Columnar NoSQL Star Schema Benchmark

Khaled Dehdouh, Omar Boussaid, and Fadila Bentayeb

ERIC Laboratory/ University of Lyon 2
5 avenue Pierre Mendes-France 69676 Bron Cedex, France
{Khaled.Dehdouh,Omar.Boussaid,Fadila.Bentayeb}@univ-lyon2.fr

Abstract. Benchmarking data warehouses is a means to evaluate the performance of systems and the impacts of different technical choices. Developed on relational models which have been for a few years the most used to support classical data warehousing applications such as Star Schema Benchmark (SSB). SSB is designed to measure performance of database products when executing star schema queries. As the volume of data keeps growing, the types of data generated by applications become richer than before. As a result, traditional relational databases are challenged to manage big data. Many IT companies attempt to manage big data challenges using a NoSQL (*Not only SQL*) database, and may use a distributed computing system. NoSQL databases are known to be non-relational, horizontally scalable, distributed. We present in this paper a new benchmark for columnar NoSQL data warehouse, namely CNSSB (Columnar NoSQL Star Schema Benchmark). CNSSB is derived from SSB and allows generating synthetic data and queries set to evaluate column-oriented NoSQL data warehouse. We have implemented CNSSB under HBase column-oriented database management system (DBMS), and apply its charge of queries to evaluate performance between two SQL skins, Phoenix and HQL (*Hive Query Language*). That allowed us to observe a better performance of Phoenix compared to HQL.

Keywords: Data warehouses, columnar databases, decisional benchmark.

1 Introduction

The benchmark databases like TPC-H, TPC-D and SSB are developed on relational model [1] which respect the ACID properties (Atomicity, Consistency, Isolation, Durability) [2] and that have been the most used in recent years to support classical data warehousing applications. Indeed, DBMS relational model are originally designed to support transactional operations then adapted to implement data warehouse [3]. However, traditional relational storage models have shown their limitations in terms of storing and managing big data [4]. The solution appears to be from NoSQL databases that are known to be non-relational, horizontally scalable, distributed and, for the most part, open source. NoSQL databases have sacrificed the ACID properties in favor of system performance and data availability to allow data warehouse architecture to be deployed in the cloud and a high scalability whilst delivering high

Y. Ait Ameur et al. (Eds.): MEDI 2014, LNCS 8748, pp. 281–288, 2014.

performance [5]. In this context, a column-oriented NoSQL database system is a storage model which is highly adapted to data warehouses and online analysis [6][7].

We present in this paper a new benchmark for columnar NoSQL data warehouse, namely CNSSB (Columnar NoSQL Star Schema Benchmark). CNSSB flattened the SSB and denormalizes the fact table to avoid using the join operation between tables, which is prohibitively slow and therefore, typically not supported by the column oriented NoSQL databases. The attributes belonging to the same dimension are grouped into a structure called the column family (CF). This helps to differentiate the attributes representing the dimensions and those representing measures. CNSSB is column-oriented NoSQL data warehouse, and allows generating synthetic data and queries set to evaluate columnar NoSQL DBMS performance. We have implemented CNSSB under HBase[1] column-oriented database management system (DBMS), and applied its charge of queries to evaluate performance between two SQL skins, Phoenix[2] and HQL[3]. That allowed us to observe a better performance of Phoenix compared to HQL. The rest of this paper is organized as follows. Section 2 gives the state of the art on decisional benchmark for evaluating the performances of data warehouses. Section 3 presents an overview of columnar NoSQL databases. Section 4 contains a detailed description of the Columnar NoSQL Star Schema Benchmark. In section 5 we implemented our CNSSB, and conducted experiments in section 6. Finally, we conclude this paper in section 7.

2 State of the Art

The benchmark edited by Transaction Processing Performance Council (TPC) is the most used to evaluate various DBMS performance, and is the efficient means for enterprise to invest in the new systems. TPC-C and TPC-D are the two main benchmarks of TPC to be published, the first is assigned to transactional system (OLTP) and the second is assigned to decisional system (OLAP). TPC-D benchmark [8] appeared in the mid-90s, and exploits a classical product-order-supplier which is relational database schema. TPC-D benchmark was replaced by TPC-H and TPC-R [9] the use of which is different. TPC-H is for ad hoc querying and queries are not known in advance, while TPC-R is for reporting and queries are known in advance. TPC-H and TPC-R are succeeded by TPC-DS, which is more normalized, and represents snowflake model of data warehouse. On the other hand, we can also cite SSB [1] which is a more denormalized model of TPC benchmarks represents a star data warehouse and manages LINEORDER (fact table) according to dimensions, PART, SUPPLIER, CUSTOMER and DATE. All these benchmarks were developed in ROLAP (Relational OLAP) environment, and cannot be implemented into NoSQL DBMS, which is based on non-relational environment.

[1] http://hbase.apache.org
[2] http://www.orzota.com/sql-for-hbase
[3] http://hive.apache.org/

3 Columnar NoSQL Database

Columnar databases store data column by column, and it helps in compressing the data greatly, which makes the columnar operations like aggregate functions very fast. This storage technique makes it possible to store values belonging to the same column into a single disk block, and only allows access to columns necessary for the decisional query execution [6]. This benefit can be efficiently exploited in the case of data warehouses and OLAP analysis. Columnar NoSQL database have non-relational data representation logic. The main benefits are their performance and capacity to process big data [10]. Compared to relational databases, the join operation between tables is prohibitively slow (thus, typically not supported) and referential integrity constraint does not exist, consequently column-oriented NoSQL database suggests grouping columns into one table. We can cite HBase and Cassandra[4] DBMS which are Apache project originally developed by Facebook, and BigTable[5] by Google.

However, native (*Application Programming Interface* - API) interfaces of the column-oriented NoSQL DBMS require several, often complex, lines to be written for using data. In order to simplify the handling of data by users, SQL (API) interfaces have been developed and adapted to column-oriented storage. This has led to a significant reduction in the quantity of code that needs to be written. Moreover, the SQL interface offers the option of making improvements to boost the performance of column-oriented NoSQL DBMS. CQL (*Cassandra Query Language*) and Phoenix for Cassandra and HBase respectively, are two examples. These SQL interfaces offer the best way of combining the features of data handling to express a selection of data and apply predicates.

4 Columnar NoSQL Star Schema Benchmark

Columnar NoSQL Star Schema Benchmark (CNSSB) is designed to measure performance of columnar NoSQL data warehouse, and consists of one table with several columns. Unlike the relational model, CNSSB flattened the SSB. The attributes belonging to the same dimension are grouped into a structure called the column family (CF). This helps to differentiate the attributes representing the dimensions and those representing measures. CNSSB is derived from SSB, and manages LINEORDER (fact table) according to dimensions, PART, SUPPLIER, CUSTOMER and DATE. LINEORDER is denormalized and dimension attributes are now grouped in column families. Thus every dimension is represented by column family, and encapsulated into fact table. CNSSB allows generating synthetic data and queries set to evaluate columnar NoSQL DBMS performance.

4.1 Data Model

CNSSB is derived from SSB, and manages LINEORDER (fact table) according to dimensions, PART, SUPPLIER, CUSTOMER and DATE. CNSSB is the result of a

[4] http://cassandra.apache.org
[5] http://fr.wikipedia.org/wiki/BigTable

denormalizing process of SSB tables into a single table. We have denormalized fact table LINEORDER, and combined dimension tables PART, SUPPLIER, CUSTOMER and DATE into a single LINEORDER table. This denormalization is standard in warehousing [3]. This simplifies the schema considerably; both for writing queries and computing queries. Thus, queries do not have to perform the join and users writing queries against the schema do not have to express the join in their queries. However, to give a better managing for CNSSB as warehouse, we grouped attribute dimensions into column families. Consequently, CNSSB comprises the following column families CF_CUSTOMER, CF_SUPPLIER, CF_PART and CF_DATE, as dimensions. The denormalization process requires some modification in particularly at key referencing (primary at level dimensions and foreign at the fact table). We merged foreign keys in the fact table Custkey, Suppkey, Partkey and Datekey which were used to ensure correspondence between the fact table and dimension, with the attributes of their dimensions. Thus, Custkey, Suppkey, Partkey and Datekey do not be considered as keys since they cannot uniquely identify all tuple values of the fact table. We decided dropping the Datekey of CF_DATE, which is a surrogate key [3], this attribute would no longer have a role in this model. The complete schema is depicted in figure 1.

LINEORDER	
ORDERKEY	
LINENUMBER	
FC_CUSTOMER	CUSTKEY
	NAME
	ADDRESS
	CITY
	NATION
	REGION
	PHONE
	MKYSEGMENT
FC_SUPPLIER	SUPPKEY
	NAME
	ADDRESS
	CITY
	NATION
	REGION
	PHONE
FC_PART	PARTKEY
	NAME
	MFGR
	CATEGOTY
	BRAND1
	COLOR
	TYPE
	SIZE
	CONTAINER

LINEORDER (suite)	
FC_DATE	ORDERDATE
	DAYOFWEEK
	MONTH
	YEAR
	YEARMONTHNUM
	YEARMONTH
	DAYNUMINWEEK
	DAYNUMINMONTH
	DAYNUMINYEAR
	MONTHNUMINYEAR
	WEEKNUMINYEAR
	SELLINGSEASON
	LASTDAYINWEEKFL
	LASTDAYINMONTHFL
	HOLIDAYFL
	WEEKDAYFL
	COMMITDATE
ORDERPRIORITY	
SHIPPRIORITY	
QUANTITY	
EXTENDEDPRICE	
ORDERTOTALPRICE	
DISCOUNT	
REVENUE	
SUPPLYCOST	
TAX	
SHIPMODE	

Fig. 1. CNSSB data model

To summarize, CNSSB consists of one large fact table called LINEORDER which contains 53 columns whose primary key consists of two attributes ORDERKEY and LINENUMBER, 41 attributes grouped into four column families CF_CUSTOMER,

CF_SUPPLIER, CF_PART and CF_DATE each representing an analysis axis. As for the ten other attributes of the fact table, they represent the measurements to analyze.

4.2 Data Population

Like in SSB, data in CNSSB is uniformly distributed, and all column families have selectivity hierarchies, which allows better control and manages the row selectivity of queries. For instance, in CF_CUSTOMER, there are dependency hierarchies REGION, NATION and CITY, noted REGION → NATION → CITY and uniform data distribution, this means that, for example ten different values for CITY, twenty five different values for NATION and five different values for REGION will result in a selectivity of 1/10 for each CITY value, (1/10 * 1/25 = 1/250) for each NATION value and (1/250 * 1/5 = 1/1250) for each REGION value.

4.3 CNSSB Queries

We decided to translate SSB queries into our schema. Our motivation is that SSB and CNSSB shared practically all attributes which are differently structured. Thus, the charge of CNSSB queries consists of thirteen decisional queries divided into four flights.

Flight 1 is composed of three queries that have a restriction on one column family, CF_DATE. They aggregate the (sum (extendedprice*discount)) with selectivity on the discount and quantity attributes, for a year in the first query, month in the second query and week for the third query.

Flight 2 is composed of three queries that perform a roll-up, and have a restriction on two column families, CF_DATE and CF_PART, and aggregate the revenue (sum (revenue)). These queries use the dependency hierarchy BRAND1 → CATEGORY → MFGR of CF_PART, and order the results by YEAR and BRAND1.

Flight 3 is composed of four queries that perform a drill down and roll-up on column families CF_CUSTOMER and CF_SUPPLIER, and aggregate the revenue (sum (revenue)), and use the dependency hierarchy CITY → NATION → REGION of both CF_CUSTOMER and CF_SUPPLIER column families

Flight 4 is composed of three queries that perform a drill down and roll-up on column families CF_PART, CF_CUSTOMER and CF_SUPPLIER, and aggregate the profit (sum (revenue - supplycost)), and use the dependency hierarchies BRAND1 → CATEGORY → MFGR of CF_PART column family and CITY → NATION → REGION of both CF_CUSTOMER and CF_SUPPLIER column families.

5 Implementation

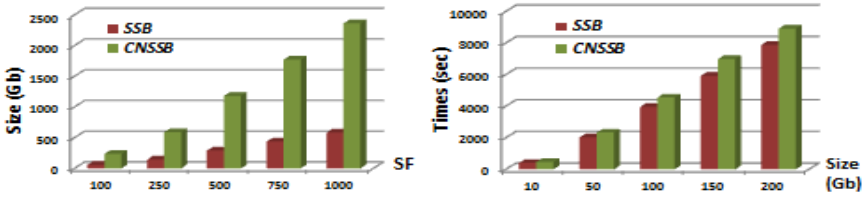
We have modified the data generator (DBGEN) of SSB to generate data according to the model CNSSB, and we have developed queries generator (QGEN) to generate the query set associated to CNSSB model. We integrated DBGEN and QGEN to an API which allows generating data and queries for Columnar NoSQL Star Schema Benchmark. DBGEN has a scale factor (SF) as parameter to define the size of the warehouse; it generates a csv file according to the CNSSB model. QGEN generates thirteen decisional queries for the data warehouse CNSSB, and generates randomly the condition values of the query (*Where Clause*) which define restrictions on column families.

6 Experiments

We conducted two experiments. The first evaluates two aspects in data generation: (1) data generated size according to the scale factor and (2) time required to generation of data. The second experiment uses the CNSSB queries to evaluate performance between two SQL skins, Phoenix and HQL (Hive Query Language) under the oriented NoSQL DBMS columns HBase. Noted that HQL is an SQL interface for Hive that is once integrated in HBase, which allows data stored in HBase to be handled using HQL [11].

6.1 Experiment 1

In this section, we evaluate the data generator of CNSSB compared to SSB, and we are interested in the size of data generated by the scale factor, and we note the different execution time for generating different sizes of data.



(a) Size of data generated by the scale factor

(b) Execution time for generating different sizes of data

Fig. 2. Performance study of the CNSSB compared to the SSB

We observe that the sizes of data generated by the data generator of CNSSB and SSB in the figure (a) are closely related with the scale factor. When a scale factor equals to 100, DBGEN of SSB generates 56.8 GB and for 1000, it generates 580 GB.

According to these results, the ratio between the scale factor and the size of generated data varies from 0.56 to 0.58. On the other hand, when a scale factor equals 100, the DBGEN of CNSSB generates 235 GB and for 1000, it generates 2360 GB, thus, the ratio between the scale factor and the size of generated data is 2.3 on average. Therefore, with the same scale factor, we note that data generator of CNSSB allows to have larger size compared to SSB, and makes it a good tool which can be used towards the simulation of large warehouses NoSQL category.

Moreover, in the figure (b), we observe that the time required to generate data with data generator of CNSSB is higher than SSB, but still it remains reasonable such as represented by linear function.

6.2 Experiment 2

In this experimentation, we use the CNSSB queries to evaluate performance in terms of execution time, between two SQL skins, Phoenix and HQL under the column-oriented NoSQL DBMS HBase in distributed environment. To conduct our experiment, we used a data sample of 1 TB, which is obtained by data generator of CNSSB, and installed a test environment which is a cluster made up of 15 machines (nodes). Each machine has an intel-Core TMi3-3220 CPU@3.30 GHZ processor with 4GB RAM. These machines operate with the operating system Ubuntu-12.10 and are interconnected by a switched Ethernet 100 Mbps in a local area network. One of these 15 machines is configured to perform the role of *Namenode* in the *HDFS* system, the master and the *Zookeeper* of HBase [12]. However, the other machines are configured to be *HDFS DataNodes* and the HBase *RegionServers*. The results we obtained are shown in the following figure.

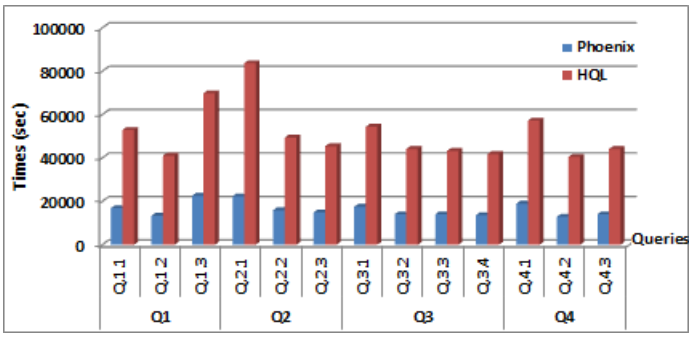


Fig. 3. Execution time of CNSSB queries with Phoenix and HQL

We showed that Phoenix allows better time of decisional queries execution compared to time HQL. Indeed, using the HQL interface requires the migration of HBase data to HIVE. This migration involves exporting data from HBase table to HIVE. This export requires the table to be scanned, which adds time and results in longer execution times. By cons, Phoenix is a skin over HBase. This does not add a new layer; rather it exposes HBase functionality through SQL using an embedded *JDBC*

driver that allows clients to run at native HBase speed. The *JDBC driver* compiles SQL into native HBase calls. Therefore, Phoenix offers the opportunity to benefit from the power of HBase. Given the results, we found that querying data in HBase with phoenix is more efficient compared to HQL.

7 Conclusion

We have presented in this paper CNSSB, which is a new benchmark for columnar NoSQL data warehouse. CNSSB is derived from SSB and allows generating synthetic data and queries set to evaluate column-oriented NoSQL data warehouse. We have used CNSSB under HBase column-oriented database management system (DBMS), and applied its charge of queries to evaluate performance between two SQL skins, Phoenix and HQL. That allowed us to observe a better performance of Phoenix compared to HQL.

References

1. O'Neil, P., O'Neil, B., Chen, X.: The Star Schema Benchmark (SSB) (2009), http://www.cs.umb.edu/~poneil/StarSchemaB.PDF
2. Codd, E.: TA Relational Model of Data for Large Shared Data Banks, pp. 377–387. Association for Computing Machinery (ACM) (1970)
3. Kimball, R., Ross, M.: The Data Warehouse Toolkit, 2nd edn. Welly (2002)
4. Leavitt, N.: Will NoSQL databases live up to their promise?, pp. 12–14. IEEE Computer Society (2010)
5. Pokorny, J.: NoSQL Databases: A Step to Database Scalability in Web Environment. In: International Conference on Management of Data, pp. 278–283. Association for Computing Machinery (ACM) (2011)
6. Matei, G.: Column-Oriented Databases, an Alternative for Analytical Environment. Database Systems Journal, 3–16 (2010)
7. Jerzy, D.: Business Intelligence and NoSQL Databases. Information Systems in Management, 25–37 (2012)
8. Ballinger, C.: TPC-D: Benchmarking for Decision Support. In: The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann (1993)
9. Poess, M., Floyd, C.: New TPC Benchmarks for Decision Support and Web Commerce. SIGMOD Record, 64–71 (2000)
10. Hecht, R., Jablonski, S.: NoSQL Evaluation: A Use Case Oriented Survey. In: Proceedings of the 2011 International Conference on Cloud and Service Computing, pp. 336–341 (2011)
11. Apache Hive (2014), https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration
12. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: ZooKeeper: Wait-free Coordination for Internet-scale Systems. In: Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, pp. 11–24 (2010)