GUN: An Efficient Execution Strategy for Querying the Web of Data

Gabriela Montoya, Luis-Daniel Ibanez, Hala Skaf-Molli, Pascal Molli, Maria-Esther Vidal

GUN: An Efficient Execution Strategy for Querying the Web of Data

Gabriela Montoya[1], Luis-Daniel Ibáñez[1], Hala Skaf-Molli[1], Pascal Molli[1], and Maria-Esther Vidal[2]

[1] LINA– Nantes University, France
{gabriela.montoya,luis.ibanez,hala.skaf,pascal.molli}@univ-nantes.fr
[2] Universidad Simón Bolívar, Venezuela
{mvidal}@ldc.usb.ve

Abstract. Local-As-View (LAV) mediators provide a uniform interface to a federation of heterogeneous data sources, attempting to execute queries against the federation. LAV mediators rely on query rewriters to translate mediator queries into equivalent queries on the federated data sources. The query rewriting problem in LAV mediators has shown to be NP-complete, and there may be an exponential number of rewritings, making unfeasible the execution or even generation of all the rewritings for some queries. The complexity of this problem can be particularly impacted when queries and data sources are described using SPARQL conjunctive queries, for which millions of rewritings could be generated. We aim at providing an efficient solution to the problem of executing LAV SPARQL query rewritings while the gathered answer is as complete as possible. We formulate the Result-Maximal k-Execution problem (ReMakE) as the problem of maximizing the query results obtained from the execution of only k rewritings. Additionally, a novel query execution strategy called GUN is proposed to solve the ReMakE problem. Our experimental evaluation demonstrates that GUN outperforms traditional techniques in terms of answer completeness and execution time.

1 Introduction

Querying the Web of Data raises the issue of semantic heterogeneity between a large number of data sources. Local-as-view (LAV) mediation [1] is a well-known and flexible approach to perform data integration over heterogeneous data sources. A LAV mediator relies on views to define semantic mappings between a uniform interface defined at the mediator level, and local schemas or views that describe the integrated data sources. A LAV mediator relies on a query rewriter to translate a mediator query into the union of queries against the local views. Additionally, new data sources can be included into LAV mediators without affecting the definition of the existing ones; thus, LAV mediators are well suitable to integrate sources from the Web of Data [2]. Nevertheless, the query rewriting problem has shown to be NP-complete, and the number of rewritings can be exponential even if mediated queries and local views are conjunctive queries [3, 4]. For example, a LAV mediator with 140 conjunctive views

can generate 10,000 rewritings for a conjunctive query with 8 goals [5]. This query rewriting problem complexity can be exacerbated by the usage of mediator queries and local views defined as SPARQL conjunctive queries. SPARQL queries are commonly comprised of a large number of triple patterns and many of them are defined on general predicates that can be answered by the majority of the data sources, i.e., `rdf:type` or `rdfs:seeAlso`. Additionally, these triple patterns can be grouped into chained connected star-shaped sub-queries [6]. Finally, a large number of variables can be projected out. Thus, the conjunction of all these properties impacts on the complexity of the query rewriting problem and conduces to the explosion of the number of query rewritings. For example, a query with 12 triple patterns that comprised three chained star-shaped sub-queries can be rewritten using 300 views in billions of rewritings, if general predicates are used in the triple patterns. This problem is even more challenging considering that statistics cannot be always collected from the sources, and there are not clear criteria to rank or prune the generated rewritings [7].

Therefore, it is not realistic to generate or execute such a huge number of rewritings, and we aim at providing an efficient solution to the problem by just considering only k LAV SPARQL query rewritings, where k corresponds to the first k rewritings produced by a LAV rewriter. Thus, we devised the Result-Maximal k-Execution Problem (ReMakE) as an extension of the Query-Rewriting-Problem (QRP) as follows: given a subset R_k of size k of a solution R of a QRP for a query Q, the ReMakE problem is to evaluate a set of rewritings R' containing R_k and contained in Q such that R' is result-maximal. Furthermore, we propose the Graph-Union execution strategy (GUN) as a solution to the ReMakE problem. Unlike traditional techniques, GUN relies on wrappers that populate an RDF graph that is locally managed by the execution engine. This approach takes advantage of the relatively low cost of the RDF-Graph union operation to construct an aggregation of the data retrieved from the views. This approach attempts at executing the original mediator query directly on the graph union and consequently, it may find results hidden to the k first rewritings. For a given set of rewritings, GUN always gathers at least all the answers collected by a traditional engine by executing the rewritings independently. If all relevant views identified by the rewriter are in R_k, GUN guarantees to return the complete answer without further processing of rewritings. Thus, the execution time of GUN depends on the number of the relevant views that comprise the rewritings in R_k, which is usually considerably lower than the total number of rewritings.

We compare GUN against traditional strategies in an experimental setup using synthetic data generated using the Berlin SPARQL benchmark [8] and views proposed by Castillo et al. [9]. We measure execution time and answer completeness for a benchmark of queries. GUN retrieves much more results in less time than existing engines. The amount of main memory required to maintain a GUN graph is in general higher than the one required to execute traditional approaches; however, improvements in execution time and results are substantial enough to consider it a good trade-off.

The paper is organized as follows: Section 2 states preliminaries, while Section 3 formalizes the ReMakE problem. Section 4 presents the GUN query execution strategy as a solution for the ReMakE problem. Section 5 reports our experimental study. Section 6 summarizes related work; and finally, conclusions and future work are outlined in Section 7.

2 Preliminaries

We assume a federation of data sources is integrated using the mediator-wrapper architecture proposed by Wiederhold [10]. Mediators provide a uniform interface to autonomous and heterogeneous data sources; mediators also implement the tasks of rewriting an input query into queries against the data sources, and merging data collected from the selected sources. Wrappers are software components that solve interoperability between sources and mediators by translating data collected from the sources into the schema and format understood by the mediators. Particularly, GUN-based mediators rely on wrappers able to solve resource identification and perform the corresponding RDF transformations to conform source data into the mediator RDF schema.

Formally, a conjunctive query Q over a database or mediator schema D has the form $Q(\bar{X})$:- $P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)$ where Q, P_1, ..., P_n are predicates name of some finite arity and $\bar{X}$, $\bar{X}_1$, ..., $\bar{X}_n$ are tuples of variables. These predicates constitute the global schema. We define the body of the query as $body(Q) = \{P_1(\bar{X}_1), \ldots, P_n(\bar{X}_n)\}$. Any non-empty subset of $body(Q)$ is called a subgoal of Q, singleton subgoals are called atomic subgoals. Predicates in the body stands for relations of D, while the head Q represents the answer relation of the query over D. We consider queries that are *safe*, i.e., $\bar{X} \subseteq \bigcup_{i=1}^{n} \bar{X}_i$, and call $Q(D)$ the result of executing Q over D.

In the spirit of [5], we define a view v as a safe query over D, we establish the difference between the *extension* of v, denoted $ext(v)$, and its evaluation over D, $v(D)$, and assume the relation $ext(v) \subseteq v(D)$ to state two important hypothesis: there may be data belonging to the database that is not available to the extensions, and the extensions never hold data that is not in the database.

A rewriting of a query Q over a database D with a set of views V is a conjunctive query $r(\bar{x})$:- $v_1(\bar{x_1}), \ldots, v_m(\bar{x_m})$ where, $v_i \in V$. A query rewriting is *contained* in Q, if for all database D and set of views V over D, the result of executing r in V is contained in the result of executing Q on D, i.e., $r(V) \subseteq Q(D)$.

Maximally Contained Query Rewriting Problem (QRP). Given a conjunctive query Q and a set of views $V = \{ v_1, \ldots, v_n \}$ over a database D, QRP is to find a set of rewritings R, called the solution of the QRP, such that:

- For all extensions of the views in the bodies of all rewritings in R, the union of the results of executing each query rewriting in the views V is contained in the result of executing Q in D, i.e., $\bigcup_{r \in R} r(ext(v_1), \ldots, ext(v_n)) \subseteq Q(D)$
- R is maximal, i.e., there is no other set R', such that:

$$\bigcup_{r \in R} r(ext(v_1), \ldots, ext(v_n)) \subset \bigcup_{r' \in R'} r'(ext(v_1), \ldots, ext(v_n)) \subseteq Q(D)$$

For a set R of rewritings, we define the set of relevant views $\Lambda(R) = \{v \mid v \in body(r) \wedge r \in R\}$ as the set of views in the rewritings in R, and its execution $R(D) = \bigcup_{r \in R} r(D)$. We also call $ext(\Lambda(R))$, the extension of the elements in $\Lambda(R)$.

The main drawback of existing query rewriting problem solutions for LAV [1, 5, 4, 11] is that the size of the set R can be exponential in the number of query subgoals [3, 11]. Considering that it is not realistic to execute or generate an enormous number of rewritings, this introduces the problem of collecting data considering only k rewritings while obtaining an answer as complete as possible.

3 Result-Maximal k-Execution Problem (ReMakE)

In this section, we formalize the problem of obtaining the maximal set of results from a given subset of the rewritings of a query over a set of views.

Result-Maximal k-Execution Problem (ReMakE). Given a subset R_k of size k of a solution R of a QRP of a query Q and a set of views V over a database D, ReMakE is to find a set of rewritings R' over the set of views in the bodies of the rewritings of R_k, such that:

$$\bigcup_{r_k \in R_k} r_k(ext(\Lambda(R_k))) \subseteq \bigcup_{r' \in R'} r'(ext(\Lambda(R_k))) \subseteq Q(D)$$

and that is result-maximal, i.e., that there is no another set R'' such that:

$$\bigcup_{r' \in R'} r'(ext(\Lambda(R_k))) \subset \bigcup_{r'' \in R''} r''(ext(\Lambda(R_k))) \subseteq Q(D)$$

We define this problem over the extensions of the views, as they are the real datasets where the query will be evaluated. It is important to note that the ReMakE problem only uses the query rewritings as an input, therefore, it is independent of the approach used to solve QRP. We also highlight that ReMakE is independent of the format of the data inside the extensions of the views.

To illustrate the problem, consider the generic set of rewritings in Figure 1, if we can only execute the first five rewritings, we may be missing a rewriting comprised of some combination of the views that we have already materialized.

4 GUN: a solution to the ReMakE problem

In this section, we explain how to solve the ReMakE problem by taking advantage of the relatively low cost of the RDF-Graph union. We use definitions of SPARQL semantics of [12]:

Definition 1. *The Sets I (IRI Identifiers), B (Blank Nodes), L (Literals) and $\mathcal{V}$ (Variables) are four infinite and pairwise disjoint sets. We also define $T = I \cup B \cup L$. An* RDF-Triple *is 3-tuple $(s, p, o) \in (I \cup B) \times I \times T$. An* RDF-Graph *is a set of RDF-Triples.*

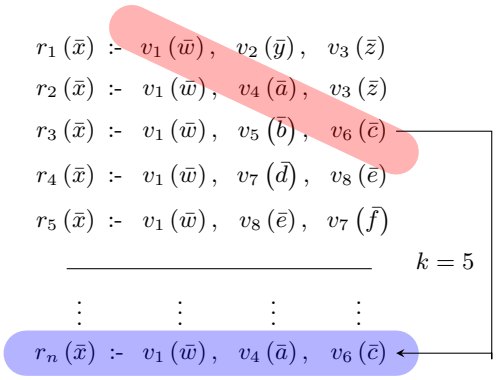

Fig. 1: Illustration of the Result-Maximal k-Execution problem. Some combinations of views materialized during the execution of a subset of rewritings are not considered, and they may be in the set of rewritings that are not taken in account.

Definition 2. *A* mapping μ *from* Υ *to* T *is a partial function* $\mu : \Upsilon \to T$. *The domain of* μ, $dom(\mu)$, *is the subset of* Υ *where* μ *is defined.*

Definition 3. *A* triple pattern *is a tuple* $t \in (I \cup \Upsilon \cup L) \times (I \cup \Upsilon) \times (I \cup \Upsilon \cup L)$. *A* Basic Graph Pattern *is a finite set of triple patterns. Given a triple pattern* t, $var(t)$ *is the set of variables occurring in* t, *analogously, given a basic graph pattern* B, $var(B) = \cup_{t \in B} var(t)$. *Given two basic graph patterns* B_1 *and* B_2, *the expression* B_1 *AND* B_2 *is a graph pattern.*

Definition 4. *Given a triple pattern* t *and a mapping* μ *such that,* $var(t) \subseteq dom(\mu)$, $\mu(t)$ *is the triple obtained by replacing the variables in* t *according to* μ. *Given a basic graph pattern* B *and a mapping* μ *such that* $var(B) \subseteq dom(\mu)$, *then* $\mu(B) = \cup_{t \in B} \mu(t)$.

Definition 5. *Two mappings* μ_1, μ_2 *are* compatible *(we denote* $\mu_1 \upuparrows \mu_2$*) iff for all* $?X \in (dom(\mu_1) \cap dom(\mu_2))$, *then* $\mu_1(?X) = \mu_2(?X)$. *This is equivalent to say that* $\mu_1 \cup \mu_2$ *is also a mapping.*

Definition 6. *Let* Ω_1, Ω_2 *two sets of mappings. The join between* Ω_1 *and* Ω_2 *is defined as:* $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \,|\, \mu_1 \in \Omega_1 \wedge \mu_2 \in \Omega_2 \wedge \mu_1 \upuparrows \mu_2\}$

Definition 7. *Given an RDF-Graph* G, *the* evaluation *of a triple pattern* t *over* G *corresponds to:* $[[t]]_G = \{\mu \,|\, dom(\mu) = var(t) \wedge \mu(t) \in G\}$. *The* evaluation *of a basic graph pattern* B *over* G *is defined as:* $[[B]]_G = \bowtie_{t \in B} [[t]]_G$. *The* evaluation *of a Graph Pattern* B' *of the form* $(B_1$ *AND* $B_2)$ *over* G *is as follows:* $[[B']]_G = [[B_1]]_G \bowtie [[B_2]]_G$

We consider that our database is an RDF-Graph G. A conjunctive query over a general database is analogous to the following query over an RDF-Graph:

$$Q(x) = \text{SELECT } x \text{ WHERE } F(p_1(\bar{x}_1)) \text{ AND } \ldots \text{ AND } F(p_n(\bar{x}_n))$$

where F is a translation function from predicates to triple patterns as defined in [13] or a customized one. The definitions of variables, head and body are the same. As the definitions of views and rewritings are based on the definition of query, they remain equivalent, together with the definitions of QRP and ReMakE. We define the evaluation of a rewriting $[[r(x)]]_G$ as:

$$[[r(x)]]_G = [[v_1(\bar{x}_1), \ldots, v_m(\bar{x}_m)]]_G = ([[p_a(\bar{x}_a)]]_{ext(v_1)} \bowtie \cdots \bowtie [[p_z(\bar{x}_z)]]_{ext(v_1)}) \\ \bowtie \cdots \bowtie ([[p_\alpha(\bar{x}_\alpha)]]_{ext(v_m)} \bowtie \cdots \bowtie [[p_\beta(\bar{x}_\beta)]]_{ext(v_m)})$$

where $p_a \ldots p_z \in body(v_1)$ and $p_\alpha \ldots p_\beta \in body(v_m)$. Note that this definition captures the practical implementation of the execution, where we materialize each call to a view (or more precisely, to its extension) and then, perform the joins between the sub-results. Traditionally, plans like Left Linear, Right Linear or Bushy Trees [14] are used to evaluate the rewritings over the extension of the views present in each rewriting; but to solve the ReMakE problem, we should ensure that any relevant combinations of obtained views are not missed, even if these combinations are not part of the rewritings in R_k.

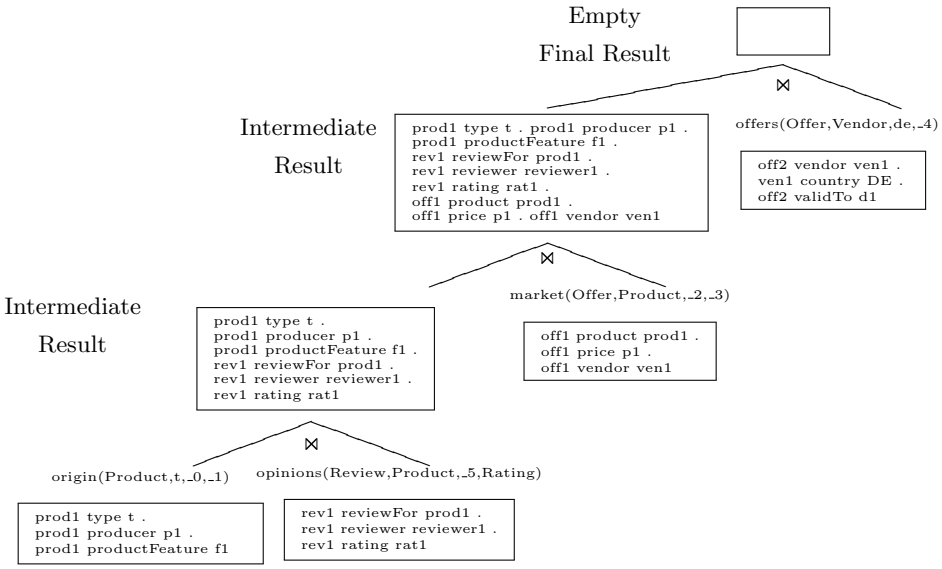

Fig. 2: Left Linear execution of the rewriting r of query Q. Views *origin*, *opinions*, *market* and *offers* are loaded, but it is not possible to produce any results since the join for *Offer* is empty. Prefixes are omitted to improve legibility.

Consider a query Q over a dataset generated with the Berlin Benchmark [8], which offers information about products, their offers and users' reviews. Q is defined as: "Products of type t that are sold by vendors from Germany, and their rating evaluation".

Q(Product, Vendor, Rating) :- type(Product, t), product(Offer, Product), vendor(Offer, Vendor), country(Vendor, de), reviewfor(Review, Product), rating1(Review, Rating). Considering the following four views: i) *origin*: "Products' type, producer, and features", ii) *market*: "Products' offers, price and vendors", iii) *offers*: "Offers' vendor, countries and validity", iv) *opinions*: "Products' reviews, ratings and reviewers". A possible rewriting of Q is:

r(Product,Vendor,Rating) :- origin(Product,t,_0,_1), opinions(Review,Product,_5,Rating), market(Offer,Product,_2,_3), offers(Offer,Vendor,de,_4).

Figure 2 shows the execution of the rewriting of the query Q following a left linear execution plan. In this execution, the RDF Graphs retrieved from the sources through the views are used to execute this rewriting. Notice that while executing the plan some intermediate results are produced, like those corresponding to the evaluation of the join between *origin* and *opinions*; however, these intermediate results are dismissed without being used to produce answers. A pertinent solution of the ReMakE problem must take advantage of these retrieved data. We now define our solution to the ReMakE problem as follows:

Graph Union (GUN). Given R_k a subset of a set of rewritings R of a query Q over a set of views V, apply Q to the union of the extensions of the views in the bodies of the elements of R_k:

$$GUN(R_k) = [[Q]]_{\bigcup ext(\Lambda(R_k))}$$

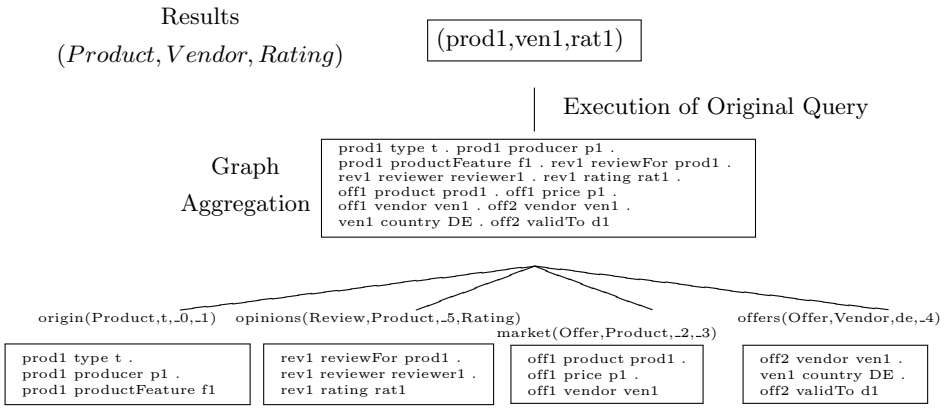

Fig. 3: GUN execution of the rewriting r of query Q. Results are produced, at the cost of building and querying over an aggregated graph. Prefixes are omitted to improve legibility.

Figure 3 shows the execution with GUN associated with the rewriting of query Q. For each view in the rewriting, we retrieve all its triples and put them into an aggregate RDF-Graph. As we are using the Local-As-View approach,

views are expressed in terms of the global schema, and we can run the original query and execute joins that are not considered in the rewritings, like the one between *market* and *offers* through *Vendor*. Therefore, GUN takes advantage of all retrieved data to produce an answer for the user. GUN is affordable in the context of the Semantic Web thanks to the simplicity of the RDF data model. Implementing the same idea in relational databases would require the creation of the universal relation, which may have a prohibitive cost.

Proposition 1. *Graph Union is a solution to the ReMakE problem, i.e.,*

$$\bigcup_{r \in R_k} [[r]]_{ext(\Lambda(R_k))} \subseteq [[Q]]_{\bigcup ext(\Lambda(R_k))} \tag{1}$$

$$[[Q]]_{\bigcup ext(\Lambda(R_k))} \subseteq [[Q]]_G \tag{2}$$

And it is result-maximal.

Proof. As by construction of the views and their extensions $\bigcup_{v \in V} [[v]]_G \subseteq G$, then $\bigcup ext(\Lambda(R_k)) \subseteq G$, making straightforward to see that (2) holds. For (1) note that the set $\Lambda(R_k)$ can be considered as a set of views over the graph $\bigcup \Lambda(R_k)$, then by the containment property, each member of R_k is contained in Q. As we are applying the original query Q, it is clear that there is no rewriting that can return more results than Q, meaning that GUN is maximal.

4.1 GUN's Properties

In this section, we state some properties of GUN. Given a query Q and a set of views V on a database D, let V_R the set of relevant views, R the set of rewritings of Q over V, and R_k a subset of R.

- *Answer Completeness:* If GUN is performed over R_k with $\Lambda(R_k) = \Lambda(R)$, then GUN will produce the complete query answer i.e., $\Lambda(R_k) = \Lambda(R) \Rightarrow GUN(R_k) = Q(D)$. By definition of QRP, only the relevant views contribute to the answer, therefore, if GUN's aggregation graph contains all the relevant views, then we can ensure that GUN will produce the complete answer.
- *Effectiveness:* We define the *Effectiveness* of GUN for a given R_k as:

$$GUNEffect(R_k) = \frac{|GUN(R_k)| - |\bigcup_{r_k \in R_k} r_k(ext(\Lambda(R_k)))|}{|Q(D)| - |\bigcup_{r_k \in R_k} r_k(ext(\Lambda(R_k)))|}$$

  intuitively, GUN has more effectiveness if it finds answers that are not found by the execution of $\bigcup_{r_k \in R_k} r_k(ext(\Lambda(R_k)))$. We say that GUN is *effective* for a given R_k if $GUNEffect(R_k) > 0$
  If $|Q(D)| - |\bigcup_{r_k \in R_k} r_k(ext(\Lambda(R_k)))| = 0$, then, the effectiveness is defined to be 0. Note that effectiveness and answer completeness are related by the following relation:

$$\bigcup_{r_k \in R_k} r_k(ext(\Lambda(R_k))) \subset Q(D) \Rightarrow (GUNEffect(R_k) = 1 \equiv GUN(R_k) = Q(D))$$

Query	Answer Size	# rewritings	# of RV
Q1	3.33E+07	1.61E+09	260
Q2	2.99E+05	6.37E+21	260
Q3	2.03E+05	3.52E+24	280
Q4	1.42E+02	6.02E+03	240
Q5	2.82E+05	1.30E+07	240
Q6	9.84E+04	1.22E+05	100
Q7	1.12E+05	1.15E+12	180
Q8	2.82E+05	4.08E+04	100
Q9	1.41E+04	2.00E+01	20
Q10	1.49E+06	9.76E+05	260
Q11	1.49E+06	3.24E+03	80
Q12	2.99E+05	2.37E+08	260
Q13	2.99E+05	2.41E+04	260
Q14	2.82E+05	8.08E+05	180
Q15	1.41E+05	4.64E+09	280
Q16	1.41E+05	8.36E+04	100
Q17	9.84E+04	2.02E+03	100
Q18	2.82E+05	3.12E+08	240

(a) Query information

Views	Size
V1-V20	147,327
V21-V40	133,992
V41-V60	41,463
V61-V80	22,410
V81-V100	4,515
V101-V120	53,131
V121-V140	32,511
V141-V160	90,873
V161-V180	21,138
V181-V200	9,836
V201-V220	4,515
V221-V240	4,515
V241-V260	67,364
V261-V280	81,313
V281-V300	840,470

(b) Views size

Table 1: Queries and their answer size, number of rewritings, number of relevant views (RV) and views size.

When the execution of R_k does not produce the complete answer, then, GUN's effectiveness for R_k equals to one iff GUN produces complete answers.

- *Execution Time Independency of k:* the execution time of GUN does not depend on the number of rewritings executed (k). It depends on the number of relevant views present in R_k. Execution time is the elapsed time between the generation of R_k rewritings and their execution time. This includes the time required to obtain the data from the wrappers, the time required to add the obtained data to the graph and the time required to execute the query plan on the graph.
- *Non-blocking:* GUN solves the ReMakE problem under the assumption that $\bigcup \Lambda(R_k)$ fits in memory. If not, GUN can only approximate it, for example, by splitting R_k into disjoint R_{k_1} and R_{k_2} such that $\bigcup \Lambda(R_{k_1})$ and $\bigcup \Lambda(R_{k_2})$ fit in memory. Then, execute $GUN(R_{k_1})$, clear the memory, and execute $GUN(R_{k_2})$. Therefore, GUN is a non-blocking execution strategy i.e., running out of memory will not prevent GUN to execute at the expense of non-maximality of the answer, as the combined effectiveness of $GUN(R_{k_1})$ and $GUN(R_{k_2})$ is in general less than this of $GUN(R_k)$.

5 Experimental Evaluation

To setup the experimental evaluation, we used the Berlin SPARQL Benchmark (BSBM) [8] to generate a dataset of 5,000,251 triples, using a scale factor of 14,091 products. We used the 18 queries and the 10 views proposed in [9]. These queries are very challenging for a query rewriter since their triple patterns can be grouped into chained connected star-shaped sub-queries, that have between 1 and 13 subgoals, with only distinguished variables.

We defined 5 additional views to cover all the predicates in the queries. From these 15 views, we produced 300 views by horizontally partitioning each original

view into 20 parts, such that each part produces 1/20 of the answers given by the original view. Queries and views information is shown in Tables 1a and 1b. The size of the complete answer was computed by loading all the views into a persistent RDF-Store (Jena-TDB) and executing the queries over it. The number of rewritings was obtained using the models counting feature of the SSDSAT [15] rewriter.

As we can see in Table 1a, the number of rewritings may be huge, making unfeasible their full execution. Furthermore, the time to generate the rewritings is not negligible, and in some cases (Q2, Q3 and Q7) SSDSAT could not generate them after 72 hours. We chose to compute 500 rewritings, as this was the best compromise we could find between number of rewritings and generation time. Additionally, we do not have any statistics about the sources to select the best rewritings or to shrink the set of relevant views. Q1 execution reached a timeout of 48 hours.

Some general predicates like *rdf:label* are present in the most of the views; therefore, the queries that have a triple pattern with these predicates will have a large number of relevant views, but not all of these views will contribute to the answer. The size of a view corresponds to the number of triples that can be accessed through that view. Detailed information about the definition of the queries and views can be found in the project website [3].

We implemented wrappers as simple file readers. For executing rewritings, we used one named graph per subgoal as done in [16]. The Jena 2.7.4 [4] library with main memory setup was used to store and query the graphs. We used the Left Linear Plans implemented by Jena as a representative of traditional query execution techniques.

5.1 Experimental Results

The analysis of our results focus on four aspects: answer completeness, effectiveness, execution time and non-blocking as defined in section 4.1.

		Q4	Q5	Q6	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18
CA	GUN	281	45	>500	381	20	21	29	36	21	>500	20	21	21	56
	Jena	281	>500	>500	383	20	141	119	>500	320	>500	>500	>500	40	>500
GUN's	k=80	0	1	0.0016	0		1	1	1	1	0.0476	1	1	0	1
Effectiveness	k=160	0	1	0.0002	0		0	0	1	1	0.0451	1	1	0	1
	k=320	0	1	0.0018	0		0	0	1	0	0.0406	1	1	0	1
	k=500	0	1	0.0024	0		0	0	1	0	0.0382	1	1	0	1

Table 2: Values of k for obtaining the Complete Answers (CA) for queries Q4-Q6, Q8-Q18; using GUN and Jena. GUN's Effectiveness for different values of k. Effectiveness for Q9 is not reported here since it only has 20 rewritings.

[3] https://sites.google.com/site/graphunion/
[4] http://jena.apache.org/

Query		Execution Time			
		K=80	K=160	K=320	K=500
Q4	GUN	39	39	63	73
	Jena	167	293	48,943	49,721
Q5	GUN	377	400	400	400
	Jena	1,155	2,302	3,848	5,935
Q6	GUN	336	337	338	339
	Jena	398	798	1,610	2,516
Q8	GUN	41	47	58	64
	Jena	190	377	751	1,278
Q10	GUN	132	132	132	132
	Jena	2,214	5,941	119,137	251,641
Q11	GUN	121	121	121	121
	Jena	1,906	3,707	9,985	16,939
Q12	GUN	28	28	28	28
	Jena	79	146	288	475
Q13	GUN	71	203	478	522
	Jena	146	352	734	2,034
Q14	GUN	328	395	395	395
	Jena	439	842	1,657	2,485
Q15	GUN	358	358	358	358
	Jena	1,207	3,000	5,812	9,160
Q16	GUN	35	35	35	35
	Jena	119	283	596	972
Q17	GUN	69	345	345	345
	Jena	168	965	2,450	4,029
Q18	GUN	324	414	415	415
	Jena	1,149	2,413	4,355	6,808

(a) Execution Time for GUN and Jena

Query		ET and # of RV			
		k=80	k=160	k=320	k=500
Q4	GUN	39	39	63	73
	# RV	23	25	31	38
Q5	GUN	377	400	400	400
	# RV	80	100	100	100
Q6	GUN	336	337	338	339
	# RV	62	63	66	69
Q8	GUN	41	47	58	64
	# RV	24	28	36	40
Q10	GUN	132	132	132	132
	# RV	79	80	80	80
Q11	GUN	121	121	121	121
	# RV	79	80	80	80
Q12	GUN	28	28	28	28
	# RV	80	80	80	80
Q13	GUN	71	203	478	522
	# RV	61	123	240	260
Q14	GUN	328	395	395	395
	# RV	81	101	101	101
Q15	GUN	358	358	358	358
	# RV	60	60	60	60
Q16	GUN	35	35	35	35
	# RV	40	40	40	40
Q17	GUN	69	345	345	345
	# RV	41	100	100	100
Q18	GUN	324	414	415	415
	# RV	80	100	100	100

(b) Execution Time and Number of Relevant Views for GUN

Table 3: Execution Time (ET) for GUN and Jena. Impact of Number of Relevant Views (RV) over Execution Time in GUN.

To study the answer completeness of GUN, we executed the GUN and Jena strategies over R_k rewritings, we counted the number of rewritings to have the complete answer. Table 2 shows that GUN is able to achieve the complete answer for 12 queries whereas Jena is able to do so only for 6 queries. For queries Q9, Q11, Q13 and Q17, GUN produced complete answers because at the reported k, $\Lambda(R_k) = \Lambda(R)$. For the rest of the queries, the non aggregated relevant views did not contribute to produce more results. Detailed information about the ratio of relevant views for each R_k can be found in the project's website.

To demonstrate the effectiveness of GUN, we executed the GUN and Jena strategies over R_k with $k \in \{80, 160, 320, 500\}$, counted the number of answers and computed the effectiveness. Table 2 shows that GUN has effectiveness 1 for $k = 80$ for half of the queries, moreover, in 5 of these 7 queries, the maximum effectiveness remains even after Jena executes 500 rewritings. In 4 cases, GUN is not effective because Jena already found the complete answer for this value of k. Finally, in Q6 and Q14, GUN found more results than Jena. Effectiveness values are not monotonic, since they can increase when considering a rewriting that contains a view that contributes to produce results in GUN and not in Jena. However, they can decrease after executing a rewriting that does not add new views to GUN, but produces results for Jena.

Regarding the execution time, we want to: *1*) demonstrate that GUN execution time does not depend on k, but on $|\Lambda(R_k)|$, and *2*) compare GUN's

execution time with Jena's. Table 3a shows total execution time, the detailed values of the execution time are available in the project website. For all queries GUN has better execution time, and for all but Q6 with $k = 80$, is more than twice faster. When $k = 500$, the difference is dramatic, varying from almost 4 times faster (Q13) to 680 times faster (Q4). Table 3b shows total execution time and number of loaded views for GUN. Execution time grows linearly in $|\Lambda(R_k)|$, this is particularly visible in Q4 and Q13.

If we compare the times detailed in section 4.1, we notice that the dominating time is the wrapper time. GUN loads views into the aggregated graph only once, whereas Jena reloads them for each executed rewriting. Note that if we try to cache the views in Jena to avoid reloading, it would consume more memory and could consume even more memory than GUN if the views have overlapped information, as it is the case in our setup.

Query	Maximal Graph Size k=80		Maximal Graph Size k=160		Maximal Graph Size k=320		Maximal Graph Size k=500	
	GUN	Jena	GUN	Jena	GUN	Jena	GUN	Jena
Q4	1,201,671	148,739	1,208,714	148,739	1,753,969	907,775	1,878,666	907,775
Q5	1,993,617	907,905	2,275,437	907,923	2,275,437	907,923	2,275,437	907,923
Q6	1,578,294	850,376	1,583,212	850,376	1,597,964	850,376	1,612,716	850,376
Q8	1,479,686	148,725	1,536,050	148,745	1,648,778	148,745	1,705,142	230,045
Q10	422,269	294,678	422,269	294,678	422,269	442,052	422,269	442,052
Q11	422,268	294,701	422,269	294,701	422,269	294,748	422,269	294,748
Q12	439,946	83,260	439,946	83,260	439,946	83,260	439,946	83,276
Q13	1,713,056	862,917	2,277,638	862,962	2,923,233	862,962	2,923,233	862,962
Q14	2,095,418	912,422	2,279,248	926,356	2,279,248	935,825	2,279,248	935,825
Q15	1,568,458	905,450	1,568,458	905,529	1,568,458	905,529	1,568,458	905,529
Q16	584,792	53,678	584,792	63,411	584,792	63,411	584,792	74,802
Q17	1,496,262	850,331	1,807,718	850,376	1,807,718	850,376	1,807,718	850,376
Q18	2,175,448	907,916	2,275,437	921,840	2,275,437	921,840	2,275,437	921,859

Table 4: Maximum number of triples loaded by a rewriting in R_k in Jena. The number of triples of the aggregated graph of GUN.

Finally, we analyzed GUN's and Jena's memory consumption to demonstrate that in spite of complex queries and many relevant views: *1*) GUN is not blocking, and *2*) to compare memory used by GUN with respect to Jena. For GUN, we count the number of triples of the aggregated graph. For Jena, we report an upper bound, that is, the maximum number of triples loaded for executing a rewriting in R_k. Table 4 summarizes the results. Neither GUN nor Jena consumes all the available memory (8GB). GUN needs to load more triples than Jena, varying from less than twice to 12 times more, in all cases except for Q10 with $k \geq 320$. GUN's aggregation is in general larger than the sum of the named graphs of the most memory-consuming rewriting in R_k.

In summary, GUN is effective with better execution time at the cost of higher memory consumption. However, in our experimentation GUN never exhausts the available memory in spite of the challenging setup. This makes it a very appealing solution for the ReMakE problem.

6 Related Work

In recent years, several approaches have been proposed for querying the Web of Data [17–21]. Some tools address the problem of choosing the sources that can be used to execute a query [20, 21]; others have developed techniques to adapt query processing to source availability [17, 20]. Finally, frameworks to retrieve and manage Linked Data have been defined [18, 20], as well as strategies for decomposing SPARQL queries against federations of endpoints [7]. All these approaches assume that queries are expressed in terms of RDF vocabularies used to describe the data in the RDF sources; their main challenge is to effectively select the sources, and efficiently execute the queries on the data retrieved from the selected sources. In contrast our approach attempts to semantically integrate data sources, and relies on a global vocabulary to describe data sources and provide a unified interface to the users. Thus, in addition to efficiently gather and process the data transferred from the selected sources, it decides which of the rewritings of the original query need to execute to efficiently and effectively produce the query answer.

Two main paradigms have been proposed to define the data sources in integration systems. The LAV approach is commonly used because it permits the scalability of the system as new data sources become available [22]. Under LAV, the appearance of a new source only causes the addition of a new mapping describing the source in terms of the concepts in the RDF global vocabulary. Under GAV, on the other hand, entities in the RDF global vocabulary are semantically described using views in terms of the data sources. Thus, the extension or modification of the global vocabulary is an easy task in GAV as it only involves the addition or local modification of few descriptions [22]. Therefore, the LAV approach is best suited for applications with a stable RDF global vocabulary but with changing data sources whereas the GAV approach is best suited for applications with stable data sources and a changing vocabulary. Given the nature of the Semantic Web, we rely on the LAV approach to describe the data sources in terms of a global and unified RDF vocabulary, and assume that the global vocabulary of concepts is stable while data sources may constantly pop up or disappear from the Web.

The problem of rewriting a global query into queries on the data sources is one relevant problem in integration systems [23], and several approaches have been defined to efficiently enumerate the query rewritings and to scale when a large number of views exists (e.g., MCDSAT [4], GQR [5], Bucket Algorithm [23], MiniCon [11]). Recently, Le et al, [16] propose a solution to identify and combine GAV SPARQL views that rewrite SPARQL queries against a global vocabulary, and Izquierdo et al [15] extends the MCDSAT with preferences to identify the combination of semantic services that rewrite a user request. A great effort has been made to provide solutions able to produce query writings in the least time possible, however, to the best of our knowledge, the problem of executing the query rewritings against the selected sources still remains open.

We address this problem and propose GUN, a query processing technique for RDF store architectures that provide a uniform interface to data sources that

have been defined using the LAV paradigm [1]. GUN assumes that the query rewriting problem has been solved using an off-the-shell query rewriter (e.g., [15, 5]), which may produce a large number of query rewritings. GUN implements a query processing strategy able to execute a reduced number of query rewritings of and generate a *more complete* answer than the rest of the engines in less time, as it was observed in our experimental results.

Since the number of valid query rewritings can be exponential in the number of sources, providing an effective and efficient semantic data management technique to reduce the number of query rewritings executed is a relevant contribution to the implementation of integration systems, and provides the basis for feasible semantic integration architectures in the Web of Data.

7 Conclusion and Future Work

Performing complex queries on different data sources raises the severe issue of semantic heterogeneity. Local-as-View mediators is one of the main approaches to solve it. However, the high number of rewritings needed to be executed represents a severe bottleneck. We proposed the ReMakE problem, that consists in maximizing the number of results obtained by considering only k rewritings (R_k). We also proposed GUN, a solution to this problem, it uses the RDF data model and takes advantage of the low cost of graph union operation.

Compared to state-of-the-art approaches, GUN provides an alternative way to improve performance at the execution engine level rather than at the rewriter level. This makes GUN usable with any LAV rewriter guaranteeing to achieve greater or equal answer completeness for the same R_k. Our experiments demonstrate that GUN gain is real, i.e., its effectiveness is equal to one for 57% of the queries for the values of k until 80. It remains equal to one for 38% of the queries for the values of k until 500.

Furthermore, GUN consumes considerably less execution time than Jena in all the cases; the difference in execution time is tremendous, up to 681 times. However, this improvement in effectiveness and execution time comes with an additional memory consumption cost of up to 12 times.

This work opens new perspectives to improve LAV approach for the Semantic Web. We would like to measure the effectiveness degradation when executing on low-memory setups, and include some heuristics to minimize it. As GUN creates materialized views for processing rewritings, we plan to evaluate the impact on the effectiveness and the execution time when performing inference tasks on the graph union. As GUN is mostly dependent on the ratio of views in rewritings divided by the number of relevant views, an interesting perspective is to modify rewriters to optimize the number of views in the first k rewritings.

References

1. Levy, A.Y., Mendelzon, A.O., Sagiv, Y., Srivastava, D.: Answering queries using views. In: Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS'95. (1995) 95–104

2. Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M., Senellart, P.: Web data management. Cambridge University Press (2011)
3. Abiteboul, S., Duschka, O.M.: Complexity of answering queries using materialized views. In: Seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS'98. (1998) 254–263
4. Arvelo, Y., Bonet, B., Vidal, M.E.: Compilation of query-rewriting problems into tractable fragments of propositional logic. In: AAAI. (2006) 225–230
5. Konstantinidis, G., Ambite, J.L.: Scalable query rewriting: a graph-based approach. In: SIGMOD. (2011) 97–108
6. Vidal, M.E., Ruckhaus, E., Lampo, T., Martinez, A., Sierra, J., Polleres, A.: Efficiently Joining Group Patterns in SPARQL Queries. In: ESWC. (2010)
7. Schwarte, A., Haase, P., Hose, K., Schenkel, R., Schmidt, M.: Fedx: Optimization techniques for federated query processing on linked data. In: ISWC. (2011) 601–616
8. Bizer, C., Shultz, A.: The berlin sparql benchmark. International Journal on Semantic Web and Information Systems **5** (2009) 1–24
9. Castillo-Espinola, R.: Indexing RDF data using materialized SPARQL queries. PhD thesis, Humboldt-Universität zu Berlin (2012)
10. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer **25** (1992) 38–49
11. Halevy, A.Y.: Answering queries using views: A survey. The VLDB Journal **10** (2001) 270–294
12. Pérez, J., Arenas, M., Gutiérrez, C.: Semantics and complexity of sparql. ACM Transactions on Database Systems (TODS) **34** (2009)
13. Baget, J.F., Croitoru, M., Gutierrez, A., Leclère, M., Mugnier, M.L.: Translations between rdf(s) and conceptual graphs. In: ICCS. (2010) 28–41
14. Chaudhuri, S.: An overview of query optimization in relational systems. In: Seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS'98. (1998) 34–43
15. Izquierdo, D., Vidal, M.E., Bonet, B.: An expressive and efficient solution to the service selection problem. In: ESWC. (2010) 386–401
16. Le, W., Duan, S., Kementsietsidis, A., Li, F., Wang, M.: Rewriting queries on sparql views. In: WWW, ACM (2011) 655–664
17. Acosta, M., Vidal, M.E., Lampo, T., Castillo, J., Ruckhaus, E.: Anapsid: An adaptive query processing engine for sparql endpoints. In: ISWC (1). (2011) 18–34
18. Basca, C., Bernstein, A.: Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In: SSWS. (2010) 64–79
19. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U., Umbrich, J.: Data summaries for on-demand queries over linked data. In: WWW. (2010) 411–420
20. Hartig, O.: Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In: ESWC. (2011) 154–169
21. Ladwig, G., Tran, T.: Sihjoin: Querying remote and local linked data. In: ESWC. (2011) 139–153
22. Ullman, J.D.: Information integration using logical views. Theoretical Computer Science **239** (2000) 189–210
23. Levy, A., Rajaraman, A., Ordille, J.: Querying heterogeneous information sources using source descriptions. In: VLDB. (1996) 251–262