Noname manuscript No.
(will be inserted by the editor)

A Workload-driven Approach for View Selection in Large Dimensional Datasets

Leandro Ordonez-Ante · Gregory Van Seghbroeck · Tim Wauters · Bruno Volckaert · Filip De Turck

Received: date / Accepted: date

Abstract The information explosion the world has witnessed in the last two decades has forced businesses to adopt a data-driven culture for them to be competitive. These data-driven businesses have access to countless sources of information, and face the challenge of making sense of overwhelming amounts of data in a efficient and reliable manner, which implies the execution of read-intensive operations. In the context of this challenge, a framework for the dynamic read-optimization of large dimensional datasets has been designed, and on top of it a workload-driven mechanism for automatic materialized view selection and creation has been developed. This paper presents an extensive description of this mechanism, along with a proof-of-concept implementation of it and its corresponding performance evaluation. Results show that the proposed mechanism is able to derive a limited but comprehensive set of views leading to a drop in query latency ranging from 80% to 99.99% at the expense of 13% of the disk space used by the base dataset. This way, the devised mechanism enables speeding up query execution by building materialized views that match the actual demand of query workloads.

Keywords Data models · information models · Data mining and (big) data analysis · Dimensional data modeling

1 Introduction

Providing instant access to data is still an open problem. A significant part of the value proposition of nowadays data-driven organizations relies on their

L. Ordonez-Ante · G. Van Seghbroeck · T. Wauters · B. Volckaert · F. D. Turck
Department of Information Technology, Ghent University - imec, IDLab
Technologiepark Zwijnaarde 126
B-9052 Gent, Belgium
Tel.: +32 9 33 14940
E-mail: {Leandro.OrdonezAnte, Gregory.VanSeghbroeck, Tim.Wauters, Bruno.Volckaert, Filip.DeTurck}@UGent.be

ability to gain prompt and actionable insight from business data, which poses stringent requirements on the response time of enterprise applications to support interactive querying and data visualization. To meet such requirements, currently available data technology offers a variety of solutions that ranges from high-level software architectural patterns such as the Lambda and Kappa architectures proposed by [1] and [2] respectively, to off-the-shelf/open-source solutions including in-memory computing platforms like Apache Ignite [3] and SQL-on-Hadoop frameworks such as Apache Impala and Apache Drill [4]. However, experimental evidence shows [5,6] that said solutions either fail to provide instantaneous query answering, or effectively achieve such interactive functioning but at the expense of flexibility by relying on hard-coded information views.

Existing enterprise information applications typically separate analytic workload processing (supported by *Online Analytical Processing systems*—OLAP) from the day-to-day *Online Transaction Processing* (OLTP). A key difference between these two types of workloads lies in the data models and structures they operate on: OLTP systems work on top of highly normalized data models, while OLAP workloads run against denormalized schemas featuring precomputed views derived from transactional business data. Results of a previous experimental study [7] evidence that using such read-optimized structures alone is not enough for analytical processing applications to meet strict response time requirements, even for small datasets.

Thanks to the wide variety of storage technologies available nowadays, more attention has been drawn towards *polystore* systems, also known as *polyglot persistence* systems. A well-thought-out polyglot persistence system is able to optimally use multiple storage technologies for managing the various types of data (with different write/access patterns) that an application requires to handle. This way, data requiring rapid access (e.g. *analytics and reporting data*) may be loaded into a columnar store, while frequently-written data (e.g. *user activity logs*) can lay on a key-value database. In this sense, [8] propose leveraging on polyglot persistence to boost the performance of legacy applications by means of a dynamic transformation process intended for translating data and queries between diverse data storage technologies.

Based on the work of Vanhove et al., a framework that serves as conceptual foundation for the mechanism this paper reports on is presented in [7]. The intuition behind that framework was to progressively optimize the schema of a base dataset by applying a sequence of data transformation operations (e.g. view materialization, table partitioning, field indexing), in response to specific query usage patterns. The experimentation conducted in that early stage evidenced that when it comes to dimensionally modeled datasets, building materialized views is the method with the most significant impact on query performance, reducing response time by several orders of magnitude, followed by table partitioning.

In this sense, this paper introduces an automatic view selection mechanism based on syntactic analysis of the queries running against dimensionally modeled datasets. The remainder of this paper is structured as follows: Section 2 addresses the related work. Section 3 introduces the dynamic data transformation framework that underlies the view selection approach proposed herein. Section 4 focuses on the main contribution of this work and elaborates on the syntactic analysis conducted on the query statements. Section 5 describes the implementation of the proposed approach, while Section 6 discusses the experimental setup and results. Finally conclusions and pointers towards future work are provided in Section 7.

2 Related Work

Dimensional data modeling is one of the foundational techniques of modern data warehousing [9]. It has been extensively applied to a wide range of domains involving data analysis and decision support, due to its inherent ability to structure data for quick access and summary [10,11,12]. View materialization is a common methodology used on top of these dimensional data schemes for speeding up query execution. The associated overhead of implementing this methodology involves computational resources for creating and maintaining the views, and additional storage capacity for persisting them. In this sense, finding the sweet spot between the benefits and costs of this method is regarded by the research community as the *view selection problem.*

Extensive research has been conducted around the materialized view selection problem, as evidenced in several systematic reviews on the topic by [13], [14], and [15] to mention some of them. The approaches surveyed in these works consist in general of two main steps:

1. Find a set of candidate views for materialization based on metrics such as *query execution frequency*, *query access costs*, and *base-relation update frequencies.*
2. Create a set of views selected out of the set of candidate views under certain resource constraints such as *view maintenance costs* and *storage space limitations.*

The review elaborated by [14] groups existing approaches in three main categories: (*i*) heuristic approaches, (*ii*) randomized algorithmic approaches, and (*iii*) data mining approaches.

Heuristic approaches uses multidimensional lattice representations [16,17], AND-OR graphs [18], or *Multiple View Processing Plan* (MVPP) graphs [19, 20] for selecting views for materialization. Issues regarding the exponential growth of the lattice structure when the number of dimensions increases, and the expensive process of graph generation for large and complex query workloads, greatly impact the scalability of these approaches and their actual implementation in consequence [21,14].

The solution space of the view selection problem grows exponentially with the number of dimensions of the data, turning this into a NP-Hard problem. Randomized algorithmic approaches [22,23,24,25,26,27,28] emerged as an attempt to provide approximate optimal solutions to this problem, by using techniques such as *simulated annealing* (SA), *evolutionary algorithms* (EA) and *particle swarm optimization* (PSO). However, since these approaches use multidimensional lattice representations, MVPP and AND-OR graphs as input data structures, they suffer the same scalability issues earlier seen in heuristic approaches [14].

On the other hand, data mining approaches are mainly workload-driven: candidate materialized views are selected and created based on the syntactic analysis of query sets representative of data warehouse usage [21,29,30]. Data-mining based solutions work with much simpler input data structures called *representative attribute matrices*, generated out of query workloads. These structures then configure a clustering context out of which candidate view definitions are derived. Aouiche et al. [21,29] propose the *KEROUAC* method, which addresses view selection as the combinatorial problem of finding the optimal partition of a set of objects (i.e. queries), according to a metric of clustering quality called *New Condorcet criterion* (*NCC*) [31,32]. In KEROUAC clustering is conducted through an iterative procedure similar to the one employed for building univariate decision trees which typically run in $O(dMNlogN)$ time [33] —with N being the number of objects, M the number of attributes, and d the number of nodes (i.e. clusters) in the tree—, and additionally involves the computation of the *NCC* metric on every iteration of the clustering process, which adds to the overall complexity of the method. Other similar data mining approaches for view selection, including the one from [30], involve identifying frequent accessed information by browsing across several intermediate and/or historical results, which is deemed to be a very costly and unscalable process [14].

In this sense, this paper delves further into the approach introduced by Ordonez et al. [7], particularly by elaborating on an automatic mechanism for materialized view selection and creation. The mechanism presented in the following sections relies also on syntactic analysis of query workloads issued against a dimensionally modeled data collection. This mechanism uses a representative attribute matrix as input data structure, assembled as a collection of feature vectors encoding all the clauses of each individual query in the workload at hand. With this input, a strategy for selecting a limited set of candidate materializable views is implemented, comprising the use of hierarchical clustering along with a custom query distance function, and the estimation of a *materializable score* on the resulting clustering configuration.

It is noteworthy that the approach in [7] was concerned with defining a data transformation framework on a conceptual level, while the mechanism discussed herein addresses an actual realization of such framework, tackling the problem of materialized view selection on large data collections. Likewise, the approach introduced by Vanhove et al. [8] —that served as inspiration for the framework proposed in [7]— is not particularly concerned with reducing query latency, but with enabling live data migration between different data storage technologies, irrespective of the query-workload. In this sense, the contribution of the proposed mechanism lies in three key features: (i) a vector representation that encodes not only the query-attribute usage, but also the basic structure of analytical queries, enabling a more precise and also regular representation of the query set, (*ii*) a measure of query distance tightly suited to the structure of the formulated feature vector representation providing a more accurate method for estimating query relatedness, instead of plain Hamming distance used in existing approaches, and (*iii*) a scalable procedure for candidate view generation that relies on a measure of cluster consistency, which in turn uses the above-mentioned query dissimilarity metric to unambiguously identify *materializable* groups of related queries.

3 Dynamic Read-Optimization Framework

3.1 Iterative data transformation

The view selection mechanism this paper presents is framed within the dynamic transformation framework introduced in a previous work [7], which aims at incrementally optimize the schema of a dimensionally modeled dataset to speed up query execution. Figure 1 presents an overview of the architecture of the mentioned framework, which operates by running an iterative process described in detail in [7].

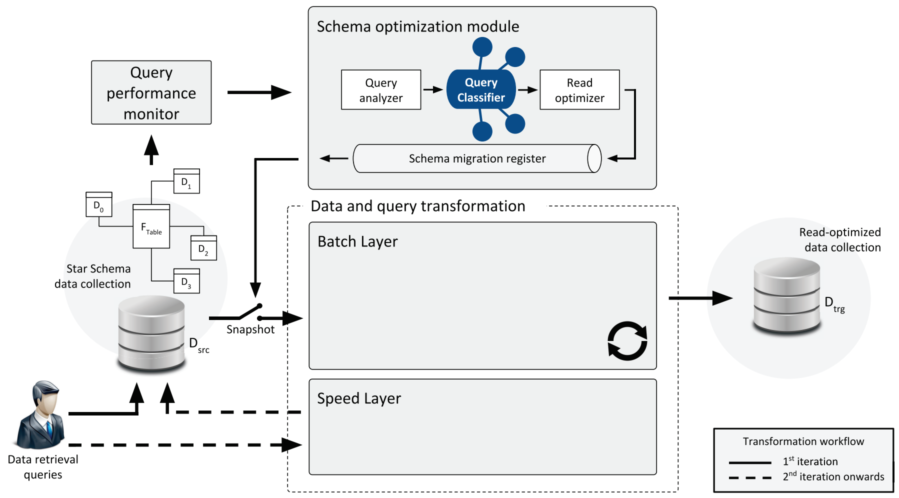

Fig. 1 Dynamic data transformation for read-optimization: architecture overview

During the first iteration of such process queries are handled by a base dataset complying a star dimensional schema (D_{src}), as their performance information is collected by the *query performance monitor*. This information is then used by the *schema optimization module* to classify incoming queries according to the read-optimization method they benefit the most, and to prescribe a *schema migration specification* defining a number of actions to be applied to a snapshot of D_{src}, to ultimately generate a read-optimized version of said dataset (D_{trg} in Fig. 1).

In subsequent iterations, the just generated D_{trg} becomes the new D_{src} and the process described for the first iteration is applied, except that this time incoming queries first have to undergo a translation procedure (*speed layer* in Fig. 1) that adjusts the query statements to the new read-optimized schema. This way client applications can issue queries to D_{trg} as if they were querying the original D_{src}.

3.2 Materialized view selection

The framework proposed in [7] contemplates two categories of methods for read-optimization: (1) *redundant structures*: *view materialization* and *indexing*, and (2) *non-redundant structures*: *table partitioning (horizontal* and *vertical)*. Redundant structures involve an overhead in both storage and computation, while methods from the second category imply reshaping the dataset schema, and rearranging the original information without incurring in any storage overhead. However, *non-redundant structure* methods are mostly intended for aiding the maintenance of large datasets (e.g. loading and removing vast amounts of data), rather than improving query performance by themselves.

These methods are not competing nor mutually exclusive. Conversely, approaches such as the one presented by [34] promote the combination of *redundant* and *non-redundant structures* to attain a better performance for a given query workload. This paper deals specifically with the design and realization of a mechanism for materialized view selection, as a first step towards the implementation of the data transformation framework depicted earlier (See Fig. 1). Before addressing an overview of the proposed approach, the *view selection problem* is defined next.

Definition 1 ***View selection problem.*** Based on the definition by [35]: Let $\mathcal{R}$ be the set of base relations, $\mathcal{S}$ the available storage space, $\mathcal{Q}$ a workload on $\mathcal{R}$, $\mathcal{L}$ the function for estimating the cost of query processing. The view selection problem is to find the set of views $\mathcal{V}$ (view configuration) over $\mathcal{R}$ whose total size is at most $\mathcal{S}$ and that minimizes $\mathcal{L}(\mathcal{R}, \mathcal{V}, \mathcal{Q})$

In the context of the view selection approach proposed herein, some assumptions are made for the system to identify and materialize candidate views:

1. The source data collection (D_{src}) is *temporary immutable.* This implies dealing with insert and update operations is out of the scope of the mechanism presented herein.
2. The query processor provides statistical information regarding D_{src}, such as the size (row count) of each of the base tables composing the schema of the dataset, as well as the cardinality of the attributes that make up the these relations.

3. Latency is favored over view storage cost. This means that the decision on materializing candidate views is driven not by storage restrictions, but by the gain in query latency.

In terms of the definition 1, given a dimensionally modeled dataset $\mathcal{R}$, and a workload $\mathcal{Q}$, the view selection mechanism starts by translating the queries in $\mathcal{Q}$ into feature vectors. In contrast to similar query representation techniques [21], the method proposed in this work accounts not only for query-attribute usage, but also for query structure by defining a number of regions/segments representative of each of the clauses of a Select-Project-Join (SPJ) query, i.e. aggregate operation, projection, join predicates and range predicates. This way, the devised query representation provides a more precise specification of the query statements in $\mathcal{Q}$. A detailed description of this query representation is provided in section 4.1.

The collection of feature vectors of $\mathcal{Q}$ configures a clustering context $\mathcal{C}$. This context is then fed to the *read optimizer* component (see Fig. 1) which implements a clustering algorithm able to identify groups of related queries based on a similarity score computed via a custom query distance function, described in detail in Section 4.2.

Upon running the clustering job, the resulting clustering configuration $\mathcal{K}$ comprises several groups of queries the algorithm deems to be similar. The idea behind building this clustering configuration is to be able to deduce view definitions covering the queries arranged under each cluster. This way, every cluster $\mathbf{c}_i \in \mathcal{K}$ would have an associated view $V_i \in \mathcal{V}$ ($\mathcal{V}$: set of candidate views), enabled to answer the queries in $\mathbf{c}_i$. Of course it is not feasible to materialize the full set of views in $\mathcal{V}$. Consider for instance the (unlikely but possible) case in which $\mathcal{Q}$ is a collection of orthogonal queries. In such situation, there would be as many clusters in $\mathcal{K}$ as queries in $\mathcal{Q}$, and in consequence one candidate view per query to be materialized, which storage-wise is not efficient at all. Now considering a more practical case, the clustering algorithm might come up with spurious clusters, i.e. groups of queries that are actually not that related. To identify those spurious clusters and setting them apart from those clusters that are worth materializing, a *materializable score* is defined, taking into account a measure of cluster consistency and the cluster size $|\mathbf{c}_i|$. Further details on this score and the clustering procedure are provided later in section 4.3.

Based on the results of the *materializable score* computed on the clustering configuration $\mathcal{K}$, a subset of the candidate views in $\mathcal{V}$, $\mathcal{V}_{mat}$, is prescribed to be materialized by defining a *schema migration specification*, which is issued and executed against the source dataset (D_{src}). With the materialized views in place, the translation of the new data-retrieval queries is performed in the speed layer (See Fig. 1). Clearly, such translation procedure involves a certain overhead, however, according to the experimental results reported by [8], the average translation time does not exceed 100 milliseconds, which is negligible when contrasted with the query execution time on the original data collection. Finally, the translated queries run against the matching materialized views, being answered in a fraction of the time it would take to process the original queries on the base dataset.

4 Syntactic analysis of query statements for view selection

Since the devised view selection mechanism aims at lowering the response time of data access operations, only Select-Project-Join (SPJ) queries are considered in this analysis, particularly those containing aggregates. Aggregate-SPJ queries are one of the foundational constructs of OLAP operations. They allow for summarization returning result sets based on multiple rows grouped together under certain criteria (column projection and range predicates). In general, these aggregate queries are computationally expensive since they require scanning several records in the data collection, hence the use of materialized views for speeding up these queries.

In this research the widely-known *Star Schema Benchmark (SSB)* [36] was adopted as a baseline schema and dataset. The SSB, defines a collection of base relations along with a set of queries typically used in data warehousing. Figure 2 shows the entity-relationship model of SSB, featuring *Lineorder* as the table of facts, and *Customer*, *DWDate*, *Part*, and *Supplier* being the dimensions describing the facts.

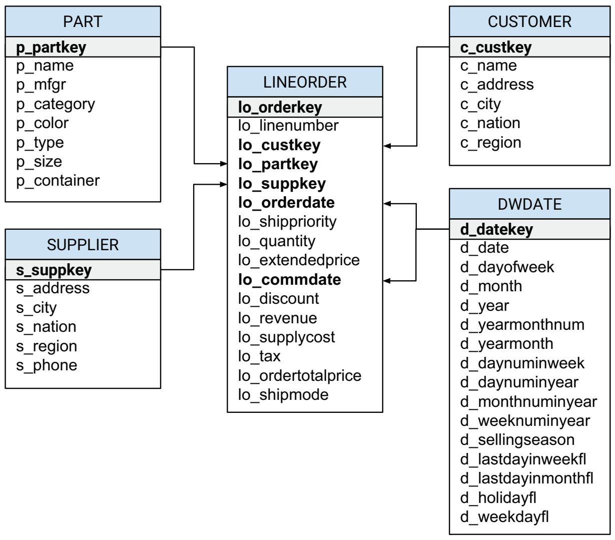

Fig. 2 The Star Schema Benchmark (SSB) data model

All statements in the SSB query set conform to the structure in listing 1.

Listing 1 SSB query structure

```
SELECT select_list
FROM table_expression
[ WHERE search_conditions ]
[ [ GROUP BY column_list]
  [ ORDER BY column_list] ]
```

The syntactic analysis this work thrives on, starts by mining the information contained in the `select_list` and `search_conditions` clauses, encoding these values in a feature vector representation that enables further query processing.

4.1 Query representation

The procedure for obtaining a text-mining-friendly representation of the queries takes each one of the `SELECT` statements from a workload $\mathcal{Q}$ and extracts the *aggregate* (agg_q) and *projection* ($proj_q$) elements, and *join* ($join_q$) and *range* ($rnge_q$) predicates, resulting in the following tuple:

$$q = (aggr_q, proj_q, join_q, rnge_q) \tag{1}$$

The tuple above is the high-level vector representation of the queries from $\mathcal{Q}$. Consider for example the following `SELECT` statement:

```
SELECT SUM(lo_revenue), d_year, p_category
FROM lineorder, dwdate, part
WHERE lo_orderdate = d_datekey
    AND lo_partkey = p_partkey
    AND d_year > 2010
GROUP BY d_year, p_category
```

For the query above:

$aggr_q = [\texttt{SUM}, \texttt{lo\_revenue}]$
$proj_q = [\texttt{d\_year}, \texttt{p\_category}]$
$join_q = [\texttt{d\_datekey}, \texttt{p\_partkey}]$
$rnge_q = [\texttt{d\_year}]$

Each element of the above high-level vector representation gets mapped to a vector using a binary encoding function, as described below.

Definition 2 ***Binary mapping function***. Let R be a relation defined as a set of m attributes $(a_1, a_2, ..., a_\mathsf{m})$ —with a_m being the primary key of R—, and given r an arbitrary set of attributes, the binary mapping of r according to R, denoted by $bm_\mathsf{R}(\mathsf{r})$, is defined as follows:

$$\begin{aligned} bm_\mathsf{R}(\mathsf{r}) &= \{b_i\}, 1 \le i \le \mathsf{m} \\ b_i &= \begin{cases} 1, & \text{if } a_i \in \mathsf{r} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

Using the mapping function above, the vector representation of each one of the query elements in Eq.1 (designated henceforth as *segments*), for a dimensional schema comprising one fact table and N dimension tables, is defined as follows:

$$\begin{aligned}
\mathbf{aggr_q} &= [aggOpCode, bm_{\mathsf{Fact}}(aggr_q)]\\
\mathbf{proj_q} &= [bm_{\mathsf{Fact}}(proj_q), bm_{\mathsf{Dim1}}(proj_q), bm_{\mathsf{Dim2}}(proj_q), ..., bm_{\mathsf{DimN}}(proj_q)]\\
\mathbf{join_q} &= [bm_{\mathsf{Fact}}(join_q), bm_{\mathsf{Dim1}}(join_q), bm_{\mathsf{Dim2}}(join_q), ..., bm_{\mathsf{DimN}}(join_q)]\\
\mathbf{rnge_q} &= [bm_{\mathsf{Fact}}(rnge_q), bm_{\mathsf{Dim1}}(rnge_q), bm_{\mathsf{Dim2}}(rnge_q), ..., bm_{\mathsf{DimN}}(rnge_q)]
\end{aligned}$$

where *aggOpCode* designates the aggregate operation using one-hot encoding, namely, `COUNT`: 00001, `SUM`: 00010, `AVG`: 00100, `MAX`: 01000, `MIN`: 10000.

A complete feature vector $\mathbf{q}$ representing a query $q \in \mathcal{Q}$ is set by putting together the above-mentioned segments, that is:

$$\mathbf{q} = [\mathbf{aggr_q}, \mathbf{proj_q}, \mathbf{join_q}, \mathbf{rnge_q}]$$

Accordingly, considering the `SELECT` statement in the example —issued against the SSB (see Fig. 2)— a complete feature vector instance is shown below (with $\mathsf{Dim_1}$=Customer, $\mathsf{Dim_2}$=DWDate, $\mathsf{Dim_3}$=Part, and $\mathsf{Dim_4}$=Supplier):

$$\mathbf{q} = [[10, 1000], [0, 0, 10000, 1000, 0], [101000, 0, 1, 1, 0], [0, 0, 10000, 0, 0]]$$

4.2 Query dissimilarity estimation

The collection of feature vectors representing the queries from $\mathcal{Q}$ are arranged as a *representative attribute matrix*, configuring a clustering context $\mathcal{C}$. To be able to identify groupings of related queries in such context, a measure of dissimilarity between observations (vectors) and sets of observations is required. In this sense, a distance function is defined in which similarity between two queries is determined to be proportional to the number of attributes they share in a per-segment (aggregation, projection, join and range predicates) and per-relation (fact and dimensions) basis.

Definition 3 Segment Distance. Let $\mathbf{x_p}$ and $\mathbf{x_q}$ be two segments of length N belonging to two distinct query vectors $\mathbf{p}$ and $\mathbf{q}$ from the clustering context $\mathcal{C}$. Distance between $\mathbf{x_p}$ and $\mathbf{x_q}$ (denoted as $sgmtDst(\mathbf{x_p}, \mathbf{x_q})$) is defined as:

$$sgmtDst(\mathbf{x_p}, \mathbf{x_q}) = \frac{1}{N'} \sum_{i}^{N} J(\mathbf{x_{p_i}}, \mathbf{x_{q_i}}) \ni (\mathbf{x_{p_i}}, \mathbf{x_{q_i}}) \neq (0, 0) \tag{3}$$

where,

- $J(\mathbf{x_{p_i}}, \mathbf{x_{q_i}})$ is the *Jaccard-Needham* distance estimated between the i-th elements of $\mathbf{x_p}$ and $\mathbf{x_q}$,
- N' is the number of pairs from $\mathbf{x_p}$ and $\mathbf{x_q}$ such that $(\mathbf{x_{p_i}}, \mathbf{x_{q_i}}) \neq (0, 0)$

This way, segment distance is defined as the average *Jaccard-Needham* dissimilarity computed between pairs of segment elements corresponding to those dimensions with at least one attribute being queried (i.e. $\mathbf{x_{p_i}} \neq 0 \vee \mathbf{x_{q_i}} \neq 0$).

The *Jaccard-Needham* distance is a widely-used method for estimating dissimilarity between sets and binary sequences. Unlike similar measurements such as the *Simple Matching coefficient* and the *Rogers-Tanimoto distance*, *Jaccard-Needham* does not count negative co-ocurrences (or mutual absences), which means that null attribute pairs $(0, 0)$ are not rated as matches in the distance estimation. This, in addition to the symmetry of the measure ($J(a, b) = J(b, a)$) and its straightforward computation, makes the *Jaccard-Needham* distance a suitable method for estimating the dissimilarity between the query segments.

Definition 4 Query Distance. Let $\mathbf{p}$ and $\mathbf{q}$ be two vectors representing queries from the clustering context $\mathcal{C}$. Distance between $\mathbf{p}$ and $\mathbf{q}$ (denoted as $qDst(\mathbf{p}, \mathbf{q})$) is defined as:

$$\begin{aligned} qDst(\mathbf{p}, \mathbf{q}) = w_{aggr} * sgmtDst(\mathbf{aggr_p}, \mathbf{aggr_q}) + w_{proj} * sgmtDst(\mathbf{proj_p}, \mathbf{proj_q}) \\ + w_{join} * sgmtDst(\mathbf{join_p}, \mathbf{join_q}) + w_{rnge} * sgmtDst(\mathbf{rnge_p}, \mathbf{rnge_q}) \end{aligned} \tag{4}$$

where $w_{aggr}, w_{proj}, w_{join}$ and w_{rnge} are arbitrary weights that add up to one (1.0), and condition the influence of each vector segment on the overall dissimiliarity measurement. These weights were estimated when tuning the clustering method the proposed view selection mechanism relies on, by binding their values to the performance of the obtained configuration of clusters estimated in terms of the *F-score* and the *Fowlkes-Mallows index* (*FMI*).

4.3 Query clustering and View materialization

The view selection approach documented herein relies on *hierarchical clustering* [37] for deriving groupings of similar queries. In contrast to other well-known clustering methods such as *K-Means* or *K-medoids*, hierarchical clustering analysis does not require to provide the number of clusters upfront. Instead, it generates a hierarchical representation of the entire clustering context in which observations and groups of observations are stacked together from lower to higher levels, according to a distance measure based on the pairwise dissimilarities among the observations. This way, the individual observations lie at the lowest level of the hierarchy as singleton clusters, while at the top level there is only one cluster holding all the observations. As an illustrative example, consider the dendrogram in Fig. 3 representing the clustering configuration obtained by applying hierarchical clustering on a workload comprising 50 queries.

There are two basic ways to perform hierarchical clustering: through an *agglomerative* procedure (i.e. starting at the bottom and recursively merging

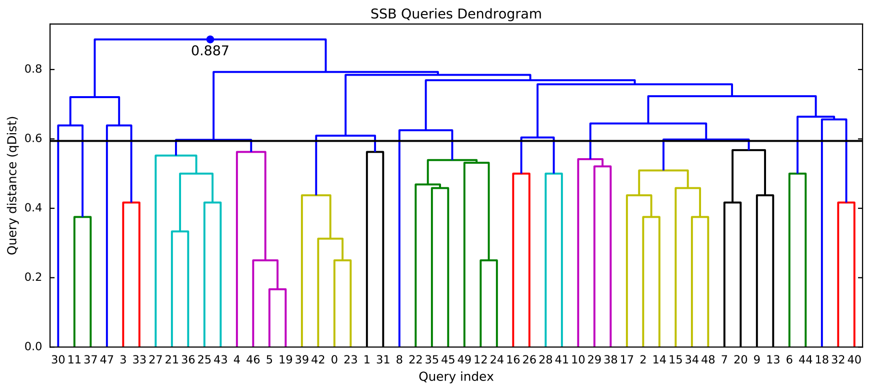

Fig. 3 Dendrogram resulting from applying hierarchical clustering analysis on a 50-query workload. Each different color indicates a group of similar queries.

pairs of similar clusters into a single cluster, while moving up the hierarchy), or by a *divisive* procedure (i.e. starting at the top of the hierarchy with all observations in one cluster, and recursively partitioning it while moving down the hierarchy). Agglomerative clustering is far more extensively used than its divisive counterpart, hence most of the hierarchical clustering algorithms available fall into this category of methods. In divisive procedures, all the possible partitions of the clustering context are considered in the first step. Since the number of combinations for a collection of N observations is $2^{N-1} - 1$, it is impractical to exhaustively implement these methods and heuristic approximations are used instead [38]. This is why agglomerative clustering was favored over divisive clustering, for analyzing the vectors in the representative attribute matrix of $\mathcal{Q}$.

In order to apply hierarchical agglomerative clustering analysis on a clustering context $\mathcal{C}$ (representative attribute matrix of $\mathcal{Q}$), it is required to specify a *dissimilarity metric* for measuring the distance between pairs of query vectors, and a *linkage criterion* which estimates the dissimilarity among groups of queries as a function of the pairwise distance computed between queries belonging to those groups. Selection of the specific technique to use as linkage criterion was made on the basis of the characteristics of the proposed feature vector representation, and results of preliminary parameter tuning tests (mentioned at the end of section 4.2). In consequence—given that the query vectors derived from $\mathcal{Q}$ do not lie on the Euclidean space—methods such as *centroid*, *median*, and *Ward's linkage* [39] were ruled out. Then, *single linkage*, *complete linkage* and *Weighted Pair Group Method with Arithmetic Mean* (WPGMA) were considered, resulting in the selection of WPGMA, since the former two methods tended to underestimate (single linkage) or overestimate (complete linkage) the dissimilarity between query groupings, according to the mentioned tests. In this way, along with WPGMA, the function defined in the previous section, *qDst*, serves as dissimilarity metric in this case. Under this set-up, the clustering procedure (detailed in algorithm 1) starts by assigning each query to its own cluster. Then, the pairwise dissimilarity matrix between such singleton clusters, $\mathbf{D}$, is computed and an empty matrix ($\mathbf{L}$) specifying the resulting dendrogram is initialized. From $\mathbf{D}$, the two most similar (nearest) clusters are merged into one, and appended to $\mathbf{L}$ along with the distance between them (see line 6 in algorithm 1). Then, the pairwise dissimilarity matrix gets updated using the WPGMA method for computing the distance between the newly formed cluster and the rest of the currently existing clusters (eq. 5):

$$\mathbf{D}[(\mathbf{a} \cup \mathbf{b}), \mathbf{x}] = \frac{qDst(\mathbf{a}, \mathbf{x}) + qDst(\mathbf{b}, \mathbf{x})}{2}, \quad (\mathbf{a}, \mathbf{b} \quad and \quad \mathbf{x} \quad being \quad clusters) \tag{5}$$

This procedure is then repeated until there is only one cluster left. Finally, both the clustering configuration ($\mathcal{K}$) and the dendrogram matrix ($\mathbf{L}$) are returned.

Algorithm 1 WPGMA clustering procedure

```
1: $\mathcal{K} \leftarrow \mathcal{C}; \mathcal{C} = \{\mathbf{q_0}, \mathbf{q_1}, \ldots, \mathbf{q_N}\}$      ▷ Initializing clusters (singleton clusters)
2: $\mathbf{D} \leftarrow qDst(\mathbf{q}_i, \mathbf{q}_j)$ for all $\mathbf{q}_i, \mathbf{q}_j \in \mathcal{K}, i \neq j$      ▷ Pairwise dissimilarity matrix
3: $\mathbf{L} \leftarrow []$      ▷ Output matrix
4: while $|\mathcal{K}| > 1$ do
5:     $(\mathbf{a}, \mathbf{b}) \leftarrow argmin(\mathbf{D})$      ▷ Get the nearest clusters
6:     append $[\mathbf{a}, \mathbf{b}, \mathbf{D}[\mathbf{a}, \mathbf{b}]]$ to $\mathbf{L}$
7:     remove $\mathbf{a}$ and $\mathbf{b}$ from $\mathcal{K}$
8:     create new cluster $\mathbf{k} \leftarrow \mathbf{a} \cup \mathbf{b}$      ▷ Merge a and b into one cluster
9:     update $\mathbf{D}$: $\mathbf{D}[\mathbf{k}, \mathbf{x}] = \mathbf{D}[\mathbf{x}, \mathbf{k}] = \frac{qDst(\mathbf{a},\mathbf{x})+qDst(\mathbf{b},\mathbf{x})}{2}$ for all $\mathbf{x} \in \mathcal{K}$
10:    $\mathcal{K} \leftarrow \mathcal{K} \cup \mathbf{k}$
11: end while
12: return $\mathcal{K}, \mathbf{L}$      ▷ L: WPGMA dendrogram: $((N-1) \times 3)$-matrix
```

The rationale behind building this clustering configuration is to deduce view definitions containing the queries grouped under each cluster, so that for each cluster $\mathbf{c}_i \in \mathcal{K}$ there exists a view $V_i \in \mathcal{V}$ able to answer the queries in $\mathbf{c}_i$. However, as stated earlier, to avoid further processing of clusters grouping queries that are not closely related (i.e. spurious clusters), a score was defined indicating to what extent it is worth to materialize the view derived from a particular cluster.

Definition 5 ***Materializable cluster.*** A cluster $\mathbf{c}$ from a clustering configuration $\mathcal{K}$ is said to be *materializable* if the following conditions are met:

1. Queries in $\mathbf{c}$ are highly similar to each other.
2. Queries in $\mathbf{c}$ are clearly separated (highly dissimilar) from queries in other clusters.
3. $\mathbf{c}$ covers as many queries as possible. In other words, $|\mathbf{c}|$ is large enough in proportion to the size of the workload ($|\mathcal{Q}|$)

For a cluster to meet the first two conditions it should be *consistent*, while the third condition prevents singleton and small clusters from being further processed. Based on the above definition, the *materializable score* of a cluster ($mat(\mathbf{c})$ in eq. 6) is computed as the product of two sigmoid functions: one on the per-cluster *silhouette score* (S) [40]—defined below in eq. 7—and the other on the per-cluster proportions (P).

$$mat(\mathbf{c}) = \Big(\frac{1}{1+e^{-k(S(\mathbf{c})-s_0)}}\Big)\Big(\frac{1}{1+e^{-k(P(\mathbf{c})-p_0)}}\Big) \tag{6}$$

with:

$$S(\mathbf{c}) = \frac{1}{|\mathbf{c}|}\sum_{\mathbf{q}_i\in\mathbf{c}}\frac{b(\mathbf{q}_i)-a(\mathbf{q}_i)}{max\,\{a(\mathbf{q}_i),b(\mathbf{q}_i)\}} \quad , \quad P(\mathbf{c}) = \frac{|\mathbf{c}|}{|\mathcal{Q}|} \tag{7}$$

where,

- k is a factor that controls the steepness of both of the sigmoid functions,
- s_0 and p_0 are the midpoints of the silhouette and cluster-proportion sigmoids respectively,
- $a(\mathbf{q}_i)$ is the average distance between $\mathbf{q}_i$ and all queries within the same cluster,
- $b(\mathbf{q}_i)$ lowest average distance of $\mathbf{q}_i$ to all queries in any other clusters.

By setting a fixed threshold on this score ($S \geq 0.5$ by default) it is possible to unambiguously set the spurious clusters apart from those whose corresponding views are worth materializing. The next step is deriving view definitions covering the queries arranged under each of the *materializable clusters* ($\mathcal{K}_{mat} \subseteq \mathcal{K}$), this is to say:

$$\forall \mathbf{c}_i \in \mathcal{K}_{mat}, \exists V_i | \forall \mathbf{q} \in \mathbf{c}_i, \mathbf{q} \subseteq V_i$$

Algorithm 2 below details the procedure conducted to derive the views V_i meeting this containment condition on each of the materializable clusters. In this procedure, the SPJ-query clauses (*aggregate*, *projection*, *join predicates*, and *group-by*) of the resulting views are defined in terms of the union of the corresponding attributes from each query in the cluster.

Algorithm 2 Procedure for deriving view definitions

```
1: Let $\mathbf{c}$ be a cluster in $\mathcal{K}_{mat}$
2: $V \leftarrow [aggr_V, proj_V, join_V, groupBy_V]$        ▷ Output view definition
3: for each query $\mathbf{q}$ in $\mathbf{c}$ do
4:     $aggr_V \leftarrow aggr_V \cup aggr_{\mathbf{q}}$
5:     $proj_V \leftarrow proj_V \cup proj_{\mathbf{q}} \cup rnge_{\mathbf{q}}$
6:     $join_V \leftarrow join_V \cup join_{\mathbf{q}}$
7:     $groupBy_V \leftarrow groupBy_V \cup proj_{\mathbf{q}} \cup rnge_{\mathbf{q}}$
8: end for
9: return $V$
```

It is worth noting that view definitions are generated without *range predicates*. Instead, the attributes used in this query clause are pushed to the *projection* and *group-by* clauses of the view definition. In this way, the resulting materialized views are able to answer not only the queries grouped under each cluster, but also unseen queries with arbitrary range predicates on the attributes listed in the projection clause of the view definition[1]. Additionally, thanks to the iterative nature of the underlying data transformation framework, further optimization of the derived views is possible. Later in this paper (Section 6.3) a view maintenance strategy is discussed which leverages on the continuous data transformation process described back in section 3.

To recap, this section developed a thorough description of the syntactic analysis of query workloads that makes up the view selection mechanism proposed here. It started by specifying the procedure for obtaining a structured representation of the queries in the form of a feature vector and a representative attribute matrix. Then, a custom query dissimilarity function was defined, tailored specifically to the structure and values of said feature vectors. Finally, a query clustering algorithm based on the pairwise dissimilarities between the analyzed queries was addressed, detailing as well the procedure for deriving materialized view definitions out of the obtained clustering configuration.

5 Proof-of-concept Implementation: Star Schema Benchmark (SSB) and workload generation

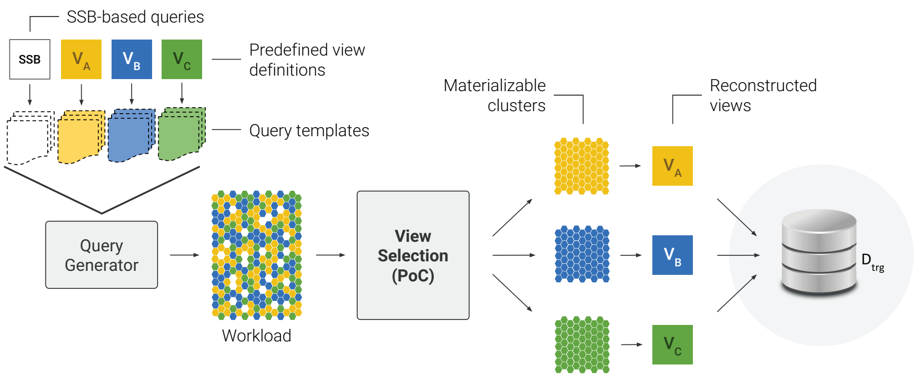

Fig. 4 Proof-of-concept implementation of the proposed view selection mechanism.

A bottom-up approach was adopted to test the principles, assumptions and procedures governing the view selection mechanism detailed in the previous sections. In this way, starting from a set of predefined view definitions, the effectiveness of the proposed mechanism is estimated in terms of its ability

[1] Although this only applies for queries with distributive aggregate functions, i.e. `SUM`, `COUNT`, `MIN`, and `MAX`.

for identifying the same set of views and reconstructing their definitions, upon analyzing a query workload generated from query templates fitting the original set of views (see Figure 4).

As stated back in section 4, this research leverages on the *Star Schema Benchmark* (SSB) as baseline schema and dataset, and therefore both the predefined views and query templates, as well as the query generator module were designed and built so they conform to the data model the SSB embodies.

The SSB comprises a dimensional data model (four dimensions, one fact table), an extensible dataset (size depending on a *scaling factor—SF*), and a set of queries typical for data warehouse applications arranged in four categories/families designated as *Query Flights* (a detailed definition of the SSB is available online [36]).

Thirteen Select-Project-Join query statements in total compose the full query set of the SSB. For the proof-of-concept that is being described, three view definitions were derived based on the original SSB query set, and from each view definition, four query templates were prepared. Additionally, one template for each one of the 13 canonical SSB queries were also composed. With this set of 25 templates as input, a module that generates random instances of runnable queries enabled the creation of query workloads of arbitrary size. Listings below present the definitions of each one of the mentioned views.

Listing 2 Definition of View A

```
SELECT sum(lo_revenue), p_brand1, c_region,
       s_region, d_year
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY p_brand1, c_region, s_region, d_year
ORDER BY p_brand1, c_region, s_region, d_year
```

Listing 3 Definition of View B

```
SELECT sum(lo_ordtotalprice), p_category, c_city,
       s_city, d_yearmonthnum
FROM lineorder, customer, dwdate, part, supplier
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY p_category, c_city, s_city, d_yearmonthnum
ORDER BY p_category, c_city, s_city, d_yearmonthnum
```

Listing 4 Definition of View C

```
SELECT sum(lo_supplycost - lo_tax), c_region, p_mfgr,
       s_region, c_nation, d_year
FROM lineorder, customer, dwdate, part, supplier
```

```
WHERE lo_custkey = c_custkey
  AND lo_orderdate = d_datekey
  AND lo_partkey = p_partkey
  AND lo_suppkey = s_suppkey
GROUP BY c_region, p_mfgr, s_region, c_nation, d_year
ORDER BY c_region, p_mfgr, s_region, c_nation, d_year
```

6 Experimental Evaluation

6.1 Experimental setup

Figure 5 depicts the arrangement of components and technologies used for conducting the experimental evaluation of the proposed view selection approach. This evaluation comprised three main stages:

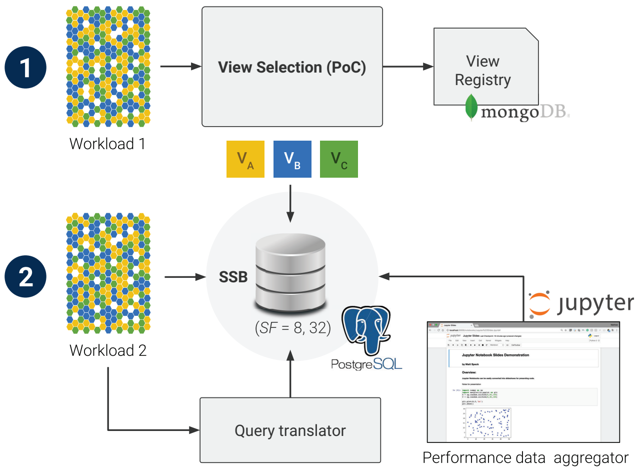

Fig. 5 Experiment set-up

(*1*) Running the view selection implementation on a 400-query workload to the point that it materializes views A, B, and C (defined earlier in section 5), while keeping track of the runtime involved in the procedures of *query clustering*, *view scoring* (using the materializable score defined in section 4.3), *view definition*, *view registering* and *view creation (i.e. materialization)*.

(*2*) Once the views are materialized, run a second 400-query workload against both the base SSB dataset and the materialized views. In doing the latter, queries first pass through a translation component that gathers the details of the available materialized views from the *view registry* (stored in a *MongoBD*[2] document database), and adapts the incoming query statements accordingly.

[2] Available at mongodb.com

(*3*) Running the two previous stages on workloads of different sizes and query distributions.

For all the stages, the performance information collected from running the tests were aggregated and visualized using *Jupyter notebook*[3]. During this evaluation, workloads of multiple sizes were run against two different dimensions of the SSB dataset: 48 million rows ($SF = 8$), and 192 million rows ($SF = 32$). These datasets were stored into *PostgreSQL*[4] databases deployed on two VMWare® virtual machines with the following specifications: Intel® Xeon® E5645 @2.40GHz CPU, 20GB RAM, 500GB hard disk.

6.2 Results

6.2.1 View selection overhead

Running the view selection implementation on a validation workload (400 queries) and a SSB dataset with $SF = 32$ took around 4200 seconds in total (i.e. 1 hour and 10 minutes). While this might be deemed as a considerable amount of time, it is worth mentioning that the actual analysis of the workload takes just a small fraction of it. As mentioned before, the view selection process involves the execution of a sequence of steps: (1) *query clustering*, (2) *view (or cluster) scoring*, (3) *view definition*, (4) *view registering* and (5) *view creation*. Out of these only the first four steps have to do with the syntactical analysis of query sets described throughout this paper, while the last one (view creation) refers to the actual materialization of the derived views in the data store.

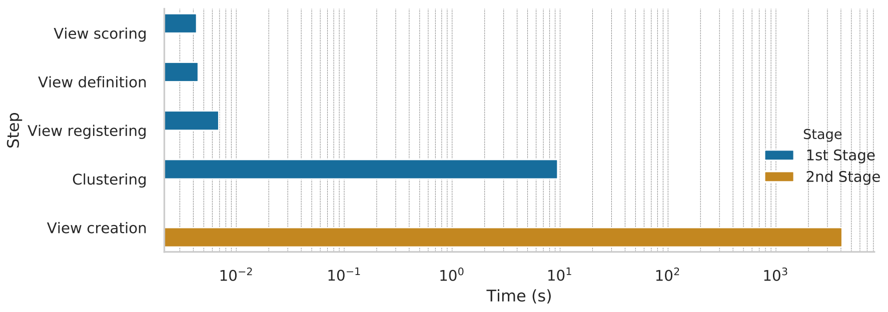

Fig. 6 Materialized view selection runtime per stage (*SSB* $SF = 32$, $|\mathcal{Q}| = 400$)

Figure 6 portrays the substantial difference between these execution times by defining two stages: the first one aggregates the initial four steps, and the second one comprises the view creation step only. Note how the steps from stage one amounts to less than 10 seconds, adding up to just the 0.23% of the

[3] Available at jupyter.org

[4] PostgreSQL v9.5.8 working with the default configuration (postgresql.org)

total execution time, while the remaining 99.77% is the time it takes for the data store (PostgresSQL in this case) to build and persist the resulting views.

While in classical data-mining methods for view selection query sets are mapped into a *query vs attribute* matrix (where statements are represented as flat binary sequences), the method proposed in this work accounts not only for query-attribute usage, but also for the basic structure of analytical queries (see *query segments* in section 4.1). The *KEROUAC* method by Aouiche et al.[21, 29] discussed back in section 2, adopts this classical approach and additionally relies on a clustering technique that can be regarded as *divisive clustering*, in the sense that it proceeds by first placing the whole collection of queries into a single cluster, and then iteratively partitioning it until convergence to a stable clustering configuration is achieved. Given that the mechanism proposed herein adopts a *agglomerative clustering* technique, it was considered relevant to contrast the performance of both methods. To this end, authors of KEROUAC were contacted to provide missing information required to replicate their approach. As a result, Figure 7 shows the variation of the overhead time of the proposed method w.r.t. the workload size, and compares it to that from KEROUAC. While the approach proposed herein features a quadratic rate of growth—since the implementation of the WPGMA method used in the clustering analysis relies on the *nearest-neighbors chain algorithm* which is known to run in $O(N^2)$ time [39]—it still outperforms similar methods running under the same conditions, as evidenced in Figure 7.

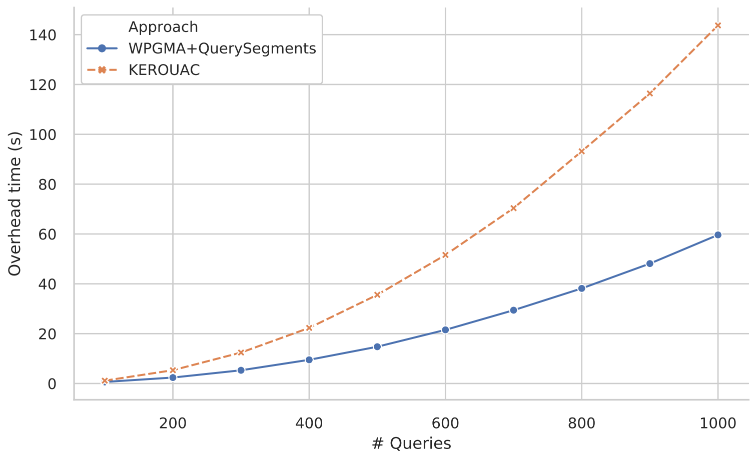

Fig. 7 View selection overhead vs Workload size.

When it comes to storage cost, view size varies depending on the cardinality of the fields used in the group-by clause of their definition [21], and whether or not there are hierarchical relations between such attributes (e.g. the one between `c_region`, `c_nation` and `c_city`). Figure 8a shows the amount of records per materialized view, in contrast to the number of rows in the SSB base schema. While the amount of records of views A (175.000) and C (4.375) is fairly negligible in comparison with the base schema (192.000.754 records), view B amounts to almost half the size of the base dataset. Nonetheless, in terms of disk space usage the proportion between views and base schema is more favorable, as evidenced in Figure 8b. This way, while the amount of records stored into the three derived views add up to 50% of the number of rows in the base schema, the disk space used by said views is only 13% the size of the base data collection.

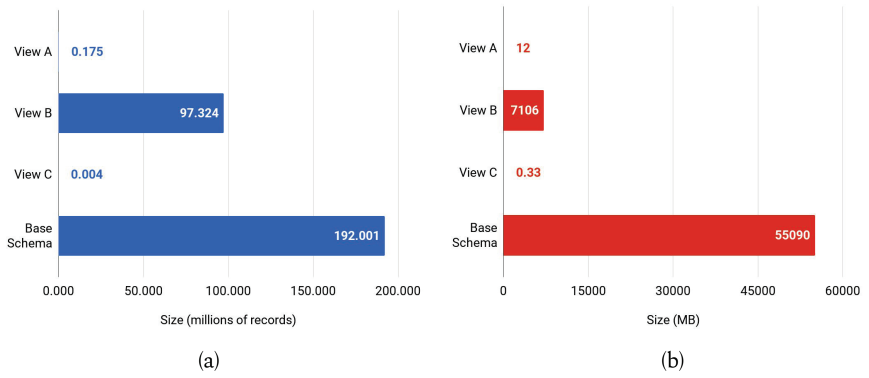

Fig. 8 Materialized views size: (a) millions of records. (b) disk space. — ($SSB\ SF = 32$).

6.2.2 View selection performance

With the selected views already materialized, a 400-query workload was run against the base SSB dataset ($SF = 32$) to get a query latency baseline. Out of those 400 queries, 300 were covered by the three available materialized views (100 queries per view), and the remaining ones were canonical SSB-based queries. Once the latency baseline was built, the same workload was issued this time with the query translation module in place, so that incoming queries matching any of the definitions of the available materialized views get rewritten and issued against them. Figure 9 illustrates the contrast between the baseline query latency (`baseline_time`) and the latency when queries run against the materialized views (`runtime`). For queries that benefit from views A and C the gain in query latency is remarkably substantial, going from hundreds of seconds in the baseline to sub-second runtime in the views. On the other hand, queries running on the view B present an important—though less striking—reduction in latency. In this case, the size of this view, which in terms of number of records is comparable to the base dataset, prevents queries from running under interactive latency constraints.

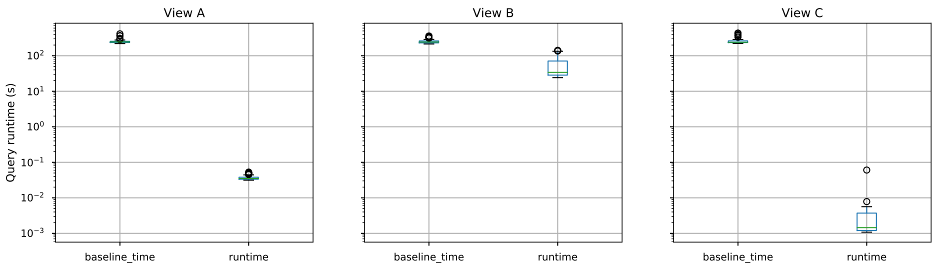

Fig. 9 Query runtime per view: Baseline runtime vs. View runtime ($SSB\ SF = 32$, $|\mathcal{Q}| = 400$)

Table 1 summarizes the results obtained from running the above test, including the reduction in latency achieved through each one of the views, relative to the average baseline query runtime.

Table 1 Query latency reduction per view ($SSB\ SF = 32$, $|\mathcal{Q}| = 400$)

	Baseline time (s)	**Translation time (s)**	**View runtime (s)**	**% Latency reduction**
View A	251.04±2.85	$(3.54 \pm 0.11) \times 10^{-3}$	$(36.89 \pm 0.49) \times 10^{-3}$	99.98
View B	253.53±3.53	$(10.7 \pm 0.45) \times 10^{-3}$	51.83±3.22	79.55
View C	256.98±4.51	$(5.18 \pm 0.45) \times 10^{-3}$	$(2.86 \pm 0.21) \times 10^{-3}$	99.99

6.2.3 Response to workload size

The purpose of this last evaluation was estimating the influence of the view selection mechanism in the workload execution time (hereafter WET), using query sets of multiple sizes and different query distributions. As with the previous tests, each workload was first run against the base SSB dataset ($SF = 8$), this time measuring the execution time for the full workload instead of the average query runtime. Afterwards, each workload was fed to the view selection implementation, measuring again the WET, as well as the view selection overhead. Three query distributions were considered for the workloads in this evaluation: (*1*) half the queries in the workload are covered by the materialized views, (*2*) 75% of the queries in the workload are covered by the materialized views, and (*3*) all the queries in the workload are covered by the available materialized views. Figure 10 presents the results obtained for different workload sizes and query distributions.

Results displayed in Figure 10 evidence a consistent decrease in the WET as the amount of queries covered by the materialized views grows: for the 50%-view-covered workloads the WET was consistently reduced by about 40%, while the 75%-view-covered and the full-view-covered workloads took 57% and 75% less time to run respectively. As the results from the previous tests

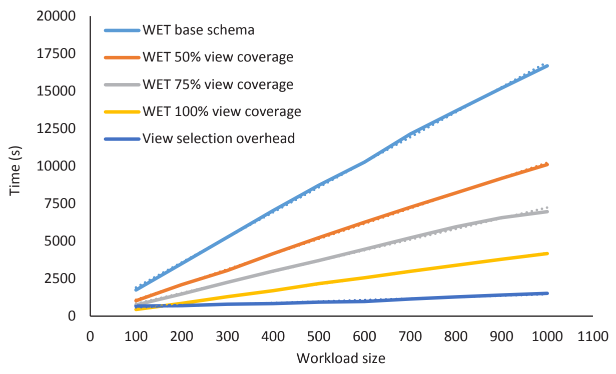

Fig. 10 Workload execution time (WET) vs. Workload size ($SSB\ SF = 8$)

indicated, the drop in the WET is closely tied to the size of the materialized views. Under the configuration used for this test, the aggregated size of the materialized views add up to 85% of the number of records of the base dataset and 46% of the disk space this last one uses. This way, in order to further reduce the workload execution time in cases like this, a better compromise between view's query coverage and view size has to be achieved. The next section elaborates on this constraint and outlines a practical strategy to address it, leveraging on the iterative nature of the dynamic transformation framework upon which the proposed view selection mechanism was conceived.

6.3 Discussion

The syntactic analysis lying at the core of the mechanism described in this paper, proved an effective method for identifying groups of related queries and deriving a limited but comprehensive set of materialized views out of them. The experimental evaluation conducted on a proof-of-concept implementation of the devised approach reported that, despite the appreciable overhead, the time it takes for the procedure to run is nearly linear on the number of queries in the workload. Moreover, most of the overhead in time is due to the materialization of the selected views in the underlying storage technology, while the time spent in the actual analysis of the workload is relatively negligible.

On the other hand, when examining the effect of using the resulting materialized views on the query latency and workload execution time, there is an evident and substantial improvement considering a drop in query latency ranging from 80% to 99.99%, and a decrease in the WET of 40% to 75% depending on the view coverage of the workload. However, there are cases where maximizing the query coverage of the derived views might lead to a prohibitively large storage overhead critically impacting the relative benefit of using materialized views. By leveraging on the iterativity inherent to the dynamic data transformation framework introduced at the beginning of section

3, it is possible to cope with the stated limitation, considering the following extra steps:

First iteration, before materializing a selected view:

1. Estimate the maximum size of the view as the product of the cardinalities of the attributes listed under the *group-by* clause of its definition.
2. When the estimated size is comparable to the amount of records of the fact table, partition the view on the attribute with the less cardinality, this way ending up with a set of size-bounded child views.

Subsequent iterations:

1. In the query translation component, anytime an incoming query matches a partitioned view rewrite it into several sub-queries targeting each one a different partition. Then, run such sub-queries concurrently and aggregate their result sets.
2. Identify *hot regions* in the available materialized views (i.e. sets of tuples that get queried the most) leveraging on the continuous monitoring of the incoming queries.
3. Use horizontal partitioning on the views to set apart the *hot regions* from the remaining tuples.
4. Keep track of the *hit ratio*[5] of each materialized view and its corresponding partitions to eventually dispose of those views/partitions that seldom get queried.

The above procedures configure a practical maintenance strategy that allows to address the storage overhead constraints and maximize the benefit of the materialized views the view selection mechanism comes up with. This strategy is currently under development and its integration to the mechanism proposed in this paper has been deferred to future work.

7 Conclusions and Future Work

Organizations nowadays face a daunting challenge when trying to make sense of the massive and ever-growing amount of data generated in their day-to-day operation. Being able to conduct ad-hoc querying and get visual insights from such data in an efficient and timely manner is key for business to support their decisions. In this regard, this paper describes an approach for automatically generating materialized views to speed up data retrieval on dimensionally modeled datasets. The proposed approach has been conceived as part of a framework for dynamic data transformation intended to generate read-optimized data schemas, and relies on syntactic analysis of query workloads

[5] Ratio of the number of queries answered by the view/partition to the total number of queries in a certain period of time

issued against the data collection. The developed method provides a way to estimate how similar two queries are, and put together clusters of related queries based on said estimation. Then, the method is able to tell out of those clusters which ones are worth materializing—based on consistency and query coverage criteria—and derive a view definition for each *materializable* cluster, based on the queries they group.

The experimental evaluation conducted on a proof-of-concept implementation was focused on measuring the overhead the proposed view selection method entails and contrasting it to the relative benefit it brings in return. Results show that in general such processing overhead pays off in query latency, leading to a drop ranging from 80% to 99.99% at the expense of 13% of the disk space used by the base dataset for persisting the derived materialized views. The evaluation also reported a few open challenges of the proposed approach when it comes to finding an adequate level of query coverage, so that the storage overhead due to the views does not compromise the benefit of using them. To deal with the declared limitation a preliminary view maintenance strategy was outlined, resorting to the iterative optimization of the available views by using horizontal partitioning and keeping track of their usage patterns over time.

In consequence, upcoming work on this research will extend the current view selection mechanism with the view maintenance strategy described earlier and evaluate its impact on query performance. Additionally, the dynamic data transformation framework—that lays the foundations for the approach proposed in this paper—will be further developed to relax the assumption of temporary immutability of the base dataset, and also incorporate polyglot persistence (i.e. intelligently scatter data over multiple data store technologies).

Acknowledgements

This work was supported by the Research Foundation Flanders (FWO) under Grant number G059615N - "*Service oriented management of a virtualised future internet*".

References

1. Marz, N., Warren, J.: Big Data: Principles and best practices of scalable realtime data systems. Manning Publications Co. (2015)
2. Fernandez, R.C., Pietzuch, P.R., Kreps, J., Narkhede, N., Rao, J., Koshy, J., Lin, D., Riccomini, C., Wang, G.: Liquid: Unifying nearline and offline big data integration. In: CIDR 2015. (2015)
3. Anthony, B., Boudnik, K., Adams, C., Shao, B., Lee, C., Sasaki, K.: In-memory computing in hadoop stack. In: Professional Hadoop®, pp. 161–182. Wiley Online Library (2016)
4. Pal, S.: Sql-on-big-data challenges & solutions. In: SQL on Big Data: Technology, Architecture, and Innovation, pp. 17–33. Apress, Berkeley, CA (2016). DOI 10.1007/978-1-4842-2247-8_2

5. AMPLab-UC-Berkeley: Amplab big data benchmark (2014). URL `https://amplab.cs.berkeley.edu/benchmark/`. Https://amplab.cs.berkeley.edu/benchmark/. Last accessed: 2017-04-18
6. Ordonez-Ante, L., Vanhove, T., Van Seghbroeck, G., Wauters, T., De Turck, F.: Interactive querying and data visualization for abuse detection in social network sites. In: ICITST 2016, pp. 104–109. IEEE (2016)
7. Ordonez-Ante, L., Vanhove, T., Van Seghbroeck, G., Wauters, T., Volckaert, B., De Turck, F.: Dynamic data transformation for low latency querying in big data systems. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 2480–2489 (2017). DOI 10.1109/BigData.2017.8258206
8. Vanhove, T., Sebrechts, M., Van Seghbroeck, G., Wauters, T., Volckaert, B., De Turck, F.: Data transformation as a means towards dynamic data storage and polyglot persistence. International Journal of Network Management **27**(4), e1976 (2017). DOI 10.1002/nem.1976
9. Kimball, R., Ross, M.: The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd edn. Wiley Publishing (2013)
10. Park, D., Yu, J., Park, J.S., Kim, M.S.: Netcube: a comprehensive network traffic analysis model based on multidimensional olap data cube. International Journal of Network Management **23**(2), 101–118 (2013)
11. Ali, O., Crvenkovski, P., Johnson, H.: Using a business intelligence data analytics solution in healthcare. In: 2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 1–6 (2016). DOI 10.1109/IEMCON.2016.7746328
12. Scriney, M., O'Connor, M.F., Roantree, M.: Generating cubes from smart city web data. In: Proceedings of the Australasian Computer Science Week Multiconference, ACSW '17, pp. 49:1–49:8. ACM, New York, NY, USA (2017). DOI 10.1145/3014812.3014863
13. Nalini, T., Kumaravel, A., Rangarajan, K.: A comparative study analysis of materialized view for selection cost. World Applied Sciences Journal (WASJ) **20**(4), 496–501 (2012)
14. Goswami, R., Bhattacharyya, D.K., Dutta, M., Kalita, J.K.: Approaches and issues in view selection for materialising in data warehouse. International Journal of Business Information Systems **21**(1), 17–47 (2016). DOI 10.1504/IJBIS.2016.073379
15. Gosain, A., Sachdeva, K.: A systematic review on materialized view selection. In: S.C. Satapathy, V. Bhateja, S.K. Udgata, P.K. Pattnaik (eds.) Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications, pp. 663–671. Springer Singapore, Singapore (2017)
16. Nadeau, T.P., Teorey, T.J.: Achieving scalability in olap materialized view selection. In: Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP, DOLAP '02, pp. 28–34. ACM, New York, NY, USA (2002). DOI 10.1145/583890.583895
17. Serna-Encinas, M.T., Hoyo-Montano, J.A.: Algorithm for selection of materialized views: based on a costs model. In: International Conference on Current Trends in Computer Science, 2007. ENC 2007., pp. 18–24. IEEE (2007)
18. Gupta, H., Mumick, I.S.: Selection of views to materialize in a data warehouse. IEEE Transactions on Knowledge and Data Engineering **17**(1), 24–43 (2005)
19. Phuboon-ob, J., Auepanwiriyakul, R.: Two-phase optimization for selecting materialized views in a data warehouse. International Journal of Computer, Electrical, Automation, Control and Information Engineering **1**(1), 119–123 (2007). URL `http://waset.org/Publications?p=1`
20. Azgomi, H., Sohrabi, M.K.: A game theory based framework for materialized view selection in data warehouses. Engineering Applications of Artificial Intelligence **71**, 125–137 (2018). DOI https://doi.org/10.1016/j.engappai.2018.02.018. URL `http://www.sciencedirect.com/science/article/pii/S0952197618300472`
21. Aouiche, K., Jouve, P.E., Darmont, J.: Clustering-based materialized view selection in data warehouses. In: Advances in Databases and Information Systems, pp. 81–95. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
22. Derakhshan, R., Stantic, B., Korn, O., Dehne, F.: Parallel simulated annealing for materialized view selection in data warehousing environments. In: International Conference on Algorithms and Architectures for Parallel Processing, pp. 121–132. Springer (2008)

23. Sun, X., Wang, Z.: An efficient materialized views selection algorithm based on pso. In: Intelligent Systems and Applications, 2009. ISA 2009. International Workshop on, pp. 1–4. IEEE (2009)
24. Zhang, Q., Sun, X., Wang, Z.: An efficient ma-based materialized views selection algorithm. In: Control, Automation and Systems Engineering, 2009. CASE 2009. IITA International Conference on, pp. 315–318. IEEE (2009)
25. Goswami, R., Bhattacharyya, D., Dutta, M.: Materialized view selection using evolutionary algorithm for speeding up big data query processing. Journal of Intelligent Information Systems **49**(3), 407–433 (2017)
26. Gosain, A., Sachdeva, K.: Materialized view selection using backtracking search optimization algorithm. In: Intelligent Engineering Informatics, pp. 241–251. Springer (2018)
27. Kumar, T.V., Kumar, A.: Materialized view selection using set based particle swarm optimization. Int. J. Cogn. Inform. Nat. Intell. **12**(3), 18–39 (2018). DOI 10.4018/IJCINI.2018070102. URL `https://doi.org/10.4018/IJCINI.2018070102`
28. Prakash, J., Kumar, T.V.: A multi-objective approach for materialized view selection. International Journal of Operations Research and Information Systems (IJORIS) **10**(2), 1–19 (2019)
29. Aouiche, K., Darmont, J.: Data mining-based materialized view and index selection in data warehouses. Journal of Intelligent Information Systems **33**(1), 65–93 (2009)
30. Kumar, T.V.V., Singh, A., Dubey, G.: Mining queries for constructing materialized views in a data warehouse. In: D.C. Wyld, J. Zizka, D. Nagamalai (eds.) Advances in Computer Science, Engineering & Applications, pp. 149–159. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
31. Jouve, P., Nicoloyannis, N.: Kerouac: an algorithm for clustering categorical data sets with practical advantages. In: International Workshop on Data Mining for Actionable Knowledge (DMAK'2003, in conjunction with PAKDD03), vol. 70 (2003)
32. Jouve, P.E.: Apprentissage non supervisé et extraction de connaissances à partir de données. Ph.D. thesis, Université Lumière-Lyon 2 (2003)
33. Yıldız, O.T., Dikmen, O.: Parallel univariate decision trees. Pattern Recognition Letters **28**(7), 825–832 (2007)
34. Du, J., Glavic, B., Tan, W., Miller, R.J.: Deepsea: Progressive workload-aware partitioning of materialized views in scalable data analytics. In: International Conference on Extending Database Technology 2017, pp. 198–209. OpenProceedings.org (2017)
35. Chirkova, R., Halevy, A.Y., Suciu, D.: A formal perspective on the view selection problem. In: 27th International Conference on Very Large Data Bases, vol. 1, pp. 59–68 (2001)
36. O'Neil, P.E., O'Neil, E.J., Chen, X.: The star schema benchmark (ssb) (2009). URL `https://www.cs.umb.edu/~poneil/StarSchemaB.PDF`. Https://www.cs.umb.edu/˜poneil/StarSchemaB.PDF. Last accessed: 2018-11-28
37. Friedman, J., Hastie, T., Tibshirani, R.: Clustering analysis. In: The elements of statistical learning: Data mining, inference and prediction, chap. 14, pp. 501–520. Springer series in statistics, New York (2009)
38. Sharma, A., López, Y., Tsunoda, T.: Divisive hierarchical maximum likelihood clustering. BMC bioinformatics **18**(16), 546 (2017)
39. Müllner, D.: Modern hierarchical, agglomerative clustering algorithms. Computing Research Repository (CoRR) **abs/1109.2378** (2011). URL `http://arxiv.org/abs/1109.2378`
40. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics **20**(1), 53–65 (1987)

Leandro Ordonez-Ante received the M.Sc. degree in Telematics Engineering from the University of Cauca, Colombia in 2014. He is currently pursuing a Ph.D. in Computer Sciences within the Internet Technology and Data Science Lab (IDLab) Research Group at Ghent University - imec, Belgium. His current research deals with optimal methods for data storing and retrieval in large distributed data collections for supporting interactive applications.

Gregory Van Seghbroeck graduated at Ghent University in 2005. After working as an IT consultant, he joined the Department of Information Technology (INTEC) at Ghent University. In January 2007, he received a PhD grant from IWT to work on theoretical aspects of advanced validation mechanism for distributed interaction protocols and service choreographies. In 2011 he received his Ph.D. in Computer Science Engineering and continued to work at Ghent University as a post-doctoral fellow.

Tim Wauters received the M.Sc. and Ph.D.degrees in electro-technical engineering from Ghent University in 2001 and 2007, respectively. He has been working as a Postdoctoral Fellow of the F.W.O.-V. with the Department of Information Technology (INTEC), Ghent University and also as Senior Researcher with imec. His main research interests focus on network and service architectures for multimedia delivery services. His work has been published in about 80 scientific publications.

Bruno Volckaert received the Ph.D. degree from Ghent University in 2006. He is currently a Professor of software engineering with the Department of Information Technology (INTEC), Ghent University and also a Senior Researcher with imec. He has worked on over 35 national and international research projects. He is author or coauthor of more than 90 articles published in international journals and conference proceedings.

Filip De Turck leads the Network and Service Management Research Group, Department of Information Technology, Ghent University–imec, Belgium. He (co) authored over 450 peer reviewed articles. His research interests include network and service management and the design of efficient virtualized networks. He serves as the Chair for the IEEE Technical Committee on Network Operations and Management (CNOM) and a TPC for many network and service management conferences.