A Data Placement Strategy for Distributed Document-oriented Data Warehouse

Abdelhak Khalil, *Member, IAENG,* Mustapha Belaissaoui, and Fouad Toufik

Abstract—**Within the big data phenomenon, cluster computing has attracted special attention for its impressive ability to process a vast amount of data. Hadoop cluster is a promising cluster computing framework for implementing big data warehouses and conducting big data analysis, thanks to its distributed file system and MapReduce paradigm. In this paper, we propose a new data placement strategy for a document-oriented data warehouse within the distributed environment of Hadoop. Our contribution includes formalizing the logical model and cube building operators. First, we present the cube building algorithm's processing using the MapReduce paradigm, and then we explore the possibility of accelerating the process by using Spark instead. To evaluate the proposed framework in terms of OLAP cube construction cost, we conducted experiments on a physical cluster, which yielded promising results, specifically, that the proposed framework enables efficient data placement and significantly speeds up cube building compared to a similar OLAP infrastructure chosen from existing literature.**

Index Terms— **Decision support systems, OLAP, Data warehouse, NoSQL, Data cube.**

I. Introduction

OLTP (Online Transactional Processing) systems have made it possible, in the era of the technological revolution, to enhance the performance of companies by automating many of their business processes. However, they are not well-suited for business intelligence analysis, especially when dealing with Big Data requirements. This is why a second decisional component called OLAP (Online Analytical Processing) is necessary in an organization's information system [1]. The role of OLAP systems is to promote the management monitoring of the activities that form the business processes thanks to Key Performance Indicators (KPIs), these KPIs are expressively grouped together in a dashboard [2]. Just like the OLTP system which relies on RDBMS, the OLAP decision-making system relies on the data warehouse which is the core component of business intelligence. OLAP systems use a multidimensional view called OLAP cube to query analytical data. The starting point of an OLAP cube is the multidimensional model which defines two essential concepts: fact and dimensions [3]. The fact contains quantitative metrics of a business subject, while the dimension provides descriptions of the facts being stored. The main problem with multidimensional data structures is that there are sparse with multiple null entries and not distributed uniformly throughout the multidimensional space, it is concentrated in groups in spaces-times where business events occur most often [4]. This issue makes it more complex to store and process cube data. Theoretically there are main four technological approaches to store and process analytical data namely Relational OLAP(ROLAP), Multidimensional OLAP(MOLAP), (HOLAP)Hybrid OLAP and Desktop OLAP(DOLAP), with these approaches, data is stored either in a relational or multidimensional database in single node architecture. With the emergence of Big Data phenomenon, storing and processing a huge amount of data with, eventually, a large number of dimensions becomes a very challenging concern for classical OLAP approaches and exponentially cost in memory and time [5], [6], [7], [8]. In order to mitigate this problem, in the last few years, technical solutions for storing and processing data in the context of big data started using a distributed approach which consists of using cluster computing [9]. With this approach, data processing is distributed and parallelized between nodes in a cluster. As a result, the multidimensionality specific to decision-making queries is easily managed thanks to the cluster's computation nodes.

Actually, Big Data decision-making solutions use either a Hadoop SQL engine which allows to write and execute MapReduce jobs using the SQL language such as Hive and Pig, or a massively parallel processing database in the same way as Teradata which allows to execute analytical queries on Hadoop. Among the cons of the existing solutions is the processing speed which is relatively time-consuming as they use classical cube computation approaches and join operations tend to be slow because multiple joins are performed to link a fact to its associated dimensions due to the logical modelling of Star/Snowflake Schema.

In this paper, we propose a distributed OLAP engine framework built atop of Hadoop which relies on MapReduce paradigm and Spark This framework allows the implementation of distributed data warehouse using the document-oriented model, and provides OLAP capability in the big data context. Our contribution can be summarized as follows:

- Data modeling formalization of document-oriented data warehouse.
- The pre-calculation of OLAP cube using MapReduce framework (MR-Cube). Then a re-implementation using Spark is given to further speed up the OLAP cube building.

The remaining of this paper is organized as follows. After the introduction, Section II presents the related works. Section III Introduces the background, the architecture, and

Manuscript received July 3, 2023; revised October 5, 2023.

Abdelhak Khalil is a software engineer and a PhD student at Hassan the First University of Settat, Morocco (corresponding author to provide phone: (+212) 523 723 577, 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)

Fouad Toufik is an assistant professor of computer science at Mohammed V University, Morocco (e-mail: toufik.fouad@gmail.com)

the formalization of the document-oriented data warehouse. Section IV introduces MR-Cube and Spark-Cube operators and details the execution steps of each operator. In Section V an implementation is done to evaluate cube building and storage cost performance metrics. Finally, Section VI concludes the paper and discusses future work.

II. Related works

As mentioned earlier, facing big data phenomenon, a new era is emerging in the field of decision support systems. Analysing a huge amount of data has been the motivation for several research studies that aimed to explore new ways for implementing data warehouses and cube building engine on big data trend. In what follows, we will discuss some related works.

In [10] , the authors present an OLAP system built on Hadoop framework called HaoLap (Hadoop based OLAP). The research includes the design, the implementation and the evaluation of the proposed OLAP engine. The MapReduce and data loading algorithm is detailed and experiments are conducted by the authors to compare HaoLap with some competitors like Hive, HadoopDB and HBase Lattice on different configurations. In the same vein, in [11] the authors propose a cloud computing architecture to set up a data warehouse and to perform OLAP analysis. The proposed environment uses MapReduce and Hive to build OLAP cubes. In [12] the authors explore the feasibility of performing small analytics queries on big data in Hadoop while benefitting from secondary indexes and partitioning in HBase. This research shows the impact of data placement strategy and indexing technique on data availability and accessibility.

In the last few years, a remarkable tendency toward using NoSQL oriented databases for implementing OLAP engines has increased [13], [14], [15], [16], [17], [18]. Column-oriented model is a promising playground that has been tested for OLAP features and data warehousing. In [19], the authors propose three approaches in order to perform the mapping from the conceptual model to logical columnar models, each one leads to a different structure and attribute types. Later the same authors proposed a new approach for building OLAP cubes from columnar NoSQL data warehouse based on an operator called MR Columnar CUBE (MC-Cube) [20], [21]. This operator allows cube construction using MapReduce paradigm according to a column-wise approach. In [22], the authors propose a novel framework to create a columnar data warehouse by applying a set of transformation mapping rules. Additionally, they introduced two aggregate operators, namely MRC-Cube and SC-Cube. These operators leverage the Hadoop MapReduce paradigm and Apache Spark to perform computations for OLAP cubes. Like columnar databases, the use of key value databases can be extended to the implementation of distributed data warehouses. In [23], [24] , a new approach is presented in order to instantiate big data mart under key value stores and to perform OLAP analysis.

Regarding OLAP implementation using document-oriented model. In [25], the authors present a new model for extracting OLAP cube from document-oriented NoSQL databases. The proposed model is based on parallel similarity algorithm implemented with MapReduce paradigm to find documents with similar attribute and use them as OLAP dimensions. In [26], The development and design of data warehouses has been suggested using a Model Driven Approach methodology.

Last but not least, in [27] the authors propose a solution to implement OLAP engines on NoSQL databases using a MapReduce like execution with Resilient Distributed Datasets (RDDs).

In summary, all the aforementioned works expresses the importance and the motivation behind using new technologies to support OLAP systems in big data era. The use of document-oriented model for the storage and analysis of big data represents a promising research direction in the field. However, there is still a gap in the development of OLAP systems based on the document-oriented model.

III. Document-oriented Datawarehouse

A. The multidimensional model

The starting point of an OLAP cube is the multidimensional model. In this section we give an overview of basic concepts related to multidimensional modelling and present the formal representation for its components.

Definition 1: The Multidimensional Schema (MS) is a logical description used to design data warehouse systems. The simplest multidimensional schema is Star schema, the two other chief types are Snowflake schema and Galaxy schema. Formally, a Multidimensional Schema denoted S is a triplet $\left(F^S, D^S, Star^S\right)$ where:

- $F^S=\{F_1,...,F_n\}$ a set of facts.
- $D^S=\{D_1,...,D_m\}$ a set of dimensions.
- $Star^S: F_i \rightarrow 2^{D_j}$ is a function that associates each fact $F_i \in F^s$ to its associated dimensions $D_j \in D^s$.

Definition 2: A fact contains measurement about a business process. It is located at the centre of the multidimensional schema. It contains quantitative information for analysis called measures or metrics used in aggregation functions according to dimension attributes. Formally, we can present a fact as a pair (Lk^F, M^F), where:

- $Lk^F=\{lk_i,...,lk_m\}$ a set of foreign attributes that link a fact to its associated dimensions.
- $M^F=\{m_1,...,m_p\}$ a finite set of measures.

Definition 3: A dimension describes an analysis coordinate of a measure. It holds dimensional attributes that are used in restriction and grouping clauses. A formal representation of a dimension is a triplet (A^D, H^D, I^D), where:

- $A^D=\{a_i{}^D,...,a_p{}^D\}$ is a finite set of attributes.
- $H^D=\{h_j,...,h_r\}$ is a finite set of hierarchies.
- $I^D=\{I_k,...,I_s\}$ a set of dimension instances.

A hierarchy is a directed tree structure for a dimension whose nodes are dimensional attributes and whose arcs represent connections between dimensional attributes. Hierarchies are used to perform aggregations with different granularity levels. A hierarchy denoted H^D is defined with a pair $(A^H, Weak^H)$, Where:

- $A^H=\{a_1^H,...,a_n^H\}$ an ordered set of attributes which are used to present the level of granularity along the dimension, the first level of granularity represents the dimensional attribute, $\forall i \in [1..n], a_i^H \in A^H$ and $a_1^D=a_1^H$.
- $Weak^H:a_i \rightarrow \{wa_1,...wa_k\}$ is a function possibly associating parameters to a set of weak attributes.

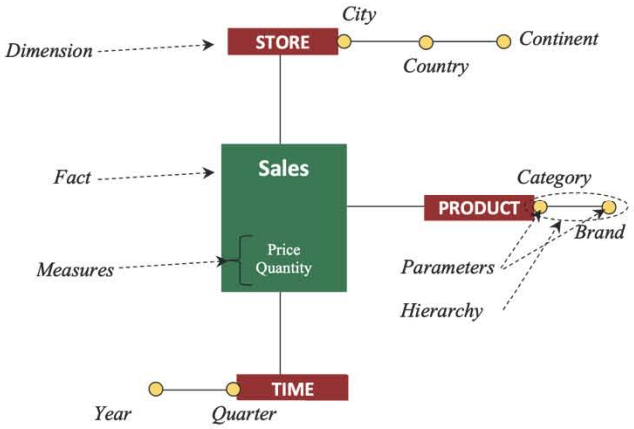

Fig. 1. The conceptual multidimensional schema representation

B. The Mapping rules

The design of a data warehouse under the document-oriented model revolves around data modelling and how the model represents relationships between fact and dimensions. Unlike the ROLAP implementation where the mapping from the conceptual model to the logical one is standardized and straightforward, the document-oriented model offers several candidate models for representing relationships between the OLAP schema components. The main challenge is to find a compromise between the data retrieval performance and the data storage capacity. As long as data storage is no longer a big issue nowadays, we are interested more in query performance against the data warehouse and especially in OLAP cube building. In this paper we propose a model which uses references to link a fact document to its associated dimensions. It corresponds to a model in which the data is stored in a hierarchical structure and mandates that each child dimension record has only one fact record. To implement a data warehouse using this model, the following rules are applied:

(R1) Each fact record, $F_i \in F^S$ is transformed into a document instance.

(R2) Each measure $m_j \in M^F$ is translated into an atomic field $af_i^{a_i}$.

(R3) Each dimension record $D_i \in D^S$ is mapped to a document instance.

(R4) Each dimension attribute $a_j \in A^D$ is mapped to an atomic field $af_j^{a_j}$.

(R5) To represent link between the fact collection and its dimensions we use a reference in the dimension collection that store the path to the associated fact instance denoted $F_i^S\_id$.

IV. OLAP CUBE CONSTRUCTION

A. Approach overview

Efficient data cube computation plays a major role in OLAP system implementation. Among the well-known techniques used in the standard database systems for such computation we can cite the Multiway Aggregation which is a typical MOLAP approach that compute the entire cube using a multidimensional array structure and The Bottom-Up Computation (known as BUC method). These techniques are suited for a limited amount of data and single node architecture. However, with the explosion in volume and variety of data that characterizes big data phenomenon, such task becomes very challenging to traditional softwires and databases. To fix a part of this issue, several OLAP solutions for big data appeared on the market. The most popular is Apache Hive which is a data warehouse infrastructure built on top of Hadoop. Hive offers several interesting features and can perform analytical queries efficiently. To perform aggregations, Hive uses MapReduce framework only with a row-wise approach. Consequently, this leads to a big cost in terms of data transfer and processing and does not allow us to benefit from the advantages offered by in-memory computation engines. To overcome this issue, MR-Cube performs OLAP cube computation from data stored row by row using a column-wise approach. The fact that the fact document and the dimensions share a driving join key, the join operation can be executed using one MapReduce join, which explains our data warehouse modelling. Hence, reducing the number of MR-jobs allows better performance and speeds up the full cube computation. Furthermore, we propose a re-implementation of MR-Cube on Spark (Spark-Cube) which allows us to take advantage of in-memory computation. Our approach relies on precomputation of aggregations over all possible dimensions after performing a query predicate that, selects only records satisfying certain conditions. The design of MR-Cube and Spark-Cube is based on cube theory, which means that the aggregation corresponding to each combination of dimensions is a cuboid and the lattice of cuboids is the full OLAP cube. As far as the technical environment is concerned, we implemented a distributed architecture based on Hadoop file system that allow the partitioning of the data warehouse across highly scalable clusters(nodes). The general architecture of the proposed OLAP system is depicted in Fig. 2

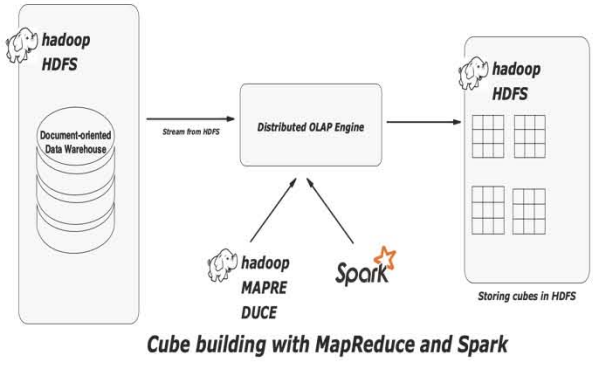

Fig. 2. The general architecture of the proposed framework

B. MR-Cube

In this operator, as the name suggests, the MapReduce framework is used to perform OLAP cube computation. Recall that, MapReduce is a distributed programming model for processing a large amount of data. One of the main characteristics of this paradigm is to allow developers to focus only on business-oriented processing because MapReduce and the environment in which it runs support a number of features such as data distribution, load balancing and parallelization. The terminologies of Map and Reduce come from the functional programming languages used for their construction.

MR-Cube builds the OLAP cube through four steps. The first one loads data from a data warehouse built on top of the Hadoop file system and performs query predicate to filter the tuple satisfying the query criteria. Then, performs a reduce side join between the fact and its associated dimensions. The output result is stored in the file system and used in the second step as input to perform an aggregation operation. This intermediate result is the first level of granularity and is considered as the starting point of the entire OLAP cube computation

In what follows the execution of MR-Cube operator is described in details:

The first step: This step consists of selecting dimension attributes and measures required to build the OLAP cube. It starts by a map phase that reads the input data using a stream from HDFS, then select, filter and emit the attributes involved in the OLAP cube computation. Since the fact and the dimensions are stored separately in two different files/collections, a reduce join operation is performed to ensure the link between documents instances. As the name implies, the join operation happens on the reduce phase side. Therefore, the mapper adds a tag to each tuple so that the reducer can distinguish where it comes from (Algorithm 1: line 5). The composite key consisting of the fact id and the tag forms the key to the partitioner function, while the measure (for the fact document) and dimension's attributes are mapped as the value. After partitioning tuples by the join key which is the fact id and sorting them by the tag value, a list of keys and associated values are generated and datasets having the same key are sent to the same reducer to be joined by fact id only (see Fig. 3)

Algorithm 1: MR-Cube – map function

1. **input:** - $(F\_id, a_i^D.value)$:
   - ∂: query predicate
2. **output**: $(compositKey, value_{tmp})$
3. **if** $a_i^D.value$ satisfy σ **do**
4. composit.key $\leftarrow F^S.\_id$
5. composit.tag $\leftarrow$ tag
6. $value_{tmp} \leftarrow a_i^D.value$
7. **end**
8. **emit** $(compositKey, value_{tmp})$

Algorithm 2: MR-Cube – reduce function

input: - compositKey (F_.id, tag)
- values: iterator

output: $(F\_.id, value_{step1})$

variables: $value_{step1} \leftarrow$ values.next()

begin

do

nextValue $\leftarrow$ values.next()

$value_{step1} \leftarrow value_{step1}$ +'',"+nextValue

while values.hasNextElement()

emit $(F\_.id, value_{step1})$

end

The second step: The second step: It consists of two MapReduce phases. The first one is a map task which takes the output entries of the first step and split the value part (of the key-value pair) into another key-value pair, where the key contains the parameters of the dimensions involved, and the value holds the measure to be aggregated. After the execution of the map function, a first aggregation is performed by the combiner. The use of a combiner in between map and reduces phases allows to reduce effectively the number of key-value pairs which will be transmitted to the shuffle/sort phase and consequently to the reducer. Reducing the number of key value pairs passing from one node to another helps optimize job execution, saves network from congestion and eases the reducer task. This later performs aggregation by key and writes the output in HDFS. At the end of this phase the first level of granularity is calculated. (see Fig. 4)

Algorithm 3: MR-Cube – map function

input: $(F\_.id, value_{step1})$ //output of step1

output: $(key_{tmp}, value_{tmp})$

variables: T: array of strings

begin

T=$value_{step1}$.split(‘’,2)

$key_{tmp} \leftarrow T[0]$

$value_{tmp} \leftarrow T[1]$

emit $(key_{tmp}, value_{tmp})$

end

Algorithm 4: MR-Cube – reduce function

input: $(key_{tmp}, value_{tmp})$

output: $(key_{step2}, value_{step2})$

variables: M: array of integers

begin

foreach v $\in value_{tmp}$

M.add(v)

end

$key_{step2} \leftarrow key_{tmp}$

$value_{step2}$=aggregate(M)

emit $(key_{step2}, value_{step2})$

The third step: Starting with the first level (REG, CUS), in this phase MR-Cube operator performs aggregation as per each dimension attribute separately, which correspond to (REG, ALL) and (ALL, CUS) levels. As mentioned earlier, at the end of the second step the first level of the OLAP cube is calculated and stored on disk. Recall that, tuples of the first level are key-value pairs where, the key is the concatenation of dimension attributes involved in the aggregation. At the map phase of the third step, this key is splitted to generate a new key value pair for each dimension attribute. This is followed by shuffle and sort phases, then, a list of output is sent to the reducer which combines all these values and performs aggregation for a specific key. After the completion of the reducer task, the second level of the OLAP cube is computed and sent back to the Hadoop file system.

Algorithm 5: MR-Cube – map function

input: $(key_{step2}, value_{step2})$
output: $(key_{tmp}, value_{tmp})$
variables: K: array of string
K ← key_{step2}.split(",")
begin
foreach k ∈ K
  key_{tmp} ← k
  $value_{tmp}$ ← $value_{step1}$
  emit $(key_{tmp}, value_{tmp})$
end
end

The fourth step: It corresponds to the final stage of the OLAP cube computation operator which performs the highest level of aggregation. The map function takes the output of the second step as input, and replaces all keys with the 'ALL' value. The reducer receives an iterator value corresponding to 'ALL' key, combines these values and performs aggregation to provide a single output key value pair corresponding to (ALL, Aggregation(measure)).

MapReduce computation paradigm has been the standard since Apache Hadoop's inception for several reasons: large scale, parallel processing and flexibility. After a Map or Reduce phase, MapReduce framework writes the intermediate results to disk. The data written to disk allows mappers and reducers to communicate with each other. It is also writing to disk that allows some fault tolerance. However, these input/output operations are costly and time-consuming. As the use of Hadoop has grown, its community has made room for other revolutionary architecture such as Spark which writes data on memory. The ability to cache intermediate result in memory has several important consequences on the processing speed of calculations as well as on the overall architecture of Spark. In the next section we look forward to use Spark instead of MapReduce algorithms to perform OLAP cube computation.

C. Spark-Cube

Apache Spark can be seen as the successor of MapReduce computation paradigm. While MapReduce operates in a series of discrete stages (map, shuffle, and reduce) which involve reading from and writing to disk at each stage, Spark embraces in-memory computation, which means it can persist intermediate data in memory between stages rather than writing to disk. That's why, theoretically, Spark's performance should be far better than MapReduce's, particularly for iterative algorithms or interactive analysis due to reduced data movement and disk I/O overhead. Fortunately, it is completely possible to re-implement MapReduce computation algorithm in Spark. The main innovation brought by Spark is the concept of Resilient Distributed Dataset (RDD). An RDD is a collection (to stick to our vocabulary) calculated from a data source (e.g., a data stream from Hadoop File System or another RDD)

Similarly, the process of computing the entire cube using Spark comprises four distinct steps. These steps are outlined below:

The first step: Spark reads text files from HDFS using an input stream from its configuration, then split and load it to multiple RDDs. Each RDD represents a dimension or a fact collection of documents, and only tuples that satisfy the query-filtering criteria are extracted. Another mapper processes the input RDD adding a tag to each element in order to distinguish its parent dataset. It then generates another RDD where the key is made up of the join key and the tag. Afterward, all RDDs are merged using a union operator to a unique RDD that contains the combination of different datasets sorted by the composite key (join key, tag) to ensure that a tuple from one collection comes before the other collection. The resulting RDD is mapped to a paired RDD which is the equivalent for a key-value pair in MapReduce. This allows the reducer to consider only the join key and to ignore the tag value, thus avoiding, for e.g., that two different reducers will be called for the same join key {key1, tag1}, {key2, tag2}. Finally, a reduceByKey function is applied to join the tuples having the same key.

The second step: In this step, an RDD transformation is applied on the output of the previous step. The result is a new RDD containing the combination of the dimension involved ({CusRegion, SupName}, revenue). Subsequently, an RDD action is performed to generate the first level of aggregation ({CusRegion, SupName}, Agg(revenue)). Following the aggregation process, this consolidated data is transmitted back to the driver node. This serves as a foundation for the subsequent computation of aggregation levels.

The third and fourth steps: These two concurrent steps run in parallel and are aimed at computing the second and the third aggregation levels corresponding to: ({CusRegion, ALL}, Agg(revenue)), ({ALL, SupName}, Agg(revenue)) and ({ALL, ALL}, Agg(revenue)). These aggregation operations are executed on an input Resilient Distributed Dataset (RDD) consisting of key-value pairs. This RDD serves as the foundational data structure for these computations, where keys represent the attributes or dimensions for aggregation and values hold the corresponding revenue data.

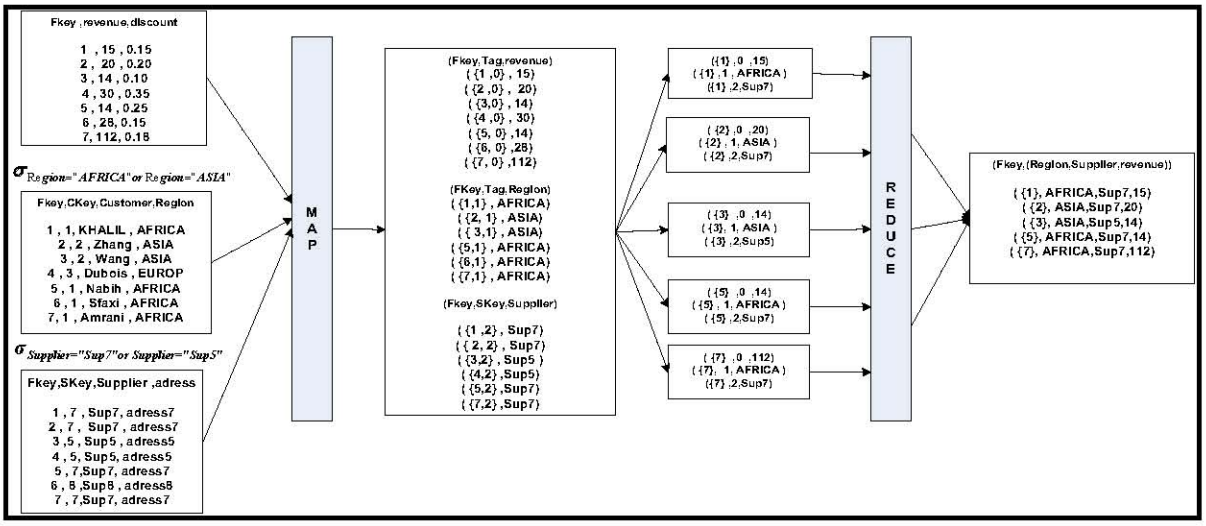

Fig. 3. Performing Reduce side join

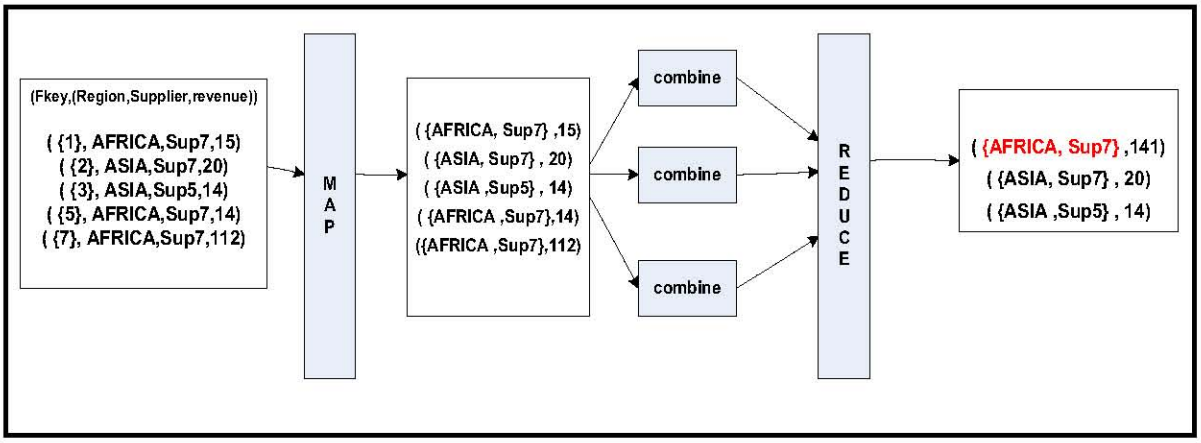

Fig. 4. Performing aggregation according to different combinations of dimension attributes

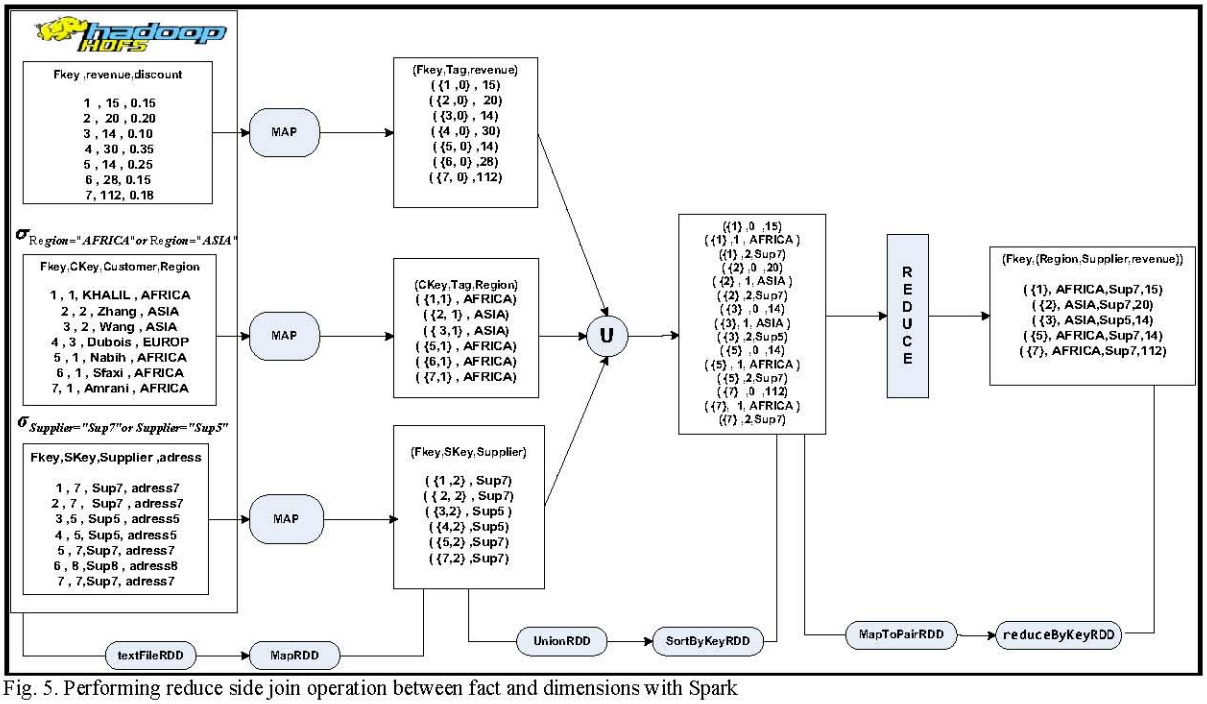

Fig. 5. Performing reduce side join operation between fact and dimensions with Spark

V. Implementation

In this section we run an implementation and report results of the experiments we conducted to validate our proposal using different test scenarios. The first experiment aims to measure the storage space metric at different scales between the proposed model and the default model of the star schema, in which the fact has references to associated dimensions (normalized model). In the second experiment we compare the execution time of the full OLAP cube building with MR-Cube and Spark-Cube from a data warehouse built using our model on one hand, then we use Apache Hive to perform cube building against the default model. The last experiment allows evaluating the response time to process analytical queries having different dimension numbers in grouping clauses.

Dataset: we used a big data benchmark generator which extends the TPC-H benchmark, a famous specification for testing decision support systems. It consists of the execution of business ad-hoc queries and concurrent data modification in order to report performance metrics of the system under test. For the experiments, we are interested in the data generation part. Basically, the official benchmark is compatible with relational databases only, that's why we used an extended version of the benchmark from the literature which is conceived to work with the most database solutions. Concretely, it is a Java application which generates data in different file formats and different scale factors. We adapted the source code to fit our proposed model. Data generation is performed by calling the DBGen Java class and loaded directly on HDFS.

Experimental environment: our experimental platform consists of 6 machines having an Intel-Core i7-6600U CPU @ 2.60GHz and running Debian-10.10 that communicate using 100 Mbps Ethernet switches. One of these machines hosts the HDFS NameNode and the JobTracker (head node), the other machines are worker nodes which host data nodes and TaskTrackers. It is worth mentioning that in a production environment, it is highly recommended having a duplicate of the NameNode which is eventually a single point of failure, otherwise if the NameNode goes down, the file system does the same.

A. First experiment

In the initial phase of our experimentation, we conducted the first experiment to assess the storage space demands of our proposed model in contrast to a normalized model. To gauge this, we varied the scalability factor, exploring three different scale factors: SF=5, SF=10, and SF=20. The primary focus of this experiment was to measure the system's capacity to efficiently store large volumes of data.

For this particular experiment, we opted not to employ a cluster setup, as it wasn't deemed necessary at this stage of our research. Instead, we utilized a single-node cluster configuration to carry out the evaluations. The results of this experiment, specifically the disk space requirements for both the proposed and normalized models, are visually represented in Fig. 6. These findings shed light on how our model compares in terms of storage efficiency under varying scalability conditions, providing valuable insights into its suitability for handling substantial data volumes.

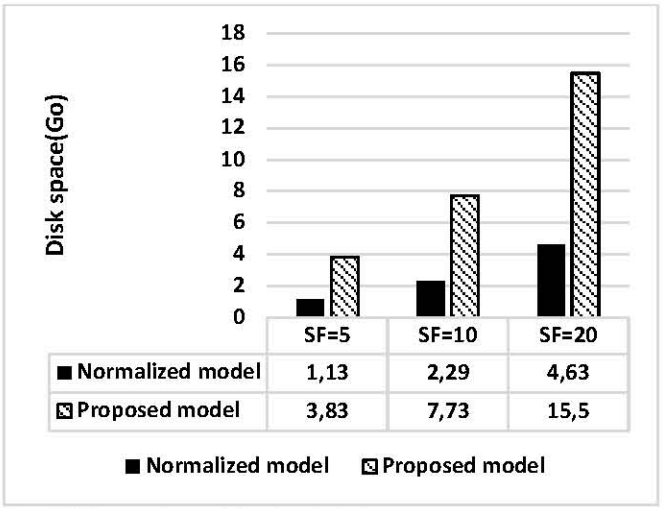

Fig. 6. Disk usage by model and scale factor

It's clear that the proposed model has a consequent storage cost due to data redundancy. Unlike the normalized model which is used by ROLAP systems, the proposed model uses a hierarchical data structure and duplicates dimension records to increase efficiency. As hard drives with big disk space are getting cheaper, storage cost is not a big issue anymore, and we are rather interested in data accessibility and reliability.

B. Second experiment

In this part we aim to compare the performance of MR-Cube and Spark-Cube. The experiment evaluates the full-OLAP cube computation on a cluster of machines described previously. We choose Apache Hive as a competitor. Our choice fell on Hive because it is an OLAP software based on MapReduce too, and it is often used to perform analytical queries from HDFS. In order to test the scalability of the system to handle a large data volume, we increased gradually the scalability factor which impacts the size of the generated data (see Fig. 7).

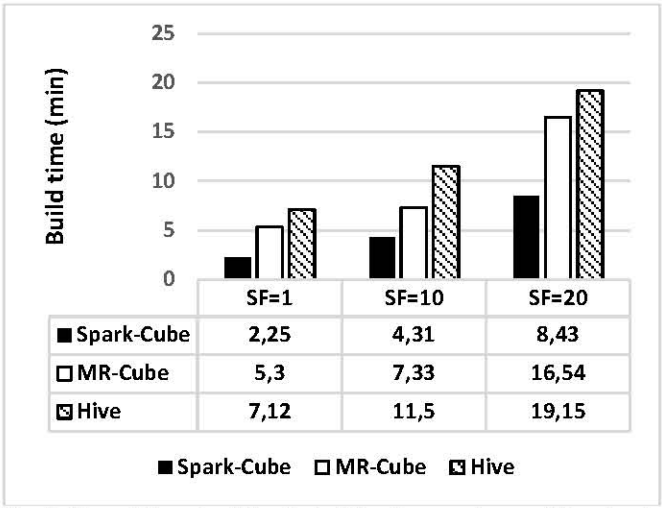

Fig. 7. Elapsed time for full cube building by operator on different scale factors.

Fig. 7 shows the elapsed time of the cube building by scale factors. From the graph we can observe that the execution time of the OLAP cube building increases proportionately with the size of the data warehouse. Furthermore, according

to Fig. 7, Spark-Cube is the fastest. For example, in a data warehouse size of 14G, Spark-Cube is 1x faster than MR-Cube and Hive. This is easily explained by the fact that in-memory processing speeds up the cube construction considerably, and allows Spark to outperform Hadoop MapReduce which is used by MR-Cube and Hive. More importantly, the results show that data storage and organization mode have a huge clout on OLAP cube building. Indeed, in the data model we are using, the fact that the dimensions inherit the key from the fact document allows to use a single join instead of multiple joins to link the OLAP schema components (the fact and its related dimensions), which reduces considerably the processing time.

Overall, we can conclude that Spark boosts the cube building and allows saving half time.

C. *Third experiment*

In this last experiment, we want to compare the response time of MR-Cube, Spark-Cube and Apache Hive to process queries with different dimensionalities at different scale factors using the same cluster of machines. Details about queries are reported in TABLE I.

TABLE I. QUERY SET DESCRIPTION

Number of dimensions	Dimension attribute	Predicate	Measure
2D	Customer: Nation Date: Year	Date: Year between 2010 and 2016	Sum(revenue)
3D	Customer: Nation Part: Type Date: Year		
4D	Customer: Nation Part: Type Supplier: Name Date: Month	Customer: Region= 'Africa'	

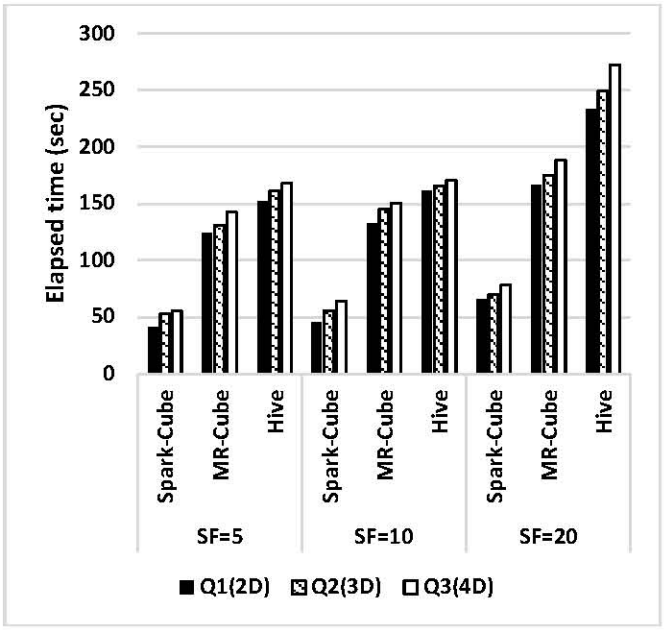

Fig. 8. Queries response time by operator faced with variation in scale factor

Fig. 8 shows time required for processing analytical queries with different dimension numbers involved. We observe that the size of the data warehouse and the variation of dimension numbers have a considerable impact on processing time. This observation is more obvious when performing queries with Apache Hive and by moving the scale factor from 5 to 10. Moreover, the speed of processing between Spark-Cube in one hand and MR-Cube and Hive in another hand differs significantly when scaling up. In fact, the read and write operations on disk performed by MapReduce jobs have a consequent execution time, while Spark keeps intermediate results in memory. More importantly, memory-based computation outperforms disk-based one for all data and query configurations. In other hand, and as expected, the number of join operations performed to fetch the query result influences the execution time (multiple joins led to low performance). Indeed, our algorithm which relies on one reduce-side join shows better performance with the equivalent implementation in Hive.

VI. CONCLUSION

On a broad scale, the process of storing and analyzing data is a complex task and a significant challenge for traditional data warehousing approaches. As a result, vendors of business intelligence solutions have begun adopting new technologies that promote distributed storage and parallel computing paradigms. This paper presents the design and implementation of a distributed document-oriented data warehouse, and aims to construct OLAP cubes using MapReduce and Spark. Our work focuses on formalizing the adopted document logical model by defining a set of migration rules from the multidimensional conceptual model. Additionally, we detail two operators for extracting OLAP cubes: MR-Cube and Spark-Cube.

In the conducted experiments, we evaluated the storage cost of the proposed model and compared the performance of MR-Cube and Spark-Cube with Apache Hive as a benchmark competitor. The experimental results showed that both operators outperform Apache Hive in terms of execution time. The logical model of the data warehouse is based on a parent-child relationship between the fact and its associated dimensions, facilitating faster join operations as the OLAP schema components share the same join key.

Undoubtedly, document-oriented storage and cluster computing offer interesting perspectives for data warehouses. In the short term, we plan to extend the experiments to include other Spark and MapReduce-based OLAP system competitors like Apache Kylin. In the medium term, we aim to study the implementation of data warehouses using graph-oriented databases.

REFERENCES

[1] C. W. Shen, "Factors of data infrastructure and resource support influencing the integration of business intelligence into enterprise resource planning systems," IJIIDS, vol. 9, no. 1, p. 1, 2015, doi: 10.1504/IJIIDS.2015.070822.

[2] S. M. Rosu, G. Dragoi, and M. Guran, "A Knowledge Management Scenario to Support Knowledge Applications Development in Small and Medium Enterprises," AECE, vol. 9, no. 1, pp. 8–15, 2009, doi: 10.4316/aece.2009.01002.

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

[4] E. J. Otoo, H. Wang, and G. Nimako, "Multidimensional Sparse Array Storage for Data Analytics," in 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), Sydney, Australia: IEEE, pp. 1520–1529, 2016, doi: 10.1109/HPCC-SmartCity-DSS.2016.0216.

[5] A. Cuzzocrea, I.-Y. Song, and K. C. Davis, "Analytics over Large-Scale Multidimensional Data: The Big Data Revolution!" in Proceedings of the ACM 14th International Workshop on Data Warehousing and OLAP, in DOLAP '11. New York, NY, USA: Association for Computing Machinery, pp. 101–104, 2011, doi: 10.1145/2064676.2064695.

[6] A. Cuzzocrea, L. Bellatreche, and I.-Y. Song, "Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions," in Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP, in DOLAP '13. New York, NY, USA: Association for Computing Machinery, pp. 67–70, 2013, doi: 10.1145/2513190.2517828.

[7] R. K. Batwada, N. Mittal, and E. S. Pilli, "Uncovering Data Warehouse Issues and Challenges in Big Data Management," in Big Data, Machine Learning, and Applications, R. Patgiri, S. Bandyopadhyay, M. D. Borah, and D. M. Thounaojam, Eds., Cham: Springer International Publishing, pp. 48–59, 2020, doi: https://doi.org/10.1007/978-3-030-62625-9_5.

[8] L. Duan and Y. Xiong, "Big data analytics and business analytics," Journal of Management Analytics, vol. 2, no. 1, pp. 1–21, 2015, doi: 10.1080/23270012.2015.1020891.

[9] I. Suh and Y. D. Chung, "A Workload Assignment Strategy for Efficient ROLAP Data Cube Computation in Distributed Systems," International Journal of Data Warehousing and Mining, vol. 12, no. 3, pp. 51–71, 2016, doi: 10.4018/IJDWM.2016070104.

[10] J. Song, C. Guo, Z. Wang, Y. Zhang, G. Yu, and J.-M. Pierson, "HaoLap: A Hadoop based OLAP system for big data," Journal of Systems and Software, vol. 102, pp. 167–181, 2015, doi: 10.1016/j.jss.2014.09.024.

[11] B. Arres, N. Kabbachi, and O. Boussaid, "Building OLAP cubes on a Cloud Computing environment with MapReduce," in 2013 ACS International Conference on Computer Systems and Applications (AICCSA), Ifrane, Morocco: IEEE, pp. 1–5, 2013, doi: 10.1109/AICCSA.2013.6616498.

[12] O. Romero, V. Herrero, A. Abelló, and J. Ferrarons, "Tuning small analytics on Big Data: Data partitioning and secondary indexes in the Hadoop ecosystem," Information Systems, vol. 54, pp. 336–356, 2015, doi: 10.1016/j.is.2014.09.005.

[13] 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., in Lecture Notes in Computer Science, vol. 9282, Cham: Springer International Publishing, pp. 79–91, 2015, doi: 10.1007/978-3-319-23135-8_6.

[14] L. C. Scabora, J. J. Brito, R. R. Ciferri, and C. D. de A. Ciferri, "Physical Data Warehouse Design on NoSQL Databases," in Proceedings of the 18th International Conference on Enterprise Information Systems, in ICEIS 2016. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, pp. 111–118, 2016, doi: 10.5220/0005815901110118.

[15] 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.

[16] I. Oditis, Z. Bicevska, J. Bicevskis, and G. Karnitis, "Implementation of NoSQL-based Data Warehouses," BJMC, vol. 6, no. 1, pp. 45–55, 2018, doi: 10.22364/bjmc.2018.6.1.04.

[17] 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.

[18] 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.

[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, 2020, doi: 10.4018/IJDSST.2020010101.

[21] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Columnar NoSQL CUBE: Agregation operator for columnar NoSQL data warehouse," in 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC), San Diego, CA, USA: IEEE, pp. 3828–3833, 2014, doi: 10.1109/SMC.2014.6974527.

[22] A. Khalil and M. Belaissaoui, "An Approach for Implementing Online Analytical Processing Systems under Column- Family Databases," IAENG International Journal of Applied Mathematics., vol. 53, no. 1, pp. 31-39, 2023.

[23] A. Khalil and M. Belaissaoui, "New approach for implementing big datamart using NoSQL key-value stores," in 2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech), Marrakesh, Morocco: IEEE, pp. 1–6, 2020, doi: 10.1109/CloudTech49835.2020.9365897.

[24] A. Khalil and M. Belaissaoui, "Key-value data warehouse: Models and OLAP analysis," presented at the 2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science, ICECOCS 2020, Institute of Electrical and Electronics Engineers Inc., pp. 1–6, 2020, doi: 10.1109/ICECOCS50124.2020.9314447.

[25] F. Davardoost, A. Babazadeh Sangar, and K. Majidzadeh, "Extracting OLAP Cubes from Document-Oriented NoSQL Database Based on Parallel Similarity Algorithms," Canadian Journal of Electrical and Computer Engineering, vol. 43, no. 2, pp. 111–118, 2020, doi:10.1109/CJECE.2019.2953049.

[26] M. Hanine, M. Lachgar, S. Elmahfoudi, and O. Boutkhoum, "MDA Approach for Designing and Developing Data Warehouses: A Systematic Review & Proposal," Int. J. Onl. Eng., vol. 17, no. 10, p. 99, 2021, doi: 10.3991/ijoe.v17i10.24667.

[27] H. Zhao and X. Ye, "A Practice of TPC-DS Multidimensional Implementation on NoSQL Database Systems," in Performance Characterization and Benchmarking, vol. 8391, R. Nambiar and M. Poess, Eds., in Lecture Notes in Computer Science, vol. 8391, Cham: Springer International Publishing, pp. 93–108, 2014, doi: 10.1007/978-3-319-04936-6_7.

Abdelhak Khalil received his engineering degree in computer science from the National School of Applied Sciences, Marrakesh, Morocco in 2014. He joined the Information Systems and Decision Support Laboratory (LEFCG-SIAD) of Hassan the First University in 2018. He participated in many international conferences and published many articles in international journals. His research interest includes business intelligence evolution in the big data era, big data analytics, value creation from big data, and cluster computing. He becomes a member of IAENG.

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.

Fouad Toufik is an assistant professor of computer science at Mohammed V University, Morocco. He works at Higher School of Technology SALE. He received his PhD degree in computer science from Hassan the First University in 2021. His interests in research focus on big data, parallel and distributed systems, cloud computing and artificial intelligence.