Transparently Offloading Data Warehouse Data to Hadoop using Data Virtualization

A Technical Whitepaper

Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

November 2014

Sponsored by

Table of Contents

1 Introduction

This whitepaper focuses on introducing the popular Hadoop data storage technology in an existing data warehouse environment with the intention to use it as a platform for data offloading. The primary reasons to offload data from current SQL database servers to Hadoop is to reduce storage costs and to speed up reports. The result is a data warehouse environment in which data is distributed across multiple data storage technologies.

The *Cisco Information Server* (CIS) data virtualization server is used to hide this *hybrid data storage system*. It allows organizations to migrate transparently from their single data storage solution to a hybrid storage system. Users and reports won't notice this offloading of data.

The whitepaper describes, step by step, how to introduce Hadoop in an existing data warehouse environment. Guidelines, do's and don'ts, and best practices are included.

As a complementary offering, Cisco also provides a packaged solution called *Cisco Big Data Warehouse Expansion*, which includes software, hardware, and services required to accelerate all the activities involved in offloading data from a data warehouse to Hadoop.

2 The Ever-Growing Data Warehouse

More, More, and More – Most data warehouse environments use the *once-in-never-out* principle to store data. In most environments there is no intention to move data from the data warehouse to an archive or to remove it entirely. When new invoice, customer, or sales data is added, old data is not removed to make room. Some data warehouses contain data that is more than twenty years old and that's barely ever used.

But it's not only new data coming from existing data sources that enlarges a data warehouse. New data sources are continuously introduced as well. An organization develops a new Internet-based transaction system, acquires a new marketing system, or installs a new CRM system; all the data produced by these new systems must be copied to the data warehouse enlarging it even further.

Besides new internal data sources, it has become very common to add data from external data sources, such as data from social media networks, open data sources, and public web services. Especially for specific forms of analytics, enriching internal data with external data can be very insightful. Again, all this external data is stored in the data warehouse environment.

All this new data and all these new data sources lead to data warehouse environments that keep growing.

The Drawbacks of an Ever-Growing Data Warehouse – In principle, data growth is a good thing. When more data is available for analysts and business users, their reporting and analytical capabilities increase. More data potentially increases the reporting and analytical capabilities of an organization.

However, there are some practical drawbacks:

- **Expensive data storage:** Storing data in databases costs money. Examples of data storage costs are storage hardware costs, management costs, and license fees for those database server especially when the fee is dependent on the database size.
- **Poor query performance:** The bigger the tables in the data warehouse are, the slower the reporting queries will be.
- **Poor loading performance:** Loading new data may slow down when tables become bigger. Especially the indexes on the tables that must be updated when the data is loaded, can have a negative effect on the loading speed.
- **Slow backup/recovery:** The larger a database is, the longer the backup process and an eventual restore of all the data takes.
- **Expensive database administration:** The larger a database is, the more time-consuming database administration will be. More and more time must be spent on tuning and optimizing the database server, the tables, the buffer, and so on.

The Solutions – There are several solutions to shrink a data warehouse environment:

- **Removing Data:** Periodically, delete some of the least-used or the less-recently used data to keep the data warehouse environment as small as possible. The drawback of this approach is that analysis of the deleted data is no longer possible, limiting, in particular, historical analysis.

- **Moving Data to an Offline Archive:** Periodically, move unused or little used data to an *offline* data storage system in which the data is *not* available online. This solution keeps the data warehouse environment small. The challenge is to "reanimate" offline data quickly and easily for incidental analysis. A crucial question to answer is whether the offline data should be reloaded in the data warehouse, or should it temporarily be copied to a separate database? In principle, this doesn't reduce the amount of data stored, it's just that a portion is stored outside the data warehouse.

- **Offloading Data to an Inexpensive Online Data Storage System:** Move a portion of the data to an online data storage system with another set of technical characteristics. Obviously, such a data storage system must support online queries, and there should be no need to "reanimate" the data before it can be used for analysis. Storing data should also be less expensive. The result is a hybrid data warehouse environment in which all the stored data is distributed over different data storage technologies.

For most organizations, the third solution is preferred.

Solving the Data Warehouse Growing Pains with Hadoop – One of the newest data storage technologies is *Hadoop*. Hadoop has been designed to handle massive amounts of stored data, it has been optimized to process complex queries efficiently, it supports a wide range of application areas, and it has a low price/performance ratio. Especially the latter characteristic makes Hadoop an attractive data storage platform to operate side by side with the familiar SQL database technology. Because the financial and technical characteristics of Hadoop are very different from those of SQL database technologies, it allows designers to choose the best fit for each data set. For example, if query speed is crucial, SQL may be selected, and when storage costs must be reduced, Hadoop can be chosen. In other words, designers can go for the best of both worlds.

Hiding the Hybrid Data Storage System with Data Virtualization – SQL database technology is used in almost all the data warehouses to store data. When data is offloaded to Hadoop, the data warehouse environment deploys both data storage systems: Hadoop and SQL database technology. The consequence is that reports and users have to use different APIs and languages depending on where data is stored. Such APIs must be studied in detail. Especially the traditional Hadoop APIs, such as HDFS, HBase, and MapReduce, are very technical and complex. Also, in such a new situation, reports and users must know in which storage system the data resides that they want to analyze. All this will raise the costs of report development and maintenance. Finally, many reporting and analytical tools do not support access to Hadoop. This means that users have to learn how to work with new reporting tools and that existing reports must be redeveloped.

A data virtualization server decouples the two data storage systems from the reports, presenting one integrated data storage environment. In fact, with data virtualization the reports and users won't notice that data has been offloaded to another system. Users don't have to learn new APIs or languages and they don't have to know in which system the data resides. Data virtualization fully hides the *hybrid data storage system*. More on this in Section 4.

3 Overview of Hadoop

Introduction to Hadoop – *Apache Hadoop* has been designed to store, process, and analyze large amounts of data from terabytes to petabytes and beyond, and to process data in parallel on a hardware platform consisting of inexpensive commodity computers. It consists of a set of software modules from which the developers can pick and choose. Figure 1 illustrates the Hadoop modules on which this whitepaper focuses.

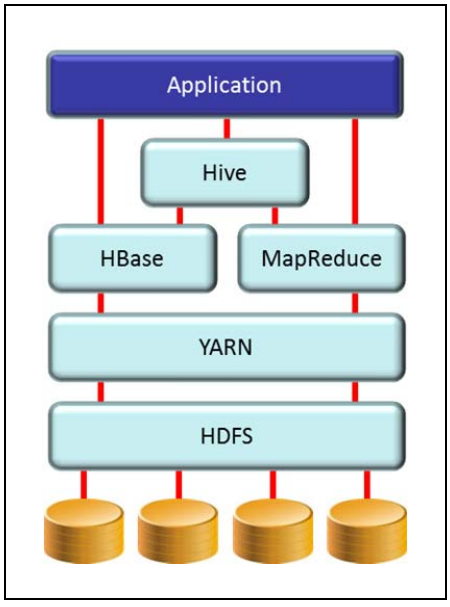

Figure 1 *Hadoop consists of a number of modules including HDFS, YARN, MapReduce, HBase, and Hive (the light blue boxes).*

The core modules are briefly introduced here. For more extensive descriptions, we refer to Tom White's book[1] on Hadoop.

- **HDFS:** The *Hadoop Distributed File System* (HDFS) forms the foundation of Hadoop. This module is responsible for storing and retrieving data. It's designed and optimized to deal with large amounts of incoming data per second and to manage enormous amounts of data up to petabytes.

- **YARN:** *YARN* (Yet Another Resource Negotiator) is a *resource manager* responsible for processing all requests to HDFS correctly and for distributing resource usage correctly. Like all other resource managers, its task is to assure that the overall performance is stable and predictable.

- **MapReduce:** *MapReduce* offers a programming interface for developers to write applications that query data stored in HDFS. MapReduce can efficiently distribute query processing over hundreds of nodes. It pushes any form of processing to the data itself, and thus parallelizes the execution and minimizes data transport within the system. MapReduce has a batch-oriented style of query processing.

- **HBase:** The *HBase* module is designed for applications that need random, real-time, read/write access to data. HBase has an API consisting of operations such as insert record, get record, and update record. HBase is usually categorized as a *NoSQL system*[2].

- **Hive:** The *Hive* module is a so-called *SQL-on-Hadoop engine* and offers a SQL interface on data stored in HDFS. It uses MapReduce or HBase to access the data. In case of the former, Hive translates each SQL statement to a MapReduce job that executes the request. Hive was the first SQL-on-Hadoop engine. Nowadays, alternative products are available, including Apache Drill, Cloudera Impala, and Spark SQL.

The Strengths of Hadoop

- **High data storage scalability:** HDFS has been designed and optimized to handle extremely large files. In real life projects, Hadoop has repeatedly proven that it's able to store, process, analyze, and manage big data.

- **High data processing scalability:** Hadoop has been designed specifically to operate in highly distributed environments in which it can exploit large numbers of nodes and drives. For example, one hundred drives working at the same time can read one terabyte of data in two minutes. In addition, MapReduce processing is moved to the nodes where the data is located. There is almost no centralized component that could become a bottleneck and lead to performance degradation.

- **High performance:** Together with HDFS, MapReduce offers high performance reporting and analytics. One of the features of HDFS is data replication, which makes concurrent access to the same data (on different nodes) possible.

[1] White, T., *Hadoop, The Definitive Guide*, O'Reilly Media, 2012, third edition.
[2] Redmond, E. and Wilson, J.R., *Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement*, Pragmatic Bookshelf, 2012.

- **Low price/performance ratio:** Hadoop has been designed to exploit inexpensive commodity hardware. The license fees of commercial Hadoop vendors are not based on the amount of data stored.

- **All data types:** HDFS is a file system that can store any type of data, including weblogs, emails, or records with sensor data. In addition, functions can be developed in MapReduce that have the same complexity found in SQL statements and beyond. MapReduce is not limited to structured data. MapReduce allows applications to access any form of data. For example, complex functions can be developed to analyze text or complex weblog records. If programmed correctly, MapReduce is able to process these complex functions completely in parallel, thus distributing this complex and I/O and resource intensive processing over a large set of nodes.

- **Fast loading:** HDFS has been designed to load massive amounts of data in real-time or in batch.

Summary – Especially the low price/performance ratio and the high scalability features make Hadoop an ideal platform for storing offloaded many types of data currently stored in the data warehouse environment.

4 The Data Warehouse Environment and Hadoop

The Databases of a Data Warehouse Environment – Traditionally, a data warehouse environment consists of several databases, such as a staging area, a central data warehouse, and many data marts; see Figure 2. In most environments, all these databases are implemented using SQL database server products, such as Oracle, Teradata, IBM PureData System for Analytics (formerly Netezza), or Microsoft SQL Server.

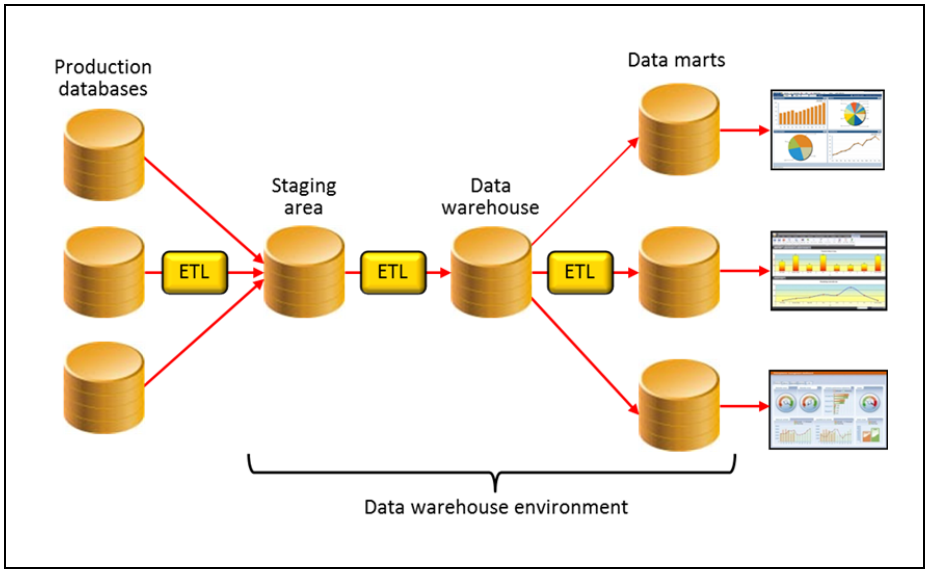

Figure 2 *Traditionally a data warehouse environment consists of several databases.*

Note that most data is stored redundantly in a data warehouse environment. For example, data coming from the source systems is first stored in a staging area, then copied to a data warehouse, and from there copied to one or more data marts. In the latter case, the data is usually stored in a slightly aggregated form, but it's still redundant data.

From SQL to Hadoop – As indicated in Section 3, the low price/performance ratio and the high scalability features make Hadoop an ideal platform for storing data warehouse data. Several application areas exist to deploy Hadoop in a data warehouse environment. For example, Hadoop can be used to develop a sand box for data scientists, or it can be used to store massive amounts of textual data. And, it can be deployed to lessen the growing pains of a data warehouse environment. By moving some of the data from the SQL databases to Hadoop, organizations can deal with the data growth more easily.

In general, there are two ways to migrate data to Hadoop: *entire tables* or *partial tables*.

Entire tables: Offloading entire tables to Hadoop implies that some of the tables in a SQL database (for example, the data warehouse itself) are moved completely to Hadoop; see Figure 3. After migrating the tables, the SQL database contains less tables. Reasons to offload an entire table are, for example, that the table is not used frequently, or the data is massive and thus too expensive to store in the SQL database.

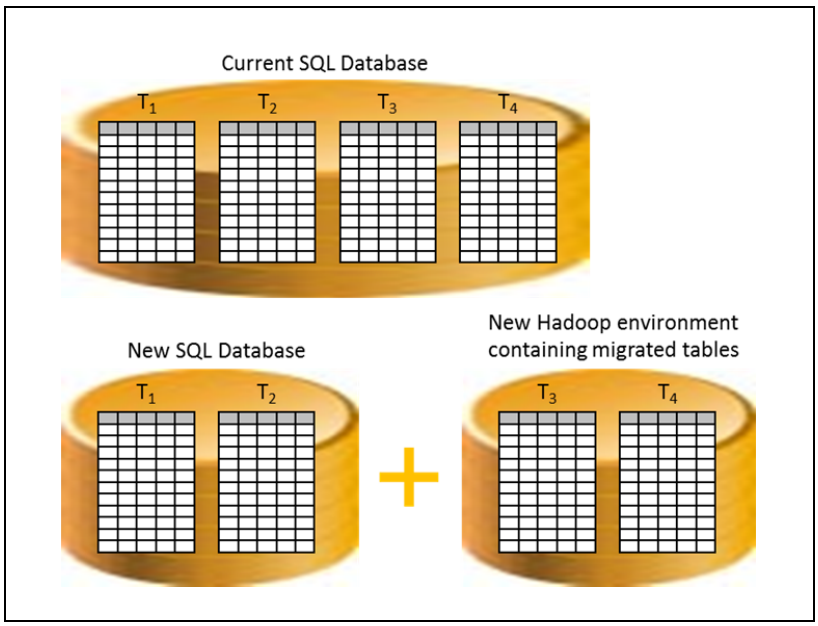

Figure 3 *Entire tables are moved from a SQL database to Hadoop.*

Partial tables: Offloading partial tables means that only a part of a table is offloaded. For example, for a number of tables in a SQL database a subset of the records is moved to a table in Hadoop; see Figure 4. Afterwards, the SQL database still contains all the tables it owned before, it's just that the tables don't contain all the records anymore. The rest of the records is stored in Hadoop. Reasons for offloading a partial table are, for example, that little-used or unused records in a table can be identified, or that the table is just massive.

In Figure 4, an alternative to offloading a subset of the rows is offloading a subset of the columns. This can be useful when some columns are barely ever used, or when they contain very large values, such as images and videos. In this case, moving those columns to Hadoop may be more cost-effective.

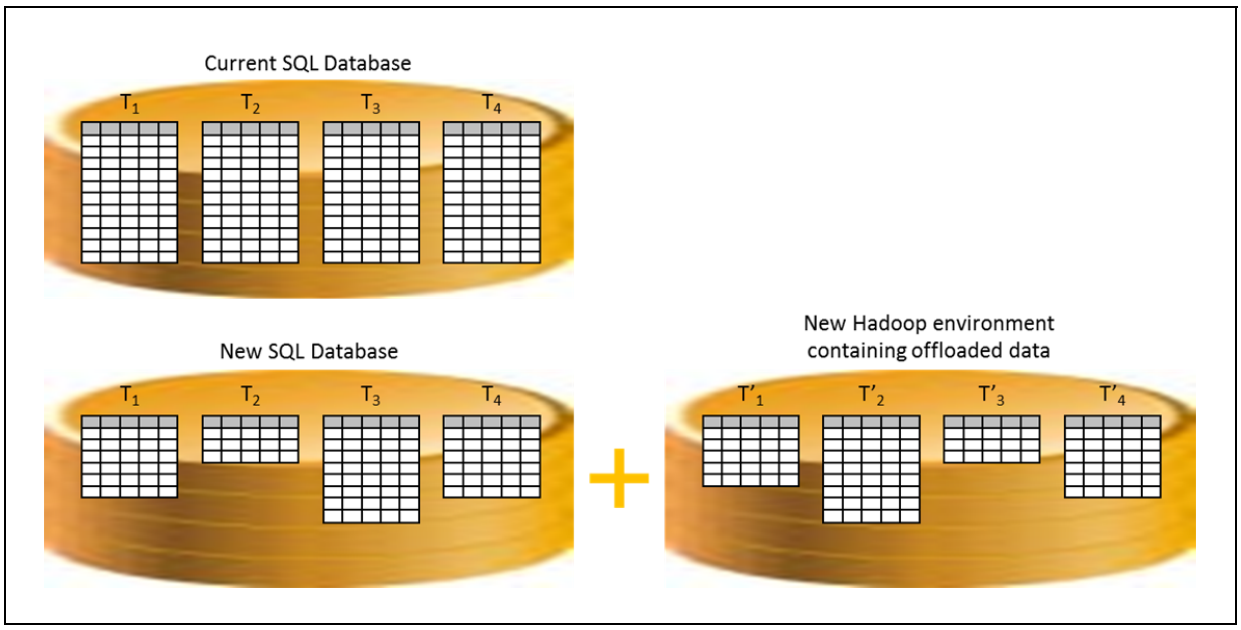

Figure 4 *Offloading partial tables means that a subset of the records is moved from a SQL table to a Hadoop table.*

A Hybrid Data Storage System – Offloading data to Hadoop results in a *hybrid* data storage system where some data is stored in SQL databases and some in Hadoop. Depending on the data and reporting characteristics, data resides in one of the two storage systems. This means, for example, when data is offloaded from a SQL database (that holds the central data warehouse) to Hadoop, that the data warehouse is no longer one physical database anymore.

The consequence of such a *hybrid data storage system* is that applications must know in which data storage system the data resides that they need, they must know which records or columns of a table reside in which storage system, they must know when data moves from one data storage system to another, and they must understand the different APIs and languages supported by the storage systems. And as already indicated, Hadoop APIs, such as HDFS, HBase, and MapReduce, are very technical and complex. Skills to work with these APIs are not commonly found in BI departments.

Data Virtualization and the Hybrid Data Storage System – All these technical aspects, such as handling different APIs, being location and SQL dialect aware, slow down report development and complicate maintenance. In particular for self-service BI users, this may all be too complex, possibly leading to incorrect report results.

Also important to note is that popular reporting and analytical tools typically support SQL but not Hadoop interfaces, such as HDFS, HBase, and MapReduce. This makes it difficult and maybe even impossible to use familiar tools on the data stored in Hadoop. This leads to redevelopment of existing reports using new tools.

These problems can be solved by placing CIS between, on the one hand, all the applications and reporting tools, and, on the other hand, all the SQL databases and Hadoop; see Figure 5. This way, reports and users still see one integrated database. CIS completely hides the hybrid data storage system. With data virtualization, the location of the data, APIs, and dialect differences are fully hidden. Users and reports don't have to know where or how tables are stored, they don't have to know where records are stored,

and users can continue using their favorite reporting tools and reports don't need to be redeveloped. In addition, CIS makes a transparent migration to a hybrid storage environment possible. The users won't notice the offloading of data to Hadoop.

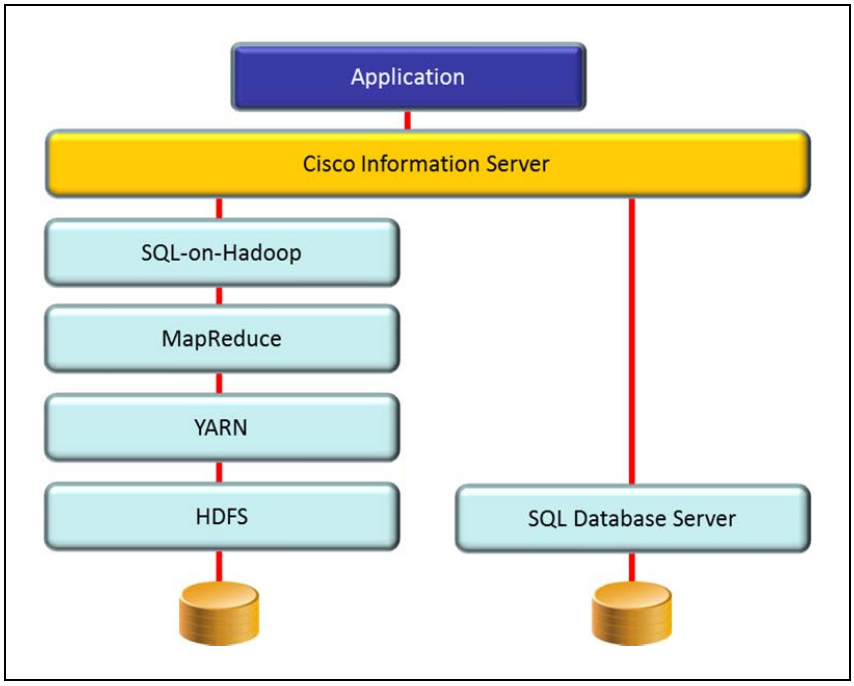

Figure 5 *By hiding all the technical details, CIS turns the hybrid data storage system into one integrated database.*

5 Examples of Offloadable Data

Infrequently-used data is the first type of data that comes to mind for offloading to Hadoop. But many more types of data exist that are well suited for offloading. This section contains examples of types of data that organizations may consider for offloading to Hadoop.

Massive Fact Data – Some tables can be just massive in size. For example, tables in a data warehouse containing call-detail records or sensor data may contain billions and billions of records. Purely because of the sheer size of those tables and possibly also the ingestion rate of new data, it's recommended to move these tables to Hadoop. These massive tables are almost always fact tables.

Semi-Structured Data – Nowadays, a lot of new data comes in a *semi-structured* form, for example, in the form of XML or JSON documents. Technically, it's possible to store such documents in SQL tables, but access is not always fast, and transforming it to flat table structures may be time-consuming during loading. Hadoop allows data to be stored in its original form. In fact, it supports file formats specifically developed for these forms of data.

Textual Data – As with semi-structured data, *textual data*, such as emails, agreements, and social media messages, can be stored in SQL tables. However, it may be more cost-effective to store this type of data in Hadoop.

Audio/Video Data – In many BI environments, *audio* and *video data* is not analyzed at all. Most reporting and analytical tools can't exploit them at all. For example, current BI tools do not have the features to detect cancer in MRI scans, to count the germs in a swab, to do optical character recognition, or to identify microstructural flaws in steel. Plus, storing this type of data in a SQL database is quite expensive. Dedicated tools to analyze this data do exist, and most of them work quite well with Hadoop.

Cold Data – Many tables contain records or columns that are, for whatever reason, rarely ever used. *Cold data* can be defined as data that was entered a long time ago and that has been used infrequently for a considerable amount of time. In a way, cold data is like the junk you have lying around in your garage and that you haven't used in ages, but is still there and is filling up your garage so that your car and barbecue are outside in the rain corroding away.

Obsolete Data – Tables may also contain data that has become irrelevant for most users of the organization: *obsolete data*. This can be, for example, descriptive data on obsolete sales products, or sales data related to retail shops that have been closed down. Obsolete data cannot be thrown away. It may still be necessary for compliancy reasons. Because of its very low data usage, it makes sense to offload obsolete data to Hadoop.

6 Steps for Offloading Data Warehouse Data to Hadoop

This section describes step by step how to offload data from one or more of the SQL databases in a data warehouse environment, such as the data warehouse or data mart, and how to move that data to Hadoop. These are the steps:

1. Identifying offloadable data
2. Installing Hadoop
3. Importing tables in CIS
4. Migrating reports to CIS
5. Creating tables in Hadoop
6. Extending views to include the Hadoop tables
7. Collecting statistical data on offloadable data
8. Initial offloading of data to Hadoop
9. Refresh of the offloaded data
10. Testing the report results
11. Adapting the backup process

We assume that all the databases making up the data warehouse environment are developed with a SQL database server, such as Oracle, Teradata, IBM PureData System for Analytics (formerly Netezza), or Microsoft SQL Server.

Step 1: Identifying Offloadable Data

It all starts with identifying offloadable data. This sounds simple, but it isn't, because the identifying process is not an exact science. For example, when is data cold or warm? There is always a set of records

that is lukewarm. Or, how much textual data financially justifies its offloading to Hadoop? When is data really obsolete? To assist, guidelines are given in this section to identify offloadable data.

Massive Fact Data – The very large tables in a data warehouse or data mart are almost always fact tables. Evaluate if some of them can be offloaded entirely. For this, study their query and load workload. Here are some reasons to offload a massive fact table:

- When the SQL database server is starting to have performance problems with loading new data in the fact table. Hadoop's load speed may be faster.
- When overall the queries on the fact table are showing performance problems due primarily to the number of records.
- When it's becoming too expensive to store the entire fact table in the SQL database server.

When the query workload can be characterized as interactive reporting and analysis, be careful with offloading to Hadoop, because Hadoop's support of this type of analysis leaves much to be desired. If most of the reporting is much more traditional, then consider offloading data.

When it's not an option to offload the entire table, offload a subset of the cold records; see the next topic.

Cold Data – Section 5 contains a definition of cold data: data that was entered a long time ago and that has been used infrequently for a considerable amount of time. Unfortunately, all the concepts used in the definition of cold data are vague. What is a long time ago? What is used infrequently? What is a considerable amount of time? To answer such questions, the *usage* of data usage must be analyzed in detail. A *data usage analysis* must show how often tables are queried, how often individual records and columns are queried, how often data is inserted, and how many users use certain records or columns. The more questions of this kind are answered, the more complete the data usage analysis will be, and the more easy it is to determine the temperature of the data.

Data usage can be determined with custom-developed tools or with dedicated monitors. Most SQL database servers support monitors that show data usage. Unfortunately, most of them do not show data usage on the level of detail required for analyzing data usage. Most of these tools only show usage per table or per column. In an ideal situation data usage analysis shows query usage per day; see Figure 6 as an example. In this diagram, the curved, black line indicates how infrequently older records are still being used and how frequently the newer ones. The alternative purple curve indicates another form of data usage where even older records are still being used frequently. In this case, offloading records probably doesn't make sense.

Based on the data usage results, define the *crossover point* (the dotted line in Figure 6). The crossover point is the age that divides the cold data from the warm data. In most cases it's easy to identify the really cold and the really warm records of a table, but there is always this group of records that is neither warm nor cold. In case of doubt, define these records as warm and don't offload them.

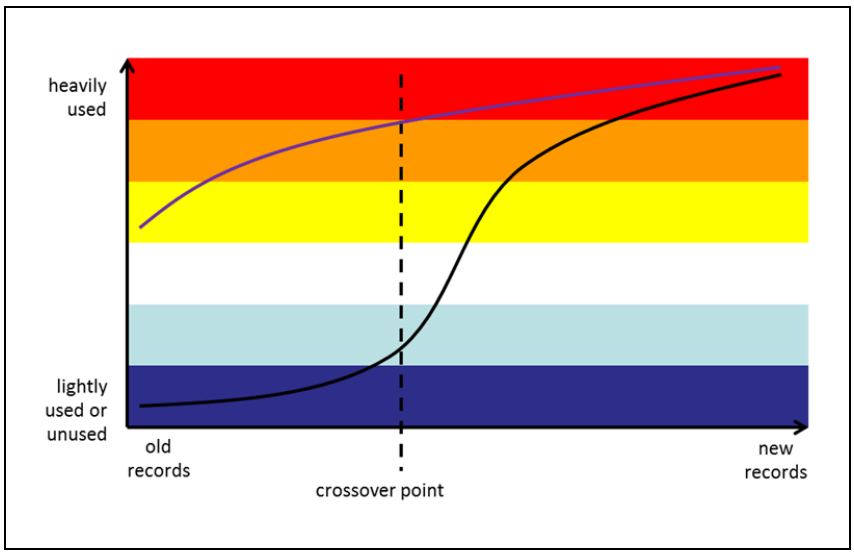

Figure 6 *A data usage analysis exercise can result in a clear presentation of usage of individual records. It shows which records in a table may be considered for offloading.*

When defining a crossover point, be careful with the following aspects:

- The crossover point is not a date, but an age.
- The crossover point may change over time. For example, initially an age of two years is identified as the crossover point, which must be changed to three years later on. The reason may be that users have started to run more forms of historical analysis leading to a change of data usage. This may slide the crossover point to an older age (to the left in Figure 6).
- Some records that have come of age and have clearly passed the crossover point may be very hot records. These records are called *outliers.* Analyze if there are outliers in tables whose usage is significantly higher than records with approximately the same age. In this case, identify them and change the crossover point accordingly.

Sometimes an entire column contains cold data. They contain data that's barely ever or never used at all. In this case, no crossover point is required. Especially offloading "wide" columns can be useful. Examples of wide columns are columns containing long text blocks, images, scanned documents, or videos. In case of an offloadable column, indicating the columns name is sufficient.

Obsolete Data – Because of its very low data usage, offload obsolete data. The difference between obsolete data and cold data is that for the latter an age-related crossover point is defined, whereas obsolete date has no relationship to the concept of age.

For obsolete records a criterion indicates the ones that are obsolete. An example of such a criterion is an indication of whether a shop has been closed. Or, all the values of a column may be obsolete when it contains characteristics of a product that are not in use anymore.

Audio/Video Data – Audio and video data is barely ever used for reporting and analysis. As indicated, most reporting tools do not even have features to analyze them. To reduce the size of the data warehouse, it's strongly recommended to offload all this data to Hadoop. Especially if separate tables have been created in a data warehouse to store these large values, offloading them entirely to Hadoop is a relatively simple exercise. In addition, in many situations this type of data is only used by a small group of users, but it may still be in the way for other users.

An additional advantage of moving audio and video data to Hadoop is that specialized tools may be available to analyze this data. This would enrich the organization's analytical capabilities.

Textual Data – For a large part, what applies to offloading audio and video data also applies to offloading textual data. Textual data may end up filling up a large part of a database, slowing down the loading processes and all the queries.

Justification of Data Offloading – After the offloadable data has been determined for each table, determine whether the amount of data justifies an offload. This depends on the following key aspects:

- **Database size:** When an entire SQL database is less than 1 Terabyte large, do not offload data at all.
- **Total number of records in a table:** The bigger a table is, the bigger the chance that performance problems exist with the query workload and the loading process, and the more relevant offloading can be. For example, do not offload a table with barely a hundred records. The overhead costs would be too high compared to the performance and storage advantages.
- **Percentage of offloadable records:** The higher the percentage of offloadable data in a table is, the higher the storage savings are and the bigger the query performance improvements are.
- **Percentage of queries accessing non-offloaded data:** The more queries exist that access only the remaining data, the bigger the query performance improvement will be after the offload.
- **Query performance:** The more queries with poor performance exist, the bigger the benefit of offloading will be. Do not offload tables with no query performance problems.
- **Total amount of offloadable data:** A final check is whether the total amount of offloadable tables is significant. To offload data from one or two tables only, is hard to justify financially. If not sufficient offloadable tables are identified, the process stops.

Summary – The result of this first step is a list of tables for which offloading can be considered plus criteria indicating the offloadable data. For cold records the criterion is identified with a crossover point, for obsolete data the criterion is a list of columns and records, and for other types of data it's an indication of the entire table or a set of columns.

Step 2: Installing Hadoop

For installing Hadoop we refer to the manuals and documentation of various vendors. However, to guarantee that it has been installed and optimized properly and that replication has been set up correctly, consult Hadoop experts. Do not leave this to beginners; your data is too valuable!

No specific Hadoop implementation is recommended. There is only one requirement. To be able to work with CIS, a *SQL-on-Hadoop engine* must be installed. Such an engine makes the Hadoop files look like SQL tables and can be accessed using SQL. Currently, CIS supports the SQL-on-Hadoop engines *Hive* (versions 1 and 2) and *Impala.* Important to understand is that Hive and Impala share the same metadata store, which is accessible via the *hCatalog* interface. For example, tables created with Hive can be queried with Impala, and vice versa. So, an initial investment in one does not require a major migration when switching to another one.

Note: The market of SQL-on-Hadoop engines is currently highly volatile. New products and new features are added almost every month. Therefore, it's hard to predict what the "best" product will be in the long run. CIS will continue to support more SQL-on-Hadoop engines in the future.

Step 3: Importing Tables in CIS

Identify all the SQL databases in the data warehouse environment that contain offloadable data. Connect CIS to all these SQL databases. Also connect CIS to the selected SQL-on-Hadoop engine to get access to Hadoop HDFS. For all these data sources organize the correct data access privileges.

Next, import in CIS the definitions of *all the tables* that are being used by the reports; see Figure 7. Then, publish them all via ODBC/JDBC. The reason that all the tables (and not only the ones with offloadable data) must be imported and published, is that we want CIS to handle all data access. If only tables with offloadable data are imported, the reports themselves would have to access the two different data sources (SQL database and CIS) and this seriously complicates report development.

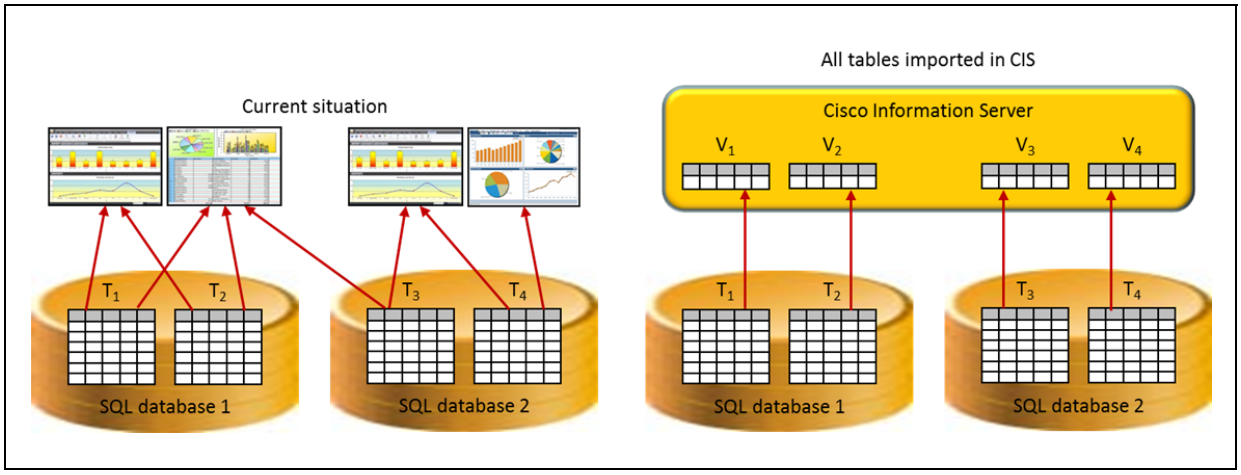

Figure 7 *All the tables (including the ones without offloadable data) accessed by the reports must be imported in CIS so that in the new situation all the data access is handled by CIS. In this diagram, for each table T a view called V is defined.*

If primary and foreign keys have been defined on the SQL tables, define the same set of keys on the views in CIS. The reason is that several reporting tools need to be aware of these keys to understand which relationships between the tables exist.

Step 4: Migrating Reports to CIS

In the new environment, CIS hides the hybrid data storage system by decoupling the reports from the data stores. For this, all the reports have to be migrated. Instead of accessing the tables in the SQL databases directly, they must access those same tables via views defined in CIS; see Figure 8. These are the views defined in the previous step.

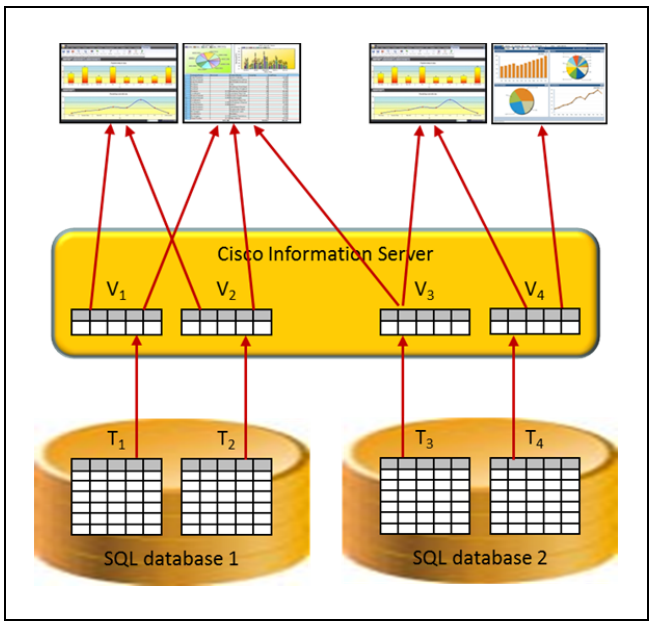

Figure 8 *All the data access of the reports must be redirected. In stead of accessing the real tables in the SQL databases, reports must access the views defined in CIS.*

This step starts with redirecting the reports from the SQL databases to CIS. Instead of using the ODBC/JDBC driver of the SQL database, they must access data through one of CIS' ODBC/JDBC drivers. This mainly involves work on the side of the reporting tool. For example, changes may have to be made to a semantic layer or in a metadata directory. For most reports this migration will proceed effortlessly. Many of them will show exactly the same results as they did before, because they're still accessing the same tables in the same SQL databases.

There may be some minor issues. For example, it could be that the reports execute SQL statements that use proprietary features not supported by CIS. In this case, try to rewrite these SQL statements and replace the proprietary features by more standard ones. If specific scalar functions are missing, develop them using CIS itself. These functions are executed by CIS itself and not by the underlying SQL database server.

If there is no solution, use the pass-through feature of CIS. In this case, CIS receives the queries from the report and passes them unchanged to the underlying database server.

At the end of this step, the reports still access the same tables they've always accessed; it's just that now their SQL statements pass through CIS. This opens the way to transparent offloading of data. In fact, all the steps described so far can be executed on the data warehouse environment without anyone noticing it.

Note: Execute this step report by report. First, apply this step to one report before moving on to the next. This way, lessons learned when migrating one report can be used when migrating the next one.

Step 5: Creating Tables in Hadoop

In this step, new tables are created in Hadoop that will hold the offloaded data. Use the selected SQL-on-Hadoop engine to create them. What these Hadoop tables will look like and how they will be handled depends completely on whether an entire table, a subset of records, or a subset of columns is offloaded.

- **Offloading an entire table or a subset of records of a table:** For each table that must be offloaded entirely, or for which a subset of records must be offloaded, define a table in Hadoop. Each Hadoop table must have a similar table structure as the original SQL table. This means that they must all have the same set of columns, identical column names, and each column must have a similar data type.

- **Offloading a subset of columns:** For each table, for which a subset of columns is offloaded, define a table in Hadoop that includes the primary key columns of the original table plus the columns that must be offloaded. Make sure that the same column names and similar data types are used when creating the Hadoop tables.

Currently, Hive and Impala, and most of the other SQL-on-Hadoop engines as well, don't support primary keys. The manuals state: "It's the responsibility of the applications to guarantee uniqueness of the primary key columns." Over time, this will change. If the SQL-on-Hadoop engine in use does support primary keys, copy the primary key definition from the original SQL table to the Hadoop table.

If indexes can be defined using the SQL-on-Hadoop engine, consider copying the indexes from the original SQL table to the Hadoop table as well.

Because most SQL-on-Hadoop engines are relatively young, they don't support all the data types that are offered by SQL database servers. Evidently, classic data types, such as integer, character, and decimal, are supported. For each SQL-on-Hadoop engine a list of supported data types is available. For example, see the Apache documentation[3] for a detailed description of the data types supported by Hive.

Physical parameters must be set for each Hadoop table. These parameters define how data is stored and accessed. Especially important is the *file format*. Examples of file formats are `SEQUENCEFILE`, `TEXTFILE`, `RCFILE`, `ORC`, and `AVRO`. The file format has an impact on the size of the files, data load speed, query performance, etcetera. For reporting and analytics of structured data, it's recommended to use the `ORC` (Optimized Row Columnar) file format[4]. This is a column-oriented file format that compresses the data, reduces I/O, and speeds up query performance. It has been designed specifically for reporting and analytical workloads. For tables that primarily contain audio and video data select the `SEQUENCEFILE` format, and select `TEXTFILE` when textual data has to be stored.

Step 6: Extending Views to Include the Hadoop Tables

The views defined in Step 3 only show data stored in the SQL databases. In this step, the views that point to SQL tables that contain offloadable data are extended with Hadoop tables that will contain the

[3] Apache, *Hive Data Types*, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Types
[4] Apache, *ORC Files*, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+ORC

offloaded data. The solution depends on whether a subset of records or the entire table is offloaded, or whether a subset of columns is offloaded.

Offload an Entire Table or a Subset of Records – Redefine each view in such a way that it includes the equivalent Hadoop table. The old definition containing only the SQL table:

```
SELECT   *
FROM     SQL_TABLE
```

The new definition after extending it with the Hadoop table:

```
SELECT   *
FROM     SQL_TABLE
UNION ALL
SELECT   *
FROM     HADOOP_TABLE
```

The table `HADOOP_TABLE` refers to the table defined with the SQL-on-Hadoop engine and `SQL_TABLE` to the equivalent table in the SQL database. The `UNION` operator is used to combine the two tables. If possible, use the `UNION ALL` operator when combining the two tables (as in the example above). `UNION` requires that duplicate rows are removed from the result. Whether there are duplicate rows or not, this requires processing. If primary keys have been defined on the tables, `UNION ALL` is recommended instead, because this operator doesn't require removal of duplicate rows, which speeds up query processing. `UNION ALL` can be used in this situation, because no duplicate rows exist.

Note that the definition of this view is changed again in Step 8. When all the data has been offloaded and `SQL_TABLE` has become empty, it can be removed from the definition.

The result of extending these views is presented in Figure 9. Here, the view called V2 is defined on the SQL table T2 combined with the Hadoop table V2. For the reports that access V2 nothing has changed. The reports still return the same results.

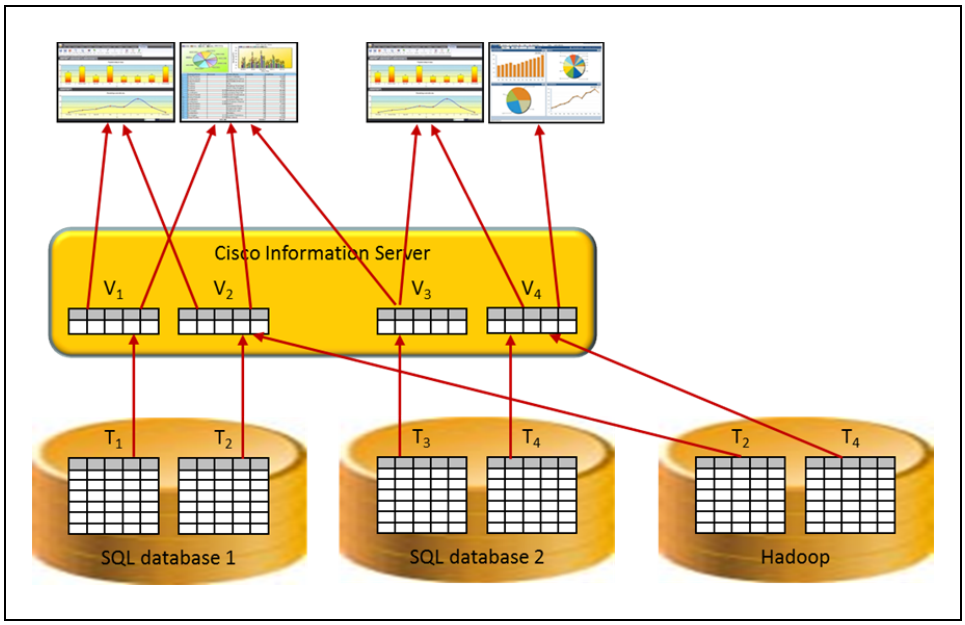

Figure 9 *The view definitions are changed so that they show data from the SQL tables and from the Hadoop tables.*

Offload Subset of Columns – When a subset of columns must be offloaded, the altered view definition looks quite different:

```
SELECT   *
FROM     SQL_TABLE LEFT OUTER JOIN HADOOP_TABLE
         ON (SQL_TABLE.PRIMARY_KEY_COLUMNS = HADOOP_TABLE.PRIMARY_KEY_COLUMNS)
```

With this definition, the offloaded columns are (virtually) added to the `SQL_TABLE` using a join.

Note that in this definition a left-outer-join is used. An inner join can be used (which may be faster), if for each row in the SQL table one row exists in the Hadoop table. If the columns to be offloaded contain many null values, it's recommended to insert a row in the Hadoop table for each row in the SQL table that contains values in those columns. And in that case, a left-outer-join is required.

Remarks – Make sure that in CIS the statistical information on the Hadoop tables has been updated. This is important, because CIS must know that the Hadoop tables are empty and that they don't have to be accessed. If for some reason, CIS accesses Hadoop (unnecessarily), then add a fake condition to the view definition that tells CIS not to access it; see the last line of code in this example:

```
SELECT   *
FROM     SQL_TABLE
UNION ALL
SELECT   *
FROM     HADOOP_TABLE
WHERE    1=2
```

Note: This step has no impact on the reporting results. When accessing the new versions of these views, the results should still be the same as before this step, because the Hadoop tables contain no data yet.

Step 7: Collecting Statistical Data on Offloadable Data

Before offloadable data is migrated, it's important that *statistical data* on that data is collected. This statistical data is required to test the report results afterwards; see Step 10.

The more statistical data is derived, the better it is, but at least collect the following statistical data on the tables containing offloadable data:

- For each table determine the number of offloadable records. This is necessary to determine if all of the records have been migrated correctly.
- For each table determine the lowest and highest key value of that data.
- For each numeric column determine the sum and the average value (with a high precision) of all the values. This is necessary to determine whether the numeric values have been copied correctly and that there have been no problems with numeric data type conversions.
- For each alphanumeric column determine the average value (high precision) of the lengths of each string value. Remove the trailing blanks before the length is calculated.

Step 8: Initial Offloading of Data to Hadoop

The Initial Offloading of Data – This step describes the initial offloading of data to Hadoop. The next section describes the refresh of the offloaded data. Some of the work done in this step is only done once, it's just to get the ball rolling. The next step deals with the continuous offloading process.

The initial offloading of data stored in the SQL tables to the Hadoop tables requires a solution that extracts data from the SQL tables, loads it into the Hadoop tables, and removes it from the SQL tables. Such a solution can be developed in several ways and several technologies are available, such as CIS itself, dedicated ETL technology, Hadoop's own ETL tool called Sqoop, or an internally-developed solution. The choice primarily depends on the amount of data to be offloaded.

Regardless of the solution, do not spend too much development time on this step, because some parts of the solution will be used only once. Therefore, even if the amount of data is considerable, go for the simplest solution, even if this means that the migration takes some time. It doesn't make sense to spend four days on development to improve the performance of the migration solution from two hours to two minutes.

Note that the views don't have to be redefined after this step, because they already access the offloaded columns from Hadoop.

Migrating an Entire Table

1) Develop the logic that moves all the data of the SQL table to the Hadoop table:

```
INSERT   INTO HADOOP_TABLE
SELECT   *
FROM     SQL_TABLE
```

2) Drop the entire SQL table:

```
DROP TABLE SQL_TABLE
```

3) Redefine the view, because there is no need to access the SQL table anymore. The new definition becomes:

```
SELECT   *
FROM     HADOOP_TABLE
```

4) Change the current ETL process that loads data into the `SQL_TABLE`. It must be redirected to load data straight into the `HADOOP TABLE` instead.

Migrating a Subset of Records

1) Develop the logic that migrates the data from the SQL table to the Hadoop table:

```
INSERT   INTO HADOOP_TABLE
SELECT   *
FROM     SQL_TABLE
WHERE    CHARACTERISTIC = 'XYZ'
```

The `WHERE` clause contains a condition that identifies the subset of records to be offloaded:

2) Develop the logic to delete offloadable data from the SQL tables:

```
DELETE   FROM SQL_TABLE
WHERE    CHARACTERISTIC = 'XYZ'
```

Note that copying an integer or string value from one system to another is straightforward. This does not apply to video and audio data. Usually, this is not just a matter of copying the audio from one system to another. It may well be that a dedicated program has to be written to extract an audio value from the SQL table and to store it in an Hadoop table, and for both dedicated logic has to be written; logic specific for that platform.

Migrating a Subset of Cold Data

1) Develop the logic that migrates the cold data from the SQL table to the Hadoop table:

```
INSERT   INTO HADOOP_TABLE
SELECT   *
FROM     SQL_TABLE
WHERE    DATE_RECORD < CURRENT_DATE -  3 YEARS
```

The table `HADOOP_TABLE` refers to the table defined with the SQL-on-Hadoop engine and `SQL_TABLE` to the equivalent table in the SQL database. This example relates to cold data, therefore the condition in the `WHERE` clause indicates a crossover point which is three years; data older than three years is moved to Hadoop.

2) Develop the logic to delete offloadable data from the SQL tables:

```
DELETE   FROM SQL_TABLE
WHERE    DATE_RECORD < CURRENT_DATE -  3 YEARS
```

Migrating a Set of Columns

1) Develop the logic to offload a subset of columns:

```
INSERT   INTO HADOOP_TABLE
SELECT   PRIMARY_KEY, OFFLOADED_COLUMN1, OFFLOADED_COLUMN2, ...
FROM     SQL_TABLE
```

This statement must be expanded with the following WHERE clause if all the offloaded columns contain a null value:

```
INSERT   INTO HADOOP_TABLE
SELECT   PRIMARY_KEY, OFFLOADED_COLUMN1, OFFLOADED_COLUMN2, ...
FROM     SQL_TABLE
WHERE    NOT(OFFLOADED_COLUMN1 IS NULL AND OFFLOADED_COLUMN2 IS NULL AND ...)
```

2) Remove the offloaded columns from the SQL table:

```
ALTER TABLE SQL_TABLE DROP OFFLOADED_COLUMN1, OFFLOADED_COLUMN2, ...
```

3) Change the current ETL process that loads data into the SQL_TABLE. Data for the offloaded columns must go straight into HADOOP TABLE instead.

Offloading Large Sets of Data – The logic above suggests that offloading can be done by using two or three SQL statements. In principle this can be done using CIS. CIS allows data to be copied from one data store to another using an INSERT/SELECT statement. If the amount of data to be copied is small, this will work, and may even work efficiently. But this is not the recommended approach when large sets of data have to be offloaded. For example, one such statement may be responsible for copying one billion rows to Hadoop in one go. The chance that problems will occur during the execution of such a simple INSERT/SELECT statement, is substantial. The above statements are purely included to show the logic to be executed, but it should not be regarded as the preferred implementation form when large sets of data have to be offloaded. When large amounts of data have to be moved to Hadoop, use dedicated tools, such as ETL tools. They have been optimized and tuned for this type of work.

Even when ETL tools are used, it may take a considerable amount of time to copy the data. In fact, it may even take days. Because offloading must be done offline (else, reports will show incorrect results), it may be required to offload the data in batches. For example, every night one month of cold data is copied, or all the transactions of a region are copied. Because the views accessed by the users include both tables, report results will not be impacted. After each night of offloading, CIS extracts more data from Hadoop and less from the SQL database server. If such a strategy is selected and possible, copy the oldest data first, and work your way to the more current data.

Differences in SQL Dialects – The SQL dialect supported by a SQL database server can differ slightly from that of a SQL-on-Hadoop engine. In most cases these differences are minor, but exist nevertheless. If differences exist, some reports won't work or won't work properly. It's important to identify these differences.

Potential differences:

- **SQL language differences:** Many vendors of SQL database products have added their own SQL proprietary constructs and statements to the language. For example, DB2 supports the MERGE statement that most others don't.
- **SQL processing differences:** Some constructs are processed differently by different vendors.
- **Function differences:** Many SQL functions are supported by all SQL products, but each product has its own proprietary functions. For example, some have added functions for manipulating XML documents, and others for statistical analysis.

- **Function processing differences:** Some SQL functions are processed differently by various products. Especially date-time related functions are notorious for their differences.
- **Data type differences:** Some SQL databases support special data types. For example, a few have data types for geographical analysis, and some allow development of special data types.
- **Data type processing differences:** Data types may be named the same, but that doesn't mean their behavior is the same. For example, the maximum time DB2 can store in a time column is `24:00:00`, while it's `23:59:59` for Oracle. This can lead to different results.
- **Null value processing differences:** The null value is not processed in exactly the same way by all SQL products.

CIS handles many of these differences. Nevertheless, these differences can lead to SQL queries that cannot be executed, or that return different results.

Step 9: Refresh of the Offloaded Data

For many forms of offloading, after the initial offload, the work is done. For example, when an entire table has been offloaded, from then on new data is added to Hadoop. The same is true for obsolete data that has to be offloaded just once. But this is not true for, for example, warm data, which eventually becomes cold data and must be moved to Hadoop. Therefore, changes must be made to the ETL logic that feeds the data warehouse environment with new data.

Periodically, data that has turned cold must be moved. Schedule this solution to run in synch with the crossover point of the cold data. For example, if the crossover point is older than 36 months, the migration must be executed every month, and if the crossover point is 8 quarters, it must be executed every quarter.

Most of the development work will deal with the ETL logic, and not with what has been defined in Hadoop or CIS.

Step 10: Testing the Report Results

When data has been migrated to Hadoop, and before it's made available to the users, it's important that extensive tests are run to determine whether all the reports still return identical results. The report results of the new situation must be 100% identical to the previous situation. We recommend a two-step testing approach where the data quality is tested first and then the reports.

For the first step the statistical data calculated in Step 7 is needed. Run the same queries again that were used to calculate that statistical data. Check if all the numbers are still identical. Differences can arise for various reasons, because differences exist between data types, the way nulls are handled, how queries are processed, and how functions are executed. For example, some SQL products return a rounded integer when an integer is divided by another integer value, while others return a decimal. Evidently, this leads to different report results.

When data quality has been tested, the real reports must be run on the new hybrid system. This is primarily a visual test. Compare the report results of the new situation with those developed for the old situation.

Step 11: Adapting the Backup Process

In most data warehouse environments, the backup process of the SQL databases has already been organized. Periodically, backups are created in case a restore is required. A restore may be needed due to a hardware failure (data corruption on disk, disk or node crash, rack failure), a user error (corrupted data writes, accidental or malicious data deletion), or a site failure due to, for example, fire.

Now that data is offloaded to Hadoop, a separate backup process must be developed for that portion of the data. Hadoop HDFS supports *data replication*. By replicating data, it's relatively easy to handle the hardware failures. Configure Hadoop in such a way that at least three replicas are stored. Be sure that the first two replicas are on different hosts and the third replica on a different rack. In addition, don't forget to back up the configuration files and the NameNode metadata.

In the case of cold data, new data is only added to the Hadoop database when data has come of age, so periodically, an incremental backup is recommended.

The order of doing a backup is as follows:

1. Load new data into the SQL database
2. Offload cold data from the SQL database
3. Do the backup of the SQL database
4. Load cold data into Hadoop
5. Do the (incremental) backup of Hadoop

If things do go wrong and a restore is required, it's important that the restore processes for the SQL database and Hadoop are synchronized. At any point in time no record may appear in both databases, because it would instantly lead to incorrect report results.

7 Getting Started

We recommend deploying a step-by-step, iterative process when implementing a hybrid data warehouse environment in which offloaded data is stored in Hadoop. Do not execute the steps described in the previous section one by one for all tables that contain offloadable data. Instead, work table by table. First, identify the table with the largest amount of offloadable data. Execute all the steps to offload this data. Check if everything works, and only if it does, move on to the second table.

When everything works, so when data has been offloaded and the reports return identical results, it's time to exploit the features of CIS more extensively. After step 11, only its abstraction and data federation capabilities are used. But CIS can do more. For example, when many report definitions contain the same data-related definitions, extract them from the reports and define them in CIS views. Or, when data security rules are defined in the reports, define them in CIS. The more specifications are extracted from

the reports and implemented in CIS, the easier they are to define and maintain and it leads to more consistent reporting results.

About the Author Rick F. van der Lans

Rick F. van der Lans is an independent analyst, consultant, author, and lecturer specializing in data warehousing, business intelligence, database technology, and data virtualization. He works for R20/Consultancy (www.r20.nl), a consultancy company he founded in 1987.

Rick is chairman of the annual European Enterprise Data and Business Intelligence Conference (organized annually in London). He writes for SearchBusinessAnalytics.Techtarget.com, B-eye-Network.com[5] and other websites. He introduced the business intelligence architecture called the *Data Delivery Platform* in 2009 in a number of articles[6] all published at BeyeNetwork.com. The Data Delivery Platform is an architecture based on data virtualization.

He has written several books on SQL. Published in 1987, his popular *Introduction to SQL*[7] was the first English book on the market devoted entirely to SQL. After more than twenty five years, this book is still being sold, and has been translated in several languages, including Chinese, German, and Italian. His latest book[8] *Data Virtualization for Business Intelligence Systems* was published in 2012.

For more information please visit www.r20.nl, or email to rick@r20.nl. You can also get in touch with him via LinkedIn and via Twitter @Rick_vanderlans.

About Cisco Systems, Inc.

Cisco (NASDAQ: CSCO) is the worldwide leader in IT that helps companies seize the opportunities of tomorrow by proving that amazing things can happen when you connect the previously unconnected. Cisco Information Server is agile data virtualization software that makes it easy for companies to access business data across the network as if it were in a single place.

For more information, please visit www.cisco.com/go/datavirtualization.

[5] See http://www.b-eye-network.com/channels/5087/articles/
[6] See http://www.b-eye-network.com/channels/5087/view/12495
[7] R.F. van der Lans, *Introduction to SQL; Mastering the Relational Database Language*, fourth edition, Addison-Wesley, 2007.
[8] R.F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann Publishers, 2012.