An Approach for Implementing Online Analytical Processing Systems under Column-Family Databases

Abdelhak Khalil and Mustapha Belaissaoui

***Abstract*—The exponential growth of business data coming from heterogeneous sources imposes the use of new generations of database management systems and new data storage architectures. The major players in the big data market have turned to NoSQL (Not only SQL) technology, which provides a flexible data model and high scalability. In this paper, we investigate OLAP (Online Analytical Processing) implementation using columnar databases (a type of NoSQL system). We provide a set of formal transformation rules in order to map the multidimensional conceptual model to a target model that is suitable for the column-oriented model. Then, we propose two OLAP cube operators called MRC-Cube and SC-Cube, which allow to build the OLAP cube using the MapReduce paradigm and Spark respectively. We conduct an experimental comparison of their performance to analogous relational implementation using Oracle OLAP, we focus particularly on read latency metric under different experimental configurations. The obtained results show a clear difference when performing the OLAP cube building between the relational implementation and the columnar one.**

***Index Terms*— OLAP, Data warehouse, Column-family databases, NoSQL, Cube model, HBase**

I. Introduction

OLAP(Online Analytical Processing) can be defined as a computing method that performs multidimensional analysis of business data in order to extract meaningful indicators. In a broader sense, it refers to a set of tools and practices that enable decision-makers to identify issues and opportunities in a business process from different perspectives. A classical OLAP system includes three main components: a data source, which could be an OLTP (Online Transaction Processing) database from which data is extracted, a data warehouse where data is loaded in a multinational model, and a data cube which is a multidimensional data structure containing aggregated measures across different dimensions. OLAP types can be categorized based on the storage techniques, that is, relational, multidimensional, or hybrid. Accordingly, we distinguish three chief types of OLAP systems, namely R-OLAP(Relational-OLAP)[1][2],MOLAP(Multidimensional-OLAP)[3] and HOLAP(Hybrid-OLAP). These classical implementations utilize traditional databases for storing and processing data. Unfortunately, despite being tried and tested, these databases can't scale when faced with enormous data volumes. Indeed, with the advent of some fields like social media and the Internet of Things (IOT)[4], data repositories are reaching critical sizes that cannot be handled using classical database management systems. Hence, to tackle this problem, business intelligence software vendors tried to accommodate OLAP with big data by developing some solutions like Apache Hive and Apache Kylin. Unfortunately, these solutions construct the cube using a row-oriented method, which is not efficient in performing online analytical processing and doesn't allow them to benefit from the greater performance they can obtain from the column-oriented approach.

Over the last decade, a strong interest toward NoSQL (Not only SQL) technology has arisen[5]. This term identifies a type of database management system that stores data in a non-relational way. One of the major NoSQL-oriented databases is the columnar data store, which saves data by columns rather than rows. This data storage type is considered as the future of Business Intelligence (BI) and the most suitable for data warehouses, and thus for processing analytical queries. Indeed, thanks to its storage model by column, data is stored sequentially on disk, which improves data access extremely. Furthermore, with this organization, columnar storage allows to ignore data that is not required for a certain query. An essential feature required for any OLAP system is its capability to perform aggregations and OLAP cube building over a large data set in a fraction of the time. Unfortunately, columnar databases lack of cube computation operators. To deal with this pitfall, a possible solution consists of integrating external tools like Hive and combining a set of "group by" clauses. This naive method requires multiple accesses to the data warehouse, which decreases performance drastically.

In this research paper, we outline a novel approach for setting up OLAP systems with columnar databases. We propose a set of transformation rules to convert the multidimensional conceptual model to a target columnar logical model. Then, we provide two OLAP operators, which allow building cubes from a data warehouse built upon the column-wise approach, and we evaluate their performance when dealing with big data volumes.

The remainder of this paper is organized as follows: Section 2 discusses the background of this work; Section 3

Manuscript received July 20, 2022; revised November 25, 2022.

Abdelhak Khalil is a software engineer and a PhD student at Hassan the First University of Settat, Morocco (e-mail: a.khalil@uhp.ac.ma).

Mustapha Belaissaoui is a Professor of computer science and a former deputy director of the National School of Business and Management of Settat, Morocco. (e-mail: mustapha.belaissaoui@uhp.ac.ma).

*Corresponding author: Abdelhak Khalil

explains the proposed approach which enables instantiating data warehouses using column-family databases; Section 4 describes the processing of MRC-Cube and SC-Cube operators through a running example; section 5 details implementation and experiments and reports the obtained results; Section 6 concludes this work and gives research perspectives.

II. Background

A. Column-Family databases

The column-family database is another NoSQL database that stores data using a column-wise approach, unlike relational ones, which organize data by rows.

Data stored in a column family database is partitioned vertically, which makes partial read more efficient as only a subset of row attributes is loaded. For this reason, columnar stores are well-suited for OLAP applications. The fundamental concepts of the column-family data model are the keyspace, the column-family, and the column:

- A **keyspace** is an object that contains a list of column families. Formally, a *keyspace* denoted *ks* can be represented by the tuple (n_{ks}, C_{CF}) where n_{ks} is its name and C_{CF} a collection of column families.
- A **column-family** $cf \in C_{CF}$ in turn is an object that holds a collection of columns that can be represented by the tuple (n_{CF}, C_{col}), where n_{CF} is the name of *cf* and C_{col} a collection of columns forming cf.
- A **row** is an instance of a *column-family* having the same columns that can be represented by the triplet $(K_{row}, C_{n_{col}}, C_{v_{col}})$, where K_{row} is a unique key to identify the row, $C_{n_{col}}$ a collection of column names and $C_{v_{col}}$ a collection of column values.
- A **column** denoted *col* is the central part of the columnar data store, formally a column is a tuple (n_{col}, v_{col}) where n_{col} is the column name and v_{col} is the column value.

The data structure of the column-family model is depicted in Fig. 1.

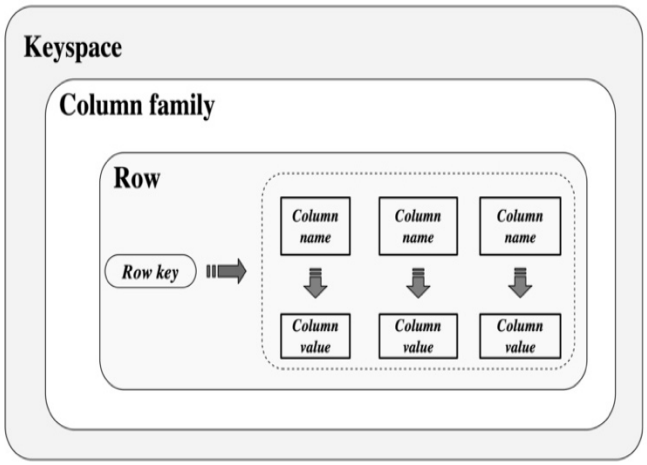

Fig. 1. A representation of the column-family data model

B. State of the art

Since its advent, NoSQL systems have aroused the interest of the research community. Several research papers have compared it to the relational model in terms of read/write latency and scalability[6], [7]. Other research works has focused on the transformation from the relational model to a target NoSQL model [8], [9], [10],[11].

Using NoSQL technology in OLAP systems is a recent research topic. Indeed, considerable research works addressed the instantiation of big data warehouses using NoSQL databases, either with document-oriented model [12], [13] or graph-oriented model [14], [15], [16], or key value stores [17].

Regarding the columnar-oriented model, previous works studied specific approaches for modeling columnar data warehouses. In [18] the authors introduce a set of mapping rules from the multidimensional conceptual model to a target logical model suited to columnar repositories. In [19] the authors propose three distinct approaches in terms of structure to map the star schema into a logical model adapted to column-oriented data warehouse.

As far as OLAP cube building from a data warehouse implemented under the NoSQL model, novel approaches have been followed. Among them, we would name three relevant works. The first one proposes an aggregate operator called MC-CUBE that allows to build OLAP cube using the column-wise approach from a columnar data warehouse [20]. The second one presents a framework for implementing OLAP systems under graph-oriented databases using Neo4j and its declarative language Cypher [21]. The last one studied an extended type of OLAP cuboids built upon a document-oriented data warehouse using nesting and array[13].

We look ahead bringing our contribution to these works by using new algorithms for OLAP cube construction from a columnar data warehouse, and we aim to use Spark instead of Hadoop MapReduce to fully take advantage of in-memory processing.

III. Column-family data warehouse

A data warehouse's logical model describes relationships between facts and dimensions in a more exhaustive manner compared to the conceptual model. Indeed, at the logical level, relationships, entity attributes, and keys are defined in order to serve as the foundation of the physical level. In the literature, there are several candidate approaches for implementing big data warehouses using columnar databases [19]. The one that fits the column family databases proposes storing both the fact and its associated dimensions in the same keyspace. This model uses the concept of "column family" to distinguish between rows. In fact, every dimension is transformed into a column family containing its attributes. The same rule applies to the fact. This modeling approach allows storing attributes belonging to the same column family on the same disk, which increases the efficiency of processing certain queries. Indeed, for instance, HBase which is a popular column-oriented NoSQL DBMS stores column families physically into separate files called *HFiles*. This means that each column family will have its own *HFile* and thus, queries involving column attributes contained in the same column family will run much faster as the other *HFiles* that doesn't

apply to the query will be ignored.

A. Formalization

The mapping from the multidimensional conceptual model to the logical model according to the aforesaid approach is performed by the application of the following rules:

- *R1*. The fact F and its corresponding dimensions are stored in the same keyspace *ks*.
- *R2*. Each fact F is converted to a column family CF^F and each measure $m \in F$ is mapped to a simple column belonging to the fact column family.
- *R3*. Each dimension D is mapped to a column family CF^D ,and each dimension attribute is converted to a simple column within the column family.

1) Case study

For a running example, we consider a decision-making system in the form of a data mart (a mini data warehouse) set up to observe the activity of a reseller of computer hardwires and accessories at different stores. Each store is recognized by its region and type (according to its area). Sales are filled in according to a period that is broken down into months, quarters, and years. Sales are observed by the number of articles according to the type and the turnover.
The logical model obtained by the application of the aforesaid mapping rules will store the fact and its related dimensions in one key space grouped by column family CF^{Sales}, CF^{Store}, $CF^{Article\_type}$, CF^{Date}. Following the second and third rules (R2 & R3), dimension attributes in the relational model will be mapped to columns in the identified column families as well as the measures of the fact e.g., (n_{region}, v_{region}), $(n_{quantity}, v_{quantity})$. The Fig. 2 illustrates the representation of the logical model.

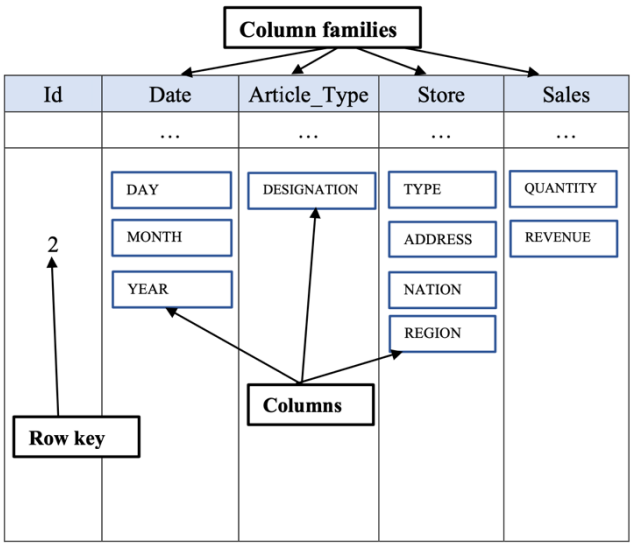

Fig. 2. The logical model representation of the columnar data warehouse.

IV. OLAP CUBE CONSTRUCTION

A. Approach overview

The underlying idea of the proposed approach is to extract all the data forming the OLAP cube and then to use the MapReduce paradigm in the first place and Spark as an enhancement in the second place to compute all possible aggregates at different levels of granularity. Once the cube is fully calculated, it is materialized according to the columnar model in order to take advantage of this architecture with regard to the manipulation of the cube.

The proposed OLAP implementation is based on a column-family architecture for data warehousing (described in Section III) and a distributed OLAP cube computation algorithm. For simplicity, we assume that the cube contains two dimensions, namely, the article type (TYPE) and the store region (REG). The workflow of the OLAP construction is depicted in Fig. 3.

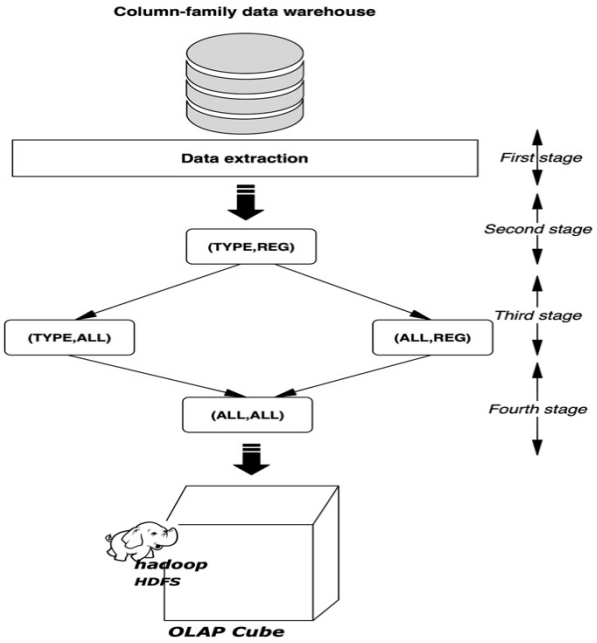

Fig. 3. OLAP Cube computation approach

B. Map-Reduce Columnar Cube (MRC-Cube)

This operator uses the MapReduce processing technique to build the cube. Recall that, MapReduce is a Java programming model within the Hadoop framework that is used for distributed computing. It contains two main tasks, the Map task, which takes input data and converts it to a key/value pair, and the Reduce task, which takes the output of the Map tasks, then iterates through all values associated with a given key and applies a reduce function to produce zero or many key/value pairs. The MRC-Cube operator computes the lattice of cuboids sequentially per stage. More precisely, it is executed in four stages:

1) The first stage

This phase consists of extracting the data that forms the cube from the column family data warehouse and performing multiple joins between the fact and its dimensions. The mappers receive rows combined based on column family from a columnar family data warehouse, then split each one into a key/value pair where the key part contains the row key (algo1: line4) and a tag (algo1: line5) to distinguish the row sources (the column family). The mapper can apply a filter to select values satisfying a certain predicate (algo1: line 3). The produced key/value pairs are buffered in memory, then partitioned by row key so that all occurrences belonging to the same row are grouped

together. The intermediate data is sorted by tag key before being sent to the reducer. The defined reducer function iterates over the sorted data and applies a reduce operation by each unique key encountered. The reduce operation consists of the concatenation of the corresponding set of values (algo2: line6). The output result is persisted on disk and serves as an input to the second stage. An illustration of the first stage is shown in Fig. 4.

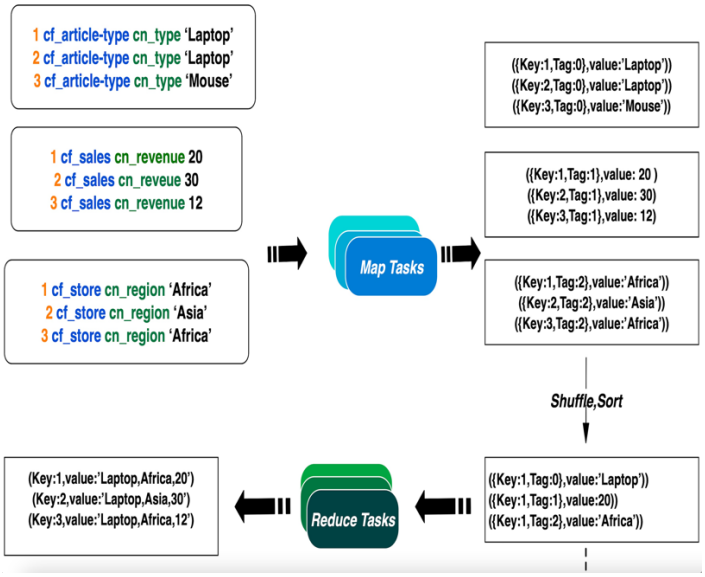

Fig. 4. Performing the reduce side join (the first stage)

Listing 1. Reduce side join - Pseudo code of the map function of the first stage.

Algorithm 1: MRC-Cube – first stage – map function

```
1  input: (rowKey,col_family col_name col_value)
         ∂ : query predicate
2  output: (CompositeKey, v_tmp );
3  if col_value satisfy ∂ do
4      CompositeKey.key ← rowKey;
5      CompositeKey.tag ← tag;
6      v_tmp ←col_value;
7  end
8  emit (CompositeKey, v_tmp )
```

Listing 2. Reduce side join - Pseudo code of the reduce function of the first stage.

Algorithm 2: MRC-Cube – first stage – reduce function

```
1  input: (CompositeKey, v_tmp )
2  output (k_stage1 , v_stage1 )
3  v_stage1 ← empty string;
4  k_stage1 ← CompositeKey
5  foreach v_i ∈ v_tmp do
6      v_stage1 = concat( v_stage1 , v_i )
7  end
8  emit (k_stage1 , v_stage1 )
```

2) The second stage

This stage aims at calculating the cube's finest level of granularity *(TYPE, REG)*. It takes the output of the first phase and iterates over each key/value pair. Following that, the mapper function maps each one encountered to another key/value pair, where the key portion carries the combination of the dimension attributes, and the value part holds the measure to be aggregated. For the sake of optimization, a first aggregation can be performed by a combiner. This is followed by a reducer function that performs an aggregation by key and persists the output pairs on the file system.

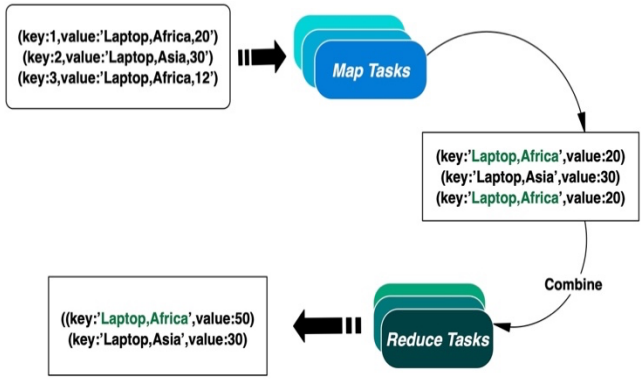

Fig. 5.Building the first level of the cube corresponding to each dimension combination.

3) The third stage

This stage uses the output of the second stage, which corresponds to the first granularity level. The mapper function splits each encountered key into two parts; each one contains a dimension attribute and emits, for a given input key/value pair, two others having the same value. The reducer function performs a simple aggregation operation for each dimension separately. At the end of this stage, the second level of granularity representing *(TYPE, ALL)* and *(ALL, REG)* is calculated and stored on the file system.

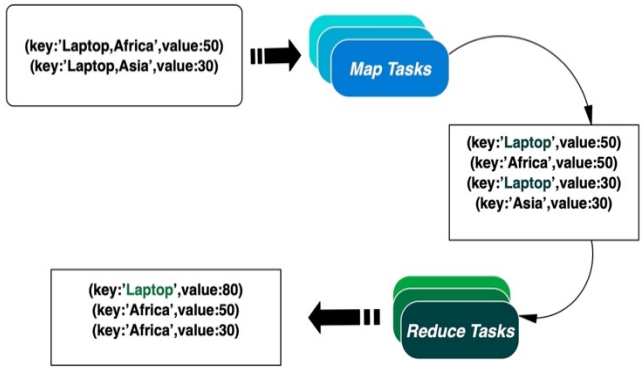

Fig. 6. Building the second level of granularity corresponding to each dimension separately.

4) The fourth stage

The execution of the fourth stage leads to the calculation of the highest level of granularity corresponding to *(ALL, ALL)*. Starting from the output of the previous stage, the mapper function processes the input datasets and replaces the key entries with the string value '*ALL, ALL*'. Therefore, all key/value pairs will be sent to the same reducer.

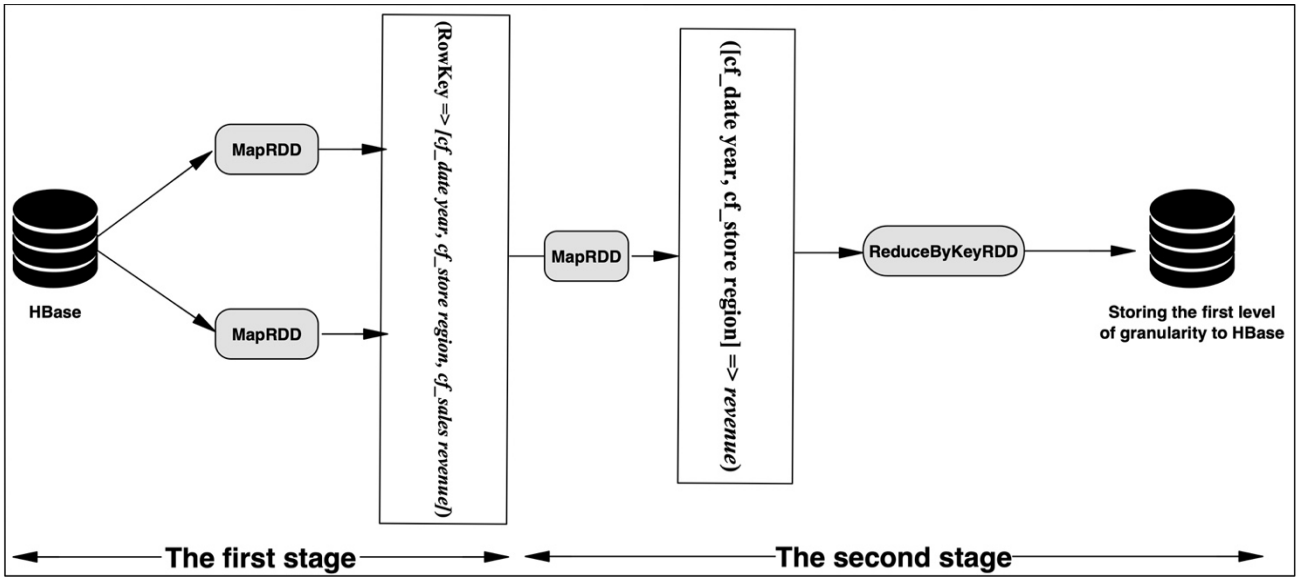

Fig. 7. Data extraction and building the first level of granularity (the first and second stage)

C. Spark Columnar Cube (SC-Cube)

Since its advent, MapReduce has been indisputably the standard parallel computation model for big data. However, the major disadvantage of MapReduce is that it processes data on disk after each iteration, which has a considerable I/O cost. Being aware of this pitfall, the community tried to overcome this problem by using random access memory (RAM) for data processing instead of disk. For example, Apache Spark keeps data in memory to speed up data flow between iterations. For this, the concept of an RDD is used, which stands for Resilient Distributed Datasets. An RDD is a data abstraction provided by Spark that allows to perform parallel calculations in memory on a cluster in a completely fault-tolerant way. Identically, Spark performs the cube computation in five stages. The processing of each stage is detailed below:

1) The first stage

First, Spark reads the input data from the columnar database by specifying the column families and the columns involved in the cube building, this avoids scanning unnecessary column values. Then Spark converts each row to a key/value pair *RDD[(K, Map[CF, Map[CN, V]])]*, where the key *K* is nothing but the row key and the value is a nested map structure that associates a given column family *CF* and a column name *CN* to a value. Afterwards, an RDD transformation is applied to fetch only the columns that compose the cube. Each column will be tagged with its column family to indicate which fact or dimension it belongs to. Therefore, the first stage will produce the following intermediate pair RDD: [rowKey,'CF1 CN1, CF2 CN2, …'].

2) The second stage

In this stage, an RDD transformation is applied to the resulting RDD from the first stage to build the lowest level of granularity. The mapper tokenizes each value of the input RDD to fetch the dimension columns along with the fact measure column. This allows to generate a new pair RDD where the key is a combination of all the dimensions involved in the cube and the value is the measure to be aggregated. At the end of this stage the first level of granularity *(YEAR, REGION)* is calculated. An illustration of the first and the second stage is shown in Fig. 7.

3) The third and fourth stage

The third and fourth stages are executed in parallel and aim at computing the second and third granularity levels, which correspond in our case to *(YEAR, ALL)* and *(ALL, REGION)*. The input RDD is the resulting one from the previous transformation. The mapper split each key into two parts; each one represents a dimension attribute. Then, duplicates the measure value for the two keys. Next, Spark uses a transformation operation called *reduceByKeyRDD* that performs aggregation by each dimension attribute separately.

4) The fifth stage

This stage computes the highest level of granularity of the cube, which corresponds to *(ALL, Sum(revenue))* by performing an aggregation by the unique key 'ALL' from the resulting RDD of the second stage.

V. Implementation and Experiments

A. Experimental setup

In order to prove the feasibility of the proposed OLAP operators, we have conducted an experimental evaluation in a distributed environment. This environment is set up using a cluster made up of three Docker containers. The *Dockerfile* used to build the image is available on GitHub repository [22]. Each container runs an instance of an image having Apache Hadoop v2.7.2, Apache Spark v2.2.1 and Apache HBase v1.4.8 installed on the same cluster as HBase, Spark can be used to perform complex processing on HBase data. For this, the different Spark Executors will be co-located with the region servers, and will be able to perform parallel processing directly where the data is stored.

We integrated into this setup Oracle OLAP, a relational OLAP technology that offers the ability to perform sophisticated computation using SQL queries. The choice fell on Oracle OLAP to compare the proposed OLAP implementation with a relational one, based on read latency criteria, in order to decide impartially which implementation is more effective when scaling up or when queries get more complicated.

Dataset: For feeding the HBase database, we used *KoalaBench* an extension of the TPC-H benchmark, which is commonly used for assessing the performance of decision support systems [23]. TPC-H encompasses a snowflake schema, including eight transactional tables that model the business process of a product seller. The *KoalaBench* is developed using the Java language and allows to generate different data sizes by specifying the scalability factor. By contrast to the TPC-H benchmark, the extended one supports NoSQL systems and performs parallel and distributed generation of data natively in HDFS. Data loading is done by executing the utility command *ImportTsv* which allows to load data in TSV format into HBase. Practically, it triggers a MapReduce job on the main file stored in HDFS to read the data and then inserts it via put commands into the database.

For illustration purposes, we append at the end of this paper the script for loading the generated data into HBase following the meta-model described previously.

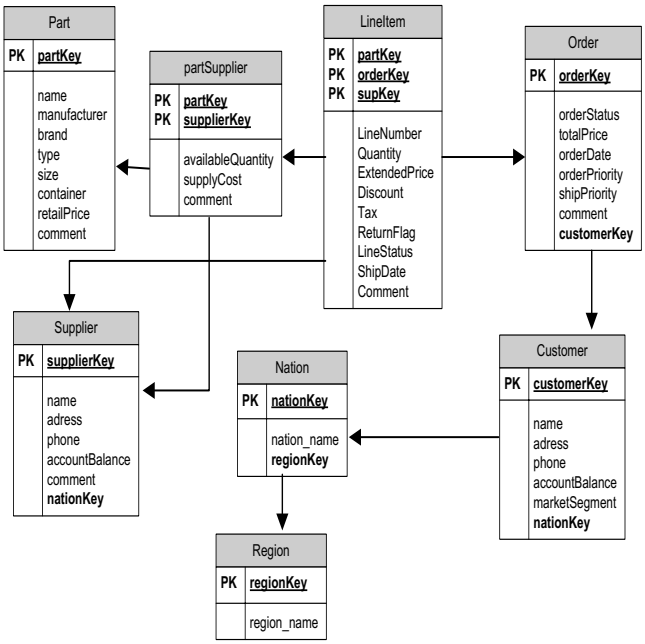

Fig. 8. The TCP-H relational data model redrawn based on [24]

According to the proposed rules, the fact table *LineItem* in the relational data model is converted to a column family in the corresponding columnar data model, and each measure is converted to a column belonging to the same column family. The foreign keys are not mapped as the matching between the fact column family and its associated dimension is assured by the row key. Following the rule R3, seven column families are identified to represent the dimensions in a normalized way, namely, CF^{ORDER}, $CF^{SUPPLIER}$, $CF^{CUSTOMER}$, CF^{NATION}, CF^{REGION}, $CF^{PARTSUPP}$, CF^{PART}. The obtained data model is depicted in Fig. 9.

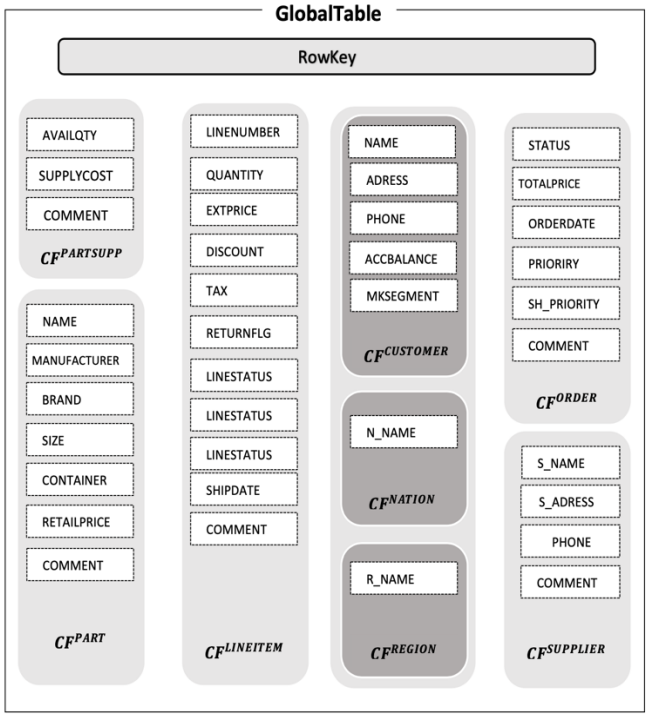

Fig. 9. The columnar data model for TPC-H Shema

The TABLE I demonstrates how the scale factor influences the generated data. We observe that data is generated proportionally to the scale factor, except for the table Part in which we don't scale linearly but logarithmically.

TABLE I. DATA SIZE PER SCALE FACTOR[23]

Table	Lines	Disk space in byte (sf=10)
Customer	300000×sf	29360128
Part	800000 × (1+log2(sf))	69206016
LineItem	6×106×sf	6227702579
Supplier	20000×sf	1782579
OrderDate	2556×sf	233472

B. Experiment 1

The metric reported by the first experiment is the capability of the system to process queries involving a gradual number of dimensions. We used a dataset having 60 million records (SF=1). Then we performed a comparative study between MRC-Cube, SC-Cube, and Oracle OLAP. In order to build the OLAP cube, we used queries that aggregate the sales revenue according to different dimensions. We distinguish three types of queries according to their dimensionality (Q1, Q2 and Q3); details about query configuration are depicted in Table 1. Each query was run three times. Fig. 10 reports the average execution time.

TABLE II. QUERIES CONFIGURATION

Query	Dimension: Attribute	Predicate	Measure
Q1(2D)	Customer: Region Supplier: Region		
Q2(3D)	Customer: Region Supplier: Region OrderDate: Year	OrderDate: Year = 2022 Customer: Region=Africa	Sum (Revenue)
Q3(4D)	Customer: Region Supplier: Region OrderDate: Year Part: Brand		

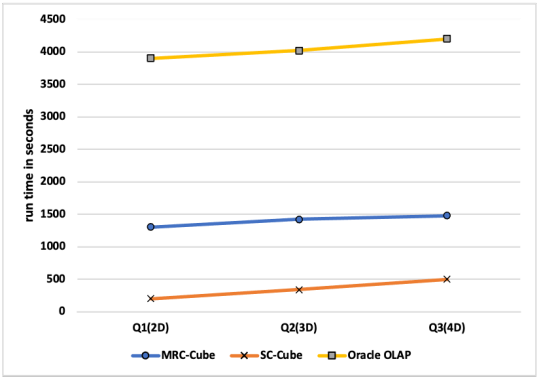

Fig. 10. OLAP cube construction time by query dimension

The runtime graph of the above-mentioned queries (Q1, Q2 and Q3) shows a slight variation when queries involve a higher number of dimensions. However, our findings demonstrate that OLAP cube computation with the relational approach is more time-consuming (up to more than four times slower). This is because the OLAP cube is built using the row-oriented approach, which is inefficient compared to the column-oriented method, especially when the select query implies a single attribute for each dimension table. Additionally, the curves of MRC-Cube and SC-Cube state clearly that Spark outperforms MapReduce. This is easily explained by the fact that Spark processes and saves data between intermediate steps in memory, whereas Hadoop MapReduce processes data on disk for subsequent steps. Which means that multiple accesses to the data warehouse are needed and imply a considerable I/O cost.

Furthermore, the way the data is organized in the data warehouse has a significant impact on OLAP cube building. Indeed, using column families enables attributes belonging to a specific dimension/fact to be stored in the same disk space, which speeds up decisional query response time, especially when they involve attributes of the same dimension.

C. *Experiment 2*

The aim of this experiment is to evaluate the scalability of MRC-Cube and SC-Cube when faced with an increasing size of the data warehouse. This is compared to analogous R-OLAP implementation: the Oracle OLAP operator. Recall that, Oracle OLAP is an embedded engine in the Oracle database which can perform complex computation with the use of straightforward SQL queries. The cube is built according to three dimensions: Customer, Supplier and *OrderDate* (Query Q2) with a scale factor gradually increasing from 1 to 10, which corresponds to a sample dataset varying approximatively from 1 GB to 10 GB. The obtained result is shown in Fig. 12.

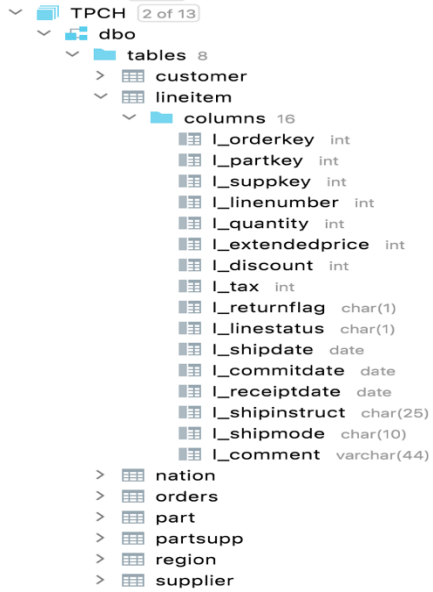

Fig. 11. Setting up the TPC-H schema with Oracle relational database management system (R-OLAP)

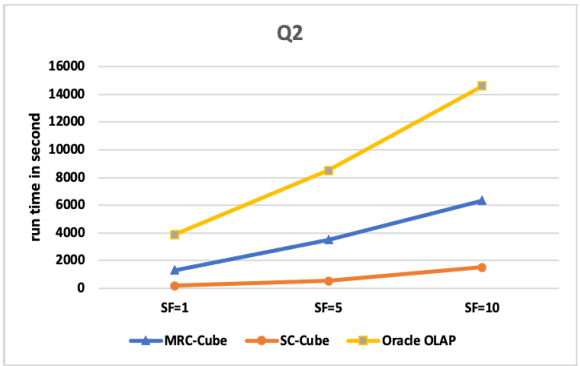

Fig. 12. OLAP cube construction time by scale factor

The curves show that the required time for building the OLAP cube with the relational approach increases exponentially when scaling up. By contrast, the curve of MRC-Cube and SC-Cube shows a slight variation when facing an increasing data volume, which is the reason for the invention of NoSQL databases, and performs OLAP cube computation up to four times faster.

Overall, the result obtained states that the underlying performance cost involved in relational implementation is considerable as it requires longer data traversal when performing the OLAP cube construction. These traversals imply join operations over multiple tables, which decrease the performance considerably. On the other hand, the performance of the columnar approach lies in the use of parallel computing and takes full advantage of the columnar

approach, which allows less disk seek to output data as only the columns that are needed are accessed.

VI. Conclusion

This paper aims at implementing OLAP systems in column-family databases. For this, we proposed the design and implementation of a columnar data warehouse using a set of transformation rules that facilitate the mapping from the multidimensional conceptual model used as the basis of data warehouses and OLAP applications to the underlying logical model. Afterward, we presented two aggregate operators called MRC-Cube and SC-Cube, which use the Hadoop MapReduce paradigm and Apache Spark to compute OLAP cubes. These operators use the reduce side join algorithm to combine the content of the fact and dimension column families based on a row key. To validate our proposal, we conducted an experimental evaluation based on the read latency criterion. We compared MRC-Cube and SC-Cube with each other and with a relational operator using Oracle OLAP. The obtained results show that both of our proposed aggregate operators perform well and exhibit the effectiveness of columnar databases for analytical purposes.

This work provides interesting perspectives for future work. In the upcoming work, we aim at doing a comparative study between the columnar implementation and other NoSQL implementations in the literature.

References

[1] R. Kimball, "Kimball Dimensional Modeling Techniques," pp. 1–24, 2013, doi: 10.1016/B978-0-12-411461-6.00009-5.

[2] O. Mangisengi and A. M. Tjoa, "A multidimensional modeling approach for OLAP within the framework of the relational model based on quotient relations," in *Proceedings of the 1st ACM international workshop on Data warehousing and OLAP - DOLAP '98*, Washington, D.C., United States, 1998, pp. 40–46. doi: 10.1145/294260.294270.

[3] B. Dinter, C. Sapia, G. Höfling, and M. Blaschka, "The OLAP market: state of the art and research issues," in *Proceedings of the 1st ACM international workshop on Data warehousing and OLAP - DOLAP '98*, Washington, D.C., United States, 1998, pp. 22–27. doi: 10.1145/294260.294268.

[4] C. Junkuo and M. Mingcai Lin Xiaojin, "A Survey of Big Data for IoT in Cloud Computing," *IAENG International Journal of Computer Science*, vol. 47, no. 3, pp. pp585-592, 2020.

[5] K. ElDahshan, E. K. Elsayed, and H. Mancy, "Enhancement Semantic Prediction Big Data Method for COVID-19: Onto-NoSQL," *IAENG International Journal of Computer Science*, vol. 47, no. 4, pp. 613–622, 2020.

[6] Y. Cheng, P. Ding, T. Wang, W. Lu, and X. Du, "Which Category Is Better: Benchmarking Relational and Graph Database Management Systems," *Data Sci. Eng.*, vol. 4, no. 4, pp. 309–322, Dec. 2019, doi: 10.1007/s41019-019-00110-3.

[7] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, "Comparison between relational and NOSQL databases," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, May 2018, pp. 0216–0221. doi: 10.23919/MIPRO.2018.8400041.

[8] G. Karnitis and G. Arnicans, "Migration of Relational Database to Document-Oriented Database: Structure Denormalization and Data Transformation," in *2015 7th International Conference on Computational Intelligence, Communication Systems and Networks*, Riga, Latvia, Jun. 2015, pp. 113–118. doi: 10.1109/CICSyN.2015.30.

[9] T. Jia, X. Zhao, Z. Wang, D. Gong, and G. Ding, "Model Transformation and Data Migration from Relational Database to MongoDB," in *2016 IEEE International Congress on Big Data (BigData Congress)*, San Francisco, CA, USA, Jun. 2016, pp. 60–67. doi: 10.1109/BigDataCongress.2016.16.

[10] S. Lee, B. H. Park, S.-H. Lim, and M. Shankar, "Table2Graph: A Scalable Graph Construction from Relational Tables Using Map-Reduce," in *2015 IEEE First International Conference on Big Data Computing Service and Applications*, Redwood City, CA, USA, Mar. 2015, pp. 294–301. doi: 10.1109/BigDataService.2015.52.

[11] C. Costa and M. Y. Santos, "Big Data: State-of-the-art Concepts, Techniques, Technologies, Modeling Approaches and Research Challenges," *IAENG International Journal of Computer Science*, vol. 44, no. 3, pp. 285–301, 2016.

[12] S. Bouaziz, A. Nabli, and F. Gargouri, "Design a Data Warehouse Schema from Document-Oriented database," *Procedia Computer Science*, vol. 159, pp. 221–230, 2019, doi: 10.1016/j.procs.2019.09.177.

[13] M. Chavalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, "Document-oriented data warehouses: Models and extended cuboids, extended cuboids in oriented document," *Proceedings - International Conference on Research Challenges in Information Science*, vol. 2016-Augus, 2016, doi: 10.1109/RCIS.2016.7549351.

[14] A. Vaisman, F. Besteiro, and M. Valverde, "Modelling and Querying Star and Snowflake Warehouses Using Graph Databases," in *New Trends in Databases and Information Systems*, vol. 1064, T. Welzer, J. Eder, V. Podgorelec, R. Wrembel, M. Ivanović, J. Gamper, M. Morzy, T. Tzouramanis, J. Darmont, and A. Kamišalić Latifić, Eds. Cham: Springer International Publishing, 2019, pp. 144–152. doi: 10.1007/978-3-030-30278-8_18.

[15] C.-H. Chou, M. Hayakawa, A. Kitazawa, and P. Sheu, "GOLAP: Graph-Based Online Analytical Processing," *Int. J. Semantic Computing*, vol. 12, no. 04, pp. 595–608, Dec. 2018, doi: 10.1142/S1793351X18500071.

[16] H. Akid, G. Frey, M. B. Ayed, and N. Lachiche, "Performance of NoSQL Graph Implementations of Star vs. Snowflake Schemas," *IEEE Access*, vol. 10, pp. 48603–48614, 2022, doi: 10.1109/ACCESS.2022.3171256.

[17] K. Abdelhak, "New Approach for implementing big datamart using NoSQL key-value stores".

[18] M. Chevalier, M. E. Malki, A. Kopliku, O. Teste, and R. Tournier, "Implementation of Multidimensional Databases in Column-Oriented NoSQL Systems," in *Advances in Databases and Information Systems*, vol. 9282, M. Tadeusz, P. Valduriez, and L. Bellatreche, Eds. Cham: Springer International Publishing, 2015, pp. 79–91. doi: 10.1007/978-3-319-23135-8_6.

[19] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Using the column oriented NoSQL model for implementing big data warehouses," *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'15)*, pp. 469–475, 2015.

[20] K. Dehdouh, O. Boussaid, and F. Bentayeb, "Big Data Warehouse: Building Columnar NoSQL OLAP Cubes," *International Journal of Decision Support System Technology*, vol. 12, no. 1, pp. 1–24, Jan. 2020, doi: 10.4018/IJDSST.2020010101.

[21] A. Khalil and M. Belaissaoui, "A Graph-oriented Framework for Online Analytical Processing," *IJACSA*, vol. 13, no. 5, pp. 547–555, 2022, doi: 10.14569/IJACSA.2022.0130564.

[22] L. Sfaxi, "Hadoop Cluster with Docker." [Online]. Available: https://github.com/liliasfaxi/hadoop-cluster-docker

[23] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, "Benchmark for OLAP on NoSQL technologies comparing NoSQL multidimensional data warehousing solutions," *Proceedings - International Conference on Research Challenges in Information Science*, vol. 2015-June, no. June, pp. 480–485, 2015, doi: 10.1109/RCIS.2015.7128909.

[24] "TPC-H Specification,." Accessed: Sep. 28, 2022. [Online]. Available: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.2.pdf

Appendix

Script loading in HBase

```
1. CREATE'GlobalTable','CF_LineItem','CF_PART','CF_CUSTOMER,
   'CF_SUPPLIER','CF_ORDER','CF_NATION','CF_REGION'
2. hbase org.apache.hadoop.hbase.mapreduce.ImportTsv
3. -Dimporttsv.separator=','
   -Dimporttsv.columns=HBASE_ROW_KEY,
4. CF_LineItem:lineNumber,
```

```
5.  CF_LineItem:quantity,
6.  CF_LineItem:extendedPrice,
7.  CF_LineItem:discount,
8.  CF_LineItem:tax,
9.  CF_LineItem:returnFlag,
10. CF_LineItem:status,
11. CF_LineItem:shipDate,
12. CF_LineItem:comment,
13. CF_ORDER:o_orderStatus,
14. CF_ORDER:o_orderDate,
15. CF_ORDER:o_orderPriority,
16. CF_ORDER:o_comment,
17. CF_ORDER:o_shipPriority,
18. CF_CUSTOMER:c_name,
19. CF_CUSTOMER:c_address,
20. CF_CUSTOMER:c_phone,
21. CF_CUSTOMER:c_accountBalance,
22. CF_CUSTOMER:c_marketSegment,
23. CF_PART:p_name,
24. CF_PART:p_manufacturer,
25. CF_PART:p_brand,
26. CF_PART:p_type,
27. CF_PART:p_size,
28. CF_PART:p_container,
29. CF_PART:p_retailPrice,
30. CF_PART:p_comment,
31. CF_SUPPLIER:s_supplierKey,
32. CF_SUPPLIER:s_name,
33. CF_SUPPLIER:s_address,
34. CF_SUPPLIER:s_phone,
35. GlobalTable, /root/flat_line_item.csv
```

Abdelhak Khalil is a researcher in computer science at Hassan the First University, Settat, Morocco. Previously, he obtained his engineering degree from the National School of Applied Science of Marrakech. His interests in research focus on business intelligence evolution in the big data era, big data analytics, value creation from big data, and cluster computing.

Mustapha Belaissaoui is a Professor of computer science at Hassan the First University, Settat, Morocco. He received his PhD in artificial intelligence from Mohammed V University. His research interests include artificial intelligence, combinatorial optimization, and information systems. He authored more than a hundred papers, including journals, conferences, chapters, and books that appeared in specialized journals and symposia. Furthermore, he was deputy director of the National School of Business and Management of Settat.