2014 IEEE International Conference on Systems, Man, and Cybernetics
October 5-8, 2014, San Diego, CA, USA

Columnar NoSQL CUBE: Agregation operator for columnar NoSQL data warehouse

Khaled Dehdouh, Fadila Bentayeb, Omar Boussaid, and Nadia Kabachi
ERIC laboratory
University of Lyon 2
5 avenue Pierre Mendes-France
69676 Bron Cedex, France
Email: name.surname@univ-lyon2.fr

***Abstract*—The emergence of large volumes of data imposed by the major players of the web requires new management models and new data storage architectures and treatment able to find information quickly in a large volume of data. The column-oriented NoSQL (Not Only SQL) database provide for big data the most suitable model to the data warehouse and the structure of multidimensional data in OLAP cube form. However, in the absence of OLAP cube computation operators, we propose in this paper, a new aggregation operator called CN-CUBE (Columnar NoSQL CUBE), which allows data cubes to be computed from data warehouses stored in column-oriented NoSQL database management system. We implemented the CN-CUBE operator using the SQL Phoenix interface of HBase DBMS and conducted experiments on a public data warehouse in a distributed environment produced using the Hadoop platform. Thus we have shown that our CN-CUBE operator has OLAP cubes computation times very suitable for NoSQL warehouses.**

I. Introduction

A data warehouse is a database for online analytical processing (OLAP) to aid decision-making [1]. It is often implemented in the relational database management system (RDBMS) [2]. However, in a constantly connected world, data sources produce increasingly massive data, namely *big data*. Traditional relational storage models have shown their limitations in terms of storing and managing big data [3]. Indeed, it is the major players of the web such as Yahoo, Google, Facebook, Twitter and LinkedIn who were the first to point out the limitations of the relational model. They found that relational DBMSs are no longer adapted to the distributed environments that are made necessary by the enormous amount of data. NoSQL database management systems emerged to meet big data storage and processing needs. They were built without ACID properties (Atomicity, Consistency, Isolation and Durability) [4], allowing a data warehouse architecture to be deployed in the cloud and a high scalability whilst delivering high performance [5].

In this article we focus on online analytical processing (OLAP) of big data. OLAP operators use data warehouses to extract data cubes corresponding to analytical contexts to aid decision-making. A data cube is a multidimensional structure which allows the fact (measure) to be represented according to several observation axes (dimensions) [6].

Cube computation produces aggregations that are beyond the limits of the Group by. For example, in the case of calculation of the sum, it computes in a multidimensional way and returns sub-totals and totals for all possible combinations. This involves performance of all aggregations according to all levels of hierarchies of all dimensions. For a cube with three dimensions A, B and C, the performed aggregations relate to the following combinations: (A, B, C), (A, B, ALL), (A, ALL, C), (ALL, B, C), (A, ALL, ALL), (ALL, B, ALL), (ALL, ALL, C), (ALL, ALL, ALL).

In this context, a column-oriented NoSQL database system is a storage model which is highly adapted to data warehouses and online analysis [7][8]. However, column-oriented NoSQL DBMSs unfortunately do not have OLAP operators.

To solve this problem, we propose an aggregation operator, called CN-CUBE (Columnar NoSQL CUBE), for column-oriented NoSQL database management systems. This operator allows OLAP cubes to be computed using column-oriented NoSQL data warehouses. CN-CUBE uses a view; result of an extraction query, based on attributes (dimensions and measures) needed to compute the OLAP cube corresponding to an analysis need. This strategy reduces disk access whilst avoid return to the warehouse data for performing different aggregations. Moreover, in order to perform the aggregations of the OLAP cube, CN-CUBE uses value positions and hash tables to take into account all dimension combinations and extract data that satisfy the predicates of the query. We implemented the CN-CUBE operator and evaluated its performance on a data warehouse created by us within the column-oriented NoSQL DBMS HBase using Hadoop. The HBase DBMS and the Hadoop platform were chosen because of their distributed context which was necessary for storing and analysing big data. The warehouse was populated with real open data describing personal injury road traffic accidents.

The rest of this paper is organised as follows. Section two gives the state of the art on column-oriented NoSQL databases. Section 3 contains a detailed description of the OLAP cube computation process and an introduction to the CN-CUBE operator. In section 4 we implemented our CN-CUBE operator and conducted experiments. Finally, we conclude this paper and present some perspectives on our work in section 5.

II. State of the Art

The column-oriented NoSQL database stores the data of a table column-by-column [9]. The data is modeled in the form of a key/value pair and stored in a distributed file system. Each value is allocated a timestamp by the system for the purposes of data consistency. However, the combination <key, column name, timestamp> represents the coordinates of the value. It is noteworthy that data replication across different machines (nodes) required by the data management in a distributed environment (Eventually consistent), sometimes leads to different versions of the same data during updates and the timestamp associated with each value means that it is the most recent version which will be taken when a query is entered into the database [10].

Furthermore, native (API[1]) interfaces of the column-oriented NoSQL DBMS require several, often complex, lines to be written in order to use the 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 DBMSs. *CQL*[2] (Cassandra Query Language) and *Phoenix*[3] for Cassandra[4] (the NoSQL database from the Apache Foundation, originally developed by Facebook) and HBase (the NoSQL database used by the Hadoop project) 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 filters [11].

However, although everyone agrees that the column-store approach is well suited to multidimensional data and as such to computing data cubes, column-oriented NoSQL DBMSs unfortunately do not have OLAP operators.

III. CN-CUBE OPERATOR

In this section, we present our OLAP operator for the cube computation in column-oriented NoSQL data warehouses. The technique used by the CN-CUBE operator we propose uses a view based on attributes (dimensions and measures) needed to compute the OLAP cube. This view is the result of a data extraction query which satisfies all the predicates of the query, grouped according to columns which represent dimensions.

It is noteworthy that in a cluster of computers, replicated data may not have the same version on different nodes (eventually consistent). The system may recover a value of columns with two versions (a key with two different values) but the timestamp associated with each value means that it is the most recent version which is taken. The result is therefore a relation R made up of columns representing dimensions and column(s) representing the measure(s) to be aggregated. This relation is the intermediate result needed to build all parts of the cube. At this stage, the relation R already allows the total aggregation and the aggregation according to all columns representing dimensions to be produced.

In order to obtain the other aggregations that make up the cube, each dimension at the relation R level is hashed with the values it contains to obtain a list of positions. The values of these lists are binary. They may correspond to 1 or 0, with 1 indicating if the hash value exists at this position and 0 if not.

[1]Application Programming Interface

[2]http ://www.datastax.com/docs/1.1/references/cql/index

[3]http ://www.orzota.com/sqlforhbase/

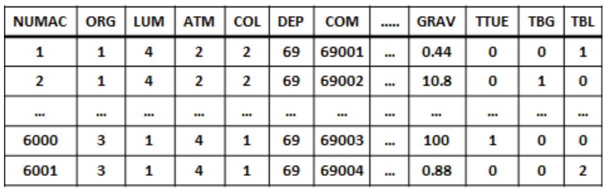
[4]http ://cassandra.apache.org/

The intersection of lists of the positions of the various columns (dimensions) with a *logical AND function* provides a set of lists of positions which represent the positions of dimension values to be combined with the measure column to be aggregated including at different levels of granularity. The grouping of all of these results is the cube.

We will use an example in order to set out the phases of execution for this technique.

A. Support example

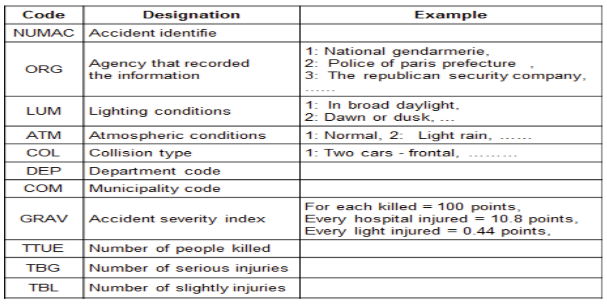
Dataset: In order to validate our approach to computing the OLAP cube using the CN-CUBE operator we decided to use real open data relating to personal injury road traffic accidents. Once implemented in a column-oriented NoSQL DBMS, this dataset allowed us to apply our OLAP cube computation operator. Of course, this amount of data is not typical example of big data, however, it is a real data and simple example which allow us to unrolling our approach. The dataset we used is an extract from the national database of personal injury road traffic accidents, administered by the National Interdepartmental Road Safety Observatory [12]. It brings together essential information collected by enforcement agencies on road traffic accident physical injury analysis forms after each personal injury road traffic accident in mainland France and involving at least one vehicle and with at least one injured person requiring medical care. The information collected in this dataset was reported by the different enforcement agencies (police, gendarmes, fire service etc.) who worked at the accident site. In figure 1 we present an extract from the dataset from this database in CSV format.



NUMAC	ORG	LUM	ATM	COL	DEP	COM	.....	GRAV	TTUE	TBG	TBL
1	1	4	2	2	69	69001	...	0.44	0	0	1
2	1	4	2	2	69	69002	...	10.8	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...
6000	3	1	4	1	69	69003	...	100	1	0	0
6001	3	1	4	1	69	69004	...	0.88	0	0	2

Fig. 1. Dataset of personal injury road traffic accidents.

The coding used is shown in figure 2.



Code	Designation	Example
NUMAC	Accident identifie	
ORG	Agency that recorded the information	1: National gendarmerie, 2: Police of paris prefecture , 3: The republican security company, ...
LUM	Lighting conditions	1: In broad daylight, 2: Dawn or dusk, ...
ATM	Atmospheric conditions	1: Normal, 2: Light rain, ...
COL	Collision type	1: Two cars - frontal, ...
DEP	Department code	
COM	Municipality code	
GRAV	Accident severity index	For each killed = 100 points, Every hospital injured = 10.8 points, Every light injured = 0.44 points,
TTUE	Number of people killed	
TBG	Number of serious injuries	
TBL	Number of slightly injuries	

Fig. 2. Codification.

Queries: In order to evaluate the CN-CUBE operator, which we propose so as to compute the OLAP cube using a column-oriented warehouse, we use four OLAP cube computation queries. These queries gradually involve the number of dimension in cube computations. These queries aggregate the number of road traffic accidents (measure) with a severity index over or equal to 100 (at least one death) that occurred in the RHONE department (69). Aggregations are performed according to different dimensions.

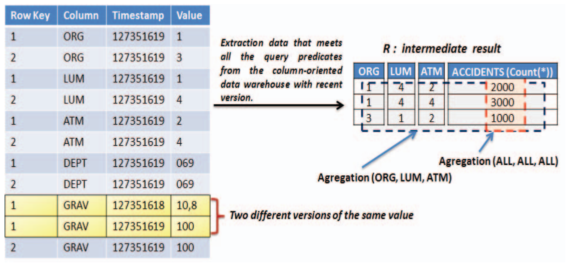
- The first query aggregates the number of accidents and performs the different aggregations in relation to the agency that recorded the information (ORG) and the lighting conditions (LUM) at the time of the accident.
- The second query performs the same aggregations with an additional dimension, atmospheric conditions (ATM).
- The third query adds the collision type (COL) when performing these aggregations.
- In addition to the above dimensions, the fourth query includes the road category (CATR).

B. *CN-CUBE operator execution phases*

The CN-CUBE computes the OLAP cube in three phases. In order to provide more detail on these phases, we illustrate the query 2, cited in section 4. In this example, the number of road traffic accidents with a severity index over or equal to 100 (at least one death) that occurred in the RHONE department (69) is computed in relation to the agency that recorded the information (ORG), the lighting conditions (LUM) and the atmospheric conditions (ATM) at the time of the accident.

First phase: This involves identifying the attributes (dimension and measures) needed to compute the OLAP cube. Data that meets all the predicates is extracted from the column-oriented data warehouse. Only values with the most recent timestamp are taken. The result is a relation R (intermediate result) which will be used to make up all parts of the cube. This strategy means there is no need for CN-CUBE to return to the warehouse data in order to perform aggregations. In our example, this means applying the query conditions (DEP = 69 and GRAV $\geq$ 100) and selecting the number of accidents and attribute values ORG, LUM and ATM. The relation R (intermediate result) is made up of columns ORG, LUM and ATM which represent dimensions and the column ACCIDENTS (number of accidents) which represents the measure to be aggregated.

At this stage of execution, the aggregations relating to the numbers of accidents (ALL, ALL, ALL) and (ORG, LUM, ATM) are performed.



Row Key	Column	Timestamp	Value
1	ORG	127351619	1
2	ORG	127351619	3
1	LUM	127351619	1
2	LUM	127351619	4
1	ATM	127351619	2
2	ATM	127351619	4
1	DEPT	127351619	069
2	DEPT	127351619	069
1	GRAV	127351618	10,8
1	GRAV	127351619	100
2	GRAV	127351619	100

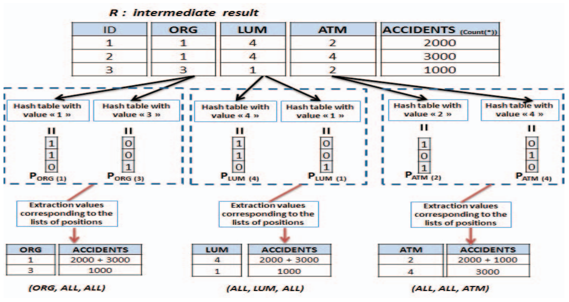
R : intermediate result

ORG	LUM	ATM	ACCIDENTS (Count(*))
1	4	2	2000
1	4	4	3000
3	1	2	1000

Extraction data that meets all the query predicates from the column-oriented data warehouse with recent version.
Agregation (ALL, ALL, ALL)
Agregation (ORG, LUM, ATM)
Two different versions of the same value

Fig. 3. Data extraction and construction of the intermediate result

Second Phase: In this phase, each relation R dimension column is hashed with the values which make it up to produce lists of the positions of these values. Indeed, the ORG dimension is hashed at values 1 and 3, producing two position lists respectively $P_{ORG(1)}$ and $P_{ORG(3)}$. LUM dimension is hashed at values 4 and 1, producing the lists $P_{LUM(4)}$ and $P_{LUM(1)}$. Finally, ATM dimension is hashed at values 2 and 4, producing the lists of the positions $P_{ATM(2)}$ and $P_{ATM(4)}$. At this stage of execution, these position lists allow the ACCIDENTS column (number of accidents) to be aggregated for each dimension separately. Indeed, they produce the aggregations of the following numbers of accidents: (ORG, ALL, ALL), (ALL, LUM, ALL) and (ALL, ALL, ATM).



R : intermediate result

ID	ORG	LUM	ATM	ACCIDENTS (Count(*))
1	1	4	2	2000
2	1	4	4	3000
3	3	1	2	1000

ORG	ACCIDENTS
1	2000 + 3000
3	1000

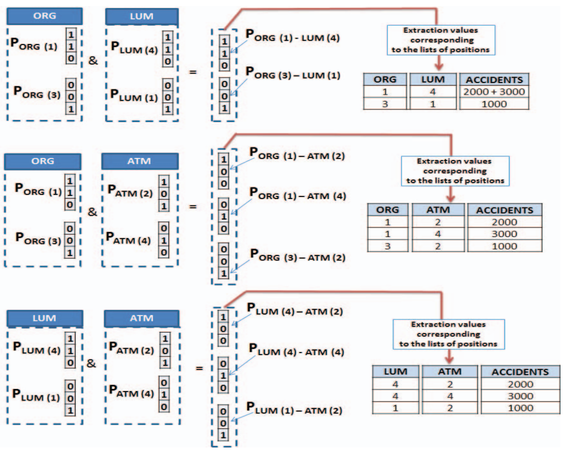
LUM	ACCIDENTS
4	2000 + 3000
1	1000

ATM	ACCIDENTS
2	2000 + 1000
4	3000

Hash table with value « 1 »
Hash table with value « 3 »
Hash table with value « 4 »
Hash table with value « 1 »
Hash table with value « 2 »
Hash table with value « 4 »
P ORG (1)
P ORG (3)
P LUM (4)
P LUM (1)
P ATM (2)
P ATM (4)
Extraction values corresponding to the lists of positions
(ORG, ALL, ALL)
(ALL, LUM, ALL)
(ALL, ALL, ATM)

Fig. 4. Constructing position lists and performing aggregations for each dimension

Third phase: At this phase, the list of dimension positions at R are associated through the *logical AND operator*. This operation identifies the values of dimensions to be combined and the values of the measure to be aggregated which correspond to the different combinations. In this case, in order to identify the various possible combinations between the ORG and LUM dimensions, the list of positions $P_{ORG(1)}$ and $P_{ORG(3)}$ are associated with $P_{LUM(4)}$ and $P_{LUM(1)}$ using the *logical AND operator*. The results are the two lists $P_{ORG(1)-LUM(4)}$ and $P_{ORG(3)-LUM(1)}$. These lists allow the values of the ORG and LUM dimensions and the measure of ACCIDENTS to be aggregated to be extracted. These operations produce the aggregations of the following accidents sums: (ORG, LUM, ALL), (ORG, ALL, ATM) and (ALL, LUM, ATM).



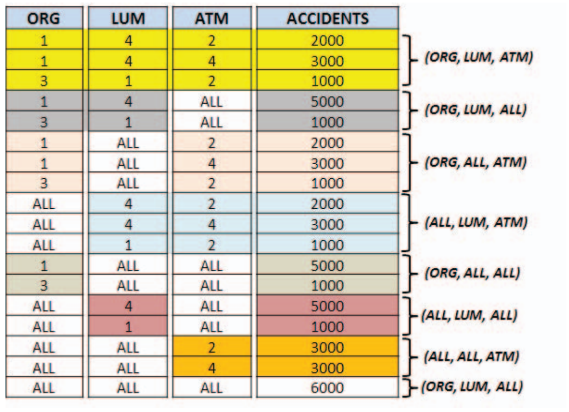
ORG	LUM	ACCIDENTS
1	4	2000 + 3000
3	1	1000

ORG	ATM	ACCIDENTS
1	2	2000
1	4	3000
3	2	1000

LUM	ATM	ACCIDENTS
4	2	2000
4	4	3000
1	2	1000

Fig. 5. Performing aggregations for the different combinations of position lists

As a result of the above, the grouping of sub-results (totals and sub-totals) of the three phases allows the cube shown in the figure below to be computed.



ORG	LUM	ATM	ACCIDENTS	
1	4	2	2000	(ORG, LUM, ATM)
1	4	4	3000	
3	1	2	1000	
1	4	ALL	5000	(ORG, LUM, ALL)
3	1	ALL	1000	
1	ALL	2	2000	(ORG, ALL, ATM)
1	ALL	4	3000	
3	ALL	2	1000	
ALL	4	2	2000	(ALL, LUM, ATM)
ALL	4	4	3000	
ALL	1	2	1000	
1	ALL	ALL	5000	(ORG, ALL, ALL)
3	ALL	ALL	1000	
ALL	4	ALL	5000	(ALL, LUM, ALL)
ALL	1	ALL	1000	
ALL	ALL	2	3000	(ALL, ALL, ATM)
ALL	ALL	4	3000	
ALL	ALL	ALL	6000	(ORG, LUM, ALL)

Fig. 6. OLAP Cube

C. *CN-CUBE operator execution process in a distributed environment*

Big data offers decision-makers the opportunity to analyse fairly broad samples thus giving the results of analysis credibility. However, unusual volumes of data become an issue when faced with the limited capacities of traditional systems.

In order to exploit big data, data warehouses need to opt for a necessarily distributed solution when scaling-up. This solution is based on a data storage architecture distributed over several machines and parallel data processing. Hadoop offers a distributed file system called HDFS [13], designed to store very large volumes of data on several machines and a system for processing distributed data called MapReduce [14]. MapReduce is a massively parallel processing model suitable for processing very large volumes of data. It is based on two main processes, Map and Reduce. The Map function breaks processing down into sub-processing on the different nodes that form the cluster and the results are grouped using the Reduce function.

Integrating a NoSQL database management system such as HBase with Hadoop allows data to be structured and indexed more effectively so that when accessing the data there is no need to perform a full scan on the whole cluster [15].

However, in the three phases which characterize the execution of the CN-CUBE operator, the CN-CUBE operator execution process on a multi-node cluster, trigger two main MapReduce jobs. The first executes the first phase which involves producing the intermediate result needed to compute all parts of the OLAP cube and consequently supply aggregations that can be performed at this level. The result of the first job, becomes the input for the second job which uses the intermediate result to perform the rest of the aggregations that make up the cube (phases 2 and 3). Thus, phases 2 and 3 can only take place after phase 1 is completed. The combined results of the jobs make up the OLAP cube.

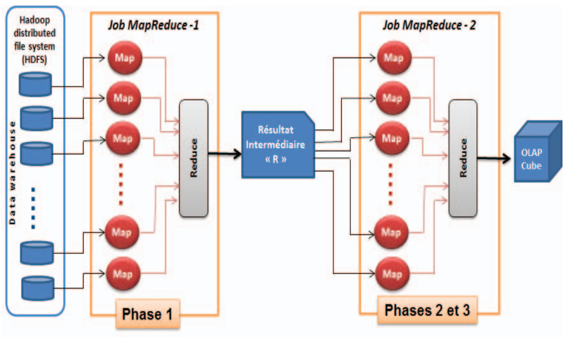


Fig. 7. CN-CUBE operator execution process with MapReduce

IV. Implementation and Experiments

In order to implement the CN-CUBE operator in a column-oriented NoSQL environment using MapReduce, we have put in place a non-relational and distributed storage and processing environment [16]. This environment is based on a private Cloud Computing architecture produced using the Hadoop-1.2.0 and a HBase-0.94.14 DBMS, for managing data in a distributed environment. In order to simplify data handling and boost the performance of the HBase DBMS, we strengthened this configuration with an SQL interface for HBase, called Phoenix-2.2.2. Phoenix-2.2.2 is open source and allows the data handling at the HBase level (scan, put and get) to be combined to express a selection of data and to apply filters.

In order to compute OLAP cubes, we implemented the CN-CUBE operator in the form of a client application based on a cube computation function (UDF: user-defined function), developed using JAVA 1.7 (java SE Runtime Environment) in Netbeans IDE 7.4. This connects to the HBase data server using API Phoenix.

A. *Test Environment*

The test environment 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 Zookeper of HBase [17]. However, the other machines are configured to be HDFS DataNodes and the HBase RegionServers.

Although the private Cloud architecture we used is limited in terms of capacity (number of nodes composing the cluster), it is sufficient to allow us to deploy a non-relational data warehouse with scaling-up and to implement the CN-CUBE operator in a distributed environment.

In order to evaluate the CN-CUBE operator, we integrated into this environment HIVE, a data warehouse software to manage large datasets. Hive provides a SQL-like language called HiveQL to query data [18]. HIVE is not column-oriented but, when integrated in HBase allows the data stored in HBase to be handled using HQL [19]. We decided to use HIVE to compare the performances of the CN-CUBE operator as in version 0.10.0 it incorporates the CUBE operator to compute the OLAP cube [20]. To conduct our experiments we used the dataset from section 4.

B. Experiments

We evaluate the performances of the CN-CUBE operator in terms of OLAP cube computation times. To do this, we conducted two experiments. The first was to evaluate the CN-CUBE operator by gradually increasing the number of dimensions when computing the OLAP cube. The second evaluates the cube computation time when scaling-up.

1) OLAP cube computation on a single-node cluster: The objective of this experiment is to empirically evaluate the CN-CUBE operator when faced with variations in the number of dimensions. For this, we compared the computation times for OLAP cubes with two to five dimensions on a sample of data comprising 60 million records and we executed the four queries from section III-A. Let us recall that these queries generate OLAP cubes with a gradually increasing number of dimensions. The results we obtained are shown in the following figure.

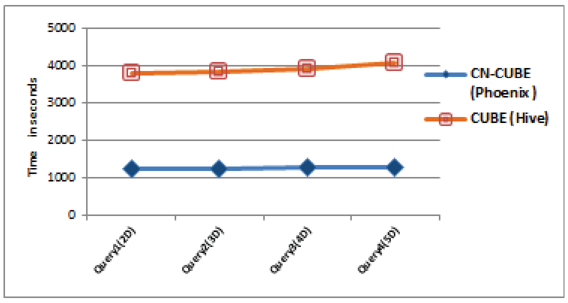


Fig. 8. Comparison of computation times for OLAP cubes with 2, 3, 4 and 5 dimensions between CN-CUBE and CUBE

We observe a slight variation in OLAP cube computation times with the CN-CUBE and CUBE operators when the number of dimensions was increased. However, CN-CUBE shows a better performance than CUBE. Indeed, the CUBE operator records times between 3801 and 4064 seconds, whereas the CN-CUBE operator records times of between 1240 and 1279 seconds.

The drawback of the HIVE CUBE operator lies in the use of data. Indeed, using the HQL interface it 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.

However, the performance of the CN-CUBE operator lies in the use of position lists when computing OLAP cubes. These lists occupy little memory space. They are perfectly suitable for memory-level operations without requesting repeated disk access. Indeed, the technical results of an increase in the number of dimensions when computing the data cube is the creation and handling of bit vectors which represent lists of value positions. This process does not have a heavy impact on memory. Moreover, combinations are produced with lists (bit vectors made up of 1 or 0) and not values and these are only extracted after the position list related to a combination has been built.

The results showed that the CN-CUBE operator computes the OLAP cube up to three times faster than the HIVE CUBE operator. As such, it is of interest to evaluate the behaviour of CN-CUBE when scaled up in a distributed environment.

2) OLAP cube computation on a multi-node cluster (scaling-up): The purpose of this experiment is to expose the CN-CUBE operator to scaling-up. The experiment involved calculating the execution time for query 2 of section III-A which computes a three dimensional OLAP cube with four different configurations, with data samples of different sizes (100 GB, 500 GB and 1 TB). The four configurations are a single-node cluster, a five-node cluster, a ten-node cluster and a fifteen-node cluster. The results we obtained are shown in the following figure:

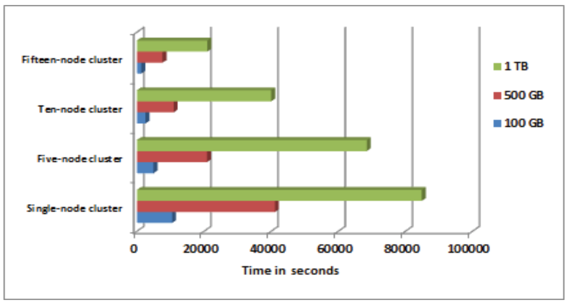


Fig. 9. Result of scalability in a multi-node cluster

We noted that computation times for OLAP cubes of different warehouse sizes decrease as the number of nodes that make up the cluster is increased. It is the 15-node cluster which records the best computation times. The difference between the OLAP cube computation times for the different clusters seems more striking for large data warehouses (500 GB and 1 TB) and the benefits of adding nodes to the cluster can be seen as the size of the warehouse increases. For a warehouse size of 100 GB, the increase in the number of nodes reduces the OLAP cube computation time to 9,188 seconds (2h and 55 mins). However, for a warehouse size of 1 TB, the increase in the number of nodes reduces the OLAP cube computation time to 63900 seconds (17h and 45 mins).

Indeed, the HBase DBMS natively adopts data storage and processing system based on the HDFS file and the MapReduce paradigm respectively. The workload, in terms of memory and computation (CPU) as well as storage is distributed across all the machines of the cluster. This presents two advantages: on the one hand, warehouse data storage is physically distributed across the machines (nodes) that make up the cluster. This prevents all tables being subject to a full scan to extract data. Moreover, the OLAP cube computation query is split into sub-processing across all machines of the cluster thus producing sub-results. These sub-results are grouped to produce the final result of the query which in this case corresponds to the OLAP cube.

Based on this, it is clear that the CN-CUBE benefits from the addition of the distributed environment of the NoSQL database in computing OLAP cubes.

V. Conclusion

The major contribution of this work is to extend the use of column-oriented NoSQL database systems to the data warehouses. Thus, in this paper we have proposed an OLAP cube computation operator, called CN-CUBE. The advantage of this operator is that it uses the positions of values to perform aggregations of the OLAP cube. This approach substantially reduces input and output flows. The implementation of this operator within the SQL interface (Phoenix) of HBase NoSQL DBMS and the experiments we conducted on the real open data warehouses clearly showed the performance of the CN-CUBE operator in a distributed environment compared with that of the Hive CUBE operator.

This work offers several interesting perspectives. In short term, we plain to scale with real big data. In long term, one possible area of research involves creating other OLAP operators to explore the cube.

References

[1] W. H. Inmon, *Building the Data Warehouse*, Information Sciences, Inc Wellesley, MA, USA, 1992.

[2] E. F. Codd, *A Relational Model of Data for Large Shared Data Banks*, Association for Computing Machinery ACM, vol. 1, pp. 377-387, 1970.

[3] N. Leavitt, *Will NoSQL databases live up to their promise ?*, IEEE Computer Society, vol. 43, pp. 12-14, 2010.

[4] R. Cattell, *Scalable SQL and NoSQL Data Stores*, Association for Computing Machinery ACM SIGMOD Record, vol. 39, pp. 12-27. 2011.

[5] J. Pokorny, *NoSQL Databases: A Step to Database Scalability in Web Environment*, Association for Computing Machinery ACM, pp. 278-283, 2011.

[6] J. Gray and S. Chaudhuri and A. Bosworth and A. Layman and D. Reichart and M. Venkatrao and F. Pellow and H. Pirahesh, *Data cube : A relational aggregation operator generalizing group-by, cross-tab, and sub totals*, Journal of Data Mining and Knowledge Discovery, vol. 1, pp. 29-53, 1997.

[7] G. MATEI, *Column-Oriented Databases, an Alternative for Analytical Environment*, Database Systems Journal, pp. 3-16, 2010.

[8] D. Jersy, *Business Intelligence and NoSQL Databases*, Information Systems in Management, vol. 1, pp. 25-37, 2012.

[9] H. Jing and E. Haihong and L. Guan and Du. Jian , *Survey on NoSQL database*, International Conference on Pervasive Computing and Applications (ICPCA), pp. 363-366, 2011.

[10] R. Cattell, *ZooKeeper: Wait-free Coordination for Internet-scale Systems*, Association for Computing Machinery ACM SIGMOD Record, vol. 39, pp. 12-27, 2011.

[11] T. James, *https://github.com/forcedotcom/phoenix/wiki/Performance*, 2013.

[12] Interior Ministry, *French public data Open platform*, http://www.data.gouv.fr/fr/dataset/base-de-donnees-accidents-corporels-de-la-circulation-sur-6-annees, 2013.

[13] K. Shvachko and H. Kuang and S. Radia and R. Chansler, *The Hadoop Distributed File System*, IEEE Computer Society, pp. 1-10, 2010.

[14] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, Association for Computing Machinery ACM Commun, vol. 51,pp. 107-113, 2008.

[15] M. N. Vora, *Hadoop-HBase for large-scale data*, Computer Science and Network Technology (ICCSNT), vol. 1, pp. 601-605, 2011.

[16] R. TAYLOR, *An overview of the Hadoop-MapReduce-HBase framework and its current applications in bioinformatics*, BMC Bioinformatics Journal, vol. 11, pp. S1, 2010.

[17] P. Hunt and M. Konar and F. P. Junqueira and B. Reed, *ZooKeeper: Wait-free Coordination for Internet-scale Systems*, Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference, pp. 11-24, 2010.

[18] A. Thusoo and J. S. Sarma and N. Jain and Z. Shao and P. Chakka and S. Anthony and H. Liu and P. Wyckoff and P. Murthy, *Hive: A Warehousing Solution over a Map-reduce Framework*, International Conference on Very Large Databases Endowment, vol. 2, pp. 1626-1629, 2009.

[19] Apache Hive, https://cwiki.apache.org/confluence/display/Hive/LanguageManual, 2012.

[20] Apache Hive, https://cwiki.apache.org/confluence/display/Hive/HBaseIntegration, 2014.