Improving the performance of querying multidimensional RDF data using aggregates

Franck Ravat, Jiefu Song, Olivier Teste, Cassia Trojahn dos Santos

Improving the performance of querying multidimensional RDF data using aggregates*

Franck Ravat
IRIT - Université Toulouse I
Toulouse Cedex 09, France
franck.ravat@irit.fr

Jiefu Song
IRIT - Université Toulouse I
Toulouse Cedex 09, France
jiefu.song@irit.fr

Olivier Teste
IRIT - Université Toulouse II
Blagnac Cedex, France
olivier.teste@irit.fr

Cassia Trojahn
IRIT - Université Toulouse II
Toulouse Cedex 9, France
cassia.trojahn@irit.fr

ABSTRACT

In this paper, we propose a novel approach to tackle the problem of querying large volume of statistical RDF cubes. Our approach relies on combining pre-aggregation strategies and the performance of NoSQL engines to represent and manage statistical RDF data. Specifically, we define a conceptual modeling solution to represent original RDF data with aggregates in a multidimensional structure. We complete the conceptual modeling with a logical design process based on well-known multidimensional RDF graph and property-graph representations. We implement our proposed model in RDF triple stores and a property-graph NoSQL database, and we compare the querying performance, with and without aggregates. Exper-imental results, on real-world datasets containing 81.92 million triplets, show that pre-aggregation allows reducing query runtime in both RDF triple stores and property-graph NoSQL databases. Neo4j NoSQL database with aggregates outperforms RDF Jena TDB2 and Virtuoso triple stores, speeding up to 99% query runtime.

KEYWORDS

multidimensional graph data, querying performance, pre-computed aggregates

1 INTRODUCTION

Multidimensional models have been largely studied in the database community and maturity on optimising multidimensional querying has been mostly reached. In the Semantic Web, exploiting such multidimensional views on RDF data has received attention over the last ten years and a growing number of RDF data sources relying on such view has being published on the Linked Open Data[1]. These data cube models[2] answer to the need of analysing statistical RDF data from different perspectives and levels of granularity [10].

However, querying multidimensional data generates high workloads on SPARQL engines. Despite the different efforts on optimising native RDF triple stores [20, 21, 27], the scalability of SPARQL engines is still limited owing further development and optimization [28]. Concurrently, in recent years, a number of new data management systems, broadly known as property-graph NoSQL databases, have offered a performing alternative to SPARQL engines. Such systems have emerged as an infrastructure for handling large amount of data outside the RDF space [8].

Different works have addressed the follow-up question on how the performance of these NoSQL engines perform compared to RDF engines [3, 8, 14]. While most of the NoSQL engines scale more gracefully than the RDF stores, the findings in these works also point out weaknesses of NoSQL engines in dealing with complex SPARQL queries involving several joins or containing complex filters on large volume of data [8]. Furthermore, as stated in [14], NoSQL engines like Neo4j could better isolate queries such that one poorly performing query does not cause a domino effect and benefit from better query algorithms.

One raised question is how triple stores performance would compare with that of NoSQL technologies, when exploiting aggregates. Our hypothesis here is a) aggregating data (i.e. summarised numeric values with grouping conditions and aggregation functions) and b) taking advantage of NoSQL performances would better scale than triple stores when dealing with large amounts of data. Here, differently from graph summarisation [7, 12, 18, 30], which mostly focus on structurally grouping vertices or edges, our aggregation strategy is based on creating new vertices whose content results from the calculating of numeric values from a set of vertices through aggregation functions. Moreover, while works have studied the

https://doi.org/10.1145/3297280.3297506

[1] https://www.w3.org/2011/gld/wiki/Data_Cube_Implementations
[2] https://www.w3.org/TR/vocab-data-cube/

problem of aggregating RDF data in RDF triple stores like [15], to the best of our knowledge, this problem has not been addressed using property-graph NoSQL systems. This has been addressed in other NoSQL engines such as MongoDB [2].

In this paper, (a) we propose a conceptual modeling solution to represent original RDF data with aggregates according to a multidimensional structure. The proposed conceptual modeling is generic enough to serves as a basis for representing graph-like multidimensional data with aggregates; (b) we couple this model with a logical translating process. The translating process takes as input well-known multidimensional RDF data and produces a conceptual graph enriched with pre-computed aggregates; (c) we compare the performance of querying data with and without aggregates. To do so, we implement the conceptual multidimensional graph in two RDF triple stores and a property-graph NoSQL system based on a real-world RDF datasets containing 81.92 millions triples.

To the best of our knowledge, this is the first systematic study of aggregated graph data including conceptual modeling, mapping rules and experimental evaluation on RDF triple stores and property-graph NoSQL engines. Our empirical results open interesting directions in ontology-based data access (OBDA) with respect to accessing data on NoSQL instead of relational databases, as mostly done in the literature [4].

The remainder of the paper is organised as follows. Section 2 summarises the related work. Section 3 introduces our multidimensional model. Section 4 presents the experimental results. Finally, Section 5 concludes the paper and describes the future work.

2 RELATED WORK

In this section we describe the main related work on (i) using NoSQL for managing RDF data, (ii) multidimensional analysis over RDF data, and (iii) graph summarisation and RDF aggregates.

NoSQL databases and RDF triple stores. Various works have investigated different aspects on using NoSQL databases to manage RDF data. In [8], the performance of NoSQL bases (HBase, Couchbase and Cassandra) is evaluated on two RDF benchmarks (Berlin SPARQL Benchmark and DBpedia SPARQL Benchmark). While most NoSQL systems scale more gracefully than RDF stores, complex SPARQL queries involving several joins, on large volumes of data, or containing complex filters perform poorly on NoSQL systems. In [14], the authors compare the performance of triple stores, relational and graph databases for querying Wikidata. Two SPARQL triple stores (Virtuoso and Blazegraph), one relational database (PostgreSQL), and one graph database (Neo4J) have been evaluated. They show that NoSQL could better isolate queries such that one poorly performing query does not cause a domino effect and benefit from better query algorithms. In [25], the Gremlinator system translates SPARQL queries to path traversals for executing graph pattern matching over graph databases. It provides the foundation for a hybrid use of RDF triple stores and property graph such as Neo4J, Sparksee and OrientDB. Finally, in [19], they address the problem of data integration by considering NoSQL MongoDB. A set of transformation rules is applied to translate MongoDB documents to SPARQL. A SPARQL query is transformed into a pivot abstract query based on the xR2RML mapping of the target database to RDF. As [8, 14] we address the evaluation of NoSQL systems as Neo4j (differently from [19]) for storing RDF data, but here we focus on a specific kind of data (multidimensional statistical data). While they have pointed out some weakness of these systems, we argue here that materialising aggregates can provide better querying performance. As [25], we consider both storage supports (triple stores and NoSQL databases) but do not combine them together.

Multidimensional analysis over RDF data. Researches on this topic mainly address different aspects of analytical queries over RDF data. In [28], the proposal maps typical OLAP operations to SPARQL and a tool named ASPG automatically generates OLAP queries from real-world Linked Data. The works in [13, 22] focus on generating SPARQL queries based on OLAP analysis, requiring data sets described in QB or QB4OLAP vocabularies. In [11], a high-level query language (COL) that operates over cubes is proposed. Using the metadata provided by QB4OLAP, COL queries are translated into SPARQL, exploiting SPARQL query optimisation techniques.

Close to ours, [15] compare the performance of SPARQL and of ROLAP SQL queries and measure the gain of RDF aggregate views that materialise parts of the RDF data cube. In that experiment, while RDF aggregate views show the capability to optimise query execution, yet, overall still take six times longer for pre-processing and not nearly reach the performance gain of aggregate tables in ROLAP. Here, we could observe that globally the aggregates can have a positive impact in the querying performance. It is important to note, however, that our experiments compared to [15] have not been executed on the same basis. On one hand, we propose a new conceptual modeling solution to include pre-computed aggregates in a multidimensional graph. On the other hand, we base our experiments on a new technical environment including both triple stores and a property-graph NoSQL database.

To address also performance, the approach in [23] is based on a refactoring of analytical queries expressed in the relational-like SPARQL algebra based on a set of logical operators. This refactoring enables parallel evaluation of groupings and aggregations, particularly beneficial for scale-out processing on distributed cloud systems. Contrary to these works, we address the performance of analytical queries exploiting materialisation of aggregates.

Graph summarisation and aggregates in RDF. Graph summarisation has been extensively studied in the literature [18]. Two main categories of approaches can be distinguished: (1) *aggregation approaches* [26, 29], which rely on strategies for grouping the graph nodes into groups based on diverse functions (e.g., similarity of values of attributes, relationships to adjacent nodes, application of aggregation functions); and (2) *structural approaches* [5, 6, 31] which rely on extracting a schema representing a summary of the graph, in general, based on equivalence relations of nodes. The approach we propose here falls into the former. This is the same for the work in [26], which produces a summary graph of K-groups by grouping nodes based on user-selected node attributes and relationships. This approach however is limited to graphs describing entities characterised by the same set of attributes. Our approach is not limited to this kind of graphs, given that RDF graphs are heterogeneous by nature. In the second category, [31] summarise graphs following a top-K approximate RDF graph pattern strategy, aiming at guiding the user in the formulation of his queries. A similar approach is adopted in [5], where sumarisation relies on a

generic graph model defining the notion of node collections i.e., set of nodes sharing similar characteristics. In [16], graphs from the LOD cloud are summarised, focusing on the distribution of classes and properties across LOD sources. The summaries are based on a mechanism that combines text labels and bisimulation contractions. The labels assigned to RDF graphs are hierarchical, enabling summarisation at different granularities. In [9] 'interesting insights' in (generic) RDF graph are automatically identified by the systems as RDF aggregate queries. The system ranks such insights and plots the most interesting ones as bar charts, and shows them to the user. Finally, hybrid approaches are proposed in [29, 30], which take into account both attribute aggregation and structure summarisation of graphs. These works rather focus on topological summarisation of graph aiming at helping users to extract and understand main characteristics of a graph. Unlike these works, our approach differs from those by nature, since it works on aggregating numeric values within vertices. Other works on RDF summarization include LODeX [1], ABSTAT [24] and SchemEX [17]. LODeX is a tool that produces a representative summary of a LOD source. Similarly to [16], the summary reports statistical and structural information regarding the LOD Dataset (number of instances of classes and attributes). Contrary to us, we apply aggregation on the values of properties. Under a different view, SchemEX extracts a concise LOD schema with a structure to be used as an index, where schema extraction means to abstract RDF instances to RDF schema concepts that represent instances with the same properties. Finally, ABSTAT takes the RDF data summarisation problem as to provide a compact but complete representation of a data set, where every relation between concepts that is not in the summary can be inferred. It adopts a minimalization mechanism based on minimal type patterns. Here, we do not exploited the inferences in our computations.

3 CONCEPTUAL MODELING OF MULTIDIMENSIONAL GRAPH WITH AGGREGATES

Here, we introduce a conceptual model based on the concept of graphs to generically represent multidimensional RDF data with their associated aggregates. This model aims to be independent of a specific storage structure (e.g. triple store, DBMS, etc.) First, we present the model and illustrate it with an example. Second, we present a logical translating process to illustrate how to build our proposed model from statistical RDF data and how the resulted model can be managed by different implementation environments.

3.1 Multidimensional Graph

A multidimensional graph enriches classical conceptual graph modeling with pre-computed aggregates. It contains individuals (ABox) organised according to three multidimensional concepts (TBox), namely numeric indicators (*measures*), descriptive properties (*attributes*) and summarised numeric indicators with grouping conditions and an aggregate function (*aggregates*).

Definition 3.1. A **multidimensional graph** is a bipartite graph composed of measures, attributes and aggregates. It is defined as (V, E) where

- $\mathtt{V}$ is a set of vertices such that $\mathtt{V} = \mathtt{M} \cup \mathtt{A} \cup \mathtt{G}$, where $\mathtt{M}$ is a set of vertices corresponding to measures, $\mathtt{A}$ is a set of vertices corresponding to attributes (parameters) and $\mathtt{G}$ is a set of vertices corresponding to aggregates. $\mathtt{M} = \{e_i^{\mathtt{M}}\}_{1 \leq i \leq |\mathtt{M}|}$ is the set of measure values, $\mathtt{A} = \{e_j^{\mathtt{A}}\}_{1 \leq j \leq |\mathtt{A}|}$ is the set of attribute values, $\mathtt{G} = \{e_k^{\mathtt{G}}\}_{1 \leq k \leq |\mathtt{A}|}$ is the set of aggregates
- $\mathtt{E} \subseteq \mathtt{V} \times \mathtt{V}$ is a set of edges such that $\mathtt{E} = \mathtt{E}_{MA} \cup \mathtt{E}_{AA} \cup \mathtt{E}_{MG} \cup \mathtt{E}_{AG} \cup \mathtt{E}_{GG}$. $\mathtt{E}_{MA}$ is an edge between a measure value $e_i^{\mathtt{M}}$ and an attribute value $e_j^{\mathtt{A}}$, $\mathtt{E}_{AA}$ is an edge between two attribute values $e_i^{\mathtt{A}}$ and $e_j^{\mathtt{A}}$, $\mathtt{E}_{MG}$ is an edge between a measure value $e_i^{\mathtt{M}}$ and an aggregate $e_i^{\mathtt{G}}$, $\mathtt{E}_{AG}$ is an edge between an attribute value $e_j^{\mathtt{A}}$ and an aggregate $e_i^{\mathtt{G}}$, $\mathtt{E}_{GG}$ is an edge between two aggregates $e_i^{\mathtt{G}}$ and $e_j^{\mathtt{G}}$

The relationships between different types of vertices and edges are presented in Figure 1.

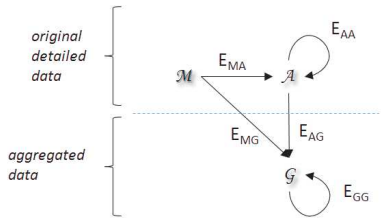


Figure 1: Relationships between different types of vertices and edges

Example. Figure 2 gives an example of a multidimensional graph. It is composed of one measure denoted *cost*, and one *geographical* analysis axis. The measure contains 3 values (i.e. $e_1^{\mathtt{M}}, e_2^{\mathtt{M}}, e_3^{\mathtt{M}}$), while the analysis axis is composed of 4 attributes (*city*, *region*, *country* and *Geography*) with 7 instances (i.e. $e_1^{\mathtt{A}}, e_2^{\mathtt{A}}, e_3^{\mathtt{A}}, e_4^{\mathtt{A}}, e_5^{\mathtt{A}}, e_6^{\mathtt{A}}, e_7^{\mathtt{A}}$). The last attribute *Geography* represents the maximal (the most general) granularity, called *ALL*. The aggregates of the measure according to different attributes are stored into one measure denoted *cost_sum*. The formal representation of this multidimensional graph is as follows:

- $\mathtt{V} = \mathtt{M} \cup \mathtt{A} \cup \mathtt{G}$ where $\mathtt{M} = \{e_1^{\mathtt{M}}, e_2^{\mathtt{M}}, e_3^{\mathtt{M}}\}$, $\mathtt{A} = \{e_1^{\mathtt{A}}, e_2^{\mathtt{A}}, e_3^{\mathtt{A}}, e_4^{\mathtt{A}}, e_5^{\mathtt{A}}, e_6^{\mathtt{A}}, e_7^{\mathtt{A}}\}$, $\mathtt{G} = \{e_1^{\mathtt{G}}, e_2^{\mathtt{G}}, e_3^{\mathtt{G}}, e_4^{\mathtt{G}}\}$
- $\mathtt{E} = \mathtt{E}_{MA} \cup \mathtt{E}_{AA} \cup \mathtt{E}_{AG}$ where $\mathtt{E}_{MA} = \{(e_1^{\mathtt{M}}, e_1^{\mathtt{A}}), (e_2^{\mathtt{M}}, e_2^{\mathtt{A}}), (e_3^{\mathtt{M}}, e_3^{\mathtt{A}})\}$, $\mathtt{E}_{AA} = \{(e_1^{\mathtt{A}}, e_4^{\mathtt{A}}), (e_2^{\mathtt{A}}, e_4^{\mathtt{A}}), (e_3^{\mathtt{A}}, e_5^{\mathtt{A}}), (e_4^{\mathtt{A}}, e_6^{\mathtt{A}}), (e_5^{\mathtt{A}}, e_6^{\mathtt{A}}), (e_6^{\mathtt{A}}, e_7^{\mathtt{A}})\}$, $\mathtt{E}_{MG} = \{(e_1^{\mathtt{M}}, e_1^{\mathtt{G}}), (e_2^{\mathtt{M}}, e_1^{\mathtt{G}}), (e_3^{\mathtt{M}}, e_2^{\mathtt{G}})\}$, $\mathtt{E}_{AG} = \{(e_4^{\mathtt{A}}, e_1^{\mathtt{G}}), (e_5^{\mathtt{A}}, e_2^{\mathtt{G}}), (e_6^{\mathtt{A}}, e_3^{\mathtt{G}}), (e_7^{\mathtt{A}}, e_4^{\mathtt{G}})\}$, $\mathtt{E}_{GG} = \{(e_1^{\mathtt{G}}, e_3^{\mathtt{G}}), (e_2^{\mathtt{G}}, e_3^{\mathtt{G}}), (e_3^{\mathtt{G}}, e_4^{\mathtt{G}})\}$.

Let $\{\preccurlyeq_1, \ldots \preccurlyeq_d\}$ a set of binary relations over $\mathtt{A}$. Each $\preccurlyeq_i$ is a binary relation over $\mathtt{D}^i_{AA} \subseteq \mathtt{A}$ defining an ordered set. The subsets $\mathtt{D}^i_{AA}$ are disjoints; $\forall i \in [1..d], j \neq i \in [1..d], \mathtt{D}^i_{AA} \cap \mathtt{D}^j_{AA} = \varnothing$.

Attributes within a multidimensional graph can be organised according to analysis axes (i.e. dimensions) :

Definition 3.2. A dimension is defined by $(\mathtt{D}^i_{AA}, \preccurlyeq_i)$ where

- $\mathtt{D}^i_{AA} \subseteq \mathtt{A}$ is a set of attributes,
- $\preccurlyeq_i$ is an ordered set over $\mathtt{D}^i_{AA}$ satisfying
  - irreflexivity: $\nexists e_j^{\mathtt{A}}, e_j^{\mathtt{A}} \preccurlyeq_i e_j^{\mathtt{A}}$;

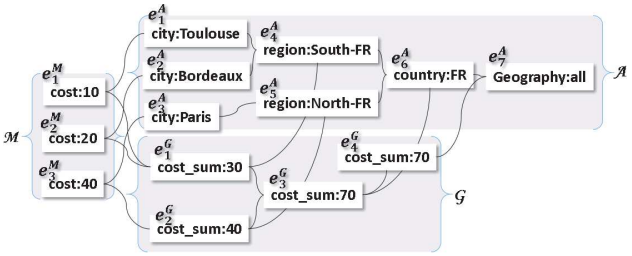


Figure 2: Example of multidimensional graph

- transitivity: if $e^{A}_{j1} \preccurlyeq_i e^{A}_{j2}$ and $e^{A}_{j2} \preccurlyeq_i e^{A}_{j3}$, then $e^{A}_{j1} \preccurlyeq_i e^{A}_{j3}$
- asymmetry: if $e^{A}_{j1} \preccurlyeq_i e^{A}_{j2}$ then not $e^{A}_{j2} \preccurlyeq_i e^{A}_{j1}$.

Attributes associated together through binary relations form one or several aggregation paths (i.e. hierarchies) within a dimension. A measure can be summarised along a hierarchy by zooming in (drilling down) or zooming out (rolling up).

Example. In the previous example, the multidimensional database is composed of one dimension with one hierarchy of four granularities (city, region, country and ALL_GEO), such as

- $\mathtt{D}^{Geography}_{AA} = \{e^{A}_1, e^{A}_2, e^{A}_3, e^{A}_4, e^{A}_5, e^{A}_6, e^{A}_7\}$,
- $\preccurlyeq_{Geography}$ is an ordered set such as $e^{A}_1 \preccurlyeq_{Geography} e^{A}_4$; $e^{A}_2 \preccurlyeq_{Geography} e^{A}_4$; $e^{A}_3 \preccurlyeq_{Geography} e^{A}_5$; $e^{A}_4 \preccurlyeq_{Geography} e^{A}_6$; $e^{A}_5 \preccurlyeq_{Geography} e^{A}_6$; $e^{A}_6 \preccurlyeq_{Geography} e^{A}_7$.

3.2 Designing a Multidimensional Graph from statistical RDF data

Based on the conceptual modeling of a multidimensional graph, we propose a designing process to pre-compute and materialise aggregates from statistical RDF data (cf. Figure 3). First, we convert statistical RDF data without aggregates at the physical level into the conceptual Multidimensional Graph (arrow 1). During this step, aggregates are computed and combined with original data in a conceptual Multidimensional Graph. Second, we convert a conceptual Multidimensional Graph into well-known RDF models and the property-graph model at the logical level (arrow 2). At last, based on the logical representation, original statistical RDF as well as pre-computed aggregates can be implemented in triple stores and property-graph NoSQL databases (arrow 3). In this section, we focus on the first and second steps of our proposed designing process. The last step will be discussed in Section 4.

Step 1. The first objective is to converting original RDF data into a conceptual Multidimensional Graph and enrich original data with all possible aggregates. To do so, we depict an algorithm which (a) takes a statistical RDF dataset in QB or QB4OLAP vocabulary as input and (b) produces a conceptual Multidimensional Graph including original data and aggregates at output (cf. algorithm 1).

The algorithm 1 first identifies attributes on each dimension according to QB (lines 3 and 4) and QB4OLAOP (line 6) vocabularies and creates attribute vertices in the Multidimensional Graph accordingly (lines 8-10). Then, it creates edges E_{AA} between attributes linked together through standardized (`skos:narrower`) or

Algorithm 1: Designing a Multidimensional Graph based on original statistical RDF data

input : Original RDF data
output: Conceptual aggregate multidimensional graph

1 **foreach** *dimension* `?dim` *in original RDF data, such as* `?dim a qb:DimensionProperty.` **do**
2 create a dimension D_i *in the multidimensional graph*;
3 **if** *there exists* `?dim qb:codeList ?lst` **then**
4 identify each attribute `?att` *within* `?dim` *, such as*
- `?lst skos:hasTopConcept ?att.` or
- `?lst qb:hierarchyRoot ?att.`
5 **else**
6 identify each attribute `?att` *within?dim* (`?level a qb4o:LevelProery. ?level qb4o:inDimension ?dim`), *such as* `?att qb4o:inLevel ?level` ;
7 **end**
8 **foreach** *identified attribute* `?att` *in original RDF data* **do**
9 create an attribute vertex a_i *such as* $a_i \in A, A \subseteq V$;
10 **end**
11 create an edge E_{AA} *between* `?att` *and* `?attUp`, *such as*
- `?attUp skos:inSchema ?lst; skos:narrower ?att.` or
- `?lst a qb:HierarchicalCodeList; qb:parentChildProperty ?p2c. ?attUp skos:inSchema ?lst; ?p2c ?att.`

12 **end**
13 **foreach** *measure* `?m` *in original RDF data, such as* `?m a qb:Observation` **do**
14 create an measure vertex m_i *such as* $m_i \in M, M \subseteq V$;
15 identify attribute `?att` *on dimension* `?dim` (`?dim a qb:DimensionProperty`) *associated with* `?m`, *such as*
- `?m ?dim ?att.` or
- `?level a qb4o:LevelPropery; qb4o:inDimension ?dim. ?att qb4o:inLevel ?level. ?m ?dim ?att.`

create an edge E_{MA} *between* `?att` *and* `?m` ;
16 **end**
17 **foreach** *measure* $m_i \in M$ **do**
18 calculate aggregated values of m_i *according to the related attributes on each dimension* $2^{A_{D_1} \times \ldots \times A_{D_n}}$, *where* $A_{D_i} \subseteq A$ *is the set of related attribute vertices on* D_i, *such as* $\forall a_j \in A_{D_i}$, a_j *is associated with* m_i *through one* E_{MA} *edge or through a set of* E_{AA} *edges and one* E_{MA} *edge* ;
19 create an aggregate vertex g_i *such as* $g_i \in G, G \subseteq V$;
20 create an edge E_{MG} *between* m_i *and* g_i. ;
21 create an edge E_{AG} *between each attribute* a_{g_i} *in the grouping conditions and* g_i. ;
22 create an edge E_{GG} *between related aggregate vertices* g_i *and* g_j, *such as there exists an attribute* a_{g_i} *associated with* g_i *and an attribute* a_{g_j}, $a_{g_i} \preccurlyeq_{D_k} a_{g_j}$;
23 **end**

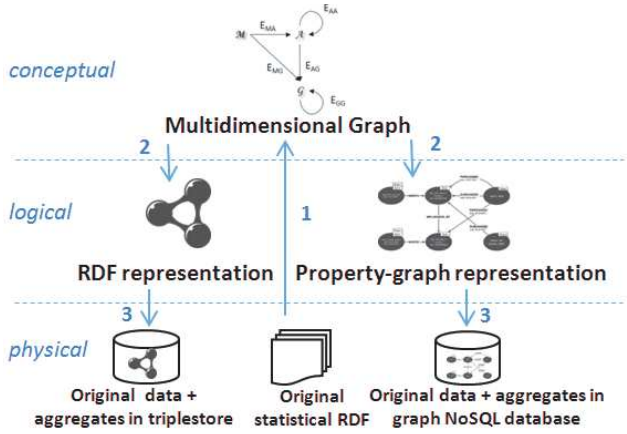


Figure 3: Designing process of Multidimensional Graph at the conceptual, logical and physical levels

customized (?p2c[3]) relations (line 11). Next, the algorithm creates measure vertices in the Multidimensional Graph (lines 13-16). At last, it calculates measure values according to attributes of different granularities on each dimension and adds pre-computed aggregates in the Multidimensional Graph (lines 17-22).

Step 2. The second objective is to transform a conceptual Multidimensional Graph including both original data and aggregates into RDF and property-graph models at the logical level. In the case of an RDF model, we have defined an underlying RDF vocabulary supporting the translation of our Multidimensional Graph model into the corresponding RDF representation. Note that a pre-computed aggregate can be represented as a `qb:Observation` and its related edges can be represented through predicates (e.g., figure 4).

```
aggregate vertex  eg:agg1 a qb:Observation;
                      MDGraph:value 96.3^^xsd:float;
E_AG edge         ->  MDGraph:dimGeo "Paris"^^xsd:String;
                      MDGraph:dimYear "2018"^^xsd:String;
E_GG edge         ->  MDGraph:GG eg:agg2;
E_MG edge         ->  MDGraph:MG eg:m1, eg:m2, eg:m3.
```

Figure 4: An example of aggregate in RDF model

To transform a conceptual Multidimensional Graph into a property-graph representation, we define the following set of mapping rules.

(1) A dimension vertex *dim* is transformed into
`(dim) - [:a] -> (:‘qb:DimensionProperty‘)`
(2) An attribute vertex *att* is transformed into
(a) `(lst) - [:a] -> (:‘skos:ConceptSchema‘),`
`(lst) - [:‘skos:hasTopConcept‘] -> (att)` or
(b) `(lst) - [:a] -> (:‘qb:HierarchicalCodeList‘),`
`(lst) - [:‘qb:hierarchyRoot‘] -> (att)` or
(c) `(level) - [:a] -> (:‘qb4o:LevelProery‘),`
`(level) - [:‘qb4o:inDimension‘] -> (dim),`
`(att) - [:‘qb4o:inLevel‘] -> (level)`
(3) An edge E_{AA} is transformed into a standard relationship `[:‘skos:narrower‘]` or a customized relationship `[:‘e_aa‘]` such as: `(lst) - [:a] -> (:‘qb:HierarchicalCodeList‘),`
`(lst) - [:‘qb:parentChildProperty‘] -> (e_aa)`
(4) An edge E_{MA} is transformed into a relationship `[:dim]` whose label corresponds to the identifier of a dimension, such as:
(a) `(dim) - [:a] -> (:‘qb:DimensionProperty‘),`
`(m) - [:a] -> (:‘qb:Observation‘),`
`(m) - [ :dim] -> (att)` or
(b) `(dim) - [:a] -> (:‘qb:DimensionProperty‘),`
`(level) - [:a] -> (:‘qb4o:LevelProery‘),`
`(level) - [:‘qb4o:inDimension‘] -> (dim),`
`(att) - [:‘qb4o:inLevel‘] -> (level),`
`(m) - [:a] -> (:‘qb:Observation‘),`
`(m) - [:dim] -> (att)`
(5) An aggregate vertex agg_i is transformed into
`(`agg_i `:‘qb:Observation‘ { ‘MDGraph:value‘ : value,`
`‘MDGraph:dim1Name‘ : ’att1Name’, . . . ,`
`‘MDGraph:dimNName‘ : ’attNName’}),`
`(`agg_i`) <- [‘aggDim1‘ : ‘MDGraph:dim1‘] - (att1),`
`. . .`
`(`agg_i`) <- [‘aggDimN‘ : ‘MDGraph:dimN‘] - (attN),`
`(`agg_i`) - [:‘MDGraph:GG‘] -> (`agg_j`)`
`. . .`
`(`agg_i`) - [:‘MDGraph:MG‘] -> (`m_i `: ‘qb:Observation‘)`

3.3 A Use Case

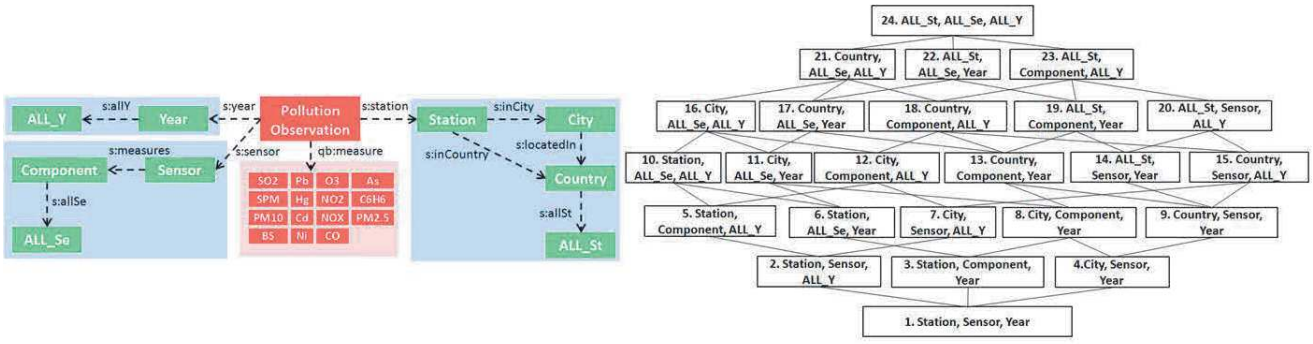
To illustrate the feasibility of our proposed translating process, in this section we describe a use case based on a real-world dataset named QBOAirbase[4]. QBOAirbase corresponds to an *RDF Cube* representation including 5.07×10^6 RDF triples describing the European air quality database on air pollution and climate change mitigation [5]. The graphical multidimensional representation of the QBAirbase dataset is shown in Figure 5(a).

First, we execute the algorithm 1 to build a conceptual Multidimensional Graph. The obtained Multidimensional Graph includes a set of *Measures* M composed of 1.6×10^6 measure vertices. Each measure vertices e_i^{M} corresponds to an observation (`qb:Observation`) in the source. Each measure vertex describe one air pollution indicator such as SO2, Pb, O3 (cf. Figure 5(a)), and it can be analyzed according to three *dimensions*, namely *s:year*, *s:station* and *s:sensor*. The set of attributes A is composed of 2.43×10^5 attribute vertices. Each attribute vertex represent an instance from the source, and it describes one granularity within a dimension. $4.8 \times 10^6 E_{MA}$ edges are created to associate an attribute vertex e_j^{A} with a measure vertex e_i^{M}. Relationships between attribute vertices are represented through 2.43×10^5 E_{AA} edges.

Besides converting original data, the algorithm 1 also creates aggregates G by rolling up on different *dimensions* at different granularities. Figure 5(b) shows 24 types of aggregates at different granularities identified by the algorithm 1. Note that the first type of aggregate *Station, Sensor, Year* corresponds to the original data,

[3] `?lst a qb:HierarchicalCodeList; qb:parentChildProperty ?p2c.`

[4] http://qweb.cs.aau.dk/qboairbase/
[5] https://www.eea.europa.eu/data-and-maps/data/airbase-the-european-air-quality-database-2



(a) Multidimensional structure of the QBOAirbase dataset

(b) The set of aggregates identified in the QBOAirbase dataset

Figure 5: QBOAribase dataset

while the 16^{th} types of aggregate compute measure values according to three distinct granularities, namely `City`, `ALL_SE` and `ALL_Y` (1.62×10^6 vertices and 3.23×10^6 edges). Consequently, the obtained Multidimensional Graph contains pre-computed aggregates represented by 3.57×10^{10} aggregate vertices, 1.045×10^{11} E_{AG} edges, 7.63×10^{10} E_{MG} edges and 3.57×10^{10} E_{GG} edges.

At last, we apply our proposed mapping rules to build an RDF and a property-graph model at the logical level for the conceptual Multidimensional Graph. In the next section, we base the experimental evaluation on the implemented Multidimensional Graph in RDF triple stores and graph-oriented NoSQL database.

4 EXPERIMENTAL EVALUATION

In this section, we present our experimental evaluation on querying statistical RDF data using the aggregate-based model described above. We aim at studying the benefits of graph aggregation for querying efficiency according to four different scenarios :

(1) querying multidimensional graphs without aggregates, using Jena TDB2 and Virtuoso triple stores;
(2) querying multidimensional graphs with pre-computed *aggregates*, using Jena TDB2 and Virtuoso triple stores;
(3) querying multidimensional graphs without aggregates, in Neo4j database;
(4) querying multidimensional graphs with pre-computed *aggregates*, in Neo4j database;

All the resources used in these experiments together with the technical documentation are available at https://iblid.weebly.com/blog/sac-2019.

4.1 Material and methods

Datasets. We built five datasets based on the QBOAirbase dataset. Each dataset includes data related to one or several European countries and weights from 0.7GB (about 4.14 million RDF triples) to 15GB (over 81.92 million RDF triples). (cf. Table 1).

The obtained logical representations are implemented at the physical level in three different data stores: Jena TDB2 (v.3.6.0), Virtuoso (v.7.2.4.2) and Neo4j (v.3.3.5). Table 2 describes different implementations of the five datasets. Note that *Orig.* implementations include only original data, while *Ag.* implementations contain both original data and pre-computed aggregates. For instance, in Jena TDB2 and Virtuoso dataset DS1 contains 4.14 million and 4.16 million triples in *Orig.* and *Ag.* implementations respectively, while in Neo4j dataset DS1 includes 1.029 million nodes with 1.88 million relationships and 1.03 million nodes with 1.89 million relationships in *Orig.* and *Ag.* implementations respectively. Finally, our experiments are carried out in 30 datasets of different data volumes with and without aggregates in the different data stores.

Table 1: QBOAirbase datasets of different volumes

Dataset	Included countries	Size(GB)
DS1	England	0.7
DS2	England, France	2.5
DS3	England, France, Spain	4.6
DS4	England, France, Spain, Germany	7.7
DS5	All countries	15

Table 2: Datasets and their size in GB after implementation

	Jena-TDB2		Virtuoso		Neo4j	
	Orig.	Ag.	Orig.	Ag.	Orig.	Ag.
DS1	0.83	0.85	0.14	0.14	0.68	0.69
DS2	2.39	2.54	0.29	0.29	1.26	1.28
DS3	4.26	4.56	0.48	0.48	2.15	2.20
DS4	6.91	7.39	0.74	0.74	7.07	7.15
DS5	13.08	13.93	1.29	1.29	13.52	13.67

Comparing the size of the original dataset (cf. Table 1) and the size of the dataset implemented in data stores (cf. Table 2), we can observe that datasets are compressed after being implemented in a data store, according to their storage optimisation strategies. In Jena and Neo4j, data are mildly compressed with a compression ratio no greater than 10% from DS1 to DS5. On the contrary, the compression ratio in Virtuoso is high; it ranges from 80% (dataset DS1) to 91.4% (dataset DS5).

Benchmark queries. We propose 24 queries covering a large range of operations during statistical RDF analyses. (cf. Figure 6). Specifically, the first 12 queries involve all data ranging from the

	Query	Remark
1	Average SO_2 by station, sensor and year	raw data without computing aggregated value
2	Average SO_2 by station, component, year	aggregate by 3 attributes on 3 dimensions
3	Average SO_2 by station and sensor	aggregate by 2 attributes on 2 dimensions
4	Average SO_2 by station and component	aggregate by 2 attributes on 2 dimensions
5	Average SO_2 by station	aggregate by 1 attribute on 1 dimension
6	Average SO_2 by city and component	aggregate by 2 attributes on 2 dimensions
7	Average SO_2 by city	aggregate by 1 attribute on 1 dimension
8	Average SO_2 by country and component	aggregate by 2 attributes on 2 dimensions
9	Average SO_2 by country	aggregate by 1 attribute on 1 dimension
10	Average SO_2	aggregate by extreme granularity on all analysis axes
11	Average SO_2 by city and year	aggregate by 2 attributes on 2 dimensions
12	Average SO_2 by country and year	aggregate by 2 attributes on 2 dimensions
13	Average SO_2 by station, sensor and year from 2006 to 2010	raw data without computing aggregated value with one condition
14	Average SO_2 recorded by one specified station	one condition on 1 attribute of low granularity on 1 dimension
15	Average SO_2 recorded by three specified stations	several conditions on 1 attribute of low granularity on 1 dimension
16	Average of SO_2 in one specified city	one condition on 1 attribute of middle granularity on 1 dimension
17	Average of SO_2 in one specified country	one condition on 1 attribute of high granularity on 1 dimension
18	Average of SO_2 in one specified city in 2010	conditions on 2 attributes of 2 dimensions
19	Average of SO_2 in one specified city from 2006 to 2010	conditions (with time interval) on 2 attributes of 2 dimensions
20	Average of SO_2 in one specified city (in 2010)	conditions on a displayed and a non-displayed attribute of 2 dimensions
21	Average of SO_2 recorded by one specified station and one specified component in 2010	conditions on 3 attributes of 3 dimensions
22	Average of SO_2 greater than 150 by city and year	conditions on a displayed measure
23	Average of SO_2 greater than 150 by year in three specified cities	conditions on a displayed measure and 1 attribute of 1 dimension
24	Average of SO_2 in three specified cities whose average NO_2 is greater than 150	conditions on a non-displayed measure

Figure 6: Benchmark Queries

most detailed granularity to the most general granularity, while the last 12 queries extract a sub-set of data according to a selection criteria on different vertex of different granularities. Specifically, queries Q1 and Q13 involve only original data without computing aggregated measure values. It takes the same form in both *orig.* and *ag.* datasets. Queries Q2-Q12 (without selection conditions) and Q14-Q24 (with selection conditions) involve aggregated measures values upon different granularities on different analysis axes. They (i) compute aggregated values on-the-fly according to some grouping conditions and an aggregation function in *orig.* datasets and (ii) directly extract pre-computed and materialized aggregates in *ag.* datasets. Each query is written with SPARQL and Cypher to be executed in triple stores and Neo4j NoSQL database, respectively.

Technical environment and evaluation metrics. The hardware configuration is as follows: OS MAC OS 10.12.5, 2 x 2,4 GHz Quad-Core Intel Xeon, 48 GB 1066 MHz DDR3, 1TB SATA Disk. For each query, we record its execution time (in millisecond) in both implementations, for ten runs. We clear the querying engine's cache before each execution, so that a previously executed query does not serve as warm-up run for the following one. The final execution time of a query corresponds to the mean time of all runs.

4.2 Results and discussion

Querying without summarising data. As the graph aggregation brings additional triples (i.e. materialized aggregates) in a dataset, we first study if queries extracting original data without computing aggregated value require longer runtime. To do so, we execute the query Q1 (without grouping condition or selection condition) and Q13 (including selection condition without grouping condition) in both *orig.* and *ag.* implementations.

Figure 7 presents the difference of runtime for the same queries executed in a *orig.* dataset and an *ag.* dataset. The difference is a positive number, which means the execution time of queries Q1 and Q13 takes more time in *ag.* datasets than in *orig.* datasets.

For query Q1 which contains neither grouping condition nor selection condition, pre-computed aggregates have remarkable impacts over query runtime in Jena TDB2 and Virtuoso. We can also notice that the impact of pre-computed aggregate over Q1 becomes greater as the data volume increases: from DS1 to DS5, the gap is widened 16.62 times in Jena TDB2 and 15.16 times in Virtuoso. The greatest gap (up to 3557ms) of Q1's runtime is found in dataset DS5 of Jena TDB (with an absolute runtime of 58790ms in *orig.* DS5 dataset and 62347ms in *ag.* DS5 dataset).

For query Q13 which contains only selection conditions without any grouping condition, its execution is greatly influenced by pre-computed aggregates in Jena TDB2 and Virtuoso. Notably, from DS1 to DS5, the gap is widened nearly 40 times in Virtuoso. The greatest gap (up to 2913 ms) of Q13's runtime is found in dataset DS5 of Virtuoso (with an absolute runtime of 7492ms in *orig.* DS5 dataset and 10405ms in *ag.* DS5 dataset).

Neo4j is the least affected by the materialization of aggregates. There is practically no difference (max. 45 ms) while querying without summarising data before and after adding pre-computed aggregates, with or without selection condition.

Querying aggregated numeric values without selection condition. The second part of the experiments focuses on the impact of graph aggregation over queries Q2 - Q12 which summarise data according to different grouping conditions without selection.

As a similar trend is found within different implementations and due to limited space available, in Figure 8, we focus only on query execution time in the smallest and largest datasets (i.e. DS1 and DS5). A complete description of our experimental assessment can be found on the following website https://iblid.weebly.com/blog/sac-2019. We can see that pre-computed aggregates allow reducing query

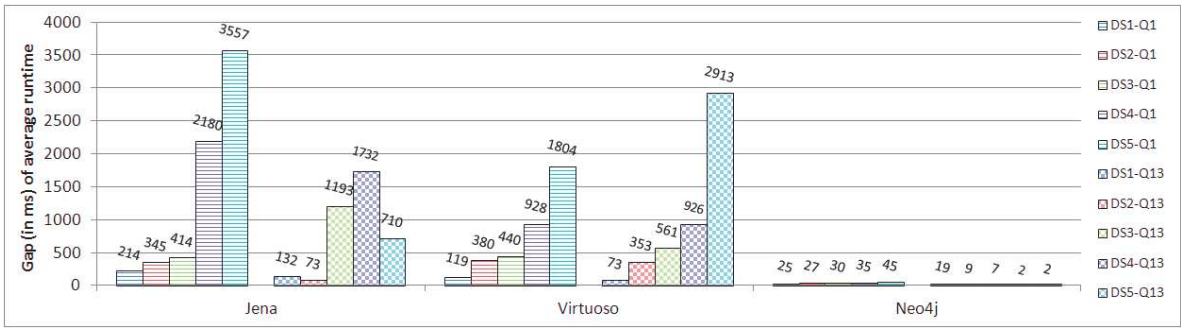


Figure 7: Gap of execution time of queries Q1 and Q13 with and without pre-computed aggregates

execution time in all data stores. Specifically, in the dataset DS1 (cf. Figure 8 (a)), the average execution time in *Jena TDB2* decreases from 1837 ms to 617 ms; the gain after aggregation is about 66%. The average execution time in Virtuoso decreases from 565 ms without aggregates to 281 ms with aggregates; the gain is about 50%. Meanwhile, Neo4j benefits greatly from aggregates: the query execution time decreases of 88% in datasets with aggregates comparing to datasets without aggregates We can see from Figures 8 (b) that the similar trend is found in the largest dataset. The same query requires less execution time in the *aggregate* implementations than the *orig.* implementation of the same dataset in all data stores. The gain of query execution time in Virtuoso becomes less important in a larger dataset (i.e. DS5): about 19.6%. We notice that the gain in Jena TDB2 and Neo4j with pre-computed aggregates becomes greater as the data volume increases; it reaches up to 83% and 99% respectively in the dataset DS5.

In Figure 9, we can see that the average execution time in most data stores increases as the dataset size becomes larger. Specifically, among the *orig.* implementations, the execution time in all three data stores increases in a quadratic polynomial manner (with R^2 close to 0.99). From DS1 to DS5, the execution time multiplies increases by 8.87 times and 10 times in Jena TDB2 and Neo4j, respectively. Comparing to Jena TDB2 and Neo4j, Virtuoso is less affected by the increasing data volume of datasets without aggregates: about 3.8 times. Among the *ag.* implementations, the average query runtime still increase in a quadratic polynomial manner in Jena TDB2 (with $R^2 = 0.99$); in Virtuoso, the increase of query execution time is linear (with R^2 close to 0.99); the execution time in Neo4j remains almost the same, even though the data volume has increased over 21 times from DS1 to DS5. The average execution time in Neo4j is about 350ms with pre-computed aggregates, corresponding to the shortest execution time among all three data stores.

Querying aggregated numeric values with selection condition. In the third part of our experiments, we study if queries with selection conditions are efficiently computed in datasets with aggregates. We analyse the runtime of queries Q14-Q24 in all datasets with and without aggregates. Due to limited space, in this section we only show results of the largest dataset (DS5). The other results are available on our website https://iblid.weebly.com/blog/sac-2019.

From Figure 10, we can see in Jena TDB2, queries including selection conditions of different types do not always benefit from pre-computed aggregates. However, pre-computed aggregates always allow decreasing the execution time of queries with selection conditions in Virtuoso and Neo4j data stores.

In Jena TDB2, queries with selection conditions on attributes (Q14-Q21) are always more efficiently computed in datasets with pre-computed aggregates. Queries with a single selection condition on measures (Q22 and Q23), however, are slightly more efficiently computed in datasets without aggregates. Query with two selection conditions on two measures benefit the most from pre-computed aggregate: its runtime in the dataset with aggregates is reduced to 1.79% of the runtime in the dataset without aggregates.

In Virtuoso and Neo4j, queries with selection conditions on attributes and measures (Q14-Q24) always take less time in dataset with aggregates than without aggregate. Especially, in Virtuoso, the runtime of query Q24 including several selection conditions on different measures decreases by 99.4% in dataset with aggregates. Among all three data stores, the lowest average runtime is recorded in Neo4j with aggregates: 393 ms. Moreover, comparing to Jena TDB2 with aggregates which has difficulty in processing selection conditions on measures, Neo4j with aggregates efficiently computes all queries with different types of selection conditions (on attribute and/or measures); comparing to Virtuoso, in Neo4j with aggregates the execution of queries with different selection conditions takes only 30.8% of the runtime in Virtuoso with aggregates.

Lessons learned. The results of our experiments are opposite to what the authors of [15] have observed: provided that aggregates are properly modeled in an RDF dataset, pre-computed aggregates can reduce query execution time in triple stores (Jena TDB2 and Virtuoso) and Neo4j NoSQL database. More importantly, we saw that even though data volume increases rapidly (by 21 times) from DS1 to DS5, the average query runtime in all three data stores with aggregates does not increase exponentially (cf. Figure 9). We can conclude that pre-computing aggregates is a promising approach to improving querying performance in large RDF datasets. We also notice that the execution of the same query does not take the same time in different data stores. Thus, the choice of data stores impacts the querying efficiency. Among the three data stores studied in this paper, Jena TDB2 performs poorly for computing aggregates on-the-fly or managing pre-computed aggregates. If summarised measure values must be calculated on-the-fly during querying, Virtuoso may consist of one efficient solution. Yet, we do not recommend Virtuoso

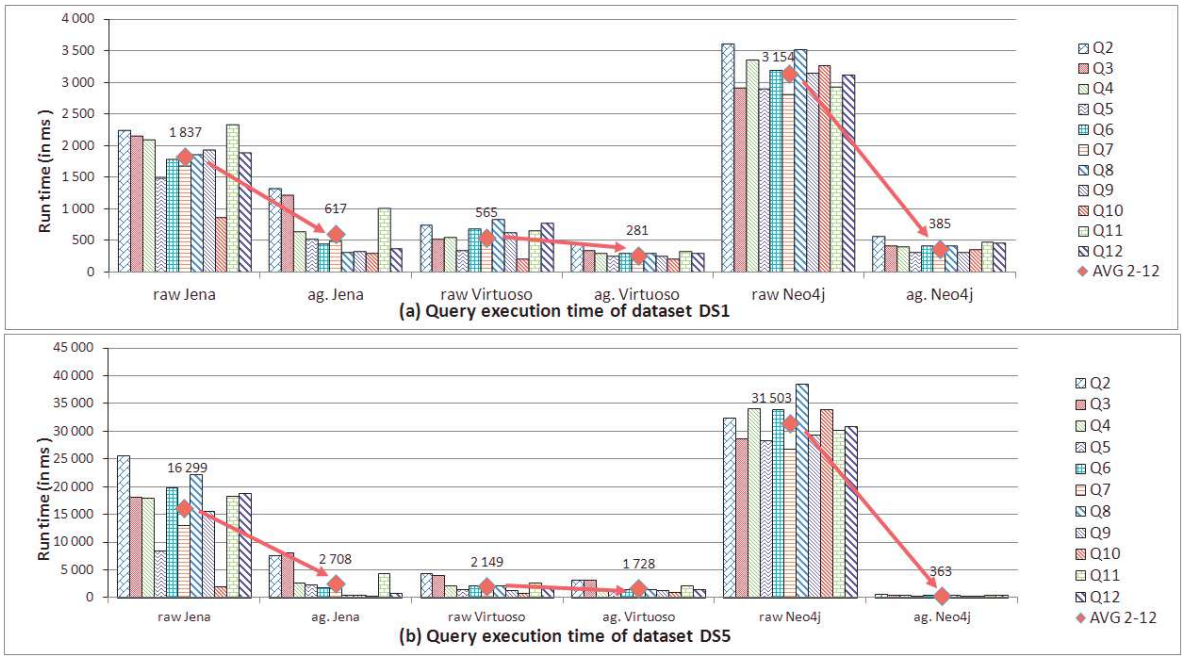


Figure 8: Query execution time with and without aggregates

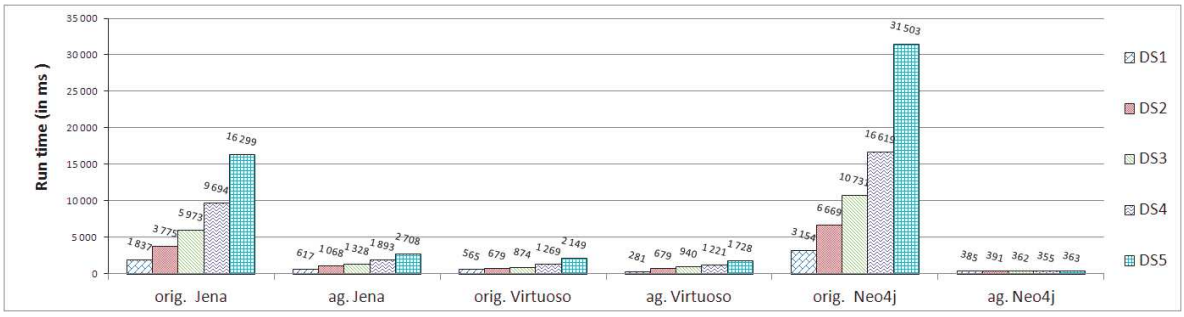


Figure 9: Average execution time of queries Q2-Q12 with and without aggregates in datasets of different volumes

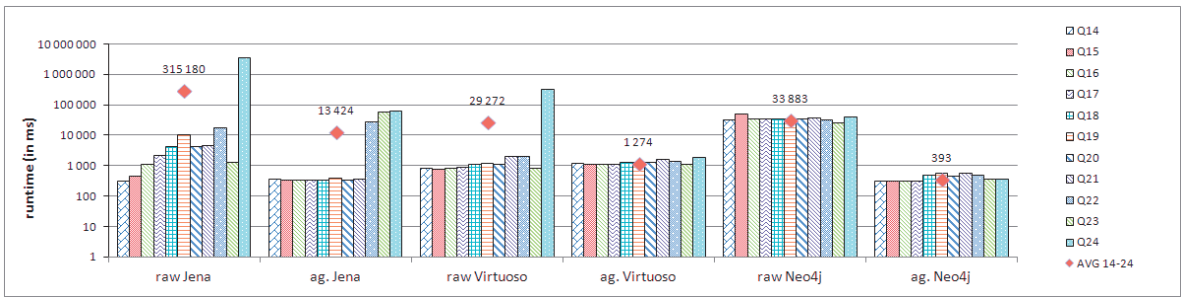


Figure 10: Execution time of queries with selection conditions in the largest dataset (DS5)

for computing complex queries with multiple selection conditions on multiple measures. The most recommended approach is to use Neo4j with pre-computed aggregates for queries with and without selection conditions. Moreover, the larger the dataset is the greater the gain of queries on aggregates becomes in Neo4j. Overall, with 81.92 million triples, queries on pre-computed aggregates in Neo4j

takes no more than 0.4 second. It provides an efficient and scalable solution to managing graph aggregation.

5 CONCLUSIONS AND FUTURE WORK

This paper presented an approach for querying large volume of statistical RDF data. The approach relies on combining pre-aggregation strategies and the performance of NoSQL engines in order to represent and manage RDF data, respectively. We have defined a conceptual model that combines original data with aggregates. We have coupled the conceptual modeling with a logical translating process based on well-known multidimensional RDF graph and property-graph representations. We have conducted experiments on a real-world RDF dataset containing 81.92 million triples. We have compared the performance of querying data on both RDF triple stores and Neo4j NoSQL database, with and without aggregates, with and without selection conditions. Our experimental results corroborate our hypothesis showing that (i) pre-computed aggregates can reduce query execution time in both triple stores and property-graph NoSQL database and (ii) the performance of analytical queries on Neo4j NoSQL database with aggregates outperforms RDF Jena TDB2 and Virtuoso triple stores.

As future work, we intend to improve the RDF modeling vocabulary of our Multidimensional Graph model. In the line of the work of [4], the objective is to support ontology-based data access to enable SPARQL querying over Neo4j NoSQL database. For that, we will propose R2RML mappings for translating the original schemes to our ontology. Another research direction consists of comparing the performance of aggregate querying in other data stores, such as relational databases, column-oriented databases, and so on. We will base our new experimental evaluation on data of larger scales (e.g., exceeding memory limits).

REFERENCES

[1] Fabio Benedetti, Laura Po, and Sonia Bergamaschi. 2014. A Visual Summary for Linked Open Data sources. In *Proceedings of the ISWC 2014 Posters & Demonstrations Track a track within the 13th International Semantic Web Conference, ISWC 2014, Riva del Garda, Italy, October 21*. 173–176.

[2] Elena Botoeva, Diego Calvanese, Benjamin Cogrel, Martin Rezk, and Guohui Xiao. 2016. OBDA Beyond Relational DBs: A Study for MongoDB. In *Proc. of the 29th Int. Workshop on Description Logics (DL 2016)*.

[3] Raouf Bouhali and Anne Laurent. 2015. Exploiting RDF Open Data Using NoSQL Graph Databases. In *AIAI: Artificial Intelligence Applications and Innovations*. Springer, Bayonne, France, 177–190.

[4] Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. 2017. Ontop: Answering SPARQL queries over relational databases. *Semantic Web* 8, 3 (2017), 471–487. https://doi.org/10.3233/SW-160217

[5] S. Campinas, T. E. Perry, D. Ceccarelli, R. Delbru, and G. Tummarello. 2012. Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation. In *2012 23rd International Workshop on Database and Expert Systems Applications*. 261–266. https://doi.org/10.1109/DEXA.2012.38

[6] Šejla Čebirić, François Goasdoué, and Ioana Manolescu. 2015. Query-Oriented Summarization of RDF Graphs. In *Data Science*, Sebastian Maneth (Ed.). Springer International Publishing, 87–91.

[7] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. 2008. Graph OLAP: Towards Online Analytical Processing on Graphs. In *2008 Eighth IEEE International Conference on Data Mining*. 103–112. https://doi.org/10.1109/ICDM.2008.30

[8] Philippe Cudré-Mauroux, Iliya Enchev, Sever Fundatureanu, Paul Groth, Albert Haque, Andreas Harth, Felix Leif Keppmann, Daniel Miranker, Juan F. Sequeda, and Marcin Wylot. 2013. NoSQL Databases for RDF: An Empirical Evaluation. In *The Semantic Web – ISWC 2013*. Springer, Berlin, Heidelberg, 310–325.

[9] Yanlei Diao, Ioana Manolescu, and Shu Shang. 2017. Dagger: Digging for Interesting Aggregates in RDF Graphs. In *Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 23rd - to - 25th, 2017.*

[10] Lorena Etcheverry, Alejandro Vaisman, and Esteban Zimányi. 2014. Modeling and Querying Data Warehouses on the Semantic Web Using QB4OLAP. In *Data Warehousing and Knowledge Discovery*. Vol. 8646. Springer International Publishing, Cham, 45–56. https://doi.org/10.1007/978-3-319-10160-6_5

[11] Lorena Etcheverry and Alejandro A. Vaisman. 2017. Efficient Analytical Queries on Semantic Web Data Cubes. *Journal on Data Semantics* 6, 4 (2017), 199–219.

[12] Amine Ghrab, Oscar Romero, Sabri Skhiri, Alejandro Vaisman, and Esteban Zimányi. 2015. A Framework for Building OLAP Cubes on Graphs. In *Advances in Databases and Information Systems*. Springer, Cham, 92–105.

[13] Nurefsan Gür, Jacob Nielsen, Katja Hose, and Torben Bach Pedersen. 2017. GeoSemOLAP: Geospatial OLAP on the Semantic Web Made Easy. In *Proceedings of the 26th International Conference on World Wide Web Companion*. 213–217.

[14] Daniel Hernández, Aidan Hogan, Cristian Riveros, Carlos Rojas, and Enzo Zerega. 2016. Querying Wikidata: Comparing SPARQL, Relational and Graph Databases. In *Proceedings of the 15th International Semantic Web Conference*. 88–103.

[15] Benedikt Kämpgen and Andreas Harth. 2013. No Size Fits All – Running the Star Schema Benchmark with SPARQL and RDF Aggregate Views. In *The Semantic Web: Semantics and Big Data*. Springer Berlin Heidelberg, 290–304.

[16] Shahan Khatchadourian and Mariano P. Consens. 2010. ExpLOD: Summary-Based Exploration of Interlinking and RDF Usage in the Linked Open Data Cloud. In *The Semantic Web: Research and Applications*. Springer Berlin Heidelberg, Berlin, Heidelberg, 272–287.

[17] Mathias Konrath, Thomas Gottron, Steffen Staab, and Ansgar Scherp. 2012. SchemEX - Efficient Construction of a Data Catalogue by Stream-based Indexing of Linked Data. *Web Semant.* 16 (Nov. 2012), 52–58. http://dx.doi.org/10.1016/j.websem.2012.06.002

[18] Yike Liu, Abhilash Dighe, Tara Safavi, and Danai Koutra. 2016. A Graph Summarization: A Survey. *CoRR* abs/1612.04883 (2016). arXiv:1612.04883

[19] Franck Michel. 2017. *Integrating heterogeneous data sources in the Web of data*. Theses. Université Côte d'Azur.

[20] Thomas Neumann and Gerhard Weikum. 2010. x-RDF-3X: Fast Querying, High Update Rates, and Consistency for RDF Databases. *Proc. VLDB Endow.* 3, 1-2 (2010), 256–263. https://doi.org/10.14778/1920841.1920877

[21] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and Complexity of SPARQL. *ACM Trans. Database Syst.* 34, 3 (2009).

[22] Franck Ravat, Jiefu Song, and Olivier Teste. 2016. Designing Multidimensional Cubes from Warehoused Data and Linked Open Data. In *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS 2016)*. IEEE, Grenoble, France, 171–182. https://doi.org/10.1109/RCIS.2016.7549337

[23] Padmashree Ravindra, HyeongSik Kim, and Kemafor Anyanwu. 2016. Optimization of Complex SPARQL Analytical Queries. In *Proceedings of the 19th International Conference on Extending Database Technology*. 257–268.

[24] Blerina Spahiu, Riccardo Porrini, Matteo Palmonari, Anisa Rula, and Andrea Maurino. 2016. ABSTAT: Ontology-Driven Linked Data Summaries with Pattern Minimalization. In *The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016*. 381–395.

[25] H. Thakkar, D. Punjani, J. Lehmann, and S. Auer. 2018. Killing Two Birds with One Stone – Querying Property Graphs using SPARQL via GREMLINATOR. *ArXiv e-prints* (Jan. 2018).

[26] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. 2008. Efficient Aggregation for Graph Summarization. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*. ACM, 567–580.

[27] Petros Tsialiamanis, Lefteris Sidirourgos, Irini Fundulaki, Vassilis Christophides, and Peter Boncz. 2012. Heuristics-based Query Optimisation for SPARQL. In *Proceedings of the 15th International Conference on Extending Database Technology*. ACM, New York, NY, USA, 324–335.

[28] Xin Wang, Steffen Staab, and Thanassis Tiropanis. 2016. ASPG: Generating OLAP Queries for SPARQL Benchmarking. In *Semantic Technology - 6th Joint International Conference, Singapore, November 2-4, 2016*. 171–185.

[29] Y. Wu, Z. Zhong, W. Xiong, and N. Jing. 2014. Graph summarization for attributed graphs. In *2014 International Conference on Information Science, Electronics and Electrical Engineering*, Vol. 1. 503–507.

[30] Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. 2011. Graph Cube: On Warehousing and OLAP Multidimensional Networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. 853–864.

[31] Mussab Zneika, Claudio Lucchese, Dan Vodislav, and Dimitris Kotzinos. 2016. Summarizing Linked Data RDF Graphs Using Approximate Graph Pattern Mining. In *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*. 684–685.