A Graph-oriented Framework for Online Analytical Processing

Abdelhak KHALIL[1]
LEFCG-SIAD Laboratory
Hassan First University of Settat
Settat, Morocco

Mustapha BELAISSAOUI[2]
LEFCG-SIAD Laboratory
Hassan First University of Settat
Settat, Morocco

***Abstract*—OLAP (Online Analytical Processing) is a tried-and-tested technology and a core concept in Business Intelligence. With data flowing from different and countless sources, exploring data in order to deliver actionable insights has become a daunting task with current OLAP tools despite the cycle of improvement that has gone through it. In the last decade, with the emergence of the big data phenomenon, NoSQL databases are seeing a spike in popularity and become more used in industry and academia as their value in handling a huge and varied amount of data become increasingly evident. Graph oriented database is one of the four chief types of NoSQL oriented databases that represent a promising technology candidate for big data analytics. In this paper we bring forward our contribution to graph-oriented analytical processing, which is twofold. First, we provide a novel approach for modeling a graph-oriented data warehouse. Second, we propose a data cube materialization through the precomputation of aggregated nodes. We present how typical OLAP queries can be performed against data warehouses stored in NoSQL graph-oriented database management systems. An implementation is conducted on a fictional data warehouse using Neo4j and the Cypher declarative language. The same dataset is stored in a relational data warehouse in order to compare storage space and query performance. Thus, the obtained results shows that graph OLAP implementation outperform clearly the relational alternative in term of query response time.**

***Keywords*—Graph OLAP; data warehousing; graph databases; NoSQL; data cube; decision support system**

I. INTRODUCTION

OLAP stands for (Online Analytical Processing) and describe a software technology dedicated to decision-making purpose. It is designed to locate meaningful intersections between multiple axes of analysis. The dimensional modelling is an integral part of OLAP systems and defines at the conceptual level the fact concept which holds measurements or metrics regarding a business process event, and the dimension concept which provides a context describing the fact. Data conversion from an OLTP (Online Transaction Processing) database of two-dimensional to the multi-dimensional model is done by an ETL (Extract, Transform, Load) tool. OLAP servers have historically been implemented mainly using four approaches: Relational-OLAP(ROLAP), Multidimensional-OLAP(MOLAP), Hybrid-OLAP(HOLAP) and Desktop-OLAP(DOLAP)[1], [2]. Each implementation has its strengths and its limitation and must be evaluated based on the business requirements.

With the IT revolution, and being aware of the potential of information, organizations around the globe has moved from the archaic age, which relies on industrial economy into a new era characterized by data driven economy. This race after technology in order to gain competitive advantages has contributed to the generation of large volumes of data. As a consequence, data analytics are becoming a huge challenge for traditional OLAP systems due its vertical scalability and its low computation ability. Indeed, earlier-generation of OLAP implementations are of poor storage and computational capacities, because they are built upon on old architectures and cannot match the requirement of big data analytics, especially data storage and data retrieval requirements. Another common problem is OLAP cube building over big data which could reach a critical complexity due to the increasing number of dimensions and the unstructured nature which characterize big data sets [3],[4].

To overcome the challenges of scale and complexity associated with today's data, OLAP researches moved in a new direction. Namely, the use of NoSQL databases in OLAP solutions which is considered as a promising alternative for traditional data storage tools [5]–[8], [9]. This revolutionary technology offers several interesting features that cannot be achieved with classical database management systems like cluster computing and the ability to process both semi-structured and unstructured data. In this paper, we are focused particularly in graph database, a class of NoSQL databases that uses a graph model composed of nodes and edges instead of relational model [10][11], and we claim that the graph data structure is suitable for data warehousing and online analysis.

Implementing an OLAP cube using a graph database is not a straightforward process. The multidimensional model used to instantiate the data cube must be converted to a logical model suitable to graph oriented database. Furthermore, typical OLAP queries must be translated to a specific language supported by this technology. The aim of this work is to illustrate the potentiality of graph databases to handle OLAP structures designed for reporting. In this context, we define a set of mapping rules in order to migrate dimensionally modelled data into the graph database. And we demonstrate how typical OLAP operations can be performed against a graph database. In Fig. 1, we position our proposal regarding the literature. The key contributions of this work can be summarized as follows:

- We propose an implementation of OLAP engines under graph database using two different logical models that are equivalent to ROLAP and MOLAP models. We define a set of rules used for the mapping from the multidimensional model to these models. An experiment is conducted to highlight the differences between the two meta-models using a case study.
- We propose an effective aggregation technique to build the lattice of cuboids from a data warehouse built upon a graph database management system.
- Then, we provide an extension of the declarative Cypher language to basic OLAP queries. We consider in this work Neo4j as a graph database engine.

The remainder of this paper is structured as follows. In the next section we present the background of our work, and we provide an overview of the state of the art related on Graph-OLAP. In Section III we present our modeling approach for graph OLAP. In Section IV we give an implementation of the proposed approach using the Cypher language. In Section V, we discuss experimental results. The last section concludes this work and suggests eventual research directions.

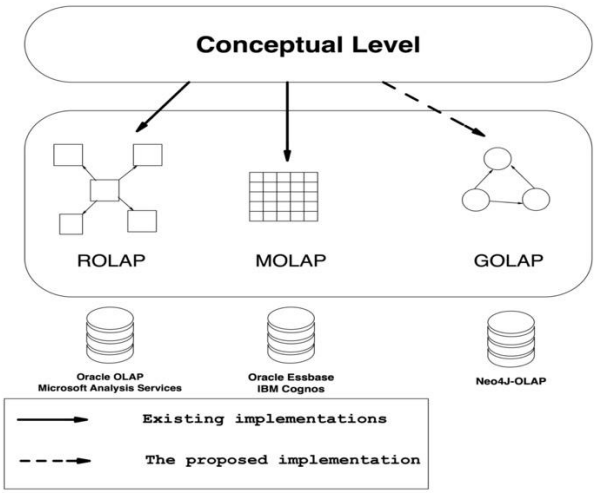

Fig. 1. Convestion from the Conceptual Level to different OLAP Implementations.

II. Related Work

A. The Multidimensional Schema

The multidimensional schema is the starting point to design and implement data warehouse systems. It defines four major concepts: fact, measures, dimensions and hierarchies [12].

Formally, a Multidimensional Schema denoted S is a triplet $\left(F^S, \mathrm{D}^S, Star^S\right)$ where:

- $F^S = \{F_1, ..., F_n\}$ a finite set of facts.
- $D^S = \{\mathrm{D}_1, ..., \mathrm{D}_m\}$ a finite set of dimensions.

$Star^S : F_i \rightarrow 2^{D_i}$ is an incidence function mapping each fact $\mathrm{F_i} \in \mathrm{F^S}$ to its associated dimensions $\mathrm{D_j} \in \mathrm{D^S}$.

A ***fact*** is the business process studied and is represented by a pair $\left(N^{F_i}, M^{F_i}\right)$ where:

- N^{F_i} is the name of the fact.
- $M^{F_i} = \left\{m_1^{F_i}, ..., m_n^{F_i}\right\}$ a finite set of measures.

A dimension $D_i \in D^S$ is defined by $\left(N^{D_i}, Att^{D_i}, H^{D_i}\right)$ where:

- N^{D_i} is the name of the dimension.
- $Att^{D_i} = \left\{a_1^{D_i}, ..., a_m^{D_i}\right\}$ a finite set of attributes.
- $H^{D_i} = \left\{l_1^{D_i}, ..., l_k^{D_i}\right\}$ a set of hierarchy levels.

A hierarchy organizes measures at different level of aggregations. A hierarchy level $l_j^{D_i} \in H^{D_i}$ can by defined by $(N^{l_j^{D_i}}, Att^{l_j^{D_i}}, Weak^{l_j^{D_i}})$ where:

- $N^{l_j^{D_i}}$ is the name of the hierarchy level.
- $Att^{l_j^{D_i}} = \left\{a_1^{l_j^{D_i}}, ..., a_m^{l_j^{D_i}}\right\}$ an ordered set of attributes.
- $Weak^{l_i^{D_i}} : a_j^{l_i^{D_i}} \rightarrow \{wa_1, ...wa_k\}$ is a function possibly associating parameters to a set of weak attributes.

B. The Graph Model

NoSQL graph-oriented database are based upon the concepts of graph model which organize data into collections of nodes and edges. Once data loaded, graph theory algorithms make it easy to handle semantic queries by calculating the shortest path between nodes. Graph database specify connections at insert time and avoid by then the problem of join index lookup performance as querying data becomes a matter of graph traversal. This makes graph engines optimum when the meta-model of data being stored has many overlapping relationships. This contrast with relational database which store the links between tables at the logical level and relies on relational algebra operations to manipulate the data stored in the database management systems in a relevant logical format.

Formally, a graph database denoted *G* is a set of properties $(N, E, \phi, L_N, L_E, P_N, P_E)$ comprising:

- *N* a set of nodes (also called vertices).
- $E \subseteq N \times N$ a set of edges (also called links).
- $\phi : E \rightarrow \{\{x, y\} \mid x, y \in N, x \neq y\}$ a function linking an edge to a pair of nodes.
- $L_N = \{l_1, ..., l_n\}$ a set of node labels.

- $L_E = \{l_1, ..., l_m\}$ a set of edge labels.
- $P_N = \{p_1^N, ..., p_j^N\}$ a set of node properties.
- $P_E = \{p_1^E, ..., p_k^E\}$ a set of edge properties.

A node $n_i \in N$ is a pair (l_i, a_{n_i}), where $l_i \in L_N$ is the node label, and $a_{n_i} = [a_1, ..., a_n]$ a set of attributes associated with the node. Identically an edge $e_j \in E$ is represented as $(l_j, a_{e_j}, \eta_x, \eta_y)$, where $l_j \in L_E$ the edge label , a_{e_i} a set of edge attributes, η_x the starting node and η_y the ending node.

C. Graph-Based OLAP

Over the last few years, big data analytics have known a meteoric adoption of NoSQL. Considerable attempts to model an OLAP cube with this technology have appeared. Several research works have been conducted to implement OLAP systems using columnar databases [5],[6],[7], others using the document-oriented database [8],[13],[14],[15],[16] and last but not least key-value stores [17],[18],[19].

Although graph databases are widely used in OLTP systems, especially when the need of modeling multiple connections is self-evident, it does not exist, to the best of our knowledge, any OLAP solution which uses a graph database at the physical level in the market. However, graph OLAP concept has been around for years. Indeed, some interesting works attempted to implement OLAP systems using graph technology. A decade ago, Chen et al.[20], [21] studied the possibility to perform multi-dimensional analysis on graph data, the authors developed a graph OLAP framework having two major subcases: Informational OLAP and Typological OLAP and proposed the basic definition of OLAP operations under this framework.

Many recent research works have been interested in implementing OLAP engines under property graph databases. In [22], the authors introduce a new data warehousing concept called Graph Cube which stands for an OLAP infrastructure that support analytical queries over a multidimensional network. In [23], the authors define the concept of GOLAP which is an extension of Online Analytic Processing(OLAP) under graph database, some features are listed such as semantics queries and structural analytics. In this work the authors address the challenges of speed and storage related to GOLAP and proposes possible solution to deal with them like graph data reduction and query result approximation when the execution time is too long, unfortunately the authors did not provide an implementation of the proposed framework and focus rather on the possible formalization. In [24], the authors propose a novel graph cube framework called Two-Step Multi-dimensional Heterogeneous(TSMH) which consists of an Entity Hyper Cube and Dimension Cube. In the Entity Hyper Cube n-meta path relation algorithm is used to guide the aggregation of the network and to extend drill-down/roll-up operations. In the Dimension Cube the efficiency of dimension operation is improved by using a hierarchical coding for entity type and dimensions.

Along the same vein, in [25] the author proposed an OLAP data structure that relies on typed nodes to store facts and dimensions, and introduced an extension of the Cypher language to basic OLAP queries. The authors didn't provide any experimental campaign to validate their proposal as they rather focused on the demonstration of its feasibility. In [26], [27], the authors proposed a formal multidimensional data model for graph analysis based on node and edge-labeled called graphoids, and presented a proof of concepted implementation using a Neo4j graph database.

Regarding the instantiation of data warehouses using property graph database, in [28] the authors define a set of transformation rules for mapping between the multidimensional conceptual model and NoSQL graph model.

All the cited works present an interesting background for graph-based online analytical processing. The majority of them addressed the issue of the adaptation of graph structure to OLAP needs. Although they share some similarities with ours, the contribution of this work is quite different as we propose a novel approach for implementing both a data warehouse and OLAP engine based on efficient data cube materialization over graph database.

III. Graph Olap Model

OLAP engines have been traditionally categorized whether they perform pre-computation of OLAP cuboids or not. Following this taxonomy, OLAP systems where all part of the cube is pre-computed and stored in memory or disk are called multidimensional OLAP systems (MOLAP) and systems where computation of OLAP cuboids is performed on-demand directly from the data warehouse are considered as Relational OLAP models (ROLAP).

In this section we define the logical graph model for data warehousing. We consider two approaches by analogy to the ROLAP and MOLAP models; each one differs in term of structure and content when the mapping from the conceptual model is performed. In the first approach, fact, dimensions and the link between them are materialized by nodes and edges following several mapping rules, while in the second approach we talk rather about an aggregate lattice modeled using the graph paradigm. In what follows, we will use a fictional electronics company as a running example. The star schema of our cube is depicted in Fig. 2:

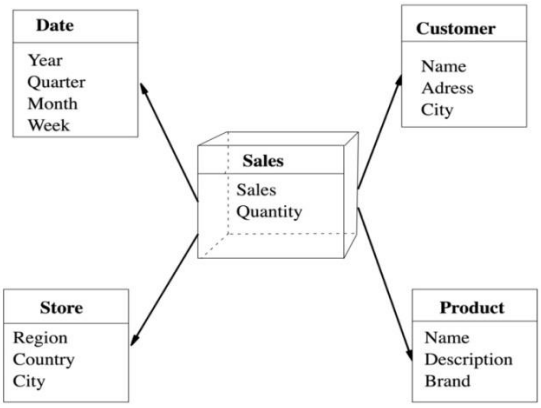

Fig. 2. The Star Schema.

A. First Approach

This approach corresponds to the lightly summarized data model. It defines a meta-model in which each component (fact and its associated dimensions) will be transformed to a node. The relation between nodes will be materialized by edges as detailed by the following mapping rules:

Rule.1. Each fact component $F_i \in F^S$ is converted to a node defined by $\left(l_i, a_{n_i}\right)$ where:

- l_i is the name of the fact.
- Each measure $m_k^{F_i} \in M^{F_i}$ is converted to a node attribute $a_k \in a_{n_i}$.

Rule.2. Each dimension component $D_j \in D^S$ is translated to a node defined by $\left(l_i, a_{n_i}\right)$ where:

- Each dimension attribute $a_k^{D_i} \in Att^{D_i}$ is mapped into a node attribute $a_k \in a_{n_i}$.
- Each hierarchy level $l_k^{D_i} \in H^{D_i}$ will be stored as a node alike dimension.
- Hierarchy levels are connected by edges to express how they are hierarchically linked.

Rule.3. The link between fact and its associated dimensions is represented by an edge (l_i, η_x, η_y) where:

- $l_j \in L_E$ is the name of the relation.
- η_x a node representing the fact.
- η_y a node representing an associated dimension

For the star schema represented in Fig. 2, the application of the aforementioned rules will give us the following meta-model, Fig. 3:

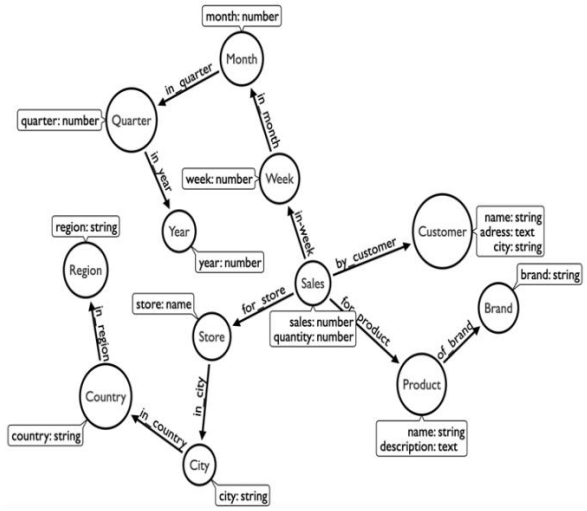

Fig. 3. The Graph-OLAP Schema According the the First Appraoch.

B. Second Approach

When we want to perform aggregation on a graph OLAP built according to the first approach, the query we should write is served on-demand and relies on fact nodes which are retrieved then aggregated using an aggregation function. This technique achieves the required result, but it is not optimized for a large data volume. Moreover, it is tending to the opposite of OLAP philosophy where data aggregation is pre-computed and stored.

The second approach corresponds to a highly summarized data model where measure aggregations are pre-calculated and directly available for the sake of query performance. The set of pre-computed aggregations is called an *aggregate lattice*. Concretely, fact measures are aggregated according to different combinations of dimensions and stored as a node with two labels.

- l_i identify the multidimensional concept l_i *='Aggregate.*
- l_j a label which follows a particular pattern that identify uniquely which cuboid the aggregate is calculated for. This label is in the form of a bitmask starting with a letter that indicate the type of the aggregate (S for Sum, A for Average, etc.). The remaining part is an ordered sequence of n position (one of each hierarchy level), each position can have three possible values: (x) if the aggregate is calculated for all occurrences of the level, (1) if the aggregate is performed for each occurrence of the level and (0) if the aggregate is not calculated for the level.

If we refer to our running example and considering only high levels of granularity. Let's assume by convention that the order of position levels is:

Product.Brand-Product.Product-Store.Region-Store.Country-Date.Year-Date.Quarter.

An example of bitmask construction is depicted in Table I and Fig. 4 displays such a representation:

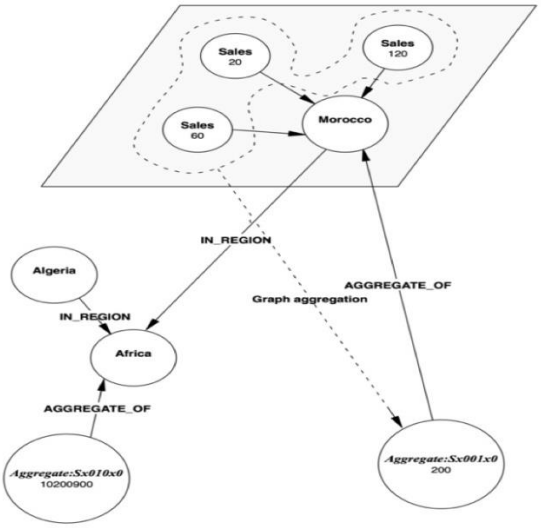

Fig. 4. .Graph Aggregation According to the Second Approach.

TABLE I. CUBOID BITMASK CONSTRUCTION

Node label	Description	Scope
Aggregate:S10x0x0	Sum by Product.Brand	One level aggregate
Aggregate:Sx010x0	Sum by Store.Region	One level aggregate
Aggregate:Ax0x010	Avg by Date.Year	One level aggregate
Aggregate:S1010x0	Sum by Product.Brand and Store.Region	Two levels aggregate

IV. IMPLEMENTATION

A. Answering Typical Analytical Operation using Cypher

OLAP operations help users to view data from different perspectives providing a convenient environment for real-time data visualization and analysis. OLAP defines several basic operations; the most popular ones are roll-up, dicing and slicing. In this section we present how these operators can be expressed over a data cube designed according to the first approach.

Queries are written using the Cypher syntax, a declarative query language intended to be executed on a database engine built on the graph model. Cypher relies on the concept of pattern matching for querying and updating graphs [12]. A detailed description of the Cypher syntax is beyond the scope of this paper.

1) Roll-up: The roll-up operation (also called consolidation or aggregation operation) performs aggregation on a data cube in two ways, either by reducing the number of dimensions or by climbing up a concept hierarchy for a dimension. It is like zooming-out feature from the most detailed granularity level to the less detailed one.

In the query given in Listing.1, the rollup operation is performed by climbing up the concept hierarchy of *Product* dimension (*Product* → *Brand*), and of *Store* dimension (*Store* → *City*). The execution of the query results in the creation of a node containing the aggregated measures and two new relations linking the created node with its associated dimension hierarchies.

Listing. 1. Roll up-Aggregation of sales and quantities by product brand and store city.

```
1. MATCH (br:Brand)<-[]-(prod:Product)<-[]-(fact:Sales)
2. MATCH (ct:City)<-[]-(st:Store)<-[]-(fact:Sales)
3. WITH DISTINCT br, ct, SUM(fact.sales) AS SumSales, SUM(fact.quantity) As
   SumQuantity
4. CREATE (br)<-[:AGGREGATE_OF]-(agg:Aggregate:S1010x0 {sales: SumSales,
   quantity: SumQuantity})-[:AGGREGATE_OF]->(ct)
5. RETURN br,agg,ct;
```

2) Dicing: Dicing is the operation of selecting a subset over all the dimensions and picking only specific dimension parameter values. We can think of dicing as zoom feature using smaller scale.

In Listing.2 the dice operation is performed using a selection criterion over *Brand* and *Year* dimensions. The generated cube has two dimensions.

Listing. 2. Dice-Selecting the sum of sales for the brand Apple in 2018.

```
1. MATCH (br:Brand {brand: 'Apple'})<-[*]-(fact:Sales)
2. MATCH (year:Year {year: 2018})<-[*]-(fact:Sales)
3. RETURN SUM(fact.sales) AS Sales, SUM(fact.quantity) AS Quantity;
```

3) Slicing: Slicing is similar to dicing with a little difference. It emphasizes one specific dimension and provides a new sub-cube by filtering on a particular attribute. It can be considered as a specialized filter for specific dimension parameter value.

In Listing.3 Slice is carried out for the dimension *Region* using the criterion *Region*= '*Asia'*.

Listing. 3. Slice- Selecting the sum of sales in region Asia

```
1. MATCH (reg:Region) <-[*]-(fact:Sales)
2. WHERE reg.name='Asia'
3. RETURN reg.region AS Region, SUM(meas.sales) AS Sales,
   SUM(meas.units) AS Units;
```

B. Aggregates Creation

We refer to the property graph in Fig. 3 and the set of aggregates in Table I, and then we show how we can perform pre-calculation of our sample cuboids.

1) Aggregate by product brand: Query results in cypher are evaluated by its core concept, namely, pattern matching. By using patterns, you describe the requested data shape, then the Cypher engine is responsible for restoring the data you are looking for. For example, to build the aggregate value *Aggregate:S10x0x0*, a join is implemented by means of matching *Sales* → *Brand* against the OLAP-graph. It is worth noting that the edge label linking the fact and the dimension nodes is not required as it is inferred from node types.

In SQL, this is equivalent to a join between the fact table *Sales* and the dimension table *Brand* followed by the aggregation function SUM and GROUP By clause over *Brand* attributes.

Listing. 4. Creation of the aggregate *Aggregate:S10x0x0.*

```
1. MATCH (brand:Brand)<-[*2]-(s:Sales)
2. WITH DISTINCT brand, SUM(s.sales) AS SumSales, SUM(s.quantity) AS
   SumQuantity
3. CREATE (a:Aggregate:S10x0x0 {sales: SumSales, quantity: SumQuantity})-
   [:AGGREGATE_OF]->(brand);
```

Fig. 5 shows how the aggregate *Aggregate:S10x0x0* (By product brand) fits in the property graph (colored in grey). It is a one-level aggragate as it is calculated against one hierachical level(colored in red).

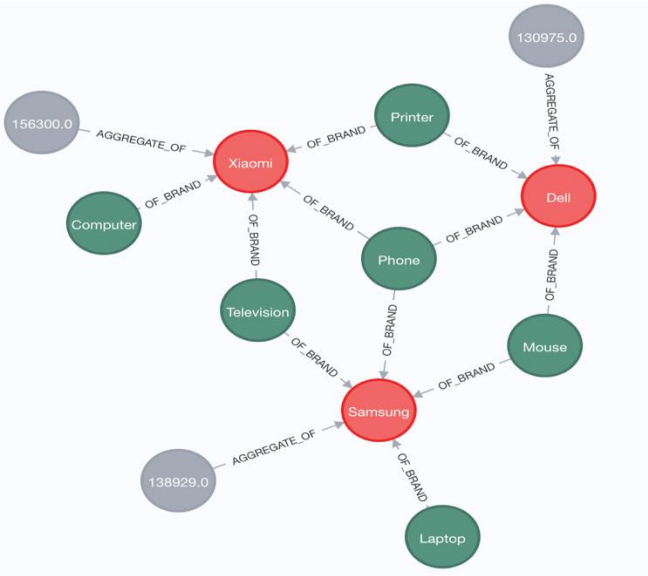

Fig. 5. Graph Visualization for Aggregates of Product Brands

2) Aggregate by region: In Listing. 5, the aggregate node *Aggregate:Sx010x0* *(*by region*)* is created:

Listing. 5. Creation of the aggregate *Aggregate:Sx010x0*

```
1. MATCH (r:Region)<-[*4]-(s:Sales)
2. WITH DISTINCT r, SUM(s.sales) AS SumSales, SUM(s.quantity) AS
   SumQuantity
3. CREATE (a:Aggregate:Sx010x0 {sales: SumSales, quantity: SumQuantity})-
   [:AGGREGATE_OF]->(r);
```

3) Aggregate by year

Listing. 6. Creation of the aggregate *Aggregate:Sx010x0*

```
1. MATCH (y:Year)<-[*3]-(s:Sales)
2. WITH DISTINCT y, AVG(s.sales) AS AvgSales, AVG(s.quantity) AS
   AvgQuantity
3. CREATE (a:Aggregate:Ax0x010 {sales: AvgSales, quantity: AvgQuantity})-
   [:AGGREGATE_OF]->(y);
```

4) Aggregate by product brand and region: In Listing.7, the two-levels aggregate node *Aggregate:S1010x0 (by product* brand *and* region*)* is created:

Listing. 7. Creation of the aggregate *Aggregate:S1010x0*

```
1. MATCH (brand:Brand)<-[*2]-(s:Sales)
2. MATCH (r:Region)<-[*4]-(s:Sales)
3. WITH DISTINCT brand, r, SUM(s.sales) AS SumSales, SUM(s.quantity) AS
   SumQuantity
4. CREATE (brand)<-[:AGGREGATE_OF]-(a:Aggregate:S1010x0 {sales:
   SumSales, quantity: SumQuantity})-[:AGGREGATE_OF]->(r);
```

Increasing the materialization of the aggregates can improve considerably query performance, but can also affect drastically storage space since aggregate nodes are stored on disk. The precalulation of all possible aggregate values is often not needed. Generally, OLAP engines chose the percentage of precompted values based on business needs, the remaining aggregates are calculated in response to a query. We can imagine a scenario in which potentially requested aggregates are infered from log files that contains previously executed queries.

V. Results and Discussion

We conducted experiments to evaluate two aspects for the OLAP implementation under graph database: storage space and query performance. For this, the solution we propose is compared with a ROLAP implementation under Oracle relational database containing the same dataset. The experiment is carried out on a Unix machine (macOS) having a core-i7 CPU,16GB of RAM and 1 TB of stockage memory and running Neo4j community edition v4.3.

A. Data Generation

The dataset used in the experiment is generated using a novel NoSQL star schema benchmark named KoalaBench [29], [30], [30]. This tool is developed with Java language and is derived from the reference benchmark TCP-H. For clarity and to fit the meta-model in our running example the Supplier is replaced with the Store dimension, LineItem is renamed with Sales, and for the equivalent graph model, only few dimension parameters are tracked. Datasets can be generated in different configurations (different file format including tab, csv, json, xml..., and multiple models). The size of the generated data by scale factor is detailed in Table II.

TABLE II. Size of the Data Generated by Scale Factor (Source[1])

		Lines	***Disk Space in Byte (SF=1)***	***Avg. Disk space/line (Byte)***
Tables	Sales (LineItem)	*SF*x6000000	862558617,6	143,76
	Product (Part)	*SF*x200000	28521267,2	142,6
	Customer	*SF*x150000	16043212,8	1069,54
	Store (Supplier)	*SF*x10000	1677721,6	167,77
	Nation	25	367	14,68
	Region	5	73,4	14,68
	Date	*SF*x2556	168522	65,93
Size on disk			0,85 GB	-

B. Experiment 1: Memory Consumption Per Scale Factor

In this experiment we use a global flat CSV file representing data in a flat meta-model. In the appendix (Listing.8), we attach the Cypher script for loading data from an CSV file into Neo4j database according to our modeling approach. A fragment of the generated graph is represented in Fig. 6. The number of nodes and edges for the corresponding graph is depicted in Table III.

[1] http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

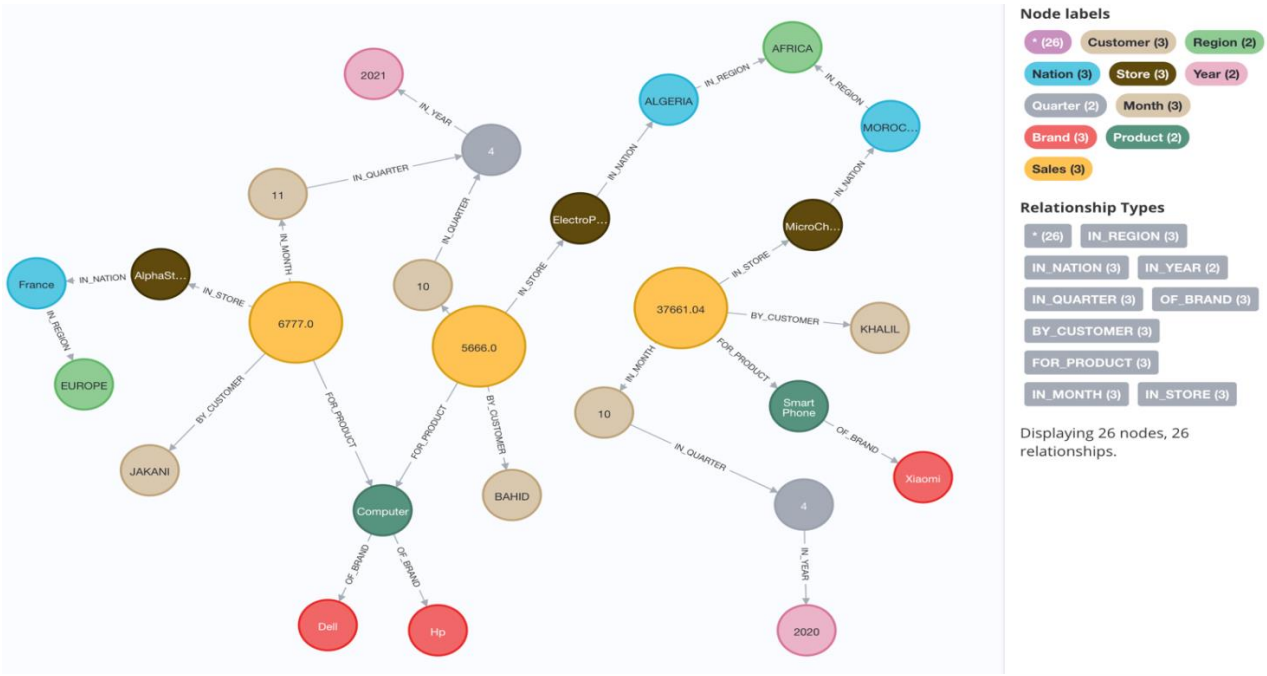

Fig. 6. A Portion of the Graph-OLAP.

TABLE III. MEMORY USAGE FOR THE GRAPH MODEL ON DIFFERENT SCALE FACTORS

		Sf=1	*Sf=5*	*Sf=10*
Nodes	Sales	6000000	30000000	60000000
	Product	200000	1000000	2000000
	Brand	100	500	1000
	Customer	150000	750000	1500000
	Store	10000	50000	100000
	Nation	25	25	25
	Region	5	5	5
	Month	79	395	790
	Quarter	27	135	270
	Year	7	35	70
Edges	FOR_PRODUCT	6000000	30000000	60000000
	OF_BRAND	200000	1000000	2000000
	BY_CUSTOMER	6000000	30000000	60000000
	IN_STORE	6000000	30000000	60000000
	IN_NATION	10000	50000	100000
	IN_REGION	25	25	25
	IN_MONTH	6000000	30000000	60000000
	IN_QUARTER	79	395	790
	IN_YEAR	27	135	270
Size on disk		3,3 GB	16.6 GB	33.2 GB

From the Table III, we can see that a snowflake schema on a graph database requires more storage space than in a relational one (more than 3 times for SF=1). This is easily explained: property graph databases store relationships physically on disk using edges while the concept of foreign key is used instead by relational databases. Furthermore the metadata is stored individually for each record in graph database unlike relational model which define the structure of the data at a higher level(the table itself). Which means that property names are repeated for each item. Indeed, graph databases are very storage intensive. This is traded for higher query performance. Since nowadays hard disks are inexpensive, it woud be worthwhile trade-off to buy more storage space than keeping users waiting.

C. Experiment 2: Query Performance

The purpose of this experiment is to measure empirically the performance of graph-OLAP to process analytical queries when scaling up in comparison with the ROLAP implementation under Oracle database. We have exposed the system to a scale factor equal to 10 wich generates 11,6 Go of random data in csv file format. Query configuration includes queries involving gradually an increasing number of dimensions as depicted in Table IV. Each query was executed three times and the average of the elapsed time is presented in Fig. 7.

Experiment results show that the relational implementation defeats the Graph alternative when the query involves one dimension, but when the query dimensionality increases the graph alternative show better performance ranging from 1,82 to 2,29 times faster. Indeed, in relational databases the deeper we go in joining tables the more queries show slower processing time because it requires scanning of all table involved in the query which has a considerable cost. Unlike relational databases which suffer the pain of joining tables, graph databases express relationship at the physical level. That means, the links between nodes exists physically on disk and are named and directed which, makes graph traversal easier.

TABLE IV. QUERY CONFIGURATION

Query	Dimensionality	Dimension attributes	Measure
Q1	1D	Date:year	Sum(sales)
Q2	2D	Product:name Store:region	Sum(sales)
Q3	3D	Product:name Store:region Date:month	Sum(sales)
Q4	4D	Product:name Store:region Date:quarter Customer:name	Sum(sales)

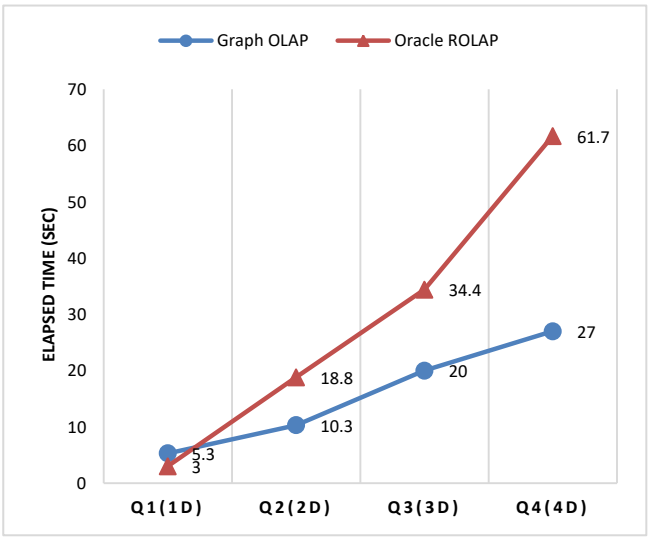

Fig. 7. Query Response Time by Dimensionality.

VI. CONCLUSION

The ability of graph technology to handle highly interconnected data makes it suitable for interactive analysis and more relevant for businesses today. In this paper, we addressed the topic of extending NoSQL graph-oriented databases to OLAP. We have proposed a modeling approach for implementing graph-based data warehouses using labeled nodes and edges. We have also shown how materialized aggregates can pre-computed across different levels to speed up query processing. At the physical level Neo4J engine is used as a graph-oriented database management system. Typical OLAP queries are rewritten using its declarative query language Cypher.

The Graph-OLAP implementation is compared to ROLAP one in terms of query performance and storage space, results show clearly that graph implementation of OLAP presents better performances than relational alternative in term of query response time when facing a huge data volume.

In the forthcoming extended work, we look forward to extending Cypher to support OLAP features by writing a user-defined aggregation function using the low-level API provided by Neo4J engine.

Without any doubt, using NoSQL technology to support OLAP features is a promising research direction. Therefore, we claim that implementing OLAP engines under column-oriented and document-oriented databases using novel frameworks would be an interesting research issue that can be addressed.

REFERENCES

[1] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," ACM SIGMOD Rec., vol. 26, no. 1, pp. 65–74, Mar. 1997, doi: 10.1145/248603.248616.

[2] A. Nanda, S. Gupta, and M. Vijrania, "A Comprehensive Survey of OLAP: Recent Trends," in 2019 3rd International conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, Jun. 2019, pp. 425–430. doi: 10.1109/ICECA.2019.8822203.

[3] 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, New York, NY, USA, 2013, pp. 67–70. doi: 10.1145/2513190.2517828.

[4] M. Tremblay and A. Hevner, "Missing Data in OLAP Cubes: Challenges and Strategies," J. Database Manag., vol. 32, p. 1, Jun. 2021, doi: 10.4018/JDM.2021070101.

[5] K. Dehdouh, O. Boussaid, and F. Bentayeb, "Big Data Warehouse: Building Columnar NoSQL OLAP Cubes," Int. J. Decis. Support Syst. Technol., vol. 12, no. 1, pp. 1–24, Jan. 2020, doi: 10.4018/IJDSST.2020010101.

[6] K. Dehdouh, F. Bentayeb, O. Boussaid, and N. Kabachi, "Using the column oriented NoSQL model for implementing big data warehouses," Int. Conf. Parallel Distrib. Process. Tech. Appl. PDPTA15, pp. 469–475, 2015.

[7] M. Boussahoua, O. Boussaid, and F. Bentayeb, "Logical Schema for Data Warehouse on Column-Oriented NoSQL Databases," in Database and Expert Systems Applications, vol. 10439, D. Benslimane, E. Damiani, W. I. Grosky, A. Hameurlain, A. Sheth, and R. R. Wagner, Eds. Cham: Springer International Publishing, 2017, pp. 247–256. doi: 10.1007/978-3-319-64471-4_20.

[8] 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," Proc. - Int. Conf. Res. Chall. Inf. Sci., vol. 2016-Augus, 2016, doi: 10.1109/RCIS.2016.7549351.

[9] Z. Challal, W. Bala, H. Mokeddem, K. Boukhalfa, O. Boussaid, and E. Benkhelifa, "Document-oriented versus Column-oriented Data Storage for Social Graph Data Warehouse," 2019, pp. 242–247. doi: 10.1109/SNAMS.2019.8931718.

[10] C. Kamphuis, "Graph Databases for Information Retrieval," in Advances in Information Retrieval, Cham, 2020, pp. 608–612.

[11] A. Bhattacharyya and D. Chakravarty, "(Graph Database: A Survey)," in 2020 International Conference on Computer, Electrical & Communication Engineering (ICCECE), Kolkata, India, Jan. 2020, pp. 1–8. doi: 10.1109/ICCECE48148.2020.9223105.

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

[13] F. Davardoost, A. Babazadeh Sangar, and K. Majidzadeh, "Extracting OLAP Cubes from Document-Oriented NoSQL Database Based on Parallel Similarity Algorithms," Can. J. Electr. Comput. Eng., vol. 43, no. 2, pp. 111–118, 2020, doi: 10.1109/CJECE.2019.2953049.
S. Bouaziz, A. Nabli, and F. Gargouri, "Design a Data Warehouse Schema from Document-Oriented database," Procedia Comput. Sci., vol. 159, pp. 221–230, 2019, doi: 10.1016/j.procs.2019.09.177.

[14] E. Gallinucci, M. Golfarelli, and S. Rizzi, "Approximate OLAP of document-oriented databases: A variety-aware approach," Inf. Syst., vol. 85, pp. 114–130, Nov. 2019, doi: 10.1016/j.is.2019.02.004.

[15] M. L. Chouder, S. Rizzi, and R. Chalal, "EXODuS: Exploratory OLAP over Document Stores," Inf. Syst., vol. 79, pp. 44–57, Jan. 2019, doi: 10.1016/j.is.2017.11.004.

[16] A. Khalil and M. Belaissaoui, "New approach for implementing big datamart using NoSQL key-value stores," presented at the Proceedings of 2020 5th International Conference on Cloud Computing and Artificial

Intelligence: Technologies and Applications, CloudTech 2020, Nov. 2020. doi: 10.1109/CloudTech49835.2020.9365897.

[17] 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, Dec. 2020. doi: 10.1109/ICECOCS50124.2020.9314447.

[18] 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. Cham: Springer International Publishing, 2014, pp. 93–108. doi: 10.1007/978-3-319-04936-6_7.

[19] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu, "Graph OLAP: a multi-dimensional framework for graph data analysis," Knowl. Inf. Syst., vol. 21, no. 1, pp. 41–63, Oct. 2009, doi: 10.1007/s10115-009-0228-9.

[20] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu, "Graph OLAP: Towards Online Analytical Processing on Graphs," in 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, Dec. 2008, pp. 103–112. doi: 10.1109/ICDM.2008.30.

[21] P. Zhao, X. Li, D. Xin, and J. Han, "Graph cube: on warehousing and OLAP multidimensional networks," in Proceedings of the 2011 international conference on Management of data - SIGMOD '11, Athens, Greece, 2011, p. 853. doi: 10.1145/1989323.1989413.

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

[23] P. Wang, B. Wu, and B. Wang, "TSMH Graph Cube: A novel framework for large scale multi-dimensional network analysis," in 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA), Campus des Cordeliers, Paris, France, Oct. 2015, pp. 1–10. doi: 10.1109/DSAA.2015.7344826.

[24] A. Castelltort and A. Laurent, "NoSQL graph-based OLAP analysis," KDIR 2014 - Proc. Int. Conf. Knowl. Discov. Inf. Retr., pp. 217–224, 2014, doi: 10.5220/0005072902170224.

[25] L. Gómez, B. Kuijpers, and A. Vaisman, "Performing OLAP over Graph Data: Query Language, Implementation, and a Case Study," in Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, Munich Germany, Aug. 2017, pp. 1–8. doi: 10.1145/3129292.3129293.

[26] L. Gómez, B. Kuijpers, and A. Vaisman, "Online analytical processsing on graph data," Intell. Data Anal., vol. 24, no. 3, pp. 515–541, May 2020, doi: 10.3233/IDA-194576.

[27] A. Sellami, A. Nabli, and F. Gargouri, "Transformation of Data Warehouse Schema to NoSQL Graph Data Base," in Intelligent Systems Design and Applications, vol. 941, A. Abraham, A. K. Cherukuri, P. Melin, and N. Gandhi, Eds. Cham: Springer International Publishing, 2020, pp. 410–420. doi: 10.1007/978-3-030-16660-1_41.

[28] M. Chevalier, M. El Malki, A. Kopliku, O. Teste, and R. Tournier, "Benchmark for OLAP on NoSQL technologies comparing NoSQL multidimensional data warehousing solutions," Proc. - Int. Conf. Res. Chall. Inf. Sci., vol. 2015-June, no. June, pp. 480–485, 2015, doi: 10.1109/RCIS.2015.7128909.

[29] M. El Malki, A. Kopliku, E. Sabir, and O. Teste, "Benchmarking Big Data OLAP NoSQL Databases," in Ubiquitous Networking, vol. 11277, N. Boudriga, M.-S. Alouini, S. Rekhis, E. Sabir, and S. Pollin, Eds. Cham: Springer International Publishing, 2018, pp. 82–94. doi: 10.1007/978-3-030-02849-7_8.

APPENDIX

Listing. 8. Script loading in Neo4J

```
UNWIND ["sales-sf1.csv"] AS sourceFile
LOAD CSV WITH HEADERS
FROM "file:///" + sourceFile
AS row
FIELDTERMINATOR ';'
MERGE (cus:Customer {cname: row.c_name})
MERGE (r:Region {region: row.s_region_name})
MERGE (n:Nation {nation: row.s_nation_name})
MERGE (st:Store {store: row.s_name})
MERGE (n)-[:IN_REGION]->(r)
MERGE (st)-[:IN_NATION]->(n)
WITH date(row.o_orderDate) AS date,row,st,cus
MERGE (y:Year {year: toInteger(date.year)})
MERGE (q:Quarter {year: date.year, quarter: date.quarter})
MERGE (m:Month {year: date.year, month: date.month, quarter:date.quarter})
MERGE (q)-[:IN_YEAR]->(y)
MERGE (m)-[:IN_QUARTER]->(q)
MERGE (b:Brand {brand: row.p_brand})
MERGE (prod:Product {product: row.p_name})
MERGE (prod)-[:OF_BRAND]->(b)
WITH
st, m, prod, row,cus,
st.store + '_' + toString(m.year) + '_' + toString(m.month) + '_' + prod.product+ '_' +
cus.cname AS SalesID
MERGE (f:Sales {fid: SalesID})
ON CREATE
SET f.sales = toFloat(row.sales),
f.quantity = toInteger(row.quantity)
ON MATCH
SET f.sales = f.sales + toFloat(row.sales),
f.quantity = f.quantity + toInteger(row.quantity)
MERGE (f)-[:IN_STORE]->(st)
MERGE (f)-[:IN_MONTH]->(m)
MERGE (f)-[:FOR_PRODUCT]->(prod)
MERGE (f)-[:BY_CUSTOMER]->(cus);
```