https://doi.org/10.48009/1_iis_2015_91-101

Issues in Information Systems
Volume 16, Issue I, pp. 91-101, 2015

ETL WORKFLOW GENERATION FOR OFFLOADING DORMANT DATA FROM THE DATA WAREHOUSE TO HADOOP

Matthew T. Hogan, Georgia Southern University, matthew_t_hogan@georgiasouthern.edu
Vladan Jovanovic, Georgia Southern University, vladan@georgiasouthern.edu

ABSTRACT

The technologies developed to address the needs of Big Data have presented a vast number of beneficial opportunities for use alongside the traditional Data Warehouse (DW). There are several proposed use cases for using Apache Hadoop as a compliment to traditional DWs as a Big Data platform. One of these use cases is the offloading of "dormant data" - that is, infrequently used or inactive - data from the DW relational database management system (RDBMS) to Hadoop Distributed File System (HDFS) for long-term archiving in an active and query-able state. The primary goal of this work is to explore and define the process by which Extract-Transform-Load (ETL) workflows can be generated utilizing applicable tools to achieve such a data offloading solution. Additional focuses of this project include conducting experiments to measure DW query performance before and after the Hadoop archive implementation and to provide analysis on the usability of the HDFS "active archive." This paper discusses the cost-savings and performance gains to the DW given this approach. The process defined in our research and experimentation suggests the feasibility of the development of fully-automated ETL workflows for the offloading and archiving of data to Hadoop.

Keywords: Data Warehouse (DW), Extract-Transform-Load (ETL), Big Data, Hadoop, Archive, Offload

INTRODUCTION

The business technology landscape has changed drastically over the past decade. Introductions of new technologies have caused major disruptions in the way in which information technology is used to drive business value. Since its birth, Hadoop has certainly had an enormous influence on the way in which information technology professionals, data scientists, and computer scientists think about the storage and processing of data. Still, even ten years later, there are many questions that remain either unanswered, partially answered, or inadequately answered with respect to its best-practice uses [6].

Big Data platforms were once viewed as competitors to relational database management systems (RDBMS) and the data systems implemented on top of them. Hadoop in particular was questioned as to whether it would be a traditional data warehouse (DW) killer. However, after careful investigation and much scrutiny, big data platforms, such as Hadoop, are now widely accepted as complimentary in nature to RDBMSs and DWs [8] – especially when it comes to optimizing the data warehouse. While it shines for many applications, some data types and problems are just a bad fit for RDBMSs. Hadoop, however, has use cases with a common theme which is to free up existing systems to allow them to perform only the tasks that they are designed to do and has a knack for tackling projects that are technologically impossible or cost-restrictive on RDBMS systems. Given this viewpoint, it is now prudent to start synthesizing and documenting the processes of Hadoop implementations focused on these complimentary-to-the-DW use cases.

Some of the more rewarding use cases of big data platforms as a new weapon in the business technology arsenal are listed below [1, 9, 12, 18]:

- Data Warehouse storage offloading and active archiving
- Enterprise-wide "data hub" or "data lake" repository
- ETL batch processing offloading (parallel processing)
- Introduction of streaming data into DW environment
- Introduction of semi- and un-structured data into DW environment
- Advanced analytics, distributed machine learning, and knowledge discovery
- Ad-hoc and exploratory analytics and data science

To this end, this paper is focused on defining a process for one of the identified use-cases of using Hadoop as an additional piece of the overall business technology puzzle to store and process data. Namely, this use case is the

offloading of inactive and infrequently used data from the DW (implemented on the RDBMS) to Hadoop for storage in an active archive capacity with fully functioning querying capabilities. The result is a hybrid data warehouse system where data is distributed across more than one storage technology that allows only the most valuable (and usually the most recent) data to be kept in the DW. Henceforth, we will use the terms "dormant," "inactive," "cold," "low-value," and "infrequently-used" interchangeably to describe the target data for offloading from the DW to Hadoop.

PROBLEM STATEMENT

There are several potential motivating factors that might drive a business to pursue an active-archival solution such as the one proposed in this paper. For most use-cases, there will likely be a combination of any of the following justifications [1, 9, 12, 10]:

- High total cost of ownership for data warehouse storage: Storing data in RDBMSs can become expensive due to hardware costs, licensing fees, and system/data management costs.
- A large amount of historical data used infrequently: Data that offers little or even negative return on the investment to host it online can be a problem.
- An increasing data growth rate: The costs will continue to go up while the performance will continue to go down.
- Poor query performance: Larger tables means slower querying.
- Poor ETL performance: Larger tables means slower loading of additional data, especially the updating of indexes.
- Slow backup and recovery processes: Larger databases take longer to backup and restore.
- Regulatory requirements to retain all data: Many forms of bureaucracy – organizational, governmental, or other – may dictate that all data must be kept indefinitely.

Regardless of the motivation, once a business has established that it is going to undertake this type of project, a process definition for achieving such a solution is needed. The main efforts within this process fall into the realm of Extract-Transform-Load (ETL). Thus, the goals of this research are:

1. To define the data offload ETL process,
2. To generate the ETL workflow software necessary to achieve a solution as defined by the process,
3. And to measure the impact that the solution has on the resulting, composite data system.

These goals are designed to offer a repeatable procedure for developing similar solutions for any traditional data warehouse implementation and demonstrate the solution's usability. While this research utilizes a data warehouse based on a dimensional (star schema) model [13, 7], it is expected that the over-arching process should remain modular enough to apply to other DW and data mart (DM) schemas, such as Data Vault [14, 11], Anchor Modeling [17, 11], Spider Schema [8], Normal Form (of varying degrees), or any combination or hybrid of these and others. Future research will need to be conducted to confirm this suspicion.

RESEARCH DESIGN AND SOLUTION IMPLEMENTATION

Project Environment

To account for fairness in certain aspects of our benchmark analysis, it was determined that a controlled-environment was to be used where the computing power of the system running the Hadoop solution was comparable to the system running the SQL Server data warehouse. While query performance comparison between the traditional DW and the Hadoop archive was not a primary objective of this research, it is still relevant to the solution analysis given that the archive needs to be operational and have the same querying functionality as the traditional DW with "reasonable" performance. In this case, the Hadoop system was configured in a single-node cluster and was installed on the exact same machine hardware as the SQL Server system in order to mimic processing power. While the data system instances were running, all other unnecessary processes on the machine were shutdown to allow maximal resource utilization. The data systems were not running simultaneously while the other was being queried and were only running simultaneously when the actual data transfer (ETL) offload was taking place.

Table 1: System Specifications

SQL Server Operating System	Windows NT 6.1.6801 64-bit
Hadoop Operating System	CentOS 6.5 Linux 64-bit
CPU	Intel Core i7 920 @ 3.2GHz

Cores	4 physical, 8 virtual
RAM	12 GB DDR3 1600MHz
Hard Disk	240 GB SSD Max Sequential Read: Up to 500 MB/s Max Sequential Write: Up to 250 MB/s 4KB Random Read: Up to 72,000 IOPS 4KB Random Write: Up to 60,000 IOPS

Test Data Warehouse

This project utilizes a familiar test data warehouse provided free by Microsoft – the AdventureWorksDW. The AdventureWorksDW database is based on a fictional bicycle manufacturing company named Adventure Works Cycles. Adventure Works produces and distributes bicycles to its customers. They sell wholesale to specialty shops and they sell to individuals through the Internet. The AdventureWorksDW is a dimensionally modeled data warehouse. That is, the data warehouse is comprised of star schemas – each of which represents one business area or subject [13]. For our purposes, we were concerned with querying on the sales-related facts and the finance fact within the warehouse.

Test Data Generation

This project was constructed with its own generated data. While the schema is based on a hypothetical business and was provided by Microsoft, the data values themselves have no real business meaning. The original records provided by Microsoft to be loaded into the Adventure Works data warehouse did not reach a storage size large enough to adequately demonstrate the magnitude of cost and performance savings. In other words, the data used in this project is not insightful in any way and it only serves to populate database for experimental analysis and benchmarking purposes. To generate this data, the SQL Data Generator tool from Redgate Software was used. Below are the rules used for the test data generation that were transposed into the tool for data generation:

- All dimension tables were filled with 1,000 generated rows. Key values ranged from 1 to 1000. The exception to this rule was the Date dimension, which was assigned 3652 rows to simulate 10-years' worth of data (including two leap years) and had key values which ranged from 1 to 3652.
- All fact tables were filled with 10,000,000 generated rows. The row counts for fact and dimension tables were designed to emulate real-world proportions and the sizes were arbitrarily selected to generate a resulting database of close to 10GB in size. This size was deemed adequate to illustrate the storage and performance savings.
- All surrogate keys referencing the Date dimension were given a random value from a range of 1 to 3652 in order to simulate a ten-year boundary of records normally distributed between 1/1/2005 and 12/31/2014.
- All surrogate keys referencing all other dimensions were given a random value from a range of 1 to 1000 to simulate records normally distributed amongst the 1000 records generated for each dimension.

Test Query Generation

To test the performance of the techniques implemented in the solution, queries were written to simulate questions that typical business users might ask in regards to this data set's schema. The queries were designed with differing number of logical joins to get an idea as to the scalability of the performance impact to the DW. These queries were first written in SQL and then translated into HiveQL for use on both the original, RDBMS-based data warehouse and also the Hadoop data warehouse archive. Differences in dialects were first noted and then applied according to the reference materials used [5]. In our case, very minimal effort was needed for proper translation, which speaks to the ease with which dimensional DWs can be implemented on Hive.

Data Warehouse Offload to Active Archive Solution

One of the main purposes of a data warehouse is to permanently store and maintain data history. However, when the DW database becomes overburdened with data and is too large, too slow, and/or too costly, it is necessary to perform some data warehouse optimization methods to meet cost and performance expectations. Traditionally, these methods have included removing some of the low-value data completely or moving the data to an offline archive storage medium [18, 2]. These methods introduce the drawbacks of either losing the historical data altogether or making this historical data inaccessible without great restoration efforts and thus losing the ability to do historical

analysis in a timely manner. In this situation, it then makes sense to offload the data to an inexpensive, fault-tolerant, and actively online data storage system. This approach offers the benefits of supporting on-the-fly queries with reasonable performance and low enough costs to justify keeping historical data indefinitely. This is where Hadoop comes in to fill this active archive role. By utilizing Hadoop, we reduce strain on the data warehouse or data mart and let it focus on what it is designed to do which is to offer high speed queries on integrated, high-value, and actively-used data.

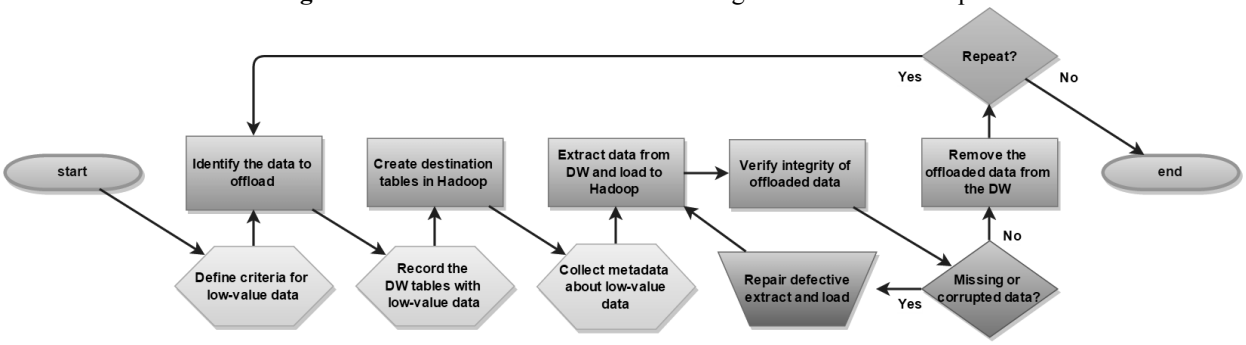
Process Definition

This process provides a step-by-step instruction set on how to offload data from within the data warehouse environment into Hadoop for archiving purposes. The steps are as follows:

Step 1: Identify the data to offload
Step 2: Create destination tables in Hadoop
Step 3: Collect metadata about candidate offload data
Step 4: Extract from the DW and load to Hadoop
Step 5: Verify the integrity of the offloaded data
Step 6: Remove the offloaded data from the DW

Step 1 in the process involves labeling which pieces of data are candidates to be offloaded and which are not. This decision is largely subjective as it depends heavily on the business requirements that dictate the situation. In other words, there is no set of objective criteria with which to apply to all DW data. A justifiable case will need to be made to assess if the identified data makes a good offload candidate. Considerations that include (but are not limited to) measurements such as the percentage of offload-able data in relation to the total table size, the frequency with which the table is queried, and performance on said queries are all factors that will determine if the identified data should be offloaded or not. Step 2 sets up the Hadoop archive to receive offload-able data. The tables created in this step should closely mirror the table definitions of those DW tables affected by the offload. There may be circumstances when subsets of tables can be considered as archive destinations, but this is wholly dependent on the specifics of the use case and for the purposes of this research, by default, it is safe to assume that we want to keep a complete historical account of our data. Step 3 is concerned with verifying the integrity of the cross-storage data as a whole. That is, the metadata collected in this step is used to ensure that data is not lost or duplicated in any way – which will affect the quality of any queries performed thereafter. At a very minimum, row counts for each table affected by the offload need to be recorded. Other metadata that might be pertinent to collect would be the range of key values (highest and lowest) and the sums of numeric columns for each table. Step 4 defines the repeatable practice of extracting data from the warehouse and loading it to the archive. Attention to detail is required with this step as it needs to be designed so that it can ensure the integrity of the data while being scheduled with some frequency. The efforts of this step center on developing ETL processes to migrate the offload data periodically. Step 5 uses the metadata gathered in Step 3 to verify the accuracy and consistency of the data in the hybrid data warehouse environment. This step must be executed before any deletes are performed or there is risk of losing or corrupting data. Finally, Step 6 is to remove the now offloaded data from the data warehouse. The deletion of the offloaded data should be followed up with taking new statistics on the table within the RDBMS. The process can be illustrated in the following process flowchart diagram:

Figure 1: Process Flowchart for Offloading DW Data to Hadoop



Apache Hive (Data Warehouse Schema) Solution

As per the defined process, before any data can be offloaded, the data warehouse destination in Hadoop must be created. This is completed by creating mirrors of the Fact and Dimension tables within Apache Hive. The raw data will be stored in HDFS and then the schema-on-read capabilities of Hive [5] allow the data to be query-able. It is important to note that big data systems have their own set of data modeling best practices which should be followed. In this case, the dimensional model of the existing DW translates well to Hadoop. The model generated for this research stayed within the guidelines of generally accepted best practices using HiveQL [13]. We defined the Hive tables corresponding to the DW tables where there was offload data identified. In our case, this happened to be our DW fact tables. The dimension tables needed to be created as well, but since we performed a full truncate and reload of each dimension for every scheduled offload, these tables were created and loaded with Apache Sqoop as illustrated in the next section.

Apache Sqoop & Apache Oozie (ETL) Solution

The first step in the process of offloading targeted data from the DW to Hadoop is to determine what data is considered inactive or low-value. To do this, specific criteria for identification needs to be developed. In the case of this research and its hypothetical business application, it was deemed that any business event records older than two years were of little value to the business analysts querying the data. Thus, these low-value records were infrequently queried and lay dormant in the warehouse offering little return on investment. After the dormant data was identified, we collected some metadata about those tables affected by the offload. This metadata was used to verify the integrity of the data after the offload took place. Specifically, to ensure that no data was lost prior to deleting any data, it was necessary at the very least to take a row count of the affected tables – for both the DW and the Hadoop archive. The affected tables are all of the tables where inactive data is shown to exist based upon the identification criteria applied. In this case, this happened to be all of our fact tables as each one has a Date Key, which is used in the identification of "cold" data. This metadata was used in our process after the offload of the dormant data by comparing the resulting row counts in both the active archive and DW for verification purposes. Additional verifications on collected metadata may be prudent in some situations, but for this research a row count check sufficed.

Table 2: Record Counts for Tables Affected By Offload

Table	Current DW Row Count	Current Archive Row Count	Identified Dormant Data Row Count
FactCallCenter	10,000,000	0	7,998,622
FactCurrencyRate	10,000,000	0	7,999,362
FactFinance	10,000,000	0	7,998,771
FactProductInventory	10,000,000	0	7,998,258
FactSalesQuota	10,000,000	0	7,998,993
FactSurveyResponse	10,000,000	0	8,001,202
FactInternetSalesReason	10,000,000	0	7,997,391
FactInternetSales	10,000,000	0	7,997,391
FactResellerSales	10,000,000	0	7,999,228

Per the defined process, once the destinations for the offloaded data have been defined in Hadoop (Hive in this case), the extraction of the DW data from the existing warehouse (SQL Server in this case) and subsequent loading of this data into Hadoop takes place. And, with the metadata (row counts in this case) recorded, the extraction and loading of the cold data from the DW to the archive can be completed with confidence. These actions were performed with the Apache Sqoop tool.

To allow for full querying capabilities on the archive, the dimension tables need to be imported in their entirety. This was achieved through Sqoop by utilizing the import-all-tables [20] (excluding the facts and metadata tables explicitly) command in conjunction with the hive-import and hive-overwrite commands. Sqoop will not only import this data to HDFS, but it will also create these tables and appropriately define their schemas in Hive if they do not already exist [20]. Note that importing of the dimension tables into Hive will be a full truncate and reload for each scheduled iteration of the DW offload process to account for Slowly Changing Dimension [13] updates and historical accountability. In the case of extremely large dimension tables, additional steps may be required to accommodate a net-change-only update to the archive. As Sqoop is designed as a bulk-loader for massive amounts

of data to be imported in parallel, it is likely that re-design inside the DW would take place before a re-design in offload procedure if performance was a true concern for such a huge dimension. Thus, this accommodation was not implemented in this research.

Once the dimension tables are loaded, the next step is to load the incremental offload data from the DW fact tables into our already-existing Hive fact tables. Sqoop also tackles this task by using the query argument. As an alternative to explicitly scanning for offload-able rows at the time of importing into Sqoop, a view on the RDBMS can be created with this logic embedded which can dynamically identify the rows to be offloaded at any given time. The explicit approach was used in this research for illustrative purposes.

After Sqoop had performed the extract and load on the dormant data, and prior to deleting this data in the DW, a comparison of row counts was required to ensure that no data would be lost in transition. The row count taken on the DW prior to the offload was now the expected sum of the changes to both the DW (which had not yet occurred) and active archive counts (which had already occurred). This can also be viewed as the sum of the resulting differences by deducting the recorded prior row counts from the new row counts (or expected new row count in the case of the DW) in both storages. If this "sum of differences" is zero, then we know that the counts of the rows to-be-deleted from the DW and the counts received by the active archive are the same for the given tables. The formula for this calculation is given below along with its illustration for each table:

$$\sum_{n=1}^{N} (((T_n - O_n) - T_n) + (A_n - P_n))$$

WHERE:
N is number of tables identified as containing dormant data
T is the current row count of the DW table
O is the row count of identified dormant data in the DW table to be offloaded
A is the new row count of the active archive table corresponding to the DW table
P is the prior row count of the active archive table corresponding to the DW table

Equation 1: Formula for Verifying No Occurrence of Data Loss or Overlap

Table 3: Record Counts Comparison for Tables After Offload

Table	(Expected) New DW Row Count Less Prior	New Archive Row Count Less Prior	Sum of Differences (Expected Zero)
FactCallCenter	(10000000 – 7998622) – 10000000	7998622 – 0	0
FactCurrencyRate	(10000000 – 7999362) – 10000000	7999362 – 0	0
FactFinance	(10000000 – 7998771) – 10000000	7998771 – 0	0
FactProductInventory	(10000000 – 7998258) – 10000000	7998258 – 0	0
FactSalesQuota	(10000000 – 7998993) – 10000000	7998993 – 0	0
FactSurveyResponse	(10000000 – 8001202) – 10000000	8001202 – 0	0
FactInternetSalesReason	(10000000 – 7997391) – 10000000	7997391 – 0	0
FactInternetSales	(10000000 – 7997391) – 10000000	7997391 – 0	0
FactResellerSales	(10000000 – 7999228) – 10000000	7999228 – 0	0

If the above computation results in an output of any number other than zero, the process should be suspended and the mismatch should be investigated – with a positive output indicating more rows were loaded into the archive than will be deleted in the DW and a negative output indicating that more rows will be deleted from the DW than were loaded into the archive. However, once it has been verified that the data has been successfully offloaded to the Hadoop system (i.e. the output of the computation was zero), deletion of the now-offloaded, original records from the DW can occur. This was completed by making a simple alteration to the original dormant data identification SQL script. Instead of simply selecting the data, the RDBMS was instructed to delete these records. Once the dormant data records have been offloaded, fresh statistics should be gathered for each affected table. This ensures that any subsequent ETL and querying processes have the most-up-to-date information to perform properly. The data offload optimization of the warehouse would now be completed for this window of time.

The Apache Oozie Workflow editor and Coordinator job scheduler (or comparable scheduling software) should be used to schedule the frequency with which this process will run. The variables for current date and number of retention days can be set by passing values in to parameters to the Sqoop jobs on the Hadoop side which then are subsequently passed on to the data warehouse RDBMS engine. The same holds true for any other variables necessary to support this process. Automating the offload process can be achieved by configuring an Oozie Workflow to call our HiveQL scripts and Sqoop ETL jobs in the appropriate sequence and then scheduling this Workflow to execute as needed within a Coordinator object. Coordinator objects can then of course be aggregated in an Oozie Bundle for batch execution [19].

EXPERIMENTAL RESULTS AND ANALYSIS

Execution

The query performance and storage space usage was monitored on the SQL Server data warehouse prior to offloading the dormant data, after offloading of the dormant data, and also on the Hadoop archive. The statistics were gathered with a number of tools including SQL Server Profiler, Performance Monitor, SQL Server Management Studio for SQL Server and Cloudera Hue for Hadoop. Execution of the queries was conducted locally on the host machine over TCP/IP to mitigate any variability in network latency between the executions.

Storage Cost Calculations

The storage cost calculations are based on size measurements taken on the DW database, both before the implemented solution and after, as well as storage size measurements taken on the HDFS equivalents of the offloaded data. In conjunction with these measurements, a storage cost calculation and subsequent storage cost savings calculation are derived using total cost of ownership (TCO) figures provided by industry research. TCO is a measure that not only includes the physical storage hardware costs, but also any software and associated overhead costs in the upkeep and maintenance of the storage [16]. The tables below outline the measurement findings for our research:

Table 4: DW Storage Size by Table

Table	Initial Row Count	Initial Storage Size (MB)	New Row Count After Offload	Storage Size After Offload (MB)
FactCallCenter	10,000,000	1,023.680	2,001,378	204.877
FactCurrencyRate	10,000,000	396.641	2,000,638	79.354
FactFinance	10,000,000	473.633	2,001,229	94.785
FactProductInventory	10,000,000	386.906	2,001,742	77.449
FactSalesQuota	10,000,000	386.906	2,001,007	77.420
FactSurveyResponse	10,000,000	691.719	1,998,798	138.261
FactInternetSalesReason	10,000,000	312.500	2,002,609	62.582
FactInternetSales	10,000,000	1,775.570	2,002,609	355.577
FactResellerSales	10,000,000	1,816.914	2,000,772	363.523
TOTAL	**90,000,000**	**7264.469**	**18,010,782**	**1453.827**

Table 5: Hadoop Storage Size by Table

Table	Archived Row Count	HDFS Storage Size (MB)
FactCallCenter	7,998,622	1054.1921
FactCurrencyRate	7,999,362	521.2466
FactFinance	7,998,771	529.6917
FactProductInventory	7,998,258	467.9742
FactSalesQuota	7,998,993	421.0336
FactSurveyResponse	8,001,202	482.7736
FactInternetSalesReason	7,997,391	106.7215
FactInternetSales	7,997,391	1778.0855
FactResellerSales	7,999,228	1800.0373
TOTAL	**71,989,218**	**7161.7561**

To calculate the storage costs and savings, the baseline TCO per storage unit for both RDBMS-based and Hadoop-based components of the hybrid data system need to be determined. We rely on previously conducted research to

make this estimation. Fairly recently quoted annual TCO per terabyte (TB) of storage space on the RDBMS ranged from $20,000 to $100,000 with a range of $20,000 to $60,000 being quoted the most frequently [1, 6, 3]. Recently quoted annual TCO per TB of storage space on Hadoop ranged from $333 to $1000 [1, 3].

We used conservative TCO estimates for this calculation to provide, to the best of our knowledge, worst-case storage savings. That is, we used the lowest quoted cost for the RDBMS storage and the highest quoted cost for the Hadoop storage. These figures are $20,000 per TB annually for RDBMS and $1,000 per TB annually for Hadoop. With the TCO for both RDBMS storage and Hadoop storage defined, the formula for the annual storage cost difference (a savings if the difference is negative) of offloading data from the DW to Hadoop is as follows:

Equation 2: Formula for Annual Storage Cost Difference

$$\sum_{o=1}^{O}\left(\sum_{n=1}^{N}\left(\left(\frac{\Delta T_n o}{C}\times R\right)+\left(\frac{\Delta S_n o}{C}\times H\times F\right)\right)\right)$$

***GIVEN*:**
R = RDBMS cost of $20,000 per TB annually
H = Hadoop cost of $1000 per TB annually
C = 1,048,576 (number of MB in 1 TB)

***WHERE*:**
N is the number of tables identified to have offload-able data
O is the number of offloads performed per year
ΔT is the change in size of the RDBMS table (in MB) post-offload
ΔS is the change in size to the HDFS-stored Hive table (in MB) post-offload
F is the replication factor for the HDFS-storage

Using this formula (and assuming that we only perform this offload once-per-year given our criteria that any data beyond two years old is inactive), we found that even the relatively-small, 10-gigabyte data warehouse in this research project was able to realize a total annual storage cost savings of 75.06% by simply offloading approximately 80% of its fact table data – data that was identified as inactive.

Performance Calculations

In order to quantify the query performance gains of the data warehouse and to demonstrate the query functionality and acceptable performance of the Hadoop archive, we measured multiple execution times for our earlier established queries. Note that we were concerned with baseline execution times given static software settings for the RDBMS, Hadoop and the operating systems. No optimization efforts were performed on either data system to maintain the status quo.

Table 6: SQL Server DW Query Execution Results – Before Offload

Query	Number of Logical Joins	Number of Executions	Mean Execution Time (ms)
1	6	10	306530
2	4	10	148852
3	2	10	42519

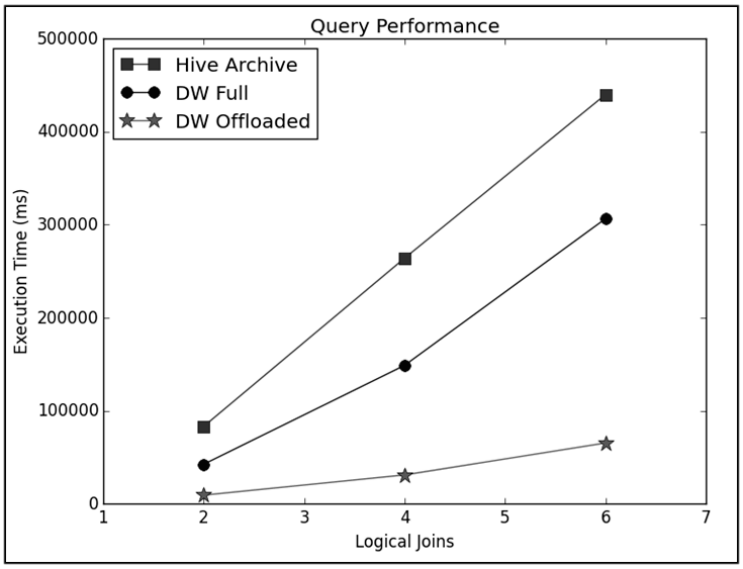
Table 7: SQL Server DW Query Execution Results – After Offload

Query	Number of Logical Joins	Number of Executions	Mean Execution Time (ms)
1	6	10	65411
2	4	10	31060
3	2	10	9319

Table 8: Apache Hive Archive Query Execution Results – Offloaded Data (Un-Indexed)

Query	Number of Logical Joins	Number of Executions	Mean Execution Time (ms)
1	6	5	439960
2	4	5	264080
3	2	5	83230

Figure 2: Plot of Query Execution Measures



Simple calculations show that by offloading the inactive data from the fact tables and repeating the same queries, we gain an average 368.04% improvement in query speed across the queries. The query speeds on the un-indexed archive tables in this configuration are 72.23% slower on average than the original queries on the full (indexed) DW tables. The vast performance improvement on the high-value data queries easily justifies the smaller hit taken in performance for data that is infrequently queried and holds little business value. Opportunities to tune Hive performance also exist to mitigate some or all of the performance loss on the archive [4]. Additionally, it should be noted that in real-world applications, some of the performance loss with respect to the Hive archive might be mitigated by the distributed nature of the stored archive. That is, the distance between the origin of the user query and the actual physical location of the node containing stored archive data may be significantly shorter than the distance to the centralized DW storage thus allowing for quicker query performance for a number of geographically diversified business intelligence users.

Given that the real cost of CPU time is calculable [21], we can calculate the estimated annual performance-based savings for projects that follow this process as well. This formula is dependent on variable factors such as frequency of queries in both data systems, query execution time in both data systems, number of offloads per year, etc. This annual performance cost difference (a savings if the difference is negative) is given by the formula:

Equation 3: Formula for Annual Query Performance Cost Difference

$$\sum_{o=1}^{O}\left(\sum_{q=1}^{Q}\left(\sum_{n=1}^{N}\left(\frac{\Delta R_{n q o}}{T}\times C\right)+\sum_{m=1}^{M}\left(\frac{\Delta H_{m q o}}{T}\times K\right)\right)\right)$$

***GIVEN*:**
T = 3,600,000 ms (in 1 hour)

***WHERE*:**
N is the number of executions per offload period of the query on the RDBMS DW
M is the number of executions per offload period of the query on the Hive archive
Q is the number of unique queries executed within the offload period
O is the number of offloads per year
ΔR is change in execution time (in ms) of the DW query post-offload
ΔH is change in execution time (in ms) of the Hive query post-offload
C is the cost per CPU hour for the RDBMS server
K is the cost per CPU hour for the Hadoop cluster

CONCLUSIONS

Given the results of the performance and storage benchmark experimentation on the implemented offload solution defined by this work's process, we conclude that:

1. The process defined in this paper has been proven adequate for use in successfully implementing an ETL solution for offloading data from the DW to Hadoop.
2. These results prove the efficacy of the offloading process as a viable solution for complimenting the traditional data warehouse to realize cost and performance gains.
3. These results are suggestive of the fact that big data systems, such as Hadoop, offer value-added use-cases as compliments to traditional Data Warehouses.
4. The successful software implementation of the offload approach outlined in this paper suggests that this process can be fully automated using the technologies demonstrated.
5. Hadoop is more than capable of acting as an active archive for offloaded DW data as it has been shown to be cost-effective and shown to offer competitive performance to the RDBMS-based DW with comparable hardware.

Future Direction

This initial exploration of only one specific use case of introducing Hadoop as a compliment to the traditional data warehouse environment gives way to numerous future research directives. These directives can take on the form of additional research regarding the data offload use case presented here or they can take on the form of investigating other hybrid data warehouse / big data system use cases.

With regard to further investigating the DW data offload to Hadoop process, it will be important to conduct similar experiments on a number of different hardware and software configurations, a number of different degrees of distribution, and a number of different data warehouse constructs with varying schemas and sizes. This will help to define the scalability and modularity of this solution. Furthermore, given the repeatable nature of the offload process and the governing rule sets that exist for translating between SQL dialects and HiveQL, it is completely feasible that research into developing a fully-automated ETL solution could be conducted. This full-automation could include implementing advanced techniques in the data warehouse for self-identifying inactive data. The RDBMS metadata can be used to look up how often data is queried and if the data meets a specified infrequency threshold, then the automated process could flag this data as dormant (thus marking it ready for offload). The automated offloader could then parse the database tables for the offload flags and launch the appropriate ETL modules to migrate the data to the active archive. Other avenues of future research with regard to the offload use-case would be investigation into securing the big data active archive and also applying this offload process to DW modeling schemas other than dimensional such as Data Vault, Anchor Modeling, Spider Schema, and Normal Form.

As the offload process can be considered a first step in a long line of DW optimization efforts using a hybrid traditional DW and big data architecture, it would be prudent to explore other complimentary use cases where cost-savings and performance-enhancements can be realized. This exploration can begin with the DW optimization use cases identified in the Introduction section of this paper. Special attention should be given to the offloading of ETL batch processing to Hadoop as it is a logical next step in optimizing the DW by letting Hadoop and MapReduce tackle the resource-heavy ETL operations [10].

REFERENCES

1. Baer, T. (2014). Hadoop: Extending your data warehouse. White Paper.
2. Balasubramanyan, A. (2014). Data warehouse augmentation, Part 3: Use big data technology for an active archive. IBM Technical Library.
3. Bandugula, N. (2015). DCM 6.1 Using Hadoop to lower the cost of data warehousing: The paradigm ahift underway. Data Center World Global Conference.
4. Bradley, C., Hollinshead, R., Kraus, S., Lefler, J., & Taheri, R. (2013). Data modeling considerations in Hadoop and Hive.
5. Capriolo, E., Wampler, D., & Rutherglen, J. (2012). *Programming hive*. Sebastopol, CA: O'Reilly & Associates.
6. Clegg, D. (2015). Evolving data warehouse and BI architectures: The big data challenge. *Business Intelligence Journal, 20*(1), 19-24.
7. Golfarelli M., & Rizzi S. (2009). Data warehouse design. McGraw Hill.

8. Hargraves, M. (2015). Spider schema data model." spider-schema.info.
9. Informatica Corporation. (2014). Data Warehouse Optimization with Hadoop. White Paper.
10. Intel Corporation. (2013). Extract, transform, and load big data with Apache Hadoop. White Paper.
11. Jovanovic, V., & Bojicic, I. (2012). Conceptual data vault model. Proceedings of the Southern Association for Information Systems Conference.
12. Kimball, R. (2011). The evolving role of the enterprise data warehouse in the era of big data analytics. Kimball Group White Paper.
13. Kimball, R., & Ross, M. (2013). The data warehouse toolkit. John Wiley and Son.
14. Lindstedt, D. (2011). Super charge your data warehouse. CreateSpace Independent Publishing Platform.
15. Lopez, K. J. (2014). The modern data warehouse--How big data impacts analytics architecture. *Business Intelligence Journal, 19*(3), 8-15.
16. Merrill, D. (2014). Storage economics: 4 principles for reducing total cost of ownership. White Paper.
17. Rönnbäck, L., Regardt, O., Bergholtz, M., Johannesson, P., & Wohed, P. (2010). Editorial: Anchor modeling: Agile information modeling in evolving data environments. *Data & Knowledge Engineering, 69*(12). Special issue on 28th International Conference on Conceptual Modeling (ER 2009), 1229-1253.
18. Russom, P. (2014). Evolving data warehouse architectures in the age of big data, TDWI Best Practices Report, 29–34.
19. Sammer, E. (2012). *Hadoop operations*. Sebastopol, CA: O'Reilly.
20. Ting, K., & Jarcec, J. (2013). Cecho. *Apache Sqoop Cookbook*. Sebastopol, CA: O'Reilly Media.
21. Walker, E. The real cost of a CPU hour. *IEEE Computer Society, 42*(1), 35-41.