Executing Queries over Schemaless RDF Databases

Güneş Aluç, M. Tamer Özsu, Khuzaima Daudjee, Olaf Hartig
Cheriton School of Computer Science, University of Waterloo
{galuc,tamer.ozsu,kdaudjee,ohartig}@uwaterloo.ca

***Abstract*—Recent advances in Linked Data Management and the Semantic Web have led to a rapid increase in both the quantity as well as the variety of Web applications that rely on the SPARQL interface to query RDF data. Thus, RDF data management systems are increasingly exposed to workloads that are far more diverse and dynamic than what these systems were designed to handle. The problem is that existing systems rely on a workload-oblivious physical representation that has a fixed schema, which is not suitable for diverse and dynamic workloads. To address these issues, we propose a physical representation that is schemaless. The resulting flexibility enables an RDF dataset to be clustered based purely on the workload, which is key to achieving good performance through optimized I/O and cache utilization. Consequently, given a workload, we develop techniques to compute a good clustering of the database. We also design a new query evaluation model, namely, schemaless-evaluation that leverages this workload-aware clustering of the database whereby, with high probability, each tuple in the result set of a query is expected to be contained in at most one cluster. Our query evaluation model exploits this property to achieve better performance while ensuring fast generation of query plans without being hindered by the lack of a fixed physical schema.**

I. Introduction

With the proliferation of very large, web-scale distributed RDF datasets such as those in the Linked Open Data (LOD) cloud [1], the demand for high-performance RDF data management systems is increasing. While multiple RDF data management approaches have been proposed [2]–[10], systems are still unable to achieve consistently good performance [11]. A major problem is that workloads that these systems service are becoming far more diverse [12]–[14] and far more dynamic [15] than what the systems have been designed to support [16]. To make matters worse, this deficiency has not been revealed in performance studies because benchmark workloads do not truly capture this diversity and dynamism [11].

To demonstrate the issue, we conducted an experiment using the Waterloo SPARQL Diversity Test Suite (WatDiv)—a new benchmark that is specifically developed for identifying physical design issues in RDF data management systems [11]. We generated 100 million RDF triples using the WatDiv data generator and measured the performance of five popular RDF data management systems, namely, RDF-3x [7], MonetDB [17], 4Store [18] and Virtuoso Open Source (VOS) versions 6.1 [19] and 7.1 [8]. In our evaluations, we used the WatDiv stress testing tool to generate a diverse workload of 12500 unique SPARQL queries. Our observations can be summarized as follows (detailed results are available in [11]): (i) no single solution performs uniformly well, that is, systems that are fastest are only so for a small percentage of queries in the workload (cf., Table I); and (ii) there can be multiple orders of magnitude difference between the execution times of the fastest and the relatively slower systems (cf., Fig. 1).

	RDF-3x	*VOS [6.1]*	*VOS [7.1]*	*MonetDB*	*4Store*
% of queries for which tested system is fastest	20.9%	0.0%	22.6%	56.5%	0.0%
Total workload execution time (hours)	27.1	20.9	20.8	38.6	72.2
Mean (per query) execution time (seconds)	7.8	6.0	6.0	11.1	20.8

TABLE I: Summary of results over WatDiv 100M RDF triples.

When the workloads are diverse, choosing the most suitable system is a difficult task. One can deploy the system that efficiently executes the most frequent queries in a given workload. However, since the same system can be very inefficient in executing the remaining queries, the overall performance of the system can be far less than optimal. In fact, Table I illustrates that *none* of the systems that we benchmarked have amortized (i.e., per query) execution times of less than six seconds, which is unacceptable for interactive web applications [20].

In earlier work, we identified 2 reasons why existing systems run into the aforementioned problems [16]: First, these systems rely on a fixed, workload-oblivious physical representation. Second, none of these systems can dynamically update their physical layout or switch to a better representation as the workload changes. For example, it may be necessary to switch from a row-oriented representation to a columnar representation, but this is not possible with existing systems. Consequently, we proposed a vision for a workload-aware and adaptive RDF data management system. Our vision consists of two parts: (i) a schemaless physical representation of RDF

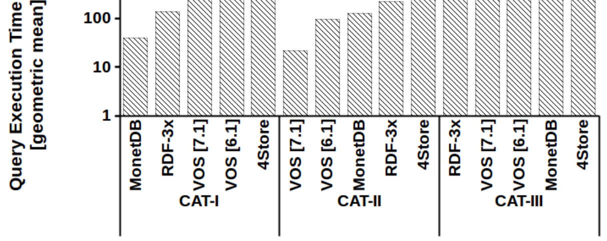
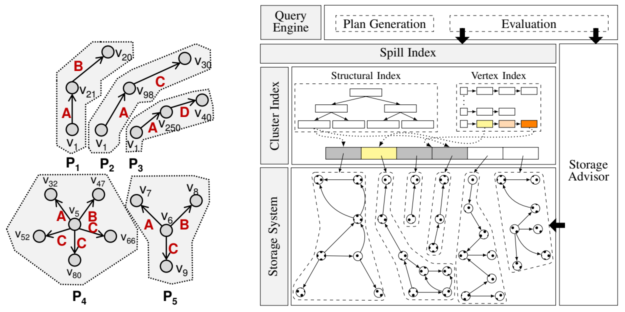


Fig. 1: Comparison of system performance. CAT-I, CAT-II and CAT-III consist of queries for which, respectively, MonetDB, VOS [7.1] and RDF-3x are the fastest systems (cf., Table I).



(a) Clusters (b) System overview

Fig. 2: Implementation of the group-by-query representation [16]

data that is purely workload driven, and (ii) partially and automatically tuning the physical representation [16].

In this paper, we focus on the first part of our vision and propose a *schemaless* **group-by-query** representation of RDF data. In this representation, based on the results of queries in the workload, RDF triples are physically clustered into what we call group-by-query clusters [16]. For example, in Fig. 2a, clusters P_1–P_3 are clustered based on a linear query, and P_4–P_5 are clustered based on a star-shaped query. These clusters are physically stored in the storage system as shown in the lower part of Fig. 2b. We create two types of indexes across these clusters: (i) cluster index and (ii) spill index. Given a query, the cluster index maintains information about which clusters are relevant to the execution of that query, and the spill index facilitates query plan generation.

In contrast to conventional (relational) records, clusters in the group-by-query representation do not have fixed sizes nor contain triples that have the same set of predicates. This schemaless representation enables easy customization of the physical data structures and indexes in the database based on the current workload, resulting in (i) more efficient I/O and cache utilization, (ii) better indexing and data localization, and (iii) fewer intermediate tuples during query evaluation.

On the other hand, there is a price for this flexibility—generating and executing valid query plans becomes more challenging than it is for fixed, non-adaptable representations. First, we do not have *any* a priori knowledge about how data will be clustered and physically organized in the storage system and within the indexes. All of these decisions depend on the current workload, and as the workload changes, the underlying group-by-query representation will change as well. This flexibility automatically rules out the possibility of designing and implementing query evaluation code in the DBMS based on a fixed representation. Systems such as RDF-3x [7], MonetDB [17] and gStore [10] are all implemented in this fashion. Second, there is no physical schema to describe the group-by-query representation that can be used to efficiently generate valid query plans, which relational systems rely on heavily for query plan generation and optimization [21]. Without addressing these two challenges, query plan generation and optimization can easily become a bottleneck.

Consequently, in addition to introducing the group-by-query representation, we make the following contributions:

1) We develop methods to quantify the "goodness" of a clustering. Using these measures of "goodness", we introduce a practical clustering algorithm to compute a suitable group-by-query representation for a given workload (Section IV).
2) We introduce a new query evaluation model, namely, schemaless-evaluation (SE), and along with it, a set of new query operators that can accommodate the following two requirements (Section V):
   - **Genericity:** For any possible group-by-query representation, SE guarantees correct evaluation of queries. This way, the underlying group-by-query representation can be adjusted safely for any type of workload.
   - **Isolation:** Without violating correctness guarantees, group-by-query representation can be updated on one part of the database, while schemaless-evaluation concurrently takes place on other parts. This allows adaptivity to workload changes, which is not addressed in this paper but is part of our vision [16] (Section V-6).
3) We propose new query optimization techniques based on the observation that in the group-by-query representation, since triples are already physically clustered, the probability that tuples in the result set of a query are contained in *at most* one cluster is high. This creates new opportunities for generating more efficient query plans that are exploited by SE. These optimizations are designed such that they can work efficiently without the full knowledge of the physical schema of the underlying database (Section V-2).
4) We experimentally quantify the benefits of the group-by-query representation and our query evaluation algorithm over workload-oblivious techniques employed by other RDF data management systems (Section VI).

II. Related Work

RDF is composed of subject-predicate-object (s, p, o) statements called *triples* [22]. Each triple describes an aspect of a web resource. The subject of the triple denotes the resource that is described, the predicate denotes a feature of that resource, and the object stores the value for that feature.

One of the major challenges in RDF data management is physical design, which has led to the development of multiple "optimal" physical representations. One option here is to represent data in a single large table with only three attributes: s, p and o [2]. As a slight variation of this representation, another option is to maintain multiple copies of the table, where each table has an index that implements a different sort-order [4], [7], [23]. It has also been argued that for different workloads, grouping data can provide performance benefits [3], [9], [24]. Therefore, two other representations were developed: (i) **grouping by predicates**, where the RDF database is partitioned into 2-column tables (one table per predicate) with the tables being stored in a column-store [5]; and (ii) **grouping by entities**, where implicit relationships within the data are determined (either as a manual or automated process) to compute a relational schema, and data are mapped to an instantiation of this schema [3], [9]. Another alternative is to rely on the native graph structure of the RDF data [6],

[10], [25], [26]. In this case, **grouping by graph vertices**, whereby edges in the RDF graph are grouped based on their incidence on a vertex, is a feasible representation.

RDF data management systems, whether single node [2]–[10] or distributed [27], rely on one of the above physical representations. Our studies have demonstrated that with increasing diversity and dynamism in SPARQL workloads, all of these existing physical representations run into serious issues [11]. This increases the need for the workload-driven **group-by-query** representation [16] (Fig. 2a).

III. Background and Preliminaries

In this paper, we use graphs to represent both RDF data and the conjunctive fragment of SPARQL queries [28], which is called basic graph patterns (BGPs); and we model query evaluation as a subgraph isomorphism problem [29]. Therefore, for the most part, we rely on the standard formalization of SPARQL [30], and introduce only the concepts necessary to capture subgraph isomorphism as it is used in evaluating BGPs over RDF graphs. In an extended version of our paper [31], we prove the equivalence between the standard formalization of SPARQL [30], [32], and our framework.

Assume two disjoint, countably infinite sets $\mathcal{U}$ (URIs) and $\mathcal{L}$ (literals) (we ignore blank nodes in our discussions). URIs uniquely denote Web resources or features of Web resources. Literals denote values such as strings, natural numbers and booleans. Then, an *RDF triple* is a 3-tuple from the set $\mathcal{T} = \mathcal{U} \times \mathcal{U} \times \big(\mathcal{U} \cup \mathcal{L}\big)$.

Definition 1: An *RDF graph* is a directed, labeled multi-graph $G = (V, E)$ where: (i) the vertices (V) are URIs or literals such that $V \subset (\mathcal{U} \cup \mathcal{L})$; (ii) the directed, labeled edges (E) are RDF triples such that $E \subset \big(V \times \mathcal{U} \times V\big) \cap \mathcal{T}$; and (iii) each vertex $v \in V$ appears in at least one edge, where for each edge $(s, p, o) \in E$, s is the source of the edge, p is the label, and o is the target of the edge. Hereafter, we use $V(G)$ and $E(G)$ to denote the set of vertices and the set of edges of an RDF graph, respectively.

To define queries, we assume a countably infinite set of variables $\mathcal{V}$ that is disjoint from both $\mathcal{U}$ and $\mathcal{L}$. Similar to RDF graphs, we use a graph-based representation for BGPs.

Definition 2: A *basic graph pattern (BGP)* is a directed, labeled multi-graph $Q = (\hat{V}, \hat{E})$ where: (i) the vertices $(\hat{V})$ are variables, URIs, or literals such that $\hat{V} \subset \mathcal{V} \cup \mathcal{U} \cup \mathcal{L}$; (ii) the directed, labeled edges $(\hat{E})$ are 3-tuples such that $\hat{E} \subset \hat{V} \times (\mathcal{V} \cup \mathcal{U}) \times \hat{V}$, where for each edge $(\hat{s}, \hat{p}, \hat{o}) \in \hat{E}$, $\hat{s}$ is the source of the edge, $\hat{p}$ is the label, and $\hat{o}$ is the target of the edge; (iii) each vertex $\hat{v} \in \hat{V}$ appears in at least one edge.

Note that each edge in a BGP represents a triple pattern. Therefore, depending on the context, we will use these two terms interchangeably.

The only deviation from the standard formalism [30] is in the way solution mappings are defined for BGPs because we rely on a graph-based formalism. That is, for BGPs, solution mappings are computed from subgraphs of a queried RDF graph that match the BGP. This is very similar to the notion of a "match" in the context of subgraph isomorphism [29] except for the presence of variables in SPARQL. To accommodate this difference, we first introduce *compatibility* between an edge in an RDF graph and an edge in a BGP (Def. 3). Informally, two edges are compatible if they have the potential to match. Formally:

Definition 3: Let $e = (s, p, o) \in E$ be an edge in an RDF graph $G = (V, E)$, and let $\hat{e} = (\hat{s}, \hat{p}, \hat{o}) \in \hat{E}$ be an edge in a BGP $Q = (\hat{V}, \hat{E})$. Edges e and $\hat{e}$ are *compatible* if either (i) $p = \hat{p}$, or (ii) $\hat{p} \in \mathcal{V}$.

Using the notion of edge compatibility, we define a match between a BGP and an RDF graph as surjection from the edges (and vertices) of a BGP onto the edges (and vertices) of an RDF graph (possibly a subgraph of the queried RDF graph) such that corresponding edges are compatible and the source (and the target) vertices of a pair of corresponding edges are also mapped onto.

Definition 4: Let $G = (V, E)$ be an RDF graph, and let $Q = (\hat{V}, \hat{E})$ be a BGP. Given a solution mapping μ, G *μ-matches* Q if (i) $\mathrm{dom}(\mu)$ is the set of variables mentioned in Q, and (ii) there exist two surjective functions $M_V : \hat{V} \to V$ and $M_E : \hat{E} \to E$ such that:

- for each $(\hat{v}_1, v_2) \in \hat{V} \times V$ with $M_V(\hat{v}_1) = v_2$: if $\hat{v}_1 \in \mathcal{V}$, then $\mu(\hat{v}_1) = v_2$, else $\hat{v}_1 = v_2$;
- for each $(\hat{e}_1, e_2) \in \hat{E} \times E$ with $M_E(\hat{e}_1) = e_2$: $\hat{e}_1 = (\hat{s}_1, \hat{p}_1, \hat{o}_1)$ and $e_2 = (s_2, p_2, o_2)$, (a) $\hat{e}_1$ and e_2 are compatible and if $\hat{p_1} \in \mathcal{V}$, then $p_2 = \mu(\hat{p_1})$, and (b) if $M_V\big(\hat{s}_1\big) = s_2$, then $M_V\big(\hat{o}_1\big) = o_2$.

G matches Q if there exists a solution mapping μ such that G μ-matches Q.

Putting it all together, we define the expected result of evaluating a BGP over an RDF graph as follows.

Definition 5: The *result* of a BGP Q over an RDF graph $G = (V, E)$, denoted by $[\![Q]\!]_G$, is defined as $[\![Q]\!]_G = \big\{\mu \,\big|\, G'$ is a subgraph of G and G' μ-matches $Q\big\}$.

BGPs can be combined using operators AND, UNION, and OPT [30]. Thus, any BGP is a *SPARQL query*, and if S_1 and S_2 are SPARQL queries, and F is a filter expression, then expressions $(S_1 \text{ AND } S_2)$, $(S_1 \text{ UNION } S_2)$, $(S_1 \text{ OPT } S_2)$, and $(S_1 \text{ FILTER } F)$ are also SPARQL queries. The semantics of these queries can be defined using the standard formalism [30], where solution mappings are combined or manipulated using union $(\cup)$, join $(\bowtie)$, difference $(\setminus)$ and selection (Θ).

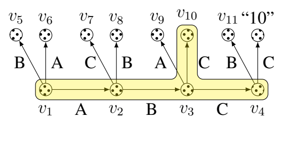
Our prototype implementation can handle full SPARQL 1.0 specification (except for complex filter expressions involving built-in functions), and additionally push filter expressions down into BGPs [32]. However, we deliberately left these definitions out to avoid complicated formalism. Furthermore, for the implementation of joins $(\bowtie)$, unions $(\cup)$ and set difference $(\setminus)$, we use existing techniques [7]. Therefore, in the remainder of this paper, we will focus on Def. 5, which defines the query result over all subgraphs of an RDF graph that match a BGP.

IV. Computing Group-by-Query Clusters

In this section, we introduce our workload-driven algorithm for computing the group-by-query clusters. First, we formulate

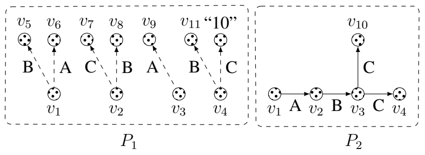
S	P	O
$\mathbf{v_1}$	$\mathbf{A}$	$\mathbf{v_2}$
v_1	A	v_6
v_1	B	v_5
$\mathbf{v_2}$	$\mathbf{B}$	$\mathbf{v_3}$
v_2	B	v_8
v_2	C	v_7
v_3	A	v_9
$\mathbf{v_3}$	$\mathbf{C}$	$\mathbf{v_4}$
$\mathbf{v_3}$	$\mathbf{C}$	$\mathbf{v_{10}}$
v_4	B	v_{11}
v_4	C	"10"

(a) Set of triples



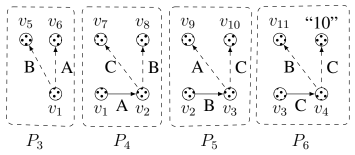
v_5 v_6 v_7 v_8 v_9 v_{10} v_{11} "10"
B A C B A C B C
v_1 A v_2 B v_3 C v_4

(b) Graph representation



v_5 v_6 v_7 v_8 v_9 v_{11} "10"
B A C B A B C
v_1 v_2 v_3 v_4
v_{10}
C
v_1 A v_2 B v_3 C v_4
P_1
P_2

(c) Clustering A



v_5 v_6 B A v_1
v_7 v_8 C B v_1 A v_2
v_9 v_{10} A C v_2 B v_3
v_{11} "10" B C v_3 C v_4
P_3
P_4
P_5
P_6

(d) Clustering B

$?w$	$?x$
$\mathbf{v_1}$	$\mathbf{v_2}$
v_1	v_6
v_3	v_9

(e) T_1

$?x$	$?y$
v_1	v_5
$\mathbf{v_2}$	$\mathbf{v_3}$
v_2	v_8
v_4	v_{11}

(f) T_2

$?w$	$?x$	$?y$
$\mathbf{v_1}$	$\mathbf{v_2}$	$\mathbf{v_3}$
v_1	v_2	v_8

(g) $T_1 \bowtie T_2$

$?y$	$?z$
v_2	v_7
$\mathbf{v_3}$	$\mathbf{v_4}$
$\mathbf{v_3}$	$\mathbf{v_{10}}$
v_4	"10"

(h) T_3

$?w$	$?x$	$?y$	$?z$
$\mathbf{v_1}$	$\mathbf{v_2}$	$\mathbf{v_3}$	$\mathbf{v_4}$
$\mathbf{v_1}$	$\mathbf{v_2}$	$\mathbf{v_3}$	$\mathbf{v_{10}}$

(i) $T_1 \bowtie T_2 \bowtie T_3$

Fig. 3: Sample dataset, their graph representation and sample group-by-query clusterings.

an objective function (Section IV-1) followed by a description of the algorithm (Section IV-2). Finally, we discuss how the underlying physical representation is updated following the computation of a group-by-query clustering (Section IV-3).

1) ***Clustering Objectives:*** Ideally, we would like to compute a group-by-query clustering in which RDF triples (or equivalently, edges in the graph representation) that are irrelevant to the evaluation of a query are clearly separated from the relevant ones, and this property should hold for as many queries in the workload as possible.

For instance, consider evaluating the linear query $Q = ?w \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$ against the graph in Fig. 3b. Any triple that lies outside the shaded region is irrelevant for the query result. Therefore, for this query, Clustering A in Fig. 3c is a better choice than Clustering B in Fig. 3d for three key reasons. First, triples from which the query result is computed are already physically clustered in the storage system. This will have significant performance benefits due to more efficient I/O and cache utilization. Second, indexes that are built over this representation will be much more efficient in localizing query evaluation to only the relevant parts of the database. In this case, P_1 does not contain any relevant triples, hence, it can be easily pruned out. Third, as a consequence of the last point, query evaluation will not produce any irrelevant intermediate results, reducing the overall cost of query evaluation.

Next, consider Clustering B (Fig. 3d), where irrelevant triples are mixed with triples relevant for query Q. In this case, the query needs to be decomposed into three sub-queries: $Q_1 = ?w \xrightarrow{A} ?x$, $Q_2 = ?x \xrightarrow{B} ?y$ and $Q_3 = ?y \xrightarrow{C} ?z$ (we will postpone the discussion of identifying this decomposition to Section V, which describes our query evaluation model). Then, each subquery is evaluated over clusters P_3–P_6, producing three tuple sets T_1, T_2 and T_3, respectively. Finally, these tuples are joined as shown in Fig. 3e–3i. In contrast to Clustering A, there are problems with Clustering B. First, triples from which the query result is computed are defragmented across the storage system, increasing I/O cost and reducing cache utilization. Second, during query evaluation, it is more difficult to distinguish between relevant and irrelevant triples, which generates unnecessary intermediate result tuples (cf., Fig. 3e–3i). In this case, reordering the join operations or applying sideways information passing [33] to early-prune some of the tuples in T_1–T_3 would not eliminate the problem. For example, while tuples (v_1, v_5) and (v_4, v_{11}) can be eliminated from T_2 as soon as v_2 is identified as the only join value in $T_1 \bowtie T_2$, tuple (v_2, v_8) remains in the pipeline until the end of joins.

Consequently, given a query, to quantify how well a group-by-query clustering separates irrelevant triples from the relevant ones, we use a combination of two measures: *segmentation* and *minimality*. Informally, segmentation is a measure of how distributed the subgraphs that match a BGP are across the group-by-query clusters. Minimality indicates how minimal clusters are with respect to those subgraphs that match a BGP. These measures are introduced formally in Def. 7, after we define our notion of a group-by-query clustering.

Definition 6: Given an RDF graph $G = (V, E)$, a *group-by-query clustering* of G is a set of RDF graphs $\mathbb{P} = \{P_1, \ldots, P_m\}$ such that (i) each P_i is a subgraph of G, (ii) P_i's are edge disjoint, (iii) $E(G) = \bigcup_{P_i \in \mathbb{P}} E(P_i)$, and (iv) $V(G) = \bigcup_{P_i \in \mathbb{P}} V(P_i)$.

Definition 7: Given a clustering $\mathbb{P}$ of an RDF graph G, let Γ_G^Q denote the set of all distinct subgraphs of G that match a BGP Q, and let $E^* = \bigcup_{G' \in \Gamma_G^Q} E(G')$. Then, *segmentation* and *minimality* of $\mathbb{P}$ with respect to Q are defined as follows:

$$segm_{\mathbb{P}}^{Q} = \left|\{(G',P) \in \Gamma_G^Q \times \mathbb{P} \mid E(G') \cap E(P) \neq \emptyset\}\right| - \left|\Gamma_G^Q\right|$$
$$minim_{\mathbb{P}}^{Q} = \left|E^*\right| / \left|\{E(P) \mid P \in \mathbb{P} \text{ and } E(P) \cap E^* \neq \emptyset\}\right|$$

The definitions of segmentation and minimality can be easily extended to a query workload $\mathbb{W} = \{Q^1, \ldots, Q^n\}$: $segm_{\mathbb{P}}^{\mathbb{W}} = \sum_{i=1}^{n} segm_{\mathbb{P}}^{Q^i} / |\mathbb{W}|$ and $minim_{\mathbb{P}}^{\mathbb{W}} = \sum_{i=1}^{n} minim_{\mathbb{P}}^{Q^i} / |\mathbb{W}|$.

Segmentation can take any positive real value, while minimality is always between [0, 1]. An *ideal clustering* for a workload is one whose segmentation is minimal (0) and minimality takes the highest possible value (1). We say that a clustering is *completely segmented* with respect to a query workload if its segmentation is maximal.

2) ***Clustering Algorithm:*** To facilitate the computation of a suitable group-by-query clustering, upon the execution of a BGP, we annotate each distinct subgraph that matches the BGP with a unique label and a timestamp of query submission.

Each annotation is of the form $\langle qid, sid, t\rangle$, where qid is the unique identifier generated by the system for every BGP that is executed, sid is the unique identifier for the corresponding subgraph, and t is the timestamp. Since matching subgraphs can be overlapping, for each annotated edge of the RDF graph we maintain the annotations in a (FIFO) queue. This way, when clustering takes place, we can ignore annotations that are too old to be relevant for the current workload.

To compute the group-by-query clustering, we use a hierarchical algorithm, which starts from a completely segmented clustering, and successively merges clusters until the clustering objective is achieved. The algorithm operates as follows:

1) Initially, each edge of the RDF graph resides in its own cluster, which corresponds to a completely segmented group-by-query clustering for any possible workload;
2) The pair of clusters, whose merging improves segmentation the most, while causing the least trade-off in minimality, is identified (race conditions will be discussed shortly);
3) Clusters found in Step 2 are merged, which results in a potential decrease in segmentation and/or minimality;
4) Steps 2–3 are repeated as long as the aggregate minimality of the clustering is greater than a threshold.

It is important to note that segmentation and minimality measures are monotonically decreasing within this algorithm. That is, whenever two clusters are merged, segmentation will potentially decrease because edges with the same qid and sid labels may be brought together. However, at the same time, edges with different qid labels may also be placed in the same cluster, which does not affect segmentation, but reduces minimality. While we would like to reduce segmentation, we would also like to increase minimality. When clusters contain too many edges that are individually irrelevant to the execution of the majority of the queries, the overhead of subgraph matching within each cluster can undermine the benefits of reduced segmentation. In our implementation, we observed that if the clusters contain on average more than 10 times as many irrelevant triples as there are relevant ones, performance of query evaluation starts to degrade. For this reason, we set the threshold on minimality as 0.1.

There are two reasons for choosing a hierarchical clustering algorithm. First, the way hierarchical clustering works is aligned with our clustering objectives as clusters are merged one pair at a time until a global objective is achieved. This is not true for centroid-based clustering [34] or spectral clustering [34]. Second, other algorithms such as k-means [34] require the final number of clusters to be known in advance, which is not possible in our case.

As an assumption that generally holds, we expect the final clustering to be fine-grained since subgraphs that match the queries in the workload are likely to be comparable in size to the query graphs, which are relatively small. Furthermore, the final clusters are not likely to be much larger than these subgraphs due to the minimality threshold. Therefore, a bottom-up (agglomerative) approach can reach the clustering objective in fewer number of iterations than a top-down (divisive) approach (hence, the reason why we start with a completely segmented clustering and employ agglomerative clustering).

A critical issue is to decide which pair of clusters to merge in each iteration. We define a distance function $\delta : \mathbb{P} \times \mathbb{P} \to [0, 1]$ over the clusters such that: (i) $\delta = 1$ is reserved for clusters that should not be merged; (ii) a smaller distance between two clusters implies that the decrease in segmentation is higher (with a lower trade-off in minimality) if these two clusters are merged.

To compute the pairwise distances between clusters, we rely on the annotations of edges in each pair of clusters, namely, the set of $\langle qid, sid, t\rangle$-tuples. We define the distance between a pair of clusters as a combination of two Jaccard distances: δ_S is defined over the sets of subgraph identifiers (sid), and δ_Q is defined over the sets of query identifiers (qid). For any cluster $P \in \mathbb{P}$, let $\pi_s(P)$ and $\pi_q(P)$ denote the set of subgraph identifiers and the set of query identifiers with which P is annotated, respectively. Given two clusters P_1 and P_2, the distances $\delta_S(P_1, P_2)$ and $\delta_Q(P_1, P_2)$ are defined as follows:

$$\delta_S = 1 - \frac{|\pi_s(P_1) \cap \pi_s(P_2)|}{|\pi_s(P_1) \cup \pi_s(P_2)|},$$
$$\delta_Q = 1 - \frac{|\pi_q(P_1) \cap \pi_q(P_2)|}{|\pi_q(P_1) \cup \pi_q(P_2)|}.$$

The two distance functions are complementary. That is, by merging P_1 with P_2, segmentation decreases by at least $|\pi_s(P_1) \cap \pi_s(P_2)|$, therefore, δ_S is more sensitive to predicting the expected change in segmentation. Likewise, $|\pi_q(P_1) \cup \pi_q(P_2)| - |\pi_q(P_1) \cap \pi_q(P_2)|$ is a more accurate approximation of the expected decrease in minimality; thus, δ_Q is more sensitive to changes in minimality. Hence, our reliance on a combination of both distances. However, in doing so, we pay particular attention to some race conditions. Specifically, the distance function is designed such that the following order, in which clusters are merged, is always preserved: (i) a pair of clusters with $\delta_S = 0$ (which also implies that $\delta_Q = 0$) are merged before any other pair of clusters; (ii) clusters with $\delta_S \neq 0$ and $\delta_Q = 0$, are merged next; (iii) finally, clusters with $\delta_S \neq 0$ and $\delta_Q \neq 0$ are merged according to a combined distance $\delta = \alpha\delta_S + (1 - \alpha)\delta_Q$, where $\alpha = 0.5$.

Note that in the first two cases, minimality will not decrease because the two clusters that are merged have subgraphs that match only a single query. Hence, they are preferred over the third case, in which minimality is expected to decrease. Furthermore, even though the first and second cases are both guaranteed to reduce segmentation (without compromising minimality), the first case can achieve the same objective with smaller clusters, hence, it is preferred over the other. When two clusters P_1 and P_2 are merged, all distances between the new cluster and any other existing cluster P_x for which $\delta(P_1, P_x) < 1$ or $\delta(P_2, P_x) < 1$ need to be updated.

*3) **Updating the Physical Representation**:* Once a suitable clustering is computed, the system performs the transformation from the current physical representation to the desired one as a set of atomic update operations (i.e., deletion and insertion) on the set of physical clusters in the storage system. Each operation has the property that before and after the operation, the database represents exactly the same RDF graph but using a different clustering.

The update operations are executed concurrently with the queries, which is possible because our schemaless evaluation

1) Compute the result of the *whole* BGP over the *whole* RDF graph according to Def. 5.

(a) Algorithm I: HolisticEvaluation

1) Decompose the BGP into its triple patterns;
2) For each triple pattern,
   a) Use indexes to locate only those clusters that have a subgraph that matches the triple pattern;
   b) For each satisfying cluster, compute the result of the triple pattern over that cluster according to Def. 5;
   c) Take the union of the results; and
3) Join the intermediate results from Step 2.

(b) Algorithm II: TriviallyDecomposedEvaluation

1) Use indexes to locate only those clusters that have a subgraph that matches the *whole* BGP;
2) For each satisfying cluster, compute the result of the *whole* BGP over that cluster according to Def. 5; and
3) Take the union of the results.

(c) Algorithm III: OptimalEvaluation

1) Preferably, do not decompose the BGP at all; however, if necessary, decompose it into as few number of segments as possible;
2) For each query segment, execute OptimalEvaluation.
3) Join the intermediate results from Step 2.

(d) Algorithm IV: SchemalessEvaluation

Fig. 4: Alternative query evaluation algorithms

approach (SE) is designed with an isolation guarantee: once results are computed within a cluster, query evaluation does not need to access that cluster anymore, allowing it to be updated while query evaluation proceeds over other clusters. In order to ensure that updates do not take place before a query has completely "consumed" the contents of a cluster, we use a two-level locking scheme (details omitted in this paper).

V. Query Evaluation

A good clustering algorithm is not sufficient for exploiting the optimizations that the group-by-query representation provides. The problem is that a query evaluation algorithm that is oblivious to the underlying group-by-query representation can easily obscure and even reverse the effects of clustering.

Consider 2 typical ways of evaluating BGPs over fixed, non-adaptable representations: In HolisticEvaluation (Fig 4a), the *whole* BGP is evaluated over the *whole* RDF graph [10], whereas in TriviallyDecomposedEvaluation (Fig. 4b), the BGP is decomposed into its triple patterns, and each triple pattern is evaluated independently over a clustering of the graph [25].

While both algorithms produce correct results over the group-by-query representation, they would be far from the optimal choice. On the one hand, as experiments show [11], HolisticEvaluation needs to consider the entire RDF graph, which may lead to processing of irrelevant parts even when the graph has been indexed (cf., gStore [10]). TriviallyDecomposedEvaluation, on the other hand, decomposes the BGP all the way down to its triple patterns, which results in suboptimal performance. Consider evaluating $Q = ?w \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$ over Clustering A (cf., Fig. 3c). Despite the fact that Clustering A is the optimal one for Q (cf., Section IV), in Step 2(a) of TriviallyDecomposedEvaluation, indexes cannot efficiently localize query evaluation to only P_2 because P_1 contains at least one edge for each label A, B and C. That is, P_1 contains a match for each triple pattern in the query. This results in the generation of irrelevant intermediate result tuples that may remain in the query evaluation pipeline until all the joins in Step 3 are completed. Thus, TriviallyDecomposedEvaluation not only performs unnecessary computations, but it also results in poor I/O and cache utilization.

In short, neither HolisticEvaluation nor TriviallyDecomposedEvaluation truly exploits the fact that triples in the group-by-query representation are already being clustered based on the results of the queries in the workload. This is a useful property because given a query from the workload, it is very likely that subgraphs that match the query (i.e., subgraphs that contribute to the result of the query as per Def. 5), are each contained within at most a single (but not necessarily the same) cluster. Recall how the two subgraphs in Clustering A in Fig. 3c that match Q are each contained in a single cluster.[1] Intuitively, if the aforementioned conditions hold, the correct result of a BGP can be obtained (i) without decomposing the BGP at all, and (ii) by evaluating the whole BGP independently over each cluster in the group-by-query representation and taking the union of the results, thereby avoiding the join step of TriviallyDecomposedEvaluation. We capture these optimizations in OptimalEvaluation (Fig. 4c).

OptimalEvaluation is much more efficient than both HolisticEvaluation and TriviallyDecomposedEvaluation. First, when evaluating Q over Clustering A, P_1 can be pruned out already in the first step of the algorithm, which results in good data localization. Second, since the query is evaluated *entirely* over P_2 (as opposed to being decomposed), query evaluation does not produce any irrelevant intermediate result tuples—in fact, there are no intermediate results. Needless to say, by fetching only a single cluster from the storage system, the algorithm also achieves better I/O and cache utilization.

Naturally, the storage advisor strives to compute a group-by-query clustering such that for every query in the workload, every subgraph that matches that query spans at most one cluster. However, sometimes, this may be too ambitious to achieve. In practice, the above condition may not hold for some queries in the workload, for which OptimalEvaluation would not be applicable. Even in that case, we argue that reverting all the way down to the decomposition into triple patterns (i.e., TriviallyDecomposedEvaluation) may be unnecessary. Therefore, we propose SchemalessEvaluation (Fig. 4d) that encapsulates both TriviallyDecomposedEvaluation and OptimalEvaluation, but depending on the underlying group-by-query representation, can

[1]Coincidentally, in this example, both subgraphs are contained also within the *same* cluster, but that is not a necessary condition.

accommodate a whole range of decompositions in between. Before any formalization, we answer some questions.

Q1— In SchemalessEvaluation, how can a decomposition of the query be found that produces the *correct* result?

A – We follow a bottom-up approach. That is, we start with the decomposition of the query into its triple patterns (i.e., TriviallyDecomposedEvaluation), which always produces correct results regardless of the underlying group-by-query representation (cf., Theorem 1 in Section V-1), and rely on equivalence rules to simplify the decomposition. These equivalence rules are conditional, and they exploit various properties about graphs to dynamically determine whether subgraphs that match the query spill into multiple group-by-query clusters (Section V-2).

Q2— How can one *efficiently* determine, at runtime, whether any of the matching subgraphs of a query spill into multiple group-by-query clusters?

A – We follow a lazy approach. Initially i.e., whenever a query is evaluated for the first time, it is assumed that all of the matching subgraphs of the query spill into multiple clusters. However, as queries are evaluated, summary information is maintained, which is used in firing the conditional equivalence rules in subsequent queries. We call this the spill index (Section V-4).

Q3— In Step 1 of OptimalEvaluation, how are the relevant clusters determined?

A – We use another index, called the cluster index. This index is also constructed in a lazy fashion. That is, given the first query, the index assumes that any of the clusters could be relevant to the evaluation of the query. However, as queries are evaluated, it uncovers more information about the clusters and indexes them (Section V-5).

Q4— What data structures are utilized to facilitate subgraph matching within a cluster?

A – To perform subgraph matching within each cluster, we represent each RDF graph in the group-by-query cluster as an adjacency list and use a variation of Ullmann's algorithm [29]. For each vertex $\hat{v}$ in the BGP, we compute candidate matching vertices in the RDF graph. If $\hat{v}$ is a URI or literal, one can directly lookup the vertex in the adjacency list. Otherwise, if $\hat{v}$ is a variable, we rely on the labels of the edges that are incident on $\hat{v}$ to prune the search space. While it is possible to build an index (other than adjacency list) over each cluster to facilitate subgraph matching, it is outside the scope of this paper.

*1) **Building Blocks of Schemaless-Evaluation (SE)**:* OptimalEvaluation and SchemalessEvaluation rely on two new operations: *prune* (Def. 8) and *clustered-match* (Def. 9). Prune corresponds to Step 1 of OptimalEvaluation, while clustered-match corresponds to Steps 2 and 3.

Definition 8: Given a clustering $\mathbb{P}$ of an RDF graph and a BGP Q, a *prune* of $\mathbb{P}$ with respect to Q, which is denoted by $\sigma_Q(\mathbb{P})$, is defined as $\sigma_Q(\mathbb{P}) = \{P \in \mathbb{P} \mid [\![Q]\!]_P \neq \emptyset\}$.

The key aspect of prune is that unless there is a subgraph in a cluster that matches the *whole* query, that cluster will be discarded, even if the query has partial matches. We will exploit this property further when building indexes over the clusters (cf., Section V-5).

Definition 9: Let Q be a BGP, $\mathbb{P}$ be a clustering of an RDF graph G and $\mathbb{P}' \subset \mathbb{P}$. The *clustered-match* of Q over $\mathbb{P}'$, denoted as $Q\lfloor\mathbb{P}'\rfloor$, is defined by $[\![Q\lfloor\mathbb{P}'\rfloor]\!]_G = \bigcup_{P\in\mathbb{P}'}[\![Q]\!]_P$.

Clustered-match is different from standard match in that $[\![Q]\!]_G = [\![Q\lfloor\mathbb{P}\rfloor]\!]_G$ will hold only if every subgraph of G that matches Q is contained in at most one cluster in $\mathbb{P}$. Note that this is *exactly* the objective of the group-by-query clustering (cf., Section IV). Thus, for most queries in the workload, we expect to rely on OptimalEvaluation to compute the correct query results. Of course, for cases when the clustering algorithm achieves its objective only partially, the query will have to be decomposed into smaller segments (cf., SchemalessEvaluation) such that for each segment Q_i, $[\![Q_i]\!]_G = [\![Q_i\lfloor\mathbb{P}\rfloor]\!]_G$ holds. To determine a good decomposition (ideally, one with the least number of segments), we start with a decomposition that always produces the correct query result regardless of how the RDF graph is clustered, and compute a better decomposition by dynamically analyzing the current state of the clustering. Next, we formalize these concepts. We start by defining so called SE *expressions*.

SE expressions can be defined recursively. Given a BGP Q and a clustering of an RDF graph $\mathbb{P}$, $Q\lfloor\mathbb{P}\rfloor$ and $Q\lfloor\sigma_Q(\mathbb{P})\rfloor$ are SE expressions (they correspond to Steps 1 and 2 of OptimalEvaluation, respectively). If M_1 and M_2 are SE expressions, then so are $(M_1 \cup M_2)$ and $(M_1 \bowtie M_2)$ (they correspond to Step 3 of OptimalEvaluation and Step 3 of SchemalessEvaluation, respectively).

We show that with the the *trivial decomposition* of a BGP, in which the BGP is decomposed into its triple patterns (Def. 10), we can guarantee the construction of an SE expression M such that $[\![Q]\!]_G = [\![M]\!]_G$ for any BGP Q and any clustering $\mathbb{P}$ of an RDF graph G (Theorem 1). We call this expression the *baseline* SE *expression* (Def. 11).

Definition 10: Given a BGP $Q = (\hat{V}, \hat{E})$, then the *trivial decomposition* of Q is defined as the set $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ of BGPs, where each $Q_i \in \mathcal{Q}$ contains exactly one edge, the set of edges in each Q_i are disjoint, $\hat{V} = \bigcup\limits_{i=1}^{k} V(Q_i)$, $\hat{E} = \bigcup\limits_{i=1}^{k} E(Q_i)$.

Definition 11: Let Q be a BGP, let G be an RDF graph, and let $\mathbb{P}$ be a clustering of G. If $\{Q_1, \ldots, Q_k\}$ is the trivial decomposition of Q, then the *baseline* SE *expression* for Q over $\mathbb{P}$ is $Q_1\lfloor\mathbb{P}\rfloor \bowtie \cdots \bowtie Q_k\lfloor\mathbb{P}\rfloor$.

Theorem 1: Given a BGP Q, an RDF graph G and a clustering $\mathbb{P}$ of G, $[\![Q]\!]_G = [\![M]\!]_G$, where M is the baseline SE expression for Q over $\mathbb{P}$.

The proof of Thm. 1 and all subsequent proofs are in the extended version of our paper [31].

*2) **Query Rewriting Rules**:* To realize the aforementioned rewrite of the baseline expression into an equivalent expression with fewer number of join operations, we introduce equivalence rules that are in two categories: generic and conditional. Generic rules are applicable irrespective of how the RDF graph is clustered, whereas the applicability of a conditional rule depends on whether the clustering satisfies certain conditions.

Assuming Q_A and Q_B are two BGPs, let $\mathbb{P}$ be a clustering of an RDF graph with subsets $\mathbb{P}_1, \ldots, \mathbb{P}_m$ ($\mathbb{P}_i \subseteq \mathbb{P}$ for all

	Name	Equivalence Rules	Condition
1	Expansion	$[\![Q_A\lfloor\mathbb{P}\rfloor]\!] = \bigcup_{i=1}^{m} [\![Q_A\lfloor\mathbb{P}_i\rfloor]\!]$	$\bigcup_{i=1}^{m} \mathbb{P}_i = \mathbb{P}$
2	Join elimination*	$[\![Q_A\lfloor\mathbb{P}_1\rfloor \bowtie Q_B\lfloor\mathbb{P}_2\rfloor]\!] = \emptyset$	Thm. 2
3	Join reduction*	$[\![Q_A\lfloor\mathbb{P}_1\rfloor \bowtie Q_B\lfloor\mathbb{P}_1\rfloor]\!]$ $= [\![(Q_A \oplus Q_B)\lfloor\mathbb{P}_1\rfloor]\!]$	Thm. 3
4	Identity ($\bowtie$)	$\Omega_1 \bowtie \emptyset = \emptyset \bowtie \Omega_1 = \emptyset$	$\Omega_1, \Omega_2, \Omega_3$
5	Identity ($\cup$)	$\Omega_1 \cup \emptyset = \emptyset \cup \Omega_1 = \Omega_1$	are sets of
6	Associativity ($\bowtie$)	$\Omega_1 \bowtie (\Omega_2 \bowtie \Omega_3)$ $= (\Omega_1 \bowtie \Omega_2) \bowtie \Omega_3$	solution mappings
7	Associativity ($\cup$)	$\Omega_1 \cup (\Omega_2 \cup \Omega_3)$ $= (\Omega_1 \cup \Omega_2) \cup \Omega_3$	
8	Distributivity ($\bowtie$ over $\cup$)	$\Omega_1 \bowtie (\Omega_2 \cup \Omega_3)$ $= (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \bowtie \Omega_3)$	
9	Reflexivity	$\Omega_1 \bowtie \Omega_2 = \Omega_2 \bowtie \Omega_1$	

TABLE II: Equivalence rules that are applicable to the evaluation of SE expressions ($\mathbb{P}_1, \ldots, \mathbb{P}_m$ represent sets of RDF graphs).

$i \in \{1, \ldots, m\}$). Table II lists the equivalence rules. Rules 1–3 are specific to the clustered-match operation, whereas rules 4–9 are derived from SPARQL algebra [35]. Rules that are marked with an asterisk (*) are conditional. Observe that the expansion rule relies on a condition that is independent of the way the graph is clustered. In other words, for any clustering $\mathbb{P}$ of an RDF graph, one may generate some $\mathbb{P}_1, \ldots, \mathbb{P}_m$, such that the condition is satisfied. On the contrary, we shall see that the conditions in join elimination and join reduction are directly related to the way the graph is clustered; therefore, they need to be checked every time a query is evaluated. The following two theorems formalize these conditions and show their correctness.

Theorem 2: Given a clustering $\mathbb{P}$ of an RDF graph G and two BGPs Q_A and Q_B with $V(Q_A) \cap V(Q_B) \neq \emptyset$, let $I = \bigcup_{(P_i,P_j)\in\mathbb{P}_1\times\mathbb{P}_2} V(P_i) \cap V(P_j)$, where $\mathbb{P}_1, \mathbb{P}_2 \subseteq \mathbb{P}$. Then, $[\![Q_A\lfloor\mathbb{P}_1\rfloor \bowtie Q_B\lfloor\mathbb{P}_2\rfloor]\!]_G = \emptyset$ if, for each vertex $v \in I$, there exists a vertex $\hat{v} \in V(Q_A) \cap V(Q_B)$ and an edge $\hat{e} \in inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$ such that $\hat{e}$ is not compatible (cf., Def. 3) with any edge in $\bigcup_{P\in\mathbb{P}_1\cup\mathbb{P}_2} inc(P, v)$, where $inc(G, v)$ denotes the set of edges that are incident on a vertex v.

Definition 12: Given two BGPs Q_A and Q_B, we define the *concatenation* of Q_A and Q_B, denoted by $Q_A \oplus Q_B$, as a BGP $Q = (\hat{V}, \hat{E})$ such that (i) $\hat{V} = V(Q_A) \cup V(Q_B)$ and (ii) $\hat{E} = E(Q_A) \cup E(Q_B)$.

Theorem 3: Given a clustering $\mathbb{P}$ of an RDF graph G and two BGPs Q_A and Q_B with $V(Q_A) \cap V(Q_B) \neq \emptyset$, $[\![Q_A\lfloor\mathbb{P}\rfloor \bowtie Q_B\lfloor\mathbb{P}\rfloor]\!]_G = [\![(Q_A \oplus Q_B)\lfloor\mathbb{P}\rfloor]\!]_G$ if, for each vertex v such that $\left|cont(\mathbb{P}, v)\right| > 1$ (where $cont(\mathbb{P}, v)$ denotes the subset of clusters in $\mathbb{P}$ that contain v), either

(i) there exists a vertex $\hat{v} \in V(Q_A) \cap V(Q_B)$ and an edge $\hat{e} \in inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$ such that $\hat{e}$ is not compatible with any edge from $\bigcup_{P\in\mathbb{P}} inc(P, v)$, or

(ii) there exists a single cluster $\bar{P} \in \mathbb{P}$ such that for every edge $e \in \bigcup_{P\in\mathbb{P}} inc(P, v)$ and for every vertex $\hat{v} \in V(Q_A) \cap V(Q_B)$, if e is compatible with an edge from $inc(Q_A, \hat{v}) \cup inc(Q_B, \hat{v})$, then $cont(\mathbb{P}, e) = \{\bar{P}\}$.

Let Q be $Q_A \oplus Q_B$ (i.e., the concatenation of two BGPs). Theorem 3 formalizes that if Q does not have any matching subgraph that spans multiple clusters, then Q can be evaluated by (i) computing the matching subgraphs of Q within each cluster in isolation, and (ii) taking the union of the results from step (i), thereby omitting the join. Thus, if partial matches of Q_A and Q_B do not join across clusters, query evaluation can be simplified. Condition (i) guarantees that clusters in consideration do not share common vertices (or if there are such vertices then they are not related to the evaluation of Q); and condition (ii) says that if there is such a vertex, the edges incident on that vertex do not match the query edges. Under these circumstances, Q cannot have any matching subgraph that spans multiple clusters and the join can be eliminated.

$inc(\mathbb{P}, v_1)$	$inc(\mathbb{P}, v_2)$	$inc(\mathbb{P}, v_3)$	in	$inc(\mathbb{P}, v_4)$
$v_1 \xrightarrow{A} v_2$	$v_2 \xleftarrow{A} v_1$	$v_3 \xrightarrow{A} v_9$	P_1	$v_4 \xrightarrow{B} v_{11}$
$v_1 \xrightarrow{A} v_6$	$v_2 \xrightarrow{B} v_3$	$v_3 \xleftarrow{B} v_2$	P_2	$v_4 \xleftarrow{C} v_3$
$v_1 \xrightarrow{B} v_5$	$v_2 \xrightarrow{B} v_8$	$v_3 \xrightarrow{C} v_4$	P_2	$v_4 \xrightarrow{C}$ "10"
	$v_2 \xrightarrow{C} v_7$	$v_3 \xrightarrow{C} v_{10}$	P_2	

TABLE III: Incident edges on v_1–v_4 in Clustering A (Fig. 3c)

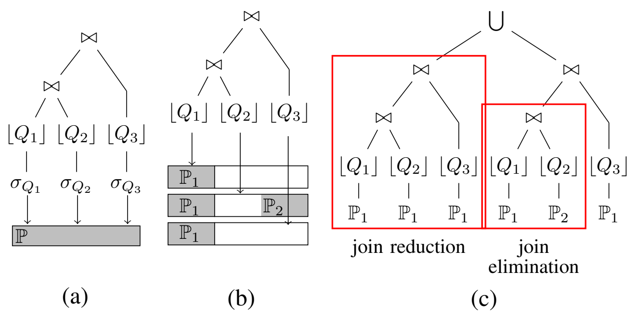


Fig. 5: Illustration of query rewriting and optimization.

Let us revisit the earlier query evaluation example and demonstrate how join reduction can be applied to the schemaless-evaluation of query $?w \xrightarrow{A} ?x \xrightarrow{B} ?y \xrightarrow{C} ?z$ over the clustering in Fig. 3c. The baseline SE expression for this BGP is $Q_1\lfloor\mathbb{P}\rfloor \bowtie Q_2\lfloor\mathbb{P}\rfloor \bowtie Q_3\lfloor\mathbb{P}\rfloor$, where $?w \xrightarrow{A} ?x$, $?x \xrightarrow{B} ?y$, $?y \xrightarrow{C} ?z$ are the three BGPs Q_1, Q_2 and Q_3 in the trivial decomposition of the query, and $\mathbb{P}$ consists of P_1 and P_2 in Fig. 3c. For simplicity, we ignore prune operations for now. If $[\![Q_2\lfloor\mathbb{P}\rfloor \bowtie Q_3\lfloor\mathbb{P}\rfloor]\!]_G = [\![(Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor]\!]_G$, this baseline SE expression can be rewritten as $Q_1\lfloor\mathbb{P}\rfloor \bowtie (Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor$. For the given clustering, the four vertices v_1, v_2, v_3 and v_4 exist in multiple clusters. Therefore, we need to check the conditions in Theorem 3. Note that $inc(Q_2, ?y) \cup inc(Q_3, ?y)$ consists of two edges, namely, $(?x, B, ?y)$ and $(?y, C, ?z)$. Condition (i) holds for v_1 because $(?y, C, ?z)$ is not compatible with any of the edges in $inc(\mathbb{P}, v_1)$, which is illustrated in Table III. For v_2, $(?y, B, ?z)$ is not compatible with any of the edges in $inc(\mathbb{P}, v_2)$ due to the direction of edges. The same argument applies to v_4. Regarding v_3, both $(?x, B, ?y)$ and $(?y, C, ?z)$ have at least one compatible edge; therefore, we also need to check condition (ii) for v_3. Since all compatible edges are from the same cluster, namely, P_2, the baseline SE expression can be simplified to $Q_1\lfloor\mathbb{P}\rfloor \bowtie (Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor$. Continuing with the process, the expression can be further simplified to $(Q_1 \oplus Q_2 \oplus Q_3)\lfloor\mathbb{P}\rfloor$.

*3) **Query Rewriting Algorithm**:* The algorithm for rewriting a baseline SE expression proceeds in three phases.

We describe this algorithm using the example illustrated in Fig. 5. Consider a baseline SE expression: $Q_1\lfloor\sigma_{Q_1}(\mathbb{P})\rfloor \bowtie Q_2\lfloor\sigma_{Q_2}(\mathbb{P})\rfloor \bowtie Q_3\lfloor\sigma_{Q_3}(\mathbb{P})\rfloor$ (Fig. 5a). First, the joins in the baseline expression are reordered according to their estimated selectivities [36]. Second, by using generic equivalence rules, the expression is transformed into a *canonical form*. An SE expression is in canonical form if it consists of the union of a set of sub-expressions $T_1 \cup \cdots \cup T_m$, where each sub-expression is made up of the exact same set of clustered-match operations, which differ only in the clusters they operate on.

To compute the canonical SE expression, first, each prune operation is evaluated, producing multiple sets of clusters with one set for each prune operation (Fig. 5b). As illustrated in Fig. 5b, these sets of clusters are factorized into maximal common subsets such that factorization produces as few segments as possible. For example, let $\mathbb{P} = \{P_a, P_b, P_c, P_d\}$ denote the set of clusters in the example; and assume that $\{P_a, P_b\}$ is a prune of $\mathbb{P}$ with respect to Q_1 and $\{P_a, P_b, P_d\}$ and $\{P_a, P_b\}$ are prunes with respect to Q_2 and Q_3, respectively. Then, $\mathbb{P}_1 = \{P_a, P_b\}$ and $\mathbb{P}_2 = \{P_d\}$ are the maximal common subsets. In the next step, each clustered-match operation is expanded across the corresponding subsets of clusters using Rule 1. Rules 4–9 are applied to the nodes of the expression-tree in a bottom-up fashion, which is repeated until no further rewriting is possible. At this stage, the canonical expression is produced (Fig. 5c).

In the third phase, each sub-expression in the canonical form is optimized independently using conditional rules (i.e., Rules 2–3) as well as Rules 4–9. In this regard, join reduction and join elimination are applied recursively to the nodes of each sub-expression until the original query is decomposed into as few segments as possible. The right-hand side of union is eliminated using join elimination (Fig. 5c) and the remaining expression is simplified to $(Q_1 \oplus Q_2 \oplus Q_3)\lfloor S_1\rfloor$ using join reduction.

For a given BGP, an SE expression is generated (which of the equivalent expressions the system chooses is the topic of Section V-2). Fig. 5a illustrates a tree representation of such an SE expression. Then, each sub-expression of the form $\sigma_{Q_i}(\mathbb{P})$ is evaluated by pruning out the irrelevant clusters using the cluster index (Fig. 2b). Consequently, each sub-expression of the form $Q_i\lfloor\sigma_{Q_i}(\mathbb{P})\rfloor$ is simplified to $Q_i\lfloor\mathbb{P}_i\rfloor$, where $\mathbb{P}_i \subseteq \mathbb{P}$. Then, for each resulting sub-expression $Q_i\lfloor\mathbb{P}_i\rfloor$, (i) the sub-expression is evaluated in isolation on each cluster using a standard subgraph matching technique [29], and (ii) the results from each evaluation are unioned. In the subsequent steps, intermediate tuples from the evaluation of each sub-expression are joined or unioned according to the standard definitions in SPARQL algebra [30].

*4) **Spill Index**:* We maintain information that is relevant to the computation of conditional equivalence rules in the spill index (cf., Fig. 2b). The spill index is constructed in a lazy fashion. As the baseline expressions are rewritten, more information is uncovered about the clusters in the underlying representation such as the vertices the clusters have in common, the labels of the edges that are incident on these vertices, etc (cf., Table III). Consequently, this information is utilized to more efficiently rewrite the subsequent baseline expressions.

*5) **Cluster Index**:* To prune the clusters irrelevant to a query, we employ another index, called the cluster index (cf., Fig. 2b). The cluster index is also constructed in a lazy fashion, which is similar to *database cracking* [37].

Before any query is evaluated, the cluster index consists of a doubly-linked list of pointers to all of the clusters. Initially, the index does not assume anything about the contents within each cluster. However, as queries are evaluated, it uncovers more information about the clusters and indexes them as follows: When the first BGP, say Q^1, is evaluated, the list is divided into two segments, namely, $\mathbb{P}_l$ and $\mathbb{P}_r$ such that every cluster that lies to the left of a pivot has matching subgraphs for the BGP, whereas those to the right do not. The overhead of restructuring the list is small. Note that the list has to be traversed anyway to compute the matching subgraphs; while doing so, clusters can be reordered in-place and in one pass over the list. For the next BGP, Q^2, there are three possible scenarios: (i) If Q^2 is the same as Q^1, query results can be computed directly from the clusters in $\mathbb{P}_l$, which does not need to be divided any further. (ii) If Q^2 is a strict supergraph of Q^1, a subset of clusters in $\mathbb{P}_l$ are relevant. Therefore, $\mathbb{P}_l$ needs to be traversed and divided into two segments: $\mathbb{P}_{l.l}$ and $\mathbb{P}_{l.r}$ like previously. (iii) For all other cases, potentially some clusters in both $\mathbb{P}_l$ and $\mathbb{P}_r$ have matching subgraphs of Q^2. Therefore, both lists need to be traversed and divided further.

As the list of cluster pointers gets divided into multiple segments, we keep track of the segments that are relevant to the particular queries using a decision tree. For every segment of clusters that contain at least one matching subgraph for any BGP in the decision tree, we also maintain two second-tier indexes. The *vertex-index* is a hash table that maps URIs to the subset of clusters that contain vertices of that URI. The *range-index* keeps track of the minimum and maximum literal values within each cluster for each distinct predicate, and it works as a filter when clusters are traversed.

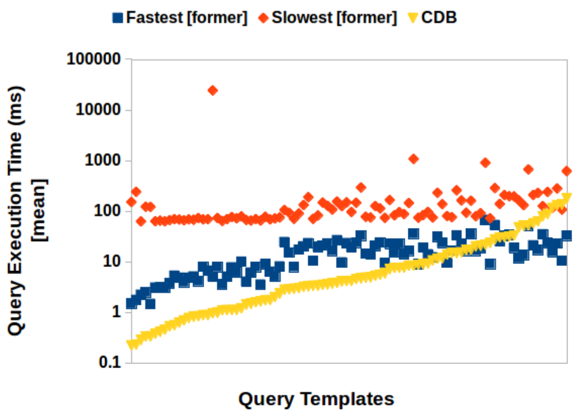
*6) **Benefits**:* In Section IV, we introduced the group-by-query representation and showed its advantages using two contrasting examples. We also described a practical clustering algorithm to compute a good group-by-query representation. In Section V, we demonstrated that a poorly designed query evaluation algorithm could easily diminish the benefits of clustering, even if the clustering were to be perfect. Therefore, we introduced schemaless-evaluation, which is specifically optimized for the group-by-query representation.

Schemaless-evaluation offers important advantages. First of all, it is possible to compute a clustering such that most of the queries in a workload do not require join operations across clusters. Even in the worst case when a query cannot be rewritten as any other expression, the baseline SE expression still guarantees correct results, thereby providing flexibility to compute a clustering that favors the most frequent queries in a workload and allowing the clustering to be updated as these frequencies change.

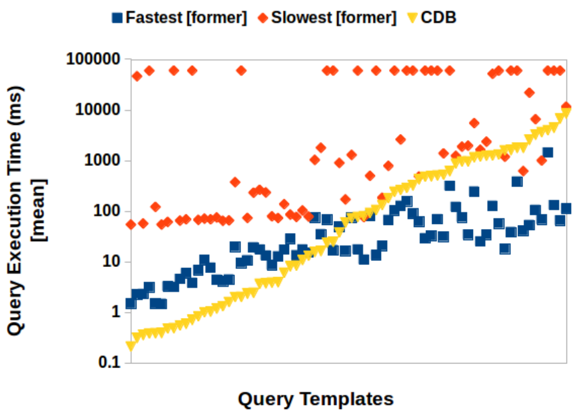
Second, we limit the scope of subgraph matching to contents within each cluster, thereby providing isolation with significant benefits. Since clusters are now truly isolated from each other, new clusters can be added and existing clusters can be split or merged without affecting the integrity of query evaluation on other parts of the graph. Consequently, query

TABLE IV: Overview of our experimental evaluation using WatDiv stress testing tool

	WatDiv 10M triples						**WatDiv 100M triples**					
	RDF-3x	***VOS [6.1]***	***VOS [7.1]***	***MonetDB***	***4Store***	***CDB***	***RDF-3x***	***VOS [6.1]***	***VOS [7.1]***	***MonetDB***	***4Store***	***CDB***
Query execution time (ms) (geometric mean)	18.8	44.0	24.5	17.0	93.0	**4.7**	71.4	210.3	96.4	62.7	767.2	**40.4**
% of query templates where system is fastest	1.1%	0.0%	2.2%	16.3%	0.0%	**80.4%**	9.7%	0.0%	0.0%	40.3%	1.4%	**48.6%**



(a) For each query template, comparison of our prototype (CDB) against fastest and slowest systems at WatDiv 10M RDF triples



(b) For each query template, comparison of CDB against fastest and slowest systems at WatDiv 100M RDF triples

Fig. 6: Detailed results

evaluation can be more easily interleaved with re-clustering of the graph, which is one of the key objectives of our work. There is also an opportunity for parallelization—subgraph matching can be performed concurrently on multiple clusters.

Third, when determining whether or not a cluster contains a subgraph that matches a query, the index needs to consider only the subgraphs that reside within a single cluster.

VI. Experimental Evaluation

For experiments, we use a commodity machine with AMD Phenom II $\times 4$ 955 3.20 GHz processor, 16 GB of main memory and a Seagate $3.AA$ hard disk drive with 100 GB of free disk space. The operating system is Ubuntu 12.04 LTS.

For our evaluations, we primarily use the Waterloo SPARQL Diversity Test Suite (WatDiv) because it facilitates the generation of test cases that are far more diverse than any of the existing benchmarks [11]. In this regard, we use the WatDiv *data generator* to create two datasets: one with 10 million RDF triples and another with 100 million RDF triples (we observe that systems under test (SUT) load data into main memory on the smaller dataset whereas at 100M triples, SUTs perform disk I/O). Then, using the WatDiv *query template generator*, we create 125 query templates and instantiate each query template with 100 queries, thus, obtaining 12500 queries (http://db.uwaterloo.ca/watdiv/stress-workloads.tar.gz).

The primary objective of our experimental evaluation is to quantify the benefits of the group-by-query clustering over fixed, workload-oblivious representations. Therefore, we compare chameleon-db (CDB) [31], which is our prototype implementation of the group-by-query clustering approach, with five popular systems, namely, RDF-3x [7], MonetDB [17], 4Store [18] and Virtuoso Open Source (VOS) versions 6.1 [19] and 7.1 [8]. RDF-3x follows the single-table approach and creates multiple indexes; MonetDB is a column-store, where RDF data are represented using vertical partitioning [5]; and the last three systems are industrial systems. Both 4Store and VOS group and index data primarily based on RDF predicates, but VOS 6.1 is a row-store whereas VOS 7.1 is a column-store. We configure these systems so that they make as much use of the available main memory as possible.

We evaluate each system independently on each query template. Specifically, for each query template, we first warm up the system by executing the workload for that query template once (i.e., 100 queries). Then, we execute the same workload five more times (i.e., 500 queries). We report average query execution time over the last five workloads.

Our prototype starts with a completely segmented clustering, where each cluster consists of a single triple. For reasons discussed throughout the paper, this clustering is bad for almost any type of workload: it potentially leads to defragmentation, poor data localization and generation of irrelevant intermediate result tuples. For each query template, we let chameleon-db execute the first 100 queries using this suboptimal clustering, but set a timeout threshold of 30 minutes. If the system manages to execute the first 100 queries within the timeout

TABLE V: Experimental evaluation over a crawl of the DBpedia dataset. For each query template, mean (geometric) query execution time is reported in milliseconds.

	$\mathbf{Q_1}$	$\mathbf{Q_2}$	$\mathbf{Q_3}$	$\mathbf{Q_4}$	$\mathbf{Q_5}$	$\mathbf{Q_6}$	$\mathbf{Q_7}$	$\mathbf{Q_8}$	$\mathbf{Q_9}$	$\mathbf{Q_{10}}$	$\mathbf{Q_{11}}$	$\mathbf{Q_{12}}$	$\mathbf{Q_{13}}$	$\mathbf{Q_{14}}$
RDF-3x	**3.5**	257.7	8.1	11.8	13.1	22.8	31.6	98.4	5761.9	81.4	**17.0**	21.8	21.4	38.0
VOS [7.1]	12.2	**19.8**	2.8	4.4	13.6	28.4	**24.2**	**27.8**	6359.6	**13.3**	19.3	29.9	27.8	**14.1**
MonetDB	41.2	29.9	27.9	22.8	54.0	57.3	30.6	60.3	**1700.0**	99.3	47.6	113.4	39.9	46.9
4Store	767.6	1176.7	502.9	519.7	550.8	739.8	1006.9	689.6	30953.5	557.6	537.4	495.5	489.5	512.6
CDB	4.0	35.1	**0.6**	**0.8**	**0.8**	**20.9**	N/A	392.3	N/A	N/A	N/A	**1.3**	**1.5**	N/A

threshold, then after the execution of the 100^{th} query, we allow the storage advisor to compute a better group-by-query clustering (on average, computation of the group-by-query clustering takes 317.6ms on the larger dataset). In that case, the last 500 queries are executed over the group-by-query clustering presented in Section IV.[2] This way, with the given timeout threshold, we were able to collect results for a majority of 92 query templates over the smaller dataset and 76 query templates over the larger one. In the remainder of this section, we focus on only these query templates.

Note that the time to update the underlying physical representation is reflected in the execution times of the first few of the last 500 queries (cf., Section IV-3). Furthermore, for the first few queries, indexes are not yet fully constructed and the cache can be cold. In particular, we observe that query execution times can improve by an order of magnitude once the indexes are fully constructed and the cache is warm.

Table IV demonstrates that with the group-by-query clustering, it is possible to achieve significantly better, consistent performance across a diverse selection of queries than any of the workload-oblivious approaches that we have compared with. That is, for both datasets, our prototype performs better with the lowest mean query execution time. Furthermore, for the 10M dataset, chameleon-db is the fastest system for over 80% of the considered 92 query templates.

Fig. 6a and 6b depict the absolute mean query execution times for each query template. To avoid cluttering the charts, for each query template, (in addition to chameleon-db) we display data points for only the fastest and the slowest systems among RDF-3x, MonetDB, 4Store, VOS [6.1] and VOS [7.1] for that particular query template, where the fastest system for one query template is not necessarily the same as that for another query template (cf., Fig. 1 and Table IV). It is important to note that for query templates in which chameleon-db is not the fastest system (i.e., the right hand sides of Fig. 6a and 6b), it is still orders of magnitude faster than the slowest system. On the other hand, going from the smaller dataset to the larger, we observe a decrease in the percentage of queries for which chameleon-db is fastest. Next, we investigate this issue to prioritize potential future work.

First, we quantify how much the group-by-query clustering improves performance and try to see if there are any anomalies specific to the 100M triples dataset. We observe that with group-by-query clustering, there is significant reduction in mean query execution time, and the scale of this reduction is consistent in both datasets. Specifically, for 10M triples, query execution becomes faster by a factor of 5.0 (from a geometric mean of 23.5ms to 4.7ms), and for 100M triples, it becomes faster by a factor of 4.8 (from a geometric mean of 195.0ms to 40.4ms). This rules out any major anomalies.

Second, we divide the queries into two groups: (i) those for which our prototype was the fastest, and (ii) all the remaining ones. We analyze the query logs, where we keep track of which query plan the system is using (i.e., OptimalEvaluation vs. SchemalessEvaluation) for a particular query, as well as the mean group-by-query cluster size (i.e., in terms of the number of triples) at the time that query is being evaluated. We make two important observations (for the 100M triples dataset): (i) in the second group, the mean cluster sizes are about an order of magnitude larger than those in the first one; and (ii) for the first group, 82.9% of the queries have been evaluated with OptimalEvaluation, whereas only 29.7% of queries have been evaluated with OptimalEvaluation in the second group. Upon further manual inspection, we also note that in some problematic cases, it would have been possible to choose OptimalEvaluation if we had stronger but potentially more compute-intensive conditional equivalence rules (cf., Section V-2). Consequently, as future work, better data structures can be developed to improve performance of subgraph matching within each cluster, especially when the clusters are large (cf., Q4 in Section V). Moreover, query rewriting rules can be extended.

Third, cache misses can lead to disproportionately large increases in query evaluation time; therefore, it is important that the system takes full advantage of the group-by-query clustering. One problem that we have omitted in this paper is the serialization of clusters across the storage system. That is, the algorithm in Section IV deals with the problem of placing triples in clusters, but it does not consider the latter problem.[3] Consequently, techniques can be developed as part of future work to decide how clusters should be serialized.

We repeat our evaluations also on a crawl of the DBpedia dataset[4] [38]. We focus on BGPs in this paper, therefore, we utilize the query logs provided by the benchmark to extract 14 BGP templates[5]. Note that most queries in the DBpedia query logs are repetitive and a large portion of the queries consist of only a single or few triple patterns. To avoid bias, for queries having 3–10 triple patterns, we pick the most frequent templates and then instantiate each query template with 25 queries, making sure that there is at least one instantiation (per template) that returns a non-empty result.

Our evaluations on DBpedia are consistent with those on WatDiv: (i) for workloads in which chameleon-db completes

[2]Improving system performance for the completely segmented clustering or choosing a different initial clustering is beyond the scope of this paper.

[3]We serialize clusters in the order they are updated, which can be random.
[4]http://benchmark.dbpedia.org/benchmark_50.nt.bz2
[5]https://cs.uwaterloo.ca/~galuc/files/dbpedia-test-queries.tar.gz

its warm-up phase without timing out, group-by-query clustering can achieve consistently good performance (cf., Table V), and (ii) chameleon-db times out on some workloads before re-partitioning kicks in due to similar reasons discussed for the WatDiv experiments. One difference is that we observe less variation among systems' query execution times across the workloads, which is consistent with our observation that DBpedia queries are not as diverse as WatDiv queries [11].

VII. Conclusions

In this paper, we develop a schemaless group-by-query representation for RDF data. The main advantage of this representation is that it is workload-driven, allowing the adjustment of physical data structures and indexes in the database according to the queries that are being executed in the database system. We also propose query optimization techniques that work effectively with our schemaless group-by-query representation that do not sacrifice correctness while supporting workload-awareness. Through implementation of our techniques in our system prototype chameleon-db and experimental evaluation against five workload-oblivious systems, we show that chameleon-db generally outperforms these other systems across a diverse selection of queries by a significant margin. Our future work consists of three milestones (some of which are discussed in more detail in [16]): (i) interleaving clustering with query evaluation so that re-clustering can take place after the execution of each query; (ii) improving the cluster index; and (iii) adaptively co-clustering group-by-query clusters on the storage system.

VIII. Acknowledgments

We thank Dr. Weddell and A. Kane for fruitful discussions.

References

[1] C. Bizer, "Web of linked data - a global public data space on the Web," in *Proc. 13th Int. Workshop on the World Wide Web and Databases*, 2010.

[2] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: implementing the semantic web recommendations," in *Proc. 13th Int. World Wide Web Conf. - Alternate Track Papers & Posters*, 2004, pp. 74–83.

[3] K. Wilkinson, "Jena property table implementation," HP-Labs, Tech. Rep. HPL-2006-140, 2006.

[4] C. Weiss, P. Karras, and A. Bernstein, "Hexastore: sextuple indexing for semantic web data management," *Proc. VLDB*, vol. 1, no. 1, pp. 1008–1019, 2008.

[5] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach, "SW-Store: a vertically partitioned DBMS for semantic web data management," *VLDB J.*, vol. 18, pp. 385–406, 2009.

[6] M. Bröcheler, A. Pugliese, and V. S. Subrahmanian, "DOGMA: A disk-oriented graph matching algorithm for RDF databases." in *Proc. 8th Int. Semantic Web Conference*, 2009, pp. 97–113.

[7] T. Neumann and G. Weikum, "The RDF-3X engine for scalable management of RDF data," *VLDB J.*, vol. 19, no. 1, pp. 91–113, 2010.

[8] O. Erling, "Virtuoso, a hybrid RDBMS/graph column store," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 3–8, 2012.

[9] M. A. Bornea, J. Dolby, A. Kementsietsidis, K. Srinivas, P. Dantressangle, O. Udrea, and B. Bhattacharjee, "Building an efficient RDF store over a relational database," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2013, pp. 121–132.

[10] L. Zou, M. T. Özsu, L. Chen, X. Shen, R. Huang, and D. Zhao, "gStore: a graph-based SPARQL query engine," *VLDB J.*, vol. 23, no. 4, pp. 565–590, 2014.

[11] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee, "Diversified stress testing of rdf data management systems," in *Proc. 13th Int. Semantic Web Conference, Part I*, 2014, pp. 197–212.

[12] M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente, "An empirical study of real-world SPARQL queries," *CoRR*, vol. abs/1103.5043, 2011.

[13] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data - the story so far." *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009.

[14] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea, "Apples and oranges: a comparison of RDF benchmarks and real RDF datasets," in *SIGMOD Conference*, 2011, pp. 145–156.

[15] M. Kirchberg, R. K. L. Ko, and B.-S. Lee, "From linked data to relevant data – time is the essence," *CoRR*, vol. abs/1103.5046, 2011.

[16] G. Aluç, M. T. Özsu, and K. Daudjee, "Workload matters: Why RDF databases need a new design," *Proc. VLDB*, vol. 7, no. 10, pp. 837–840, 2014.

[17] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten, "MonetDB: Two decades of research in column-oriented database architectures," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 40–45, 2012.

[18] S. Harris, N. Lamb, and N. Shadbolt, "4store: The design and implementation of a clustered RDF store," in *Proc. 5th Int. Workshop on Scalable Semantic Web Knowledge Base Systems*, 2009, pp. 81–96.

[19] O. Erling and I. Mikhailov, "RDF support in the Virtuoso DBMS," *Networked Knowledge-Networked Media*, pp. 7–24, 2009.

[20] F. F.-H. Nah, "A study on tolerable waiting time: how long are Web users willing to wait?" *Behaviour & IT*, vol. 23, no. 3, pp. 153–163, 2004.

[21] A. Y. Halevy, "Answering queries using views: A survey," *VLDB J.*, vol. 10, no. 4, pp. 270–294, 2001.

[22] G. Klyne and J. J. Carroll, "Resource Description Framework (RDF): Concepts and abstract syntax," http://www.w3.org/TR/rdf-concepts/, 2004.

[23] A. Harth and S. Decker, "Optimized index structures for querying RDF from the Web," in *Proceedings of the 3rd Latin American Web Congress (LA-Web)*, 2005, pp. 71–80.

[24] L. Sidirourgos, R. Goncalves, M. Kersten, N. Nes, and S. Manegold, "Column-store support for RDF data management: not all swans are white," *Proc. VLDB*, vol. 1, no. 2, pp. 1553–1563, 2008.

[25] J. Huang, D. J. Abadi, and K. Ren, "Scalable SPARQL querying of large RDF graphs," *Proc. VLDB*, vol. 4, no. 11, pp. 1123–1134, 2011.

[26] K. Hose and R. Schenkel, "WARP: Workload-aware replication and partitioning for rdf," in *Proc. 4th Int. Workshop on Data Engineering Meets the Semantic Web*, 2013, pp. 1–6.

[27] Z. Kaoudi and I. Manolescu, "RDF in the clouds: a survey," *VLDB J.*, 2014, forthcoming.

[28] E. Prudhommeaux and A. Seaborne, "SPARQL Query Language for RDF," http://www.w3.org/TR/rdf-sparql-query/, 2008.

[29] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, 1976.

[30] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of SPARQL," *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 1–45, 2009.

[31] G. Aluç, M. T. Özsu, K. Daudjee, and O. Hartig, "chameleon-db: a workload-aware robust RDF data management system," University of Waterloo, Tech. Rep. CS-2013-10, 2013.

[32] M. Schmidt, M. Meier, and G. Lausen, "Foundations of SPARQL query optimization," in *Proc. 13th Int. Conf. on Database Theory*, 2010, pp. 4–33.

[33] T. Neumann and G. Weikum, "Scalable join processing on very large RDF graphs," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2009, pp. 627–640.

[34] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," vol. 31, pp. 264–323, 1999.

[35] M. Arenas and J. Pérez, "Querying semantic web data with SPARQL," in *Proc. 30th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, 2011, pp. 305–316.

[36] M. Stocker, A. Seaborne, A. Bernstein, C. Kiefer, and D. Reynolds, "Sparql basic graph pattern optimization using selectivity estimation," in *Proc. 17th Int. World Wide Web Conf.*, 2008, pp. 595–604.

[37] S. Idreos, M. L. Kersten, and S. Manegold, "Self-organizing tuple reconstruction in column-stores," in *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2009, pp. 297–308.

[38] M. Morsey, J. Lehmann, S. Auer, and A.-C. N. Ngomo, "DBpedia SPARQL benchmark - performance assessment with real queries on real data," in *Proc. 10th Int. Semantic Web Conference*, 2011, pp. 454–469.