibilities that the bond issuer will not pay the interest as planned or the principal at maturity. This arises when a borrower is expecting to use future cash flows to pay a current debt.

2.7.2 Assessing Risk

There are three basic steps to assessing risk. Firstly, one will have to understand the risk associated with certain categories of investments. Secondly, an investor needs to determine the kind of risk they are comfortable taking and lastly, they will have to evaluate specific investments. It is usually advised that these steps be followed with an investment professional to effectively assess risk.

Categorizing Risk

The first step is to understand the types of risk that is posed by a particular category of group of investments that one might be exposed to. These categories are called “asset classes”. The asset classes that will be addressed are stocks, bonds, and cash. These are separate classes because each of them uses money in different ways making their associated risks different.

Stocks

One of the greatest risks associated with investing in stock is volatility. They do not have a fixed value because they reflect investor demand, which can result in significant price changes in short periods of

time. The other side of this is that in the long run, short term price changes can smooth out to show gradual price changes or stability.

Bonds

Bonds have a fixed value, and if held until maturity, one will get back the par value plus the interest the bond earns. The most common risk associated with this investment is default risk, which was discussed in the previous section. This is the risk that the issue fails to pay the interest and/or principal. Another risk posed by this investment is market risk. This is a threat when the bond is sold before maturity. The price in the secondary market is usually lower than the par value, so the sale may result in a loss. The market value may also decrease if there is an interest increase between the time the bond was issued and the time it matures.

Cash

The primary risk associated with cash investments, including U.S. treasury bills and money market mutual funds, is inflation. Also, money in money market funds is not usually insured.

Accepting Risk

The second step is to decide which risks one is comfortable with. This decision is usually driven by age, goals and timeline to meet them, financial responsibilities, and other financial resources. In general, the younger an investor is, the more investment risk they can afford to take, simply because they expect that there is more time to make up for any short term losses. Also, the younger an investor is the more stock and stock funds they may consider buying. As one gets closer to retirement, managing risk means moving some assets out of more volatile stock and stock funds into equities and bonds that produce income. These investments should have great growth potential since the demand for money will increase should the investor live longer than expected. An investor who is the primary source of financial support for a number of people may be willing to take less investment risk than they would if

they were only financial responsible for themselves. On the other hand, the larger their source of income, the more willing they will be to take on additional risks.

Evaluating Investments

The third and final step to assessing risk is evaluating specific investments. For stocks and bonds, it is wise to get as much information as possible about the issuer, since the value of these investment are directly linked to the strength of the company or the government agency behind them. Every public company is required to register with the Securities and Exchange Community (SEC) and provide updated information on a periodic basis. This information includes detailed information about the company, the people who run it, the risks of investing in the company and much more. These documents are usually for finding information about pending lawsuits, regulatory investigations or other issues that could have a negative impact on the company's bottom line. With all this information at investors' disposal, they need to keep in mind that past results cannot predict future performance or a new manager's performance. It is also important to check independent rating services that review specific bonds. Rating companies provide information about issuers from a different point of view than the company documents. They are more focused on the issuer's ability to meet its financial obligations. Even with rating companies, investors should keep in mind that ratings are not perfect and do not indicate which the investment will go up or down. Most importantly, investors should remember that managing risk does not mean avoiding risk altogether.

Chapter 3: Methodology

The main goal of the project is to develop a generic mechanism for cache-based large scale aggregation and compare OLAP and Oracle Coherence systems. To accomplish the end goal, the following steps were needed:

1. Extracting data from current database and converting them into .csv files,
2. Loading data from .csv files into caches,
3. Connecting data from different tables in the caches to create a joint table,
4. Using Coherence aggregators to do aggregations on the data from joint table,
5. Taking notes of execution times of queries and analyzing data to compare the two systems.

3.1 Converting Database to .csv Files

There are many ways to convert data from a database into .csv files. The two most commonly used methods will be introduced. The first method is piping SQL SELECT output to a file. The other is using SQL Server Import and Export Wizard in SQL Server Management Studio.

3.1.1 Piping SQL SELECT Output to a File

MySQL allows users to store the results of a SELECT statement in a text file on a server. Using this mechanism, it is possible to create a .csv file. So, piping SQL SELECT statement output to a file could be one method for extracting data from SQL Server database. Below is a simple example.

In the example below, the data is being extracted from a table, called “Products”, and saved in a .csv file, called “products.csv” folder in a folder called “tmp” (/tmp/products.csv).

Example:

```
SELECT *
INTO OUTFILE '/tmp/products.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
ESCAPED BY '\\'
LINES TERMINATED BY '\n'
```

```
FROM products
```

For the method to work, the directory used above must be created by the MySQL database server and the file that will be used to store the data should not already exist.

3.1.2 Using SQL Server Import and Export Wizard in SQL Server Management Studio

SQL Server Management Studio is an integrated environment for manipulating all components of SQL Server. The Import and Export Wizard allows the user to extract data into Excel spreadsheet easily because it is a very interactive method.

Comparing the two methods mentioned earlier, it has been found that the SQL Server Import and Export Wizard is the most ideal and efficient way to extract the data from the database and convert it into .csv files. For this reason, this method will be implemented for this project and the detailed steps for this method will be explained in Section 4.1.1 for review.

3.2 Loading Files in to Caches

In this step, the cache-based environment is created. This allows the user to execute the aggregation in the cluster using Coherence.

To avoid constantly loading repeated data into cache, the structure of the data will be need to be modified. This way server space and time can be saved.

Because of the amount of data being loaded into cache, the program may crash during the loading process. So after loading all the data into cache, the size of cache needs to be verified to ensure that all the data was loaded into the new environment.

When running the program to load all the data into cache, the process will be done on the server using command prompt. This way up to eight processes can be run simultaneously, which will make the loading process take much less time.

3.3 Aggregating Data in Caches

3.3.1 Querying Data to Create Joint Table

A joint table will be created using information that is highly requested by the users from several different caches in the cluster. The users will later run aggregators and filters on this joint table. To do this, values of the highly requested variables will be extracted from the various caches, and then the data will be joined together based on a specified condition. The joint table will be put in a separate cache as an output.

3.3.2 Implementing Aggregators and Filters

To complete this step, queries will be requested from the current users of the cube so that the cache-based system can be set up to meet those needs. This will also create a basis for testing. The current users run MDX queries to get information from the cube. The MDX queries will be rewritten so that they can run in the cache-based environment and return the same results. To do this, aggregators will be needed to do calculations and filter will be needed to requests specific information.

After queries have been rewritten, they will be tested to ensure that they return the same results as the MDX queries. By doing this, the comparison between the two systems will be fair. The team will be recording the run times for each query and analyze the returns to figure out what factors change the execution speeds or cause the difference between the two systems.

Chapter 4: Implementation

4.1 Converting Database to .csv Files

4.1.1 Using SQL Server Import and Export Wizard in SQL Server Management Studio

Based on the workstation CPU allowance, a maximum of 8 extracting programs ran at the same time. Once a program ran completely, another one would be started immediately. It took roughly two and a half hours for extracting all the data we needed. The total output was172 files and size is was363 gigabytes. Below are the steps of the whole process. Some of the information was censored as a requirement of the company's confidentiality policy. Below are the steps taken while using wizard to extract the data.

Step 1:To launch the SQL Server Import and Export Wizard, connect to the database engine server , then right-click on 'Expand the databases', select 'Tasks', and then click on 'Export Data'.

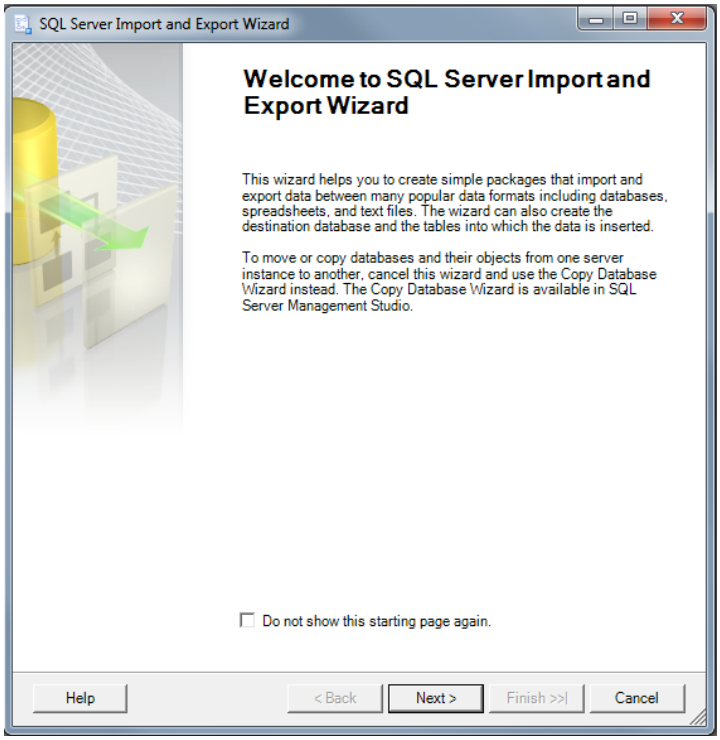


Figure 12: SQL Server Import and Export Wizard

Step 2: Choose the data source and the server name, and then click ‘Next’.

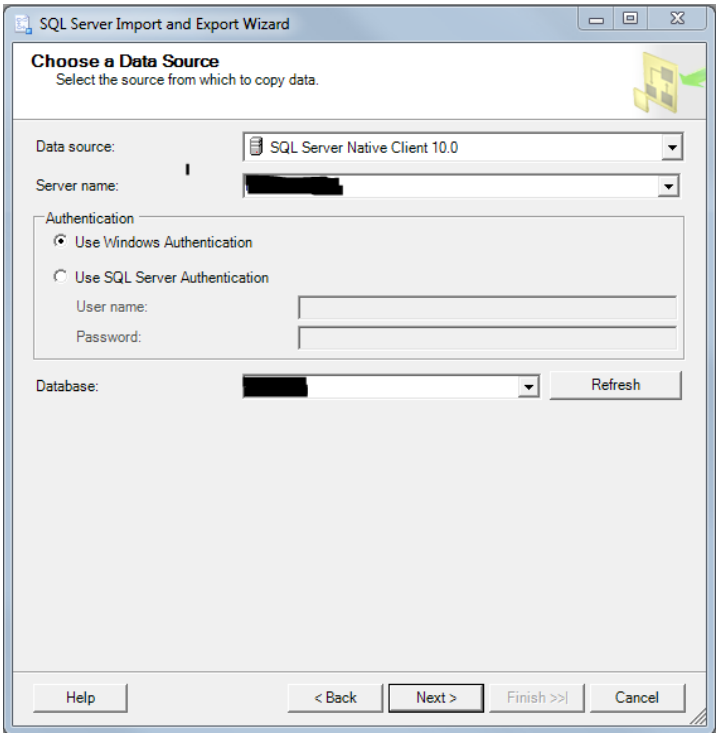


Figure 13: SQL Server Import and Export Wizard Step 2

Step 3: Select ‘Destination’ from the list as ‘Flat File Destination’, then browse the file name by creating a new file name which matches the data set. Also, check the box for ‘Column names in the first data row’. Click ‘Next’.

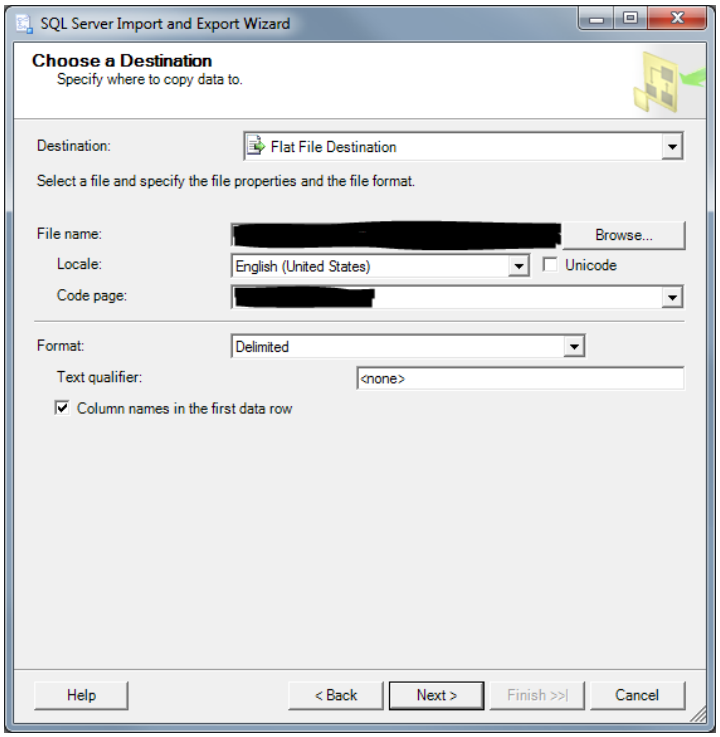


Figure 14: SQL Server Import and Export Wizard Step 3

Step 4: Choose ‘Copy data from one or more tables or views’, then click ‘Next’.

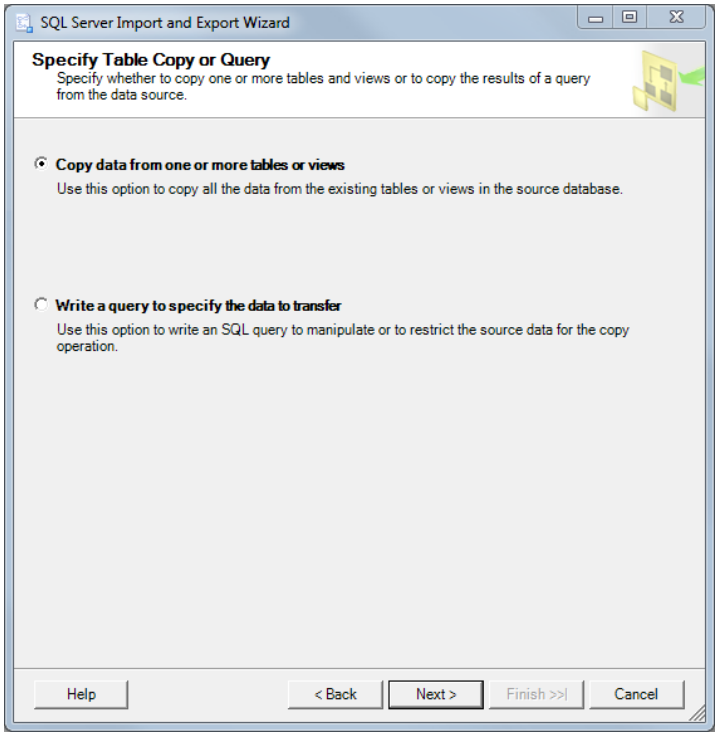


Figure 15: SQL Server Import and Export Wizard Step 4

Step 5: Select the exact data set name from the list of ‘Source table or view’, and then click ‘Next’.

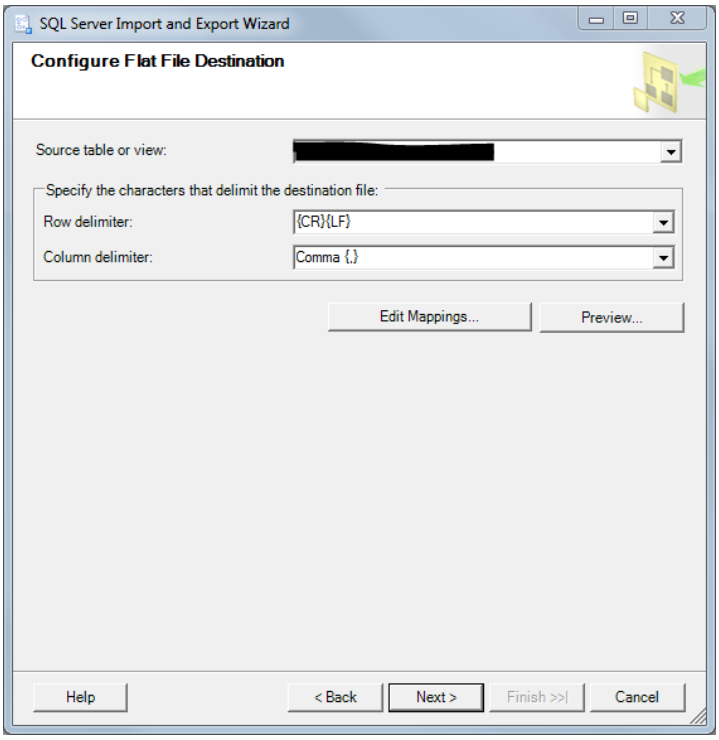


Figure 16: SQL Server Import and Export Wizard Step 5

Step 6: Check the box ‘Run immediately’, click ‘Next’.

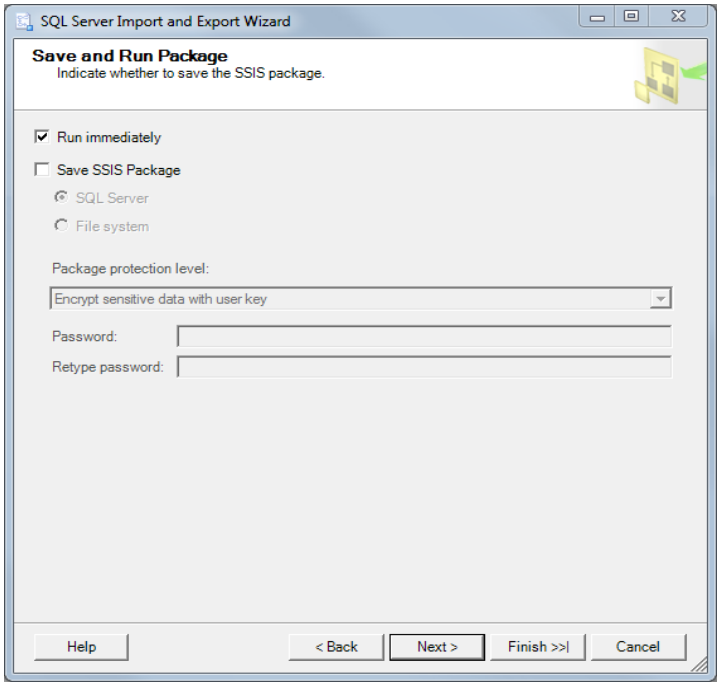


Figure 17: SQL Server Import and Export Wizard Step 6

Step 7: The screenshot below is an indication that the process is almost complete. Click ‘Finish’ and it will be run automatically.

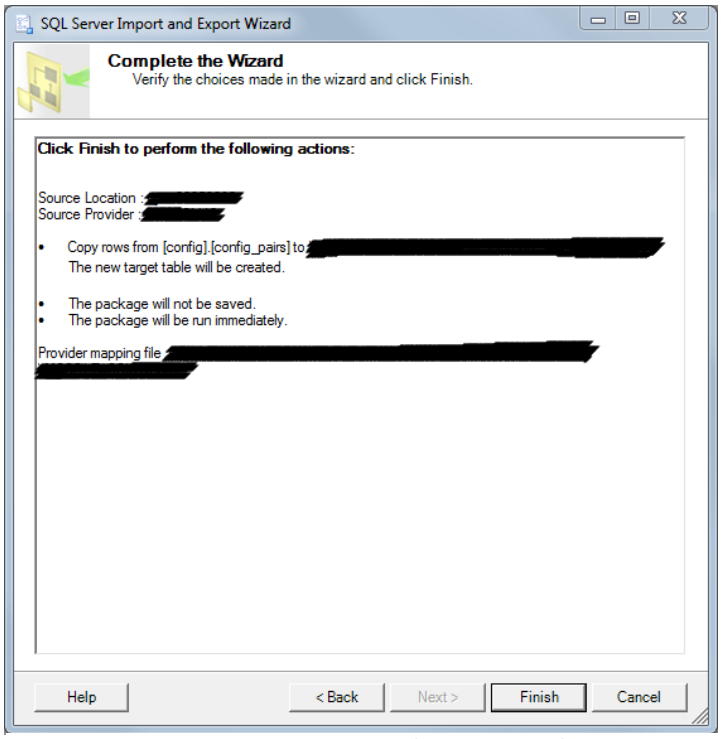


Figure 18: SQL Server Import and Export Wizard Step 7

Step 8: Below is a screenshot of the complete window that the wizard returns once the process is complete.

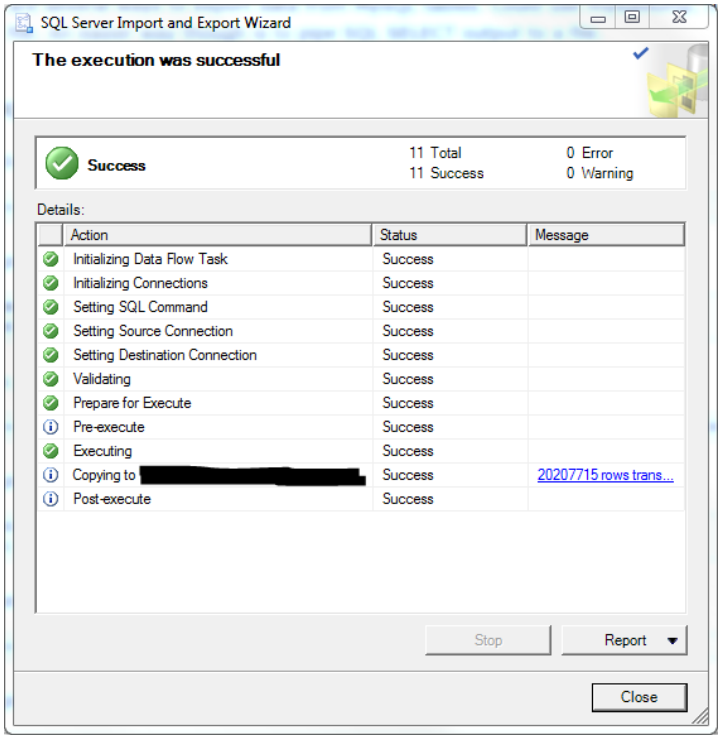


Figure 19: SQL Server Import and Export Wizard Step 8

The 8 steps above were repeated until the entire database was extracted.

4.2 Loading Files in to Caches

After extracting data from database to .csv file, next step was to load the data into cache. An example program was tested on a small sample database (countries.csv) to load data into cache. Sample code was then modified to fit the data from current database.

4.2.1 Explanation of Program Used to Load Data

A Java program was implemented to insert sample data shown in Figure 20 into cache. An object class made up of the variables was created to define and represent the data types. In this example, a *country* object class was created. *PortableObject* was implemented to speed up the reading and loading process.

countries.csv

	A	B	C	D	E
1	code	name	capital	currencySymbol	currencyName
2	AFG	Afghanistan	Kabul	AFN	Afghani
3	ALB	Albania	Tirana	ALL	Lek
4	DZA	Algeria	Algiers	DZD	Dinar
5	AND	Andorra	Andorra le Vella	EUR	Euro
6	AGO	Angola	Luanda	AOA	kwanza

Figure 20: Screenshot of Sample Data File

After defining the object class that represents the type of data, a program was developed which used the class to put data into a cache in the cluster.

Below is the framework to read .csv files (Figure 21). It read each line in the file, and then separated the values by commas. After reading, the program assigned each value in the row to their corresponding fields defined in the object class.

```
public ArrayList<Country> readCountries(){
    ArrayList<Country> CountryInCSV = new ArrayList<Country>();
    String csvFile = "countries.csv";
    BufferedReader br = null;
    String line = "";
    String cvsSplitBy = ",";
    int firstline = 1;

    try {
        br = new BufferedReader(new FileReader(csvFile));
        while ((line = br.readLine()) != null) {
            System.out.println("   Load Line");
            if (firstline == 1){
                firstline = 0;
                continue;}
            // use comma as separator
            String[] row = line.split(cvsSplitBy);
            String code = row[0];
            String name = row[1];
            String capital = row[2];
            String currencySymbol = row[3];
            String currencyName = row[4];
            Integer yearFounded = Integer.parseInt(row[5]);
            Country country = new Country(code, name, capital, currencySymbol, currencyName,yearFounded);
            CountryInCSV.add(country);
        }
```

Figure 21: Framework to Read Files

In the main method shown in Figure 22, after calling the function to read the .csv file, the cache was given a name in order to reference the cache using *NamedCache and CacheFactory.getCache* methods.

With the existing cache, a function is written to take country objects in the .csv file and store them in the cache as values with unique keys. In this example (Figure 22), country code is used as a unique key for each object.

```
public static void main(String[] args) {
    CoherenceHelloWorld hello = new CoherenceHelloWorld();
    ArrayList<Country> CountryInCSV = hello.readCountries();
    NamedCache countries = CacheFactory.getCache("countries");
    int index = CountryInCSV.size();
    for(int i = 0; i < index;i++){
        String countryCode = CountryInCSV.get(i).getCode();
        countries.put(countryCode, CountryInCSV.get(i));
    }
```

Figure 22: Putting Data into Cache

After putting data into a cache, all country objects in the cache was printed out to verify that everything had been put into the cache. In order to do this *null* was passed as the filter as shown in Figure 23. Later on, when specific entries needed to be retrieved, other filters would replace *null*.

```
Set<Map.Entry> entries = countries.entrySet(null, null);
for (Map.Entry entry : entries) {
    System.out.println(entry.getKey() + "=" + entry.getValue());
}
```

Figure 23: Code Returning Value from Cache

4.2.2 Modify example program

After testing on the sample .csv file, the program was modified to meet the needs of the BNP databases. In the following sections the major changes are discussed.

Change of data type

In the sample data file (countries.csv), there was only one data type to be represented in the *country* object class, which was a string, as shown in the figure earlier. In BNP databases, there were several data types in each object. Below are screenshots of the start-of-day trade file showing some of the fields to be considered when create the object class.

O	P
risk_id	scale
BTAZ14	64

Figure 24: Sample Fields of the Start-of-Day Trade File – Risk ID and Scale

Z
settled
FALSE

Figure 25: Sample Fields of the Start-of-Day Trade File - Settled

AD	AE
received_datetime	fact_trade__load_datetime
11:12.4	11:13.5

Figure 26: Sample Fields of the Start-of-Day Trade File – Date of Receipt and Load

Figures 24, 25 and 26 show several sample fields contained in the start-of-day trade file. The data type for *risk ID* is a string, while the data type for *Scale* is an integer. Likewise, the data type for *Settled* is a Boolean, and the data type for *Received Datetime* and *Fact Trade Load Datetime* is a raw date.

When creating the object class for the start-of-day trade file (FactTradeSod), these fields needed to be defined individually as shown in Figure 27 below. However, fields that related to date and time were not going to be aggregated, so they were simply defined as a string. The return type of *get* and *set* functions also needed to be kept the same as the type of the corresponding data.

```
public class FactTradeSod implements PortableObject, Comparable {

    private Integer trade_key;
    private Integer trade_id;
    private Integer date_key;
    private Integer security_key;
    private Integer currency_key;
    private Integer portfolio_key;
    private Integer trade_flags_key;
    private Integer pos_location_key;
    private Integer delivery_date_key;
    private Integer ref_rate_key;
    private Integer product_key;
    private Integer instrument_key;
    private Integer counterparty_key;
    private Integer maturity_date_key;
    private String risk_id;
    private float scale;
    private float notional;
    private float amount;
    private String front_office_id;
    private String settlement_system_id;
    private String universal_id;
    private Integer trade_version;
    private String contract_number;
    private Integer contract_version;
    private String trade_description;
    private Boolean settled;
    private String counter_rebook;
    private String cancel_rebook_link_id;
    private String trade_datetime;
    private String received_datetime;
    private String fact_trade__load_datetime;
```

Figure 27: Screenshot of Object Class for Start-of-Day Trade Cache

When implementing PortableObject, the *pofRead* and *pofWrite* methods were implemented. These methods were modified to allow reading and writing the various data types. Figure 28 and 29 shows an example of pofReader and pofWriter and how they were modified according to different data types. When setting up the *pofRead* and *pofWrite* functions, the variables were numbered in the same order they were in the constructor.

```
contract_version = pofReader.readInt(23);
trade_description = pofReader.readString(24);
settled = pofReader.readBoolean(25);
```

Figure 28: Example of PofReader

```
pofWriter.writeInt(23,contract_version);
pofWriter.writeString(24,trade_description);
pofWriter.writeBoolean(25,settled);
```

Figure 29: Example of PofWriter

Recall that the files were in the format of comma separate value files, therefore each row was split by a comma. These files were converted to a list with several string elements, which means all the data in each row were, by default, set to a string. When reading the start-of-day trade file, each element in the row needed to be parsed into corresponding data types if the original type of data was not a string. Figure 30 shows how to parse a string to other data types.

```
Integer contract_version = Integer.parseInt(row[23]);
String trade_description = row[24];
Boolean settled = Boolean.parseBoolean(row[25]);
```

Figure 30: Parse String in Each Row to Other Data Type

Each element in a row populated the object. The numbers in the brackets represented the locations of each element in the row, and the order of those elements remained the same as the object was constructed. To put the objects from the start-of-day trade file in a map, the *trade key* was used as the unique for each object as shown in Figure 31.

```
FactTradeSod sods = new FactTradeSod(trade_key, trade_id,
        date_key, security_key, currency_key, portfolio_key,
        trade_flags_key, pos_location_key, delivery_date_key,
        ref_rate_key, product_key, instrument_key,
        counterparty_key, maturity_date_key, risk_id, scale,
        notional, amount, front_office_id,
        settlement_system_id, universal_id, trade_version,
        contract_number, contract_version, trade_description,
        settled, counter_rebook, cancel_rebook_link_id,
        trade_datetime, received_datetime,
        fact_trade__load_datetime);
tradeSod.put(trade_key, sods);
```

Figure 31: Creating Object and Put It into Map

Change of data structure

The structure of the 150 risk files in the BNP database was very different from that of the start-of-day trade file. In risk files, many of the values for most of the fields remained the same. The differences between these rows were the *tenor key* and *delta value*.

risk_id	data_set_key	currency_key	tenor_key	delta_value
AAA	123	999	1	1.4
AAA	123	999	2	1.5
AAA	123	999	3	6.4
AAA	123	999	4	2.5
AAA	123	999	5	-3.5
AAA	123	999	6	6.7
AAA	123	999	7	5.7
AAA	456	999	1	1.4
AAA	456	999	2	1.5
AAA	456	999	3	6.4
AAA	456	999	4	2.5
AAA	456	999	5	-3.5
AAA	456	999	6	6.7
AAA	456	999	7	5.7

Figure 32: Sample of Risk File

Figure 32 is a simplified example of the structure of a risk file; original files and contents are not shown due to the company's confidentiality policy. As the figure shows, values under *risk ID*, *data set key* and

currency key were the same for the first seven risk entries, *tenor key* and *delta value* were different for each entry. Starting at the 8th entry, the *data set key* changed from 123 to 456, the *tenor key* and *delta value* repeated the previous values, while *risk ID* and *currency key* remained the same. If the data in risk files is to be inserted the same way as the start-of-day trade file, this would lead to a large amount of repeating data, which is not efficient in both time and space.

To avoid constantly loading repeating values, the data structure of the risk object was modified to keep all the variables except *tenor key* and *delta value*. To do this, a HashMap was used in the risk object to store *tenor key* and *delta value* as key-value pairs. After this modification, risk objects resembled the following two figures.

risk_id	data_set_key	currency_key	tenor_key	delta_value
AAA	123	999	1	1.4
			2	1.5
			3	6.4
			4	2.5
			5	-3.5
			6	6.7
			7	5.7

Figure 33: Condensed Risk Object 1

risk_id	data_set_key	currency_key	tenor_key	delta_value
AAA	456	999	1	1.4
			2	1.5
			3	6.4
			4	2.5
			5	-3.5
			6	6.7
			7	5.7

Figure 34: Condensed Risk Object 2

This way, the sample risk file was condensed to contain only 2 objects instead of 14 objects. The size of the cache which stores the data in these risk files was decreased. Figure 35 shows the variables contained in the condensed risk object class.

```
public class FactRisk implements PortableObject, Comparable{
    private String risk_id;
    private Integer data_set_key;
    private Integer currency_key;
    private Integer risk_notional_currency_key;
    private Integer main_batch_key;
    private Integer master_tenor_set_key;
    private Integer risk_method_type_key;
    private Integer curve_key;
    //private String tenor_key;
    //private String delta_value;
    private String fact_risk__available_datetime;
    private String fact_risk__received_datetime;
    private String fact_risk__load_datetime;
    private HashMap<Integer,Double> deltaTable = new HashMap<Integer,Double>();
```

Figure 35: Code for Condensed Risk Object Class

To create new a risk object, the program would check values of each variable except the *tenor key* and *delta value*. If they do not change, the program would keep adding the *tenor key* and *delta value* to the HashMap in the object. Once the value of any of the other variable changed, the program would stop adding the key-value pair to the HashMap and create a new risk object. The program would continue reading the remaining data in the file and repeat this procedure.

The following is an example of how the program would check for changes in the constant variables.

```
br = new BufferedReader(new FileReader(risks));
String risk_id = "";
String curr_risk_id = "";
Integer data_set_key = 0;
Integer curr_data_set_key = 0;
```

Figure 36: Initializing Variables

To track the changes, two variables were used; one to record the original value and the other to go through each row in the risk file. This is shown in Figure 36. There is *risk ID* and *current risk ID* for the field *risk ID*. In Figure 36, these variables were given an initial value because when reading the first line

of the file, there were no previous values for each field. For the field which is a string, it was initialized to be an empty string. For the field which is an integer, it was initialized to be 0.

The first row in the file is always skipped because they are headers for the file. After skipping first row, each row was split by commas. Values for each field were checked if they were an empty string or 0 depending on the data type. Then, the variables were assigned to each corresponding value in the row. Figure 37 shows the algorithm:

```
String[] row = line.split(cvsSplitBy);

if (risk_id == ""){
    risk_id = row[0];
}
curr_risk_id = row[0];


if (data_set_key == 0){
    data_set_key = Integer.parseInt(row[1]);
}
curr_data_set_key = Integer.parseInt(row[1]);
```

Figure 37: Assign Variables

When going through the first line, *risk ID* and *data set key* were initialized as an empty string and 0 respectively. So, the value in row[0] (first element in the row) was assigned to *risk ID*, and value in row[1] (second element in the row) was assigned to d*ata set key.* Outside the '*if*' statement, *current risk ID* and *current data set key* were assigned to first and second element in the row respectively. Since *current risk ID* and *current data set key* were outside the *'if'* statement, they would always store the first and second values in each row. This '*if*' statement did not apply to *tenor key* and *delta value* as they just needed to be assigned to the ninth and tenth element in the row (row [8] and row [9]) and put into the HashMap in the risk object. When processing the rest of the rows in the file, the *risk ID* and *data set key* were not an empty string or zero any more. They had been assigned the values in the previous row. If the *'if'* statements were skipped, *risk ID* and *data set key* would retain the values in the first row. As

mentioned earlier, the program checked if there has been any change in values of any constant variables and then created a new object. When looping to the end of the file, the last risk object would be created with the rest of the values and put it into the HashMap. Figure 38 shows the algorithm in detail.

```
if ((!(risk_id.equals(curr_risk_id))) || (!(data_set_key.equals(curr_data_set_key)))){
    FactRisk X = new FactRisk(risk_id,data_set_key,...);
    factrisk.put(new UUID(), X);
    risk_id = curr_risk_id;
    data_set_key = curr_data_set_key;
    HM = new HashMap<Integer, Double>();
}
HM.put(tenor_key, delta_value);
FactRisk Y = new FactRisk(curr_risk_id, curr_data_set_key,...);
factrisk.put(new UUID(), Y);
```

Figure 38: Check Changes

Load Risk Object

There were 150 risk files with a total of 3,043,265,584 risk objects before condensing. After condensing, it was still too large to put all the risk objects into a single Map then into the cache which stores risk objects. Because of this, the Java Virtual Machine (JVM) would crash during the loading process. To solve this problem, the Map would need to be split into many segments. An *'if'* statement was used to check if the size of the Map reached 1000 objects. Once the size of Map reached 1000, the objects would be put into cache and then the Map would be cleared to continue adding other risk objects. Figure 39 shows the detail of '*if*' statement.

```
FactRisk X = new FactRisk(risk_id, data_set_key
factrisk.put(new UUID(), X);
if (factrisk.size() == 1000) {
    factRisk.putAll(factrisk);
    factrisk.clear();
}
risk_id = curr_risk_id;
data_set_key = curr_data_set_key;
```

Figure 39: Check Size of risk Map

Verify size of cache

After loading all the data into cache, the size of cache needed to be verified to ensure all the information from the files were loaded. For risk files, the size of risk cache would not match the total number of rows in all the risk files since the risk objects were condensed. So, to verify the loading of data from the risk files the number was estimated. The start-of-day trade file was too large to open completely in either Excel or Notepad++, so the total number of lines could not be shown directly. In this case, a counter was added to the '*while*' loop, to count every time when the 'loop processed a new line. The value of counter was printed out to show the number of rows in the file. Figure 40 and Figure 41 show where the counter was added (countLine).

```
int firstline = 1;
int countLine = 0;

try {
    System.out.println("reading: " + sodfile);

    br = new BufferedReader(new FileReader(sodfile));
    while ((line = br.readLine()) != null) {
        if (firstline == 1) {
            firstline = 0;
            continue;
        }
        countLine++;
```

Figure 40: Add Counter Inside While Loop

```
FactTradeSod sods = new FactTradeSod(trade_key, trade_id, date_
tradeSod.put(trade_key, sods);
System.out.println("Number of Line: " + countLine);
```

Figure 41: Print Out Value of Counter

Run the program to load data in server

The loading process was done on the server using command prompt. The files which contained data were stored one folder. The class files of the code, to load data, were copied into a separate folder.

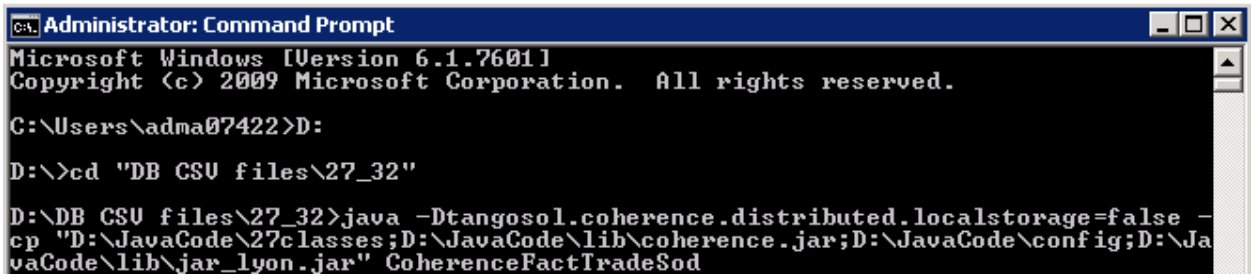


Figure 42: Command Prompt to Run Program

The command prompt was firstly directed to the drive where all the data and class files were saved (D drive). Then it was directed to the folder which contained all the data files. The following command line was used to run the program:

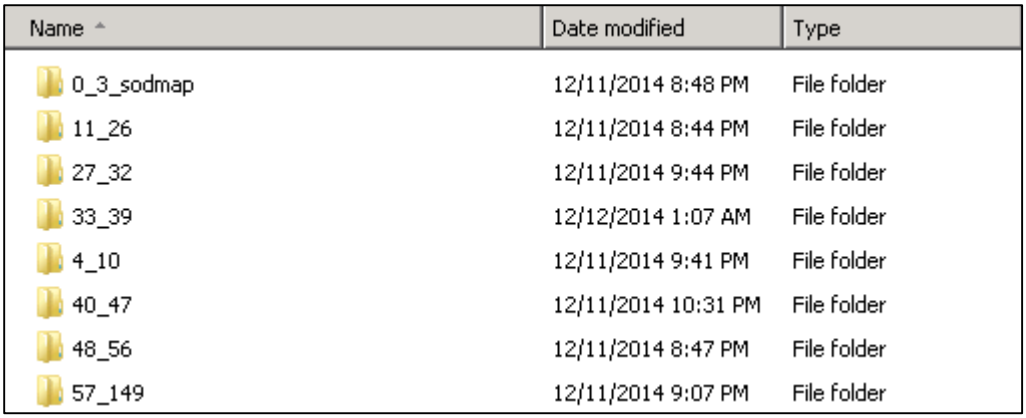
java -Dtangosol.coherence.distributed.localstorage=false -cp

*"D:\JavaCode**classes**;D:\JavaCode\lib\coherence.jar;D:\JavaCode\config;D:\JavaCode\lib\jar_lyon.jar"*

CoherenceFactTradeSod

The class name, in bold, was the name of folder where the corresponding class files are stored. The *'CoherenceFactTradeSod'* was the name given to the program to load the data.

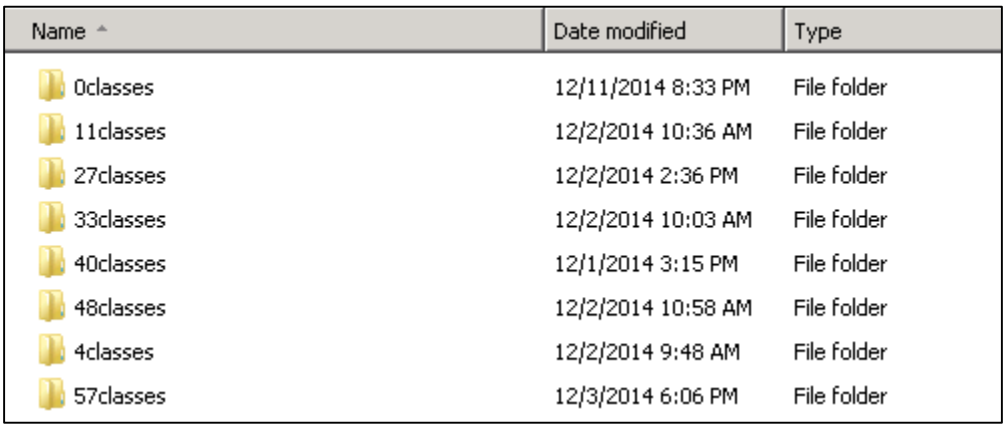
Reading and loading each folder of files, one by one, would take a lot of time. To speed up this process, files were split into eight folders to read and load, in parallel, as Figure 43 shows:



Name	Date modified	Type
0_3_sodmap	12/11/2014 8:48 PM	File folder
11_26	12/11/2014 8:44 PM	File folder
27_32	12/11/2014 9:44 PM	File folder
33_39	12/12/2014 1:07 AM	File folder
4_10	12/11/2014 9:41 PM	File folder
40_47	12/11/2014 10:31 PM	File folder
48_56	12/11/2014 8:47 PM	File folder
57_149	12/11/2014 9:07 PM	File folder

Figure 43: Separate Folders for Files

This way, multiple processes were run simultaneously. The program was modified to read and load different files in the eight folders, and .class files were then copied to each corresponding class folder as Figure 44 shown below:



Name	Date modified	Type
0classes	12/11/2014 8:33 PM	File folder
11classes	12/2/2014 10:36 AM	File folder
27classes	12/2/2014 2:36 PM	File folder
33classes	12/2/2014 10:03 AM	File folder
40classes	12/1/2014 3:15 PM	File folder
48classes	12/2/2014 10:58 AM	File folder
4classes	12/2/2014 9:48 AM	File folder
57classes	12/3/2014 6:06 PM	File folder

Figure 44: Separate Class Files Folders Corresponding to File Folders

The name of the class folder needed to be change each time when starting another process to load the files in other folders.

4.3 Aggregating Data in Caches

4.3.1 Querying Data to Create Joint Table

Not all the information loaded into cache was useful to the users. So, the highly requested information needed to be extracted to do further aggregations. To do this, a joint table was created with variables and data from several different tables in the caches.

Create JointRisk Object type

Like the other files, each row in the joint table is an object that needs a defined object class.

```
public class JointRisk implements PortableObject, Comparable {
    private Integer trade_key;
    private Integer portfolio_key;
    private Integer currency_key;
    private Integer data_set_key;
    private Integer risk_method_type_key;
    private HashMap<Integer,Double> deltaTable = new HashMap<Integer,Double>();
    private Integer curve_key;
    private String position_type;
    private Integer product_key;
    private Integer master_tenor_set_key;
    private Integer date_key;
    private String metier;
    private String sub_activity;
    private String activity;
```

Figure 45: JointRisk

Figure 45shows the definition of the joint risk object class. These variables came from different objects; namely risk objects, start-of-day trade objects and portfolio objects.

Retrieve value from cache

If everything in the cache was printed out directly, they would be all in *ConverterEntry* format where data could not be extracted from. To extract data, the first step was to retrieve value from cache. Take start-of-day trade cache as an example (Figure 46):

The first line invoked one of the QueryMap.entrySet methods and returned a set of Map.Entry instances representing the key and the value of a cache entry. This would allow user to filter and sort the entrySet. Here, all the entries were required to be retrieved so, like before, '*null*' was passed as a filter.

The print statement would print out the size of the Set, which was the same as size of the cache. Then each entry in this set was looped through and the getValue() method was used to extract only the value in the entry in the start-of-day trade object.

Then data could be retrieved from those values using the *'get'* methods defined in the object class. For example, the getPortfolio_key() method would return the portfolio key of a start-of-day trade object.

```
Set<Map.Entry> allTrades = factTradeSODcache.entrySet(null);
for (Map.Entry sTrade : allTrades)
{
    FactTradeSod sodObj = (FactTradeSod) sTrade.getValue();
```

Figure 46: Retrieve Value from Cache

Algorithm to join table and create JointRisk Object

Since data in the joint table came from different tables in the cache, there were certain criteria to select some risk and portfolio objects to join together. The algorithm looped through all the trades in start-of-day trade cache. For each of the trade objects, the algorithm retrieved the *risk ID* from them and found the corresponding risk objects that had the same *risk ID*. At the same time, the process described above was repeated to retrieve three following fields from the portfolio cache based on the object's *portfolio key*: *Métier*, *Subactivity* and *Activity*. The *EqualsFilter* was used to select those objects which met the specified condition. After selecting the risk objects, *data set key, risk method type key, master tenor set key, curve key* and *delta table* were retrieved for further use.

Next, the algorithm multiplied the *delta value* in the *delta table* by the *scale* from the selected trade objects. Lastly, each variable retrieved from the selected objects would be put together to create the joint table and put all joint risk objects into a separate cache. There existed some *risk IDs* in the trade objects that did not have a corresponding risk object. So, before retrieving *scale* and other required fields from trade objects, an '*if*' statement was used to check when the *risk ID* in a selected trade object is the same as *risk ID* in a selected risk object. Below is a diagram (Figure 47) shows the connection between the three different caches and the joint table.

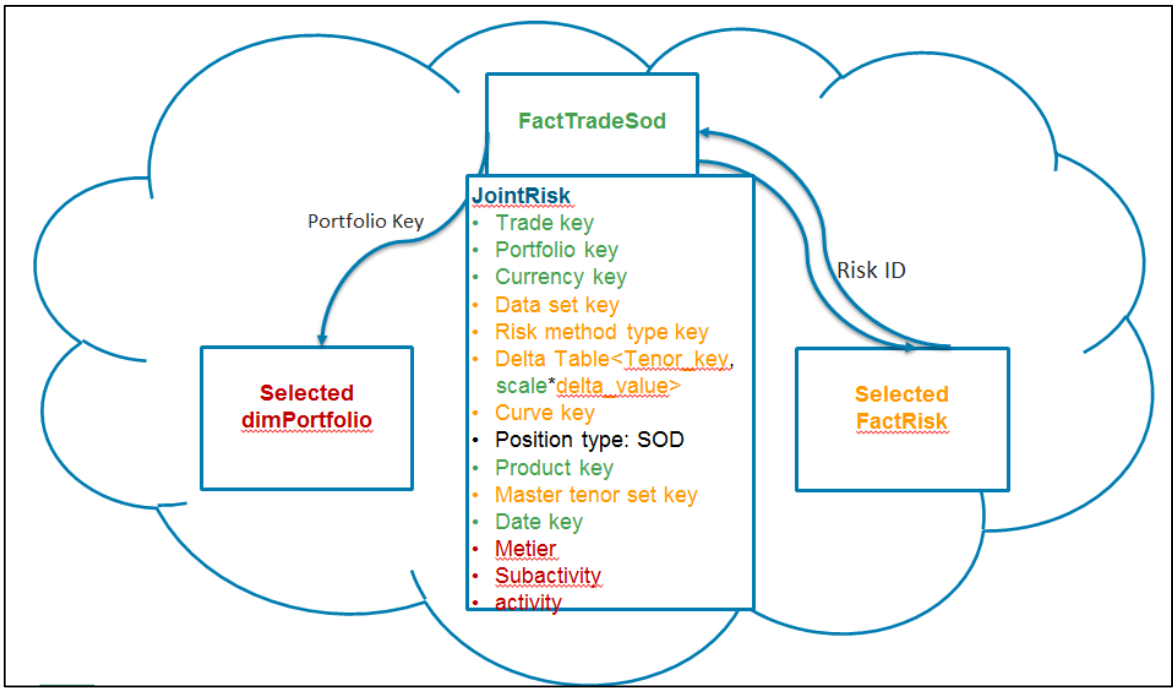


Figure 47: Connection of Different Caches

4.3.2 Implementing Aggregators and Filters

Three sample MDX queries from the current users of the OLAP system were received to being this portion of the project. A simple query that filtered on one currency value, one master tenor set key, a risk method type and 8 portfolios. There was also a complex query that filtered on 7 currency values, 7 master tenor set keys, a risk method type, an activity, a sub activity and a métier. The last query was in the within of this spectrum, much closer to the complex query, previously mentioned. This query filtered on 7 currency values, 7 master tenor set keys, a risk method type, and 8 portfolios. All of three queries returned a table with 9 columns. To get a wider range of queries, the queries were edited to return less information and/or filter on more or fewer specific conditions.

		Average Run Time
Simple	Orginal	0:02
	No Portfolios	0:07
	1 Portfolio	0:01
	10 Portfolios	0:03
	50 Portfolios	0:05
	100 Portfolios	0:06
	200 Portfolios	0:05
	300 Portfolios	0:07
	All Portfolios (336 Portfolios)	0:07
	Less Display (Original Conditions)	0:01

Table 2: Average Execution Time of Simple MDX Query (With Cache)

		Average Run Time
Suppo	Orginal	0:05
	No Portfolios	3:37
	1 Portfolio	0:02
	10 Portfolios	0:03
	50 Portfolios	0:08
	100 Portfolios	0:15
	500 Portfolios	0:40
	1000 Portfolios	2:54
	All Portfolios (1185 Portfolios)	3:13
	Less Display (Original Conditions)	0:01

Table 3: Average Execution Time of Suppo MDX Query (With Cache)

		Average Run Time
Complex	Original	2:43
	No Sub Activity	2:48
	No Activity	1:38
	No Metier	1:37
	5 Sub Activity	2:06
	All (8) Sub Activity	2:56
	All (6) Activity	1:43
	All (2) Metier	1:42
	5 Sub Activity and All (6) Activity	2:15
	All (8) Sub Activity and All (6) Activity	2:33
	Less Display (Original Conditions)	0:01

Table 4: Average Execution Time of Complex MDX Query (With Cache)

At the end of the edits, a total of 31 variations of the original queries were made, with execution times ranging from as little as one second to as much as 3 minutes and 37 seconds, as seen above. After running the query once, the execution time would decrease up to 80% and remain at this minimized time for all the subsequent runs. This was because after the first run, the cube would save the results in a cache so that the system can retrieve the information more efficiently. Two tests were done; where the 31 queries ran with cache and without cache. Without the cache the range of the execution time decreased to as little as one second to as much as 3 minutes and 13 seconds, as seen below.

		Average Run Time
Simple	Orginal	0:01
	No Portfolios	0:07
	1 Portfolio	0:01
	10 Portfolios	0:03
	50 Portfolios	0:05
	100 Portfolios	0:06
	200 Portfolios	0:05
	300 Portfolios	0:07
	All Portfolios (336 Portfolios)	0:07
	Less Display (Original Conditions)	0:01

Table 5: Average Execution Time of Simple MDX Query (Without Cache)

		Average Run Time
Suppo	Orginal	0:05
	No Portfolios	3:13
	1 Portfolio	0:01
	10 Portfolios	0:03
	50 Portfolios	0:08
	100 Portfolios	0:15
	500 Portfolios	0:40
	1000 Portfolios	2:54
	All Portfolios (1185 Portfolios)	3:13
	Less Display (Original Conditions)	0:01

Table 6: Average Execution Time of Suppo MDX Query (Without Cache)

		Average Run Time
Complex	Original	1:42
	No Sub Activity	2:52
	No Activity	1:41
	No Metier	1:41
	5 Sub Activity	2:06
	All (8) Sub Activity	2:56
	All (6) Activity	1:43
	All (2) Metier	1:42
	5 Sub Activity and All (6) Activity	2:07
	All (8) Sub Activity and All (6) Activity	2:34
	Less Display	0:02

Table 7: Average Execution Time of Complex MDX Query (Without Cache)

After fully understanding what the MDX queries were doing and how they worked, aggregators and filters were written for Coherence that returned the same results as the MDX queries. Initially, a very simple aggregator was written to return a double representing the total risk based on the specified filters. Another aggregator was written that returned a hash map that represented a total of all the delta values for every tenor key based on the specified filters. Soon, it was evident, that parameters would have to be passed to the aggregators to get the 9 columns that the MDX queries returned. So new objects had to be made that contained the variables that the MDX query would return in the result. These objects would later represent the keys of a hash map with values being hash maps that represented the total delta values for each tenor key. This made it possible to change how much information was being returned from Coherence just like the current users are doing with OLAP. In the end, 4 aggregators were made: one that returned a double, another which returned a hash map with just the tenor keys and delta values, another that would display the information the original MDX queries display and another that displayed less information. Filters were written to narrow the result down the same way the MDX queries narrowed them down based on currency, master tenor set key, risk method type, portfolio, métier, sub activity and activity. A comparison between the results being

return by the two systems validated the results returned in Coherence. Also, the results of each aggregator were compared to ensure they were related. For example, the total risk aggregator should return the total of the risk values that the tenor risk aggregator would return, given that the filters are the same.

Chapter 5: Conclusion

5.1 Limitations

Here are the main limitations encountered while working on this project.

5.1.1 Losing Data When Restarting Cluster

Over time, the joint table evolved to meet the needs of the current users of the OLAP cubes. Any time an object class needed to be modified the .jar file would also need to be modified. When the .jar files are modified, for the cluster to run with the modified files, the cluster would have to be restarted. During this process, some of the data was lost. The amount of data loaded into the cluster was approximately 400GB across 3 servers. Each server needed to shut down, one by one, when restarting the cluster. This would give the data a chance to rebalance before shutting down another server. It is suspected that data lost during the rebalancing process. However, more investigation is needed to figure out exactly why data was lost when restarting the cluster.

5.1.2 Time Taken to Load Data and Create Joint Table

Another limitation is the time it took to load all the data into the cluster and create the joint table. To load all the data into the cluster, it took approximately five hours. After spending five hours loading, it took at least an additional six hours to query data in the cluster and connect several tables to create the joint table. Restarting cluster took approximately 1 hour. The restarting, loading and querying processes were performed several times due to the following situations: automatic restart of computer, cache ran out of space, and frozen system. This took a lot of time.

5.1.3 Unable to Run Aggregators as an Extend Client

The aggregators were able to run as a member of the cluster but not as an extend client, which is how the users of the system would run their queries. A member is a node in the cluster with access to all the information from that has been stored. An extend client would run queries on the cluster to request

information from the nodes from the outside of the cluster. Every time the aggregators ran as an extend client, there was error message stating that the program no longer understands the operations that was hard coded into the cluster. This is an issue which needs further investigation.

5.1.4 Unreliability of Coherence System

The following figure is a screen shot of the activity in the risk cache. A program allows the users to see details for each cache created in Coherence. The yellow graph on the top shows the history and size of the cache from 18:59 at December 17th to 10:48 at December 18th.

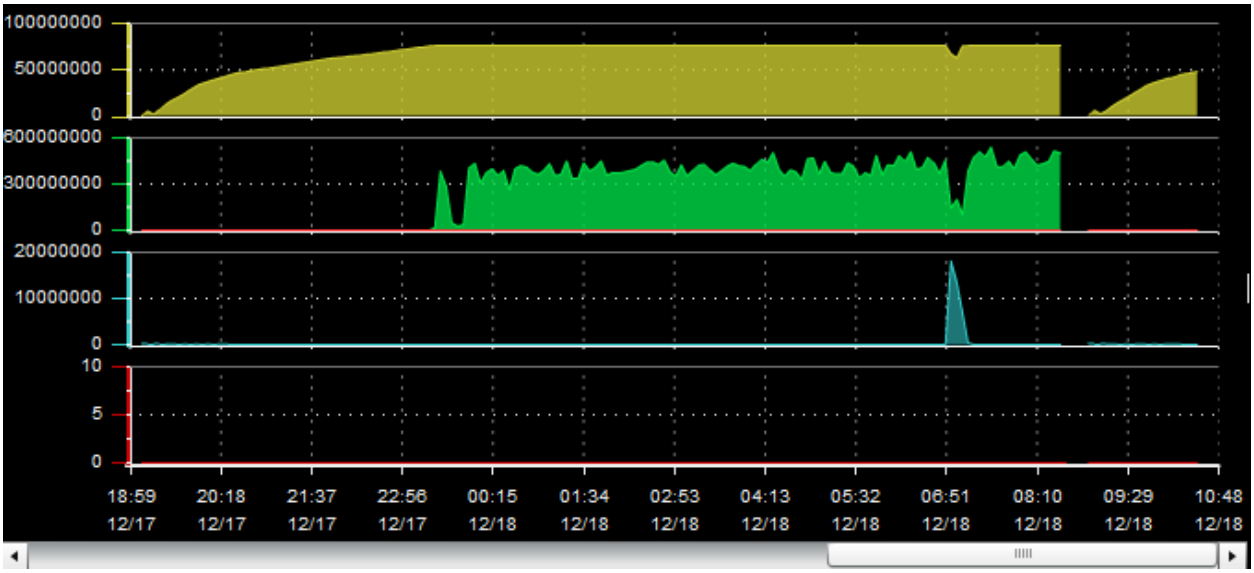


Figure 48: History of risk cache

During this time the loading of data was processed twice and the cache was cleared once between the two loading process. From this diagram two details were noticed:

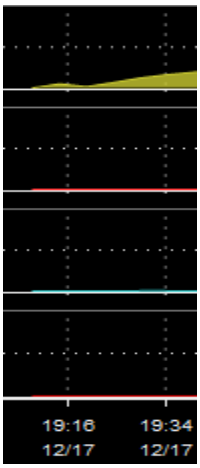


Figure 49: Lost and recover of data 1

Immediately after start loading the risk information into the cluster, there was a fluctuation where data was lost and then recovered (Figure 49, yellow diagram on the top). The amount of data lost each time when loading risk information was different. After completing the loading process, there was, again, a gap where data was lost and recovered (Figure 50, yellow diagram on the top).

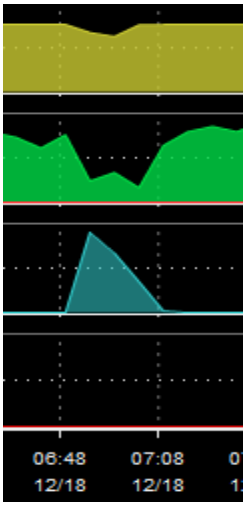


Figure 50: Lost and recover of data 2

This phenomenon shows that the Oracle Coherence system is unreliable. This may due to the huge amount of data stored. Further development and investigation is needed to fix this issue of Coherence system.

5.2 Extension

As an extension to the project, the system can be more generic to allow for various aggregations to be done based the users' needs. Aggregators can be written to collect an object that has all the fields the users are concerned about and decipher which ones will be displayed and which ones will be used for filtering. There will also need to be a way to get more information from the system without extending the joint risk object class. This would mean that there would need to be a way to link the joint risk object to other caches in the clusters so that information can be retrieved without storing everything in the joint risk cache, which may not be possible with the limited memory. Also, the current system runs in real time. This is a feature our implemented process does not have. This can be solved by implementing listeners for the aggregators. With the listeners being implemented, the programmer will need to figure out what will be done with the real time information, i.e. where will it be stored and how the user will know that something has changed and so on. To increase Coherence execution speed, indexes can also be added to the code so that over time, values that are frequently requested can be located much faster. This can be very useful with the more generic version of the current system by indexing values in other caches so that the time to retrieve more information can be reduced.

Bibliography

24 Processing Data In a Cache. (2008, January 1). Retrieved Novemeber 3, 2014, from Oracle: http://docs.oracle.com/middleware/1212/coherence/COHDG/api_processcache.htm#COHDG5201

Amazon. (2014, September 30). *Clusters*. Retrieved November 24, 2014, from Amazon ElatiCache: http://docs.aws.amazon.com/AmazonElastiCache/latest/UserGuide/CacheCluster.html

Batch Processing. (2014, October 24). Retrieved November 3, 2014, from Wikipedia: http://en.wikipedia.org/wiki/Batch_processing

BNP Paribas. (2014, May 5). *A Group with Global Reach*. Retrieved November 10, 2014, from Banque BNP Paribas: http://www.bnpparibas.com/en/about-us/corporate-culture/history

FINRA. (n.d.). *Managing Investment Risk*. Retrieved November 20, 2014, from FINRA: http://www.finra.org/Investors/SmartInvesting/AdvancedInvesting/ManagingInvestmentRisk/

Holton, G. A. (2014). *Delta and Gamma*. Retrieved November 3, 2014, from Risk Encyclopedia: http://riskencyclopedia.com/articles/delta_and_gamma/

Investopedia. (n.d.). *Liquidity Risk*. Retrieved November 20, 2014, from Investopedia: http://www.investopedia.com/terms/l/liquidityrisk.asp

Investopedia. (n.d.). *Risk Management*. Retrieved from Investopedia: http://www.investopedia.com/terms/r/riskmanagement.asp

IT Toolbox Popular Q&A Team. (2014, September 13). *Advantages and Disadvantages of MOLAP, ROLAP and HOLAP*. Retrieved November 13, 2014, from Toolbox: http://businessintelligence.ittoolbox.com/documents/advantagesdisadvantages-of-molap-rolap-and-holap-15897

Khan, I. (2012, May 8). *Real-time risk management: At decision time, every moment counts*. Retrieved November 3, 2014, from IT World: http://www.itworld.com/article/2726293/big-data/real-time-risk-management--at-decision-time--every-moment-counts.html

Lamb, K. (n.d.). *Measuring and Managing Investment Risk*. Retrieved November 20, 2014, from Investopedia: http://www.investopedia.com/articles/08/risk.asp

OLAP.com Team. (n.d.). *Types of OLAP Systems*. Retrieved from OLAP: http://olap.com/types-of-olap-systems/

Oltsik, J. (2010, September 1). *Real-Time Risk Management.* Retrieved Novemeber 5, 2014, from McAfee: http://www.mcafee.com/us/resources/solution-briefs/sb-esg-real-time-risk-management.pdf

Oracle. (2006, January 1). *Cache Clustering*. Retrieved November 6, 2014, from Oracle: http://docs.oracle.com/cd/B14099_19/caching.1012/b14046/cluster.htm

Oracle. (2010, July 1). *Loading Data Into a Cache*. Retrieved November 20, 2014, from Oracle Coherence: https://docs.oracle.com/cd/E15357_01/coh.360/e15831/loaddata.htm#BGBCFBFE

Oracle. (n.d.). *Using Portable Object Format*. Retrieved November 20, 2014, from Oracle Coherence: https://docs.oracle.com/cd/E24290_01/coh.371/e22837/api_pof.htm

Pandre, A. (2014). *OLAP Cubes*. Retrieved November 3, 2014, from Data Visualization: http://apandre.wordpress.com/data/datacube/

Paski, D., & Simic, R. (20123, October 30). *High Performance Risk Calculations with Coherence.* Retrieved November 3, 2014, from Oracle: http://coherence.oracle.com/download/attachments/5112972/Oracle+SIG+Sydney+Oct+2013+-+High+Performance+VaR.pdf

Rouse, M. (2012, April). *What is OLAP Cube?* Retrieved November 3, 2014, from WhatIS: http://searchdatamanagement.techtarget.com/definition/OLAP-cube

Saganich, A., Howes, J., Middleton, T., Arliss, N., & Team, O. (n.d.). *Using Coherence Portable Object Format Annotations*. Retrieved November 18, 2014, from Using Coherence Portable Object Format Annotations: http://www.oracle.com/technetwork/tutorials/tutorial-1836512.html

Seovic, A., & Falco, M. (2010). *Oracle Coherence 3.5 create internet-scale applications using Oracle's high-performance data grid.* Birmingham, U.K.: Packt Pub.

Sigito. (2013, May 23). *Aggregation and Processing of Data with Oracle Coherence*. Retrieved November 4, 2014, from BlogSpot: http://it-routings.blogspot.com/2013/05/aggregation-and-processing-of-data-with.html

Street, N. (2012, March 9). *System Diagram*. Retrieved November 10, 2014, from London Echonet.

Swap Pricing. (2014). Retrieved November 3, 2014, from FinancialCAD: http://www.fincad.com/derivatives-resources/wiki/swap-pricing.aspx

Tu, R. (2012, July 17). *Exporting SQL Server Data with Import and Export Wizard*. Retrieved November 3, 2014, from Winhost: http://blog.winhost.com/exporting-sql-server-data-with-import-and-export-wizard/

Vaselenko, S. (2011, January 1). *SQL Server Export to Excelusing bcp/sqlcmd Utilities and CSV Files*. Retrieved October 31, 2014, from http://www.excel-sql-server.com/sql-server-export-to-excel-using-bcp-sqlcmd-csv.htm

Williams, B. (2004, August 14). *Designing OLAP Cubes*. Retrieved November 12, 2014, from Database Answers: http://www.databaseanswers.org/designing_olap_cubes.htm

Yield Curve. (2014). Retrieved November 3, 2014, from Investopedia: http://www.investopedia.com/terms/y/yieldcurve.asp