Efficiently Querying Archived Data Using Hadoop

Rajeev Gupta, Himanshu Gupta, Ullas Nambiar, Mukesh Mohania
IBM Research, India
+91-11-41292100
{grajeev, higupta8, ubnambiar, mkmukesh} @in.ibm.com

ABSTRACT
The need to analyze structured data for various business intelligence applications such as customer churn analysis, social network analysis, telecom network monitoring etc., is well known. However, the potential size to which such data will scale in future will make solutions that revolve around data warehouses hard to scale. As data sizes grow the movement of data from the warehouse to archives becomes more frequent. Current file based archive models make the archived data unusable for any type of insight extraction. In this paper, we present an active archival solution for data warehouses that makes use of Hadoop distributed file system (HDFS) to store the data in an always available and cost-effective manner. We investigate various structured data storage schemes within HDFS and empirical evaluations show that a combination of *Universal scheme model* and *column store* is best suited for the active archival solution.

Categories and Subject Descriptors
H.2.7 [Database Administration]: Data warehouse and repository

General Terms
Algorithms, Measurement, Performance, Design.

Keywords
Data warehouse, data archival, Hadoop, query, response time.

1. INTRODUCTION
Many industries, such as telecom, health care, retail, pharmaceutical, financial services, etc., are generating large amounts of data. Telecom companies are enhancing their portfolio of services while number of subscribers are increasing at the rate of 21% [8]. Call data records (CDRs) generated in telecom network of an Indian telecom company often range between 500 million and 2 billion a day just for voice calls. Gaining critical business insights by querying and analyzing such massive amounts of data is becoming the need of the hour. One option is to archive the data which is not frequently used. Traditional archiving saves data to a flat file. Whenever the archived data needs querying (e.g., a telecom company querying about one particular person who is no longer a customer) one needs to restore the entire archive in a database and then query the restored data. This process requires lots of time and capacity, making the querying over the archived data an infrequent activity. Various business intelligence applications such as customer churn analysis, social network monitoring, etc. may gain from the older data (probably archived) for better insight. In this paper, we propose an archival solution such that the archived data is kept on scalable low-cost storage but it can be queried easily. *Active Archiving* is the process of removing a precise set of infrequently used reference data from an overloaded relational database and keeping it active in an archive where it can be easily and quickly retrieved when needed. Specifically, we present a *Hadoop* [1] based active archive solution that provides low cost, scalable, fault-tolerance and quick retrieval features to data management.

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.

1.1 Contributions
We describe a MapReduce cluster based, scalable active archive platform that can seamlessly co-exist with existing data warehouse eco-system. We compare ways for retrieving structured data stored in various storage mechanisms. Using the results of the data retrieval performance evaluation, we design the architecture of our active archival solution. Our solution mitigates the need for any data migration - often a time consuming and lossy operation, by providing integrated and seamless querying capability over both the live data stored in the warehouse and the archived data stored in the Hadoop cluster. In our solution, we show how schema of the archived data should be changed to reduce the storage requirements and the query response time compared to the techniques proposed in literature.

Outline: In Section 2 we describe the Hadoop and its related technologies. In Section 3, we compare performances of various data access technologies to show support for structured data processing in Hadoop. In Section 4 we present various schema options for the archived data which can be used for storage-performance tradeoff. In Section 5, we use representative TPC-DS queries to measure performance of various archival data storage schemes to show that proposed combination of *universal schema* with *column store* is best suited for the active archival solution. The paper concludes in Section 6.

2. HADOOP TECHNOLOGIES
MapReduce framework developed by Google has been identified as a fit platform for data analysis at the Petabyte scale. The MapReduce framework provides simple model to write distributed programs to be run over a large number of (cheap) machines.

Hadoop is the most popular open source implementation of MapReduce framework. It is used for writing applications which process vast amount of data in parallel on large clusters of hardware in a fault-tolerant manner. In Hadoop, data is stored on Hadoop Distributed File System (HDFS) which is a massively distributed file system designed to run on cheap commodity hardware.

Hive [2] is a data-warehousing tool developed at Facebook which puts relational structure on the data and gives capability to query and analyze large amount of data stored in HDFS. Hive defines a simple SQL like query language called HQL (Hive query language). Hive uses database schema to convert HQL queries into corresponding MapReduce jobs which are executed on the underlying Hadoop system. Hive is mainly used for log processing, text mining, document indexing, etc.

Jaql [4] is query language for processing structured and semi-structured data written using Java Script Object Notification (JSON) [3] data model. In JSON data model, data is represented as an array of objects. An object contains a series of {*name: value*} pairs where the value can be atomic, array or nested type. Jaql is compiled into a series of MapReduce tasks which can be executed over a Hadoop system. Jaql has some of the best features of SQL and XQuery making it an easy to use yet powerful language to query JSON data.

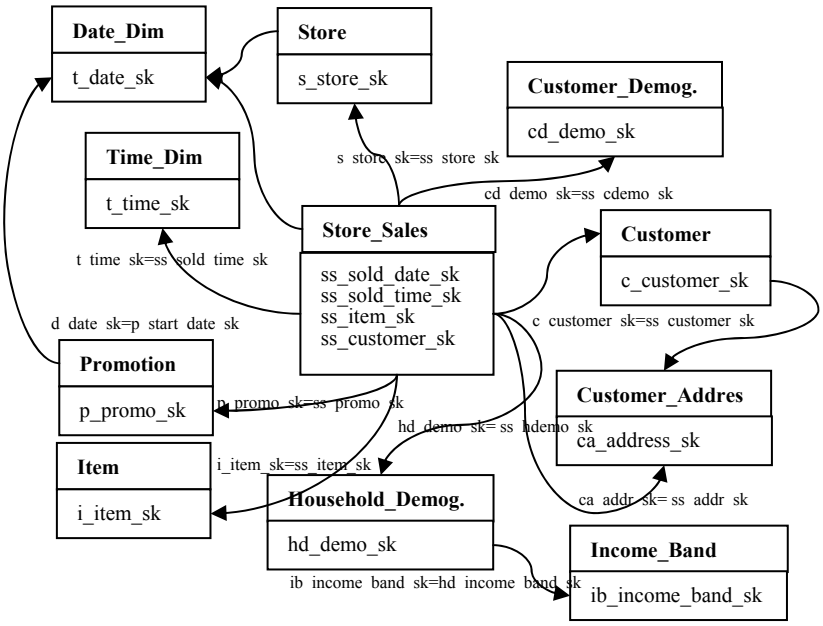


Figure 1: Schema of the TPC-DS data warehouse

3. STRUCTURED DATA MANAGEMENT

To decide on the solution architecture to be used for Hadoop based archival solution, we performed various experiments using queries that reflect the reporting needs of the business analysts and are likely to be issued over the data stored and/or the models (customer profiles, loyalty, churn likelihood, segmentation etc) mined from them. Here are details of our evaluation:

Evaluation setup: The analytics platform was setup over a 20 node Hadoop cluster built using Blade Servers with four 3 GHz Xeon processors having 8GB memory and 200 GB SATA drives. These machines ran Red Hat Linux 5.2. The software stack comprised of Hadoop 0.18.3 with HDFS, Jaql 0.4 and Hive.

Data: The dataset used for the evaluation was the one given by the TPC-DS [5] benchmark. The benchmark models a decision support system of a retail product supplier and includes typical queries and data generation utility. We selected the star schema representing fact and dimension tables used for storing line-items of sales made through *store_sales* channel. Figure 1 shows a simplified version of the data schema. It has one fact table (*store_sales*) and 10 dimension tables. We used the data generator provided as part of the benchmark to create the sample data.

Queries: There are 37 queries specified in the TPC-DS benchmark that focus on extracting data over the selected schema. We selected three representative queries from them for performance comparison purposes. Table 1 gives the details of the selected queries. Query Q_1 is a simple selection query which is likely to output large data as result. Q_2 is an aggregation query returning very small size result. Q_3 is more complex with more Joins across fact and dimension tables.

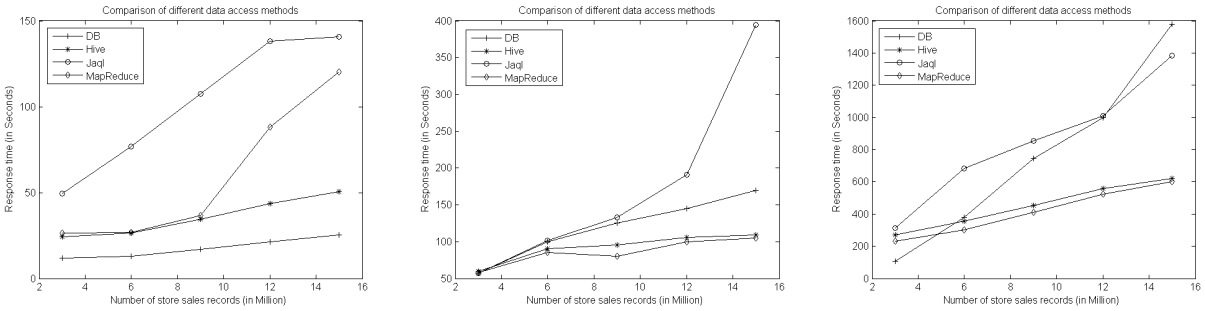
Data Access Techniques: We evaluated the performance of above mentioned queries using the four structured data access mechanisms:

1) *SQL query over RDBMS*: We loaded the test data on a IBM DB2 v9 running on Windows 2003 server with 4 GB RAM to get a benchmark value for performance. Our implementation is not comparable to the shared-nothing distributed data warehouse usually implemented in a typical enterprise. Nonetheless our motivation was to have a representative baseline.

2) *Hive*: Queries were executed on TPC-DS tables after being hand converted to HQL.

Table 1: Queries used for Data Retrieval Comparison

Query	Description	Characteristics	O/P size (*i/p*=3M records)
Q_1	List all sales with the sales price between two given values	Select-project on 1 table with large o/p size	401287
Q_2	Revenue for a given manager for all items	An aggregation with a join	11
Q_3	For each store, compare the weekly sales for two given years	An aggregation and multiple joins and sub-queries	372



(a) Query Q_1 (b) Query Q_2 (c) Query Q_3

Figure 2: Data access method comparison for various queries

3) Jaql: In this setup data was converted into JSON [3] format and written into HDFS.

4) *MapReduce programs*: In this case we wrote tailored MapReduce programs for computing the answers for each of the queries. This was added to quantify the effects of high level query language.

Figure 2 shows the performance results of the four data access techniques. The evaluations were conducted by varying the *store_sales* data warehouse size from 3 million to 15 million records. For Q_1, relational database works best among the four options. Since Q_1 is a simple selection query, this can be explained with the existence of corresponding indices on the database. Raw MapReduce and Jaql give bad performance in this case. The bad performance of MapReduce can be attributed to large output size of Q_1 which leads to lots of data exchange between nodes. Since Map produces large number of records, each Map has to write large amount of data to the disk. Hive performed better than the raw MapReduce as Hive does a better job of optimizing performance using suitable numbers of Map and Reduce jobs. For example, for MapReduce number of maps was 9 whereas for Hive it was 35. For other queries implementing raw MapReduce was very cumbersome with each query requiring its custom implementation. Thus, programmers writing raw MapReduce jobs to extract data, and optimizing them for performance, might end-up doing a very bad job. For these queries Hive and MapReduce perform better than SQL and Jaql. This shows that the cases where aggregation is required over partition-able attributes, MapReduce and Hive perform better compared to database access using SQL. Jaql is designed to query the semi-structured data in JSON format. Hive's engine is designed to optimize simple SQL like queries compared to the more generic engine of Jaql which is more suitable for semi-structured data processing. From this evaluation, we can conclude that asking developers to provide custom data extraction routines to answer this class of queries would be an inefficient way to go. Hence, for the active archive solution proposed in the next section, we use Hive for data extraction.

4. DATA SCHEMA OPTIONS

We are convinced about the value of building a cost-effective and scalable active archive solution over Hadoop. However, as stated earlier, our needs are quite different than most existing implementations of Hadoop. Specifically, our domain of interest is structured data generated by the transactions in an enterprise. The data will move out of the enterprise data warehouse and into our active archive once the transaction moves out of the purview of the warehouses window of storage. We want to make the system easy to access and use for business users who are used to existing data management tools provided over the warehouse. A critical aspect of providing seamless access to the active reference data is to maintain the Hadoop archive schema similar to that in the relational data warehouse. Data warehouse schemas are normalized for saving space and usually are star shaped with a fact table connected to many dimension tables. However we archive the structured data on the HDFS platform and the HDFS file system does not have any support for indexing. Specifically, the MapReduce paradigm of chopping a task into several smaller tasks with each getting a subset of independent data to deal with makes it harder to use indexes. Due to these reasons join performance suffers on Hadoop. We consider the following changes in the data organization:

1) **Universal schema**: While storing the archive data in the HDFS storage, we *de-normalize* or flatten the data warehouse (star) schema and create a single table – *a universal relation* [6]. Essentially, this amounts to converting the *join queries* over a star schema into simple *select project* queries over the universal schema. However, one could argue that doing so for large sized schemas will blow-up the data size as well the amount of data read and processed by the *mappers* and *reducers* while answering queries. We handle that using *column store*.

2) **Column store**: To reduce the amount of (*universal relation*) data that is required to be read by the Hadoop we take recourse to the idea that a typical query on the warehouse accesses only few columns from the fact and dimension tables. So if we can ensure that only the columns that are needed to compute the query answer are read, it will significantly reduce the data read and query evaluation times [7]. This can be achieved by organizing the data with every column stored in a separate table.

5. PERFORMANCE OF ARCHIVE DATA QUERYING

In this section we present evaluation results showing the efficiency and effectiveness of our approach for providing an active archival solution over Hadoop. The active archival platform was setup over the same 20 node cluster that we used for the performance study in Section 3.1. Same datasets and queries were used for this evaluation as well. We ran the queries on these four data storage and processing schemes: (a) *Star* schema of TPC-DS schema given by Figure 1; (b) *Universal* schema as

explained in the previous section; (c) *UCStore* which combines universal schema with column storage; and (d) *SCStore* which combines star schema with column storage technique. In this section, first we measure the storage space required for these 4 schemes for varying number of *store_sales* records and then we compare the performance of these schemes to show that our proposed scheme performs better compared to the *star* schema for the archive data storage and processing.

5.1 Storage Space Comparison

Figure 3 gives the space requirements for all the above mentioned schemes as number of *store_sales records* are varied between 3 and 15 million. As expected *Star* schemes are more efficient

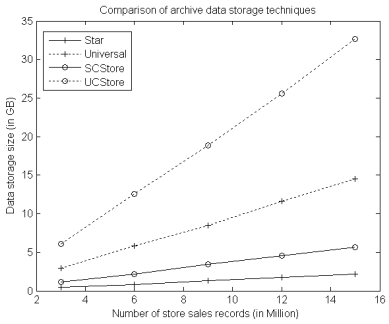


Figure 3: Data storage requirements for various schemes

compared to their corresponding *Universal* schemes from space utilization point of view. For the TPC-DS data we used, all the schemes with *Universal* schema requires approximately 4 times space compared to the *Star* schema. The *UCStore* schema consumes almost double the space compared to *Universal*. This happens because in the column store we store primary key with each of the columns so that Hive can perform join on that.

5.2 Query Performance of Different Schemes

Figures 4 (a)-(c) show the performance of four storage schemes described above for 3 queries given in Table 1. Continuous lines are used to represent the *star* schema based schemes whereas dotted lines are used to represent the *universal* schema based schemes. For queries Q1 and Q2 *Star* performs better than other schemes. But for Q3 *UCStore* performs better as we increase the data size. It should be noted that using *column store* we reduce the read cost but increase the join cost, thus depending on number of records, *Universal* can perform better or worse compared to *UCStore*. In general, for larger data sizes (when read cost dominates) *UCStore* performs better compared to *Universal*. Thus, our experimental evaluation shows that as data size increases, *UCStore* scheme scales well and achieves the query response times outperforming the *star schema*. This comes at a cost of increased storage space requirement approximately by a factor of 4 which in our opinion is a good trade-off.

6. CONCLUSION

In this paper we present an active archive solution that uses the distributed storage capability provided by HDFS and parallel computing support of Hadoop to store and retrieve massive amounts of data. Our solution is designed to seamlessly co-exist with the data warehouse implementation existing with the business. Our approach is domain independent and can cater to all types of businesses, be it the large enterprise with data centers or the small business that can only use a few machines for archiving.

7. REFERENCES

[1] Apache Foundation. Hadoop. http://hadoop.apache.org/core/.

[2] Hive- Hadoop wiki. http://wiki.apache.org/hadoop/Hive

[3] JSON. http://www.json.org

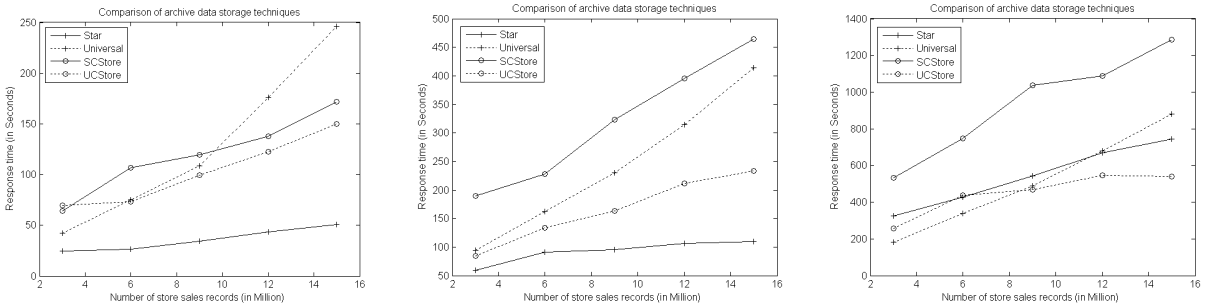
[4] Jaql Project hosting. http://code.google.com/p/jaql/

[5] M. Poess and R.O. Nambiar and D. Walrath. Why You Should Run TPC-DS: A Workload Analysis. *In Proceedings of VLDB*, 2007.

[6] M. Vardi. The Universal-Relation Data Model for Logical Independence. *IEEE Software*, Vol.5, No.2, 1988.

[7] M. Stonebraker et al. C-STORE: A Column-oriented DBMS. *In Proceedings of VLDB*, 2005.

[8] http://en.wikipedia.org/wiki/Communications_in_India



(a) Query Q_1 **(b) Query Q_2** **(c) Query Q_3**

Figure 4: Performance of different active archival schemes