Storing and Querying Scientific Workflow Provenance Metadata Using an RDBMS

Artem Chebotko, Xubo Fei, Cui Lin, Shiyong Lu, and Farshad Fotouhi
Department of Computer Science, Wayne State University
5143 Cass Avenue, Detroit, Michigan 48202, USA
{artem, xubo, cuilin, shiyong, fotouhi}@wayne.edu

Abstract

Provenance management has become increasingly important to support scientific discovery reproducibility, result interpretation, and problem diagnosis in scientific workflow environments. This paper proposes an approach to provenance management that seamlessly integrates the interoperability, extensibility, and reasoning advantages of Semantic Web technologies with the storage and querying power of an RDBMS. Specifically, we propose: i) two schema mapping algorithms to map an arbitrary OWL provenance ontology to a relational database schema that is optimized for common provenance queries; ii) two efficient data mapping algorithms to map provenance RDF metadata to relational data according to the generated relational database schema, and iii) a schema-independent SPARQL-to-SQL translation algorithm that is optimized on-the-fly by using the type information of an instance available from the input provenance ontology and the statistics of the sizes of the tables in the database. Experimental results are presented to show that our algorithms are efficient and scalable.

1 Introduction

Today, many significant scientific discoveries are achieved through complex and distributed scientific computations. More and more scientists start to use workflow technologies to automate the steps they need to go through from raw datasets to potential scientific discovery. As a result, scientific workflow has emerged as a new field to address the new requirements from scientists [15, 16]. A scientific workflow is a formal specification of a scientific process, which represents, streamlines, and automates the steps from dataset selection and integration, computation and analysis, to final data product presentation and visualization. A Scientific Workflow Management System (SWMS) supports the specification, execution, re-run, and monitoring of scientific processes [15, 17, 9, 12, 10, 25, 11].

Provenance management is essential for scientific workflows to support scientific discovery reproducibility, result interpretation, and problem diagnosis [19, 4]; such a facility is not necessary for business workflows. Provenance metadata captures the derivation history of a data product, including the original data sources, intermediate data products, and the steps that were applied to produce the data product. The provenance management problem concerns about the efficiency and effectiveness of the recording, representation, storage, querying, and visualization of provenance metadata.

This paper proposes an approach to provenance management that seamlessly integrates the interoperability, extensibility, and reasoning advantages of Semantic Web technologies with the storage and querying power of an RDBMS. The motivation of using Semantic Web technologies is three folds. First, large-scale e-science applications can span multiple domains and can involve global distributed workflows that consist of several heterogeneous local workflows orchestrated by different workflow engines, each of which has its own provenance manager [26]. The integration of these heterogenous provenance systems is important for global provenance analysis. A Semantic Web approach promotes interoperability and facilitates such provenance integration. Second, RDF is a property-centric, extremely flexible and dynamic data model, which captures well the dynamic and heterogeneous nature of data, services, and metadata in e-science applications. Finally, we can use the inference capability of Semantic Web for deriving metadata for various provenance dependency graphs [5].

The main contributions of this paper are: i) we propose two schema mapping algorithms to map an arbitrary OWL provenance ontology to a relational database schema that is optimized for common provenance queries; ii) we propose two efficient data mapping algorithms to map provenance RDF metadata to relational data according to the generated relational database schema, and iii) we propose a schema-independent SPARQL-to-SQL translation algorithm that is optimized on-the-fly by using the type information of an instance available from the input provenance ontology and the statistics of the sizes of the tables in the database. Experimental results are presented to show that our algorithms are efficient and scalable.

Organization. The rest of the paper is organized as follows. Related work is described in Section 2. Provenance ontology to database schema mapping is introduced in Section 3. Provenance metadata to relational data mapping is presented in Section 4. SPARQL-to-SQL query translation is presented in Section 5. Conclusions and future work are discussed in Section 6.

2 Related Work

Provenance management has become an important functionality for most scientific workflow management systems; see [4, 19] for surveys. Kepler [3] implements a *provenance recorder* to record information about a workflow run, including the context, data derivation history, workflow definition, and workflow evolution. myGrid/Taverna [24] uses Semantic Web technologies for representing provenance metadata at four levels: process, data, organization, and knowledge. Chimera [11] introduces a Virtual Data Catalog (VDC) consisting of a set of relations to store the description of executable programs as transformations, their actual invocations as derivations, and input/outputs as data objects. VisTrails [12] is the first system that supports provenance tracking of workflow evolution in addition to tracking the data product derivation history. A couple of stand-alone provenance systems have also been developed, including PReServ [13] and Karma [20].

Although our approach is similar to Taverna's in that Semantic Web technologies are used for provenance modeling and representation, our contributions are different as we focus on the efficiency of provenance queries using an RDBMS. First, our strategy for ontology to database schema mapping results in a database schema that is optimized for common provenance queries. Second, our RDF to relational data mapping procedure employs our incremental strategy that takes advantage of the order of inserting various kinds of provenance metadata. Finally, we use SPARQL for provenance querying and propose a schema-independent SPARQL-to-SQL translation algorithm; the translation algorithm is optimized by using the type information of an instance and the statistics of the size of the tables in the database.

In the context of RDBMS-based RDF storage and query systems [23, 6, 21, 2, 18, 22, 14, 1], not only our approach employs a novel database schema to store RDF graphs, but also our SPARQL-to-SQL translation features innovative optimizations and a schema-independent algorithm. Our algorithms and optimizations, although aimed for provenance metadata, are applicable for RDF in general. Although not reported in this paper, the comparison with Jena [23] and Sesame [6] showed an improved query performance of our system.

3 Provenance Ontology to Database Schema Mapping

In this section, we propose two provenance ontology to database schema mapping algorithms using our developed provenance ontology PO as a running example.

3.1 A running example of provenance ontology

We have developed a provenance ontology called PO that not only supports the semantic, syntactic, and structural description of workflows, tasks, and data objects, but also their execution instances. Our goal is to support queries across workflow definitions, workflow runs, and data objects. Currently PO includes over 30 classes and 40 properties. Figure 1 illustrates an excerpt of PO which sketches concepts for workflow definition, workflow execution, workflow evolution, and definition-execution relationships. More details are available in [7].

Consider a sample workflow in Figure 2(a). It consists of three tasks, two workflow inputs, one workflow input parameter, and two workflow outputs. Each task represents a computational or analytical step of a scientific process. A task has input ports and output ports that provide the communication interface to other tasks. Tasks are linked together into the workflow as an acyclic graph via data channels. During workflow execution, tasks communicate with each other by passing data via their ports through data channels. An RDF graph that describes this workflow is drawn in Figure 2(b), and an execution of this workflow produces the RDF graph in Figure 2(c). Both RDF graphs are stored into our provenance database.

An important advantage of using a formal ontology is the ability to derive new RDF descriptions using semantic inference. Out of two inference support mechanisms, *forward-chaining*, in which all inferences are precomputed and stored, and *backward-chaining*, in which inferences are computed dynamically for each query, we adopt the first one, which can be efficiently implemented outside of an RDBMS. We support: (1) *OWL semantics inference*, which uses A-Box inference rules for OWL constructs, such as *rdfs:subClassOf*, *owl:TransitiveProperty*, and *owl:SymmetricProperty*, and (2) *Provenance dependency graph inference*, which uses our rules to derive various provenance graphs. For example, our rule for deriving a data dependency graph is as follows. If TR *input* $D1$ and

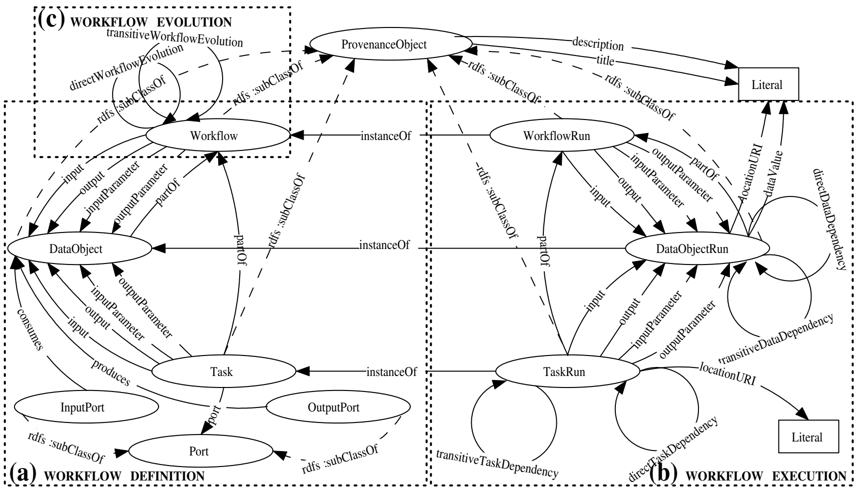


Figure 1. An excerpt of the provenance ontology

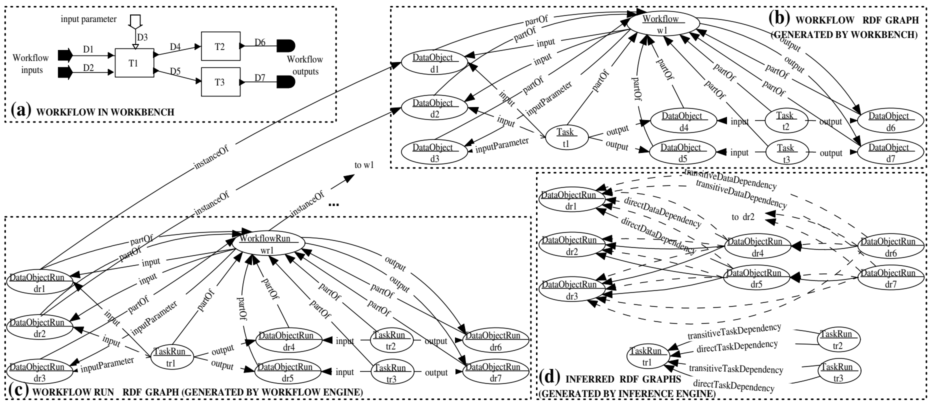


Figure 2. A sample workflow and its RDF graphs

TR *output* $D2$, then $D2$ *directDataDependency* $D1$ and $D2$ *transitiveDataDependency* $D1$, where TR is an instance of *TaskRun* that has input and output data object runs $D1$ and $D2$, respectively, and the *transitiveDataDependency* property is defined as *owl:TransitiveProperty*. Task invocation dependency and workflow evolution graphs can be inferred based on similar rules. Given the workflow run RDF graph in Figure 2(c), data dependency and task invocation dependency graphs are shown in Figure 2(d). These RDF graphs are precomputed by an inference engine and stored into our provenance database.

3.2 Schema mapping algorithms

Database schema design is of decisive importance to support efficient processing of provenance queries. In general, generating an optimal schema for a given set of queries under certain time and space efficiency constraints is a hard problem. Our schema design aims at supporting efficient processing of the following queries that are common in provenance retrieval and browsing: (1) retrieve RDF graph nodes of a given type, such as *Workflow*, *WorkflowRun*, and *Task*, (2) retrieve all the immediate neighboring nodes of a given node reachable through incoming or outgoing edges, (3) retrieve nodes that are directly related to a given node by some properties, such as *input*, *output*, and *partOf*, and (4) a combination of the above queries.

Our two alternative schema mapping algorithms are presented in Figure 3. SchemaMapping-V generates one table and four kinds of views from a given ontology with class-set $\mathcal{C}$ and property-set $\mathcal{P}$: (1) a single table *Triple(s,p,o)* that stores all RDF triples in the database, (2) a view $\$c$*(i)* that captures all instances of class $\$c \in \mathcal{C}$, (3) a view $\$c$*Subject(i,p,o)* that captures all triples whose subjects are instances of class $\$c \in \mathcal{C}$, (4) a view $\$c$*Object(s,p,i)* that captures all triples whose objects are instances of class

```
Algorithm SchemaMapping-V
Input: ontology with class-set $\mathcal{C}$ and property-set $\mathcal{P}$
Output: relational database schema
Begin
  Create table Triple(s,p,o)
  For each class $\$c \in \mathcal{C}$
    Create view $\$c$(i) = Select s From Triple Where p='rdf:type' And o='$\$c$'
    Create view $\$c$Subject(i,p,o) = Select s,p,o From Triple, $\$c$ Where s=i
    Create view $\$c$Object(s,p,i) = Select s,p,o From Triple, $\$c$ Where o=i
  End For
  For each property $\$p \in \mathcal{P}$
    Create view $\$p$(s,o) = Select s,o From Triple Where p='$\$p$' End For
End Algorithm

Algorithm SchemaMapping-T
Input: ontology with class-set $\mathcal{C}$ and property-set $\mathcal{P}$
Output: relational database schema
Begin
  Create table Triple(s,p,o)
  For each class $\$c \in \mathcal{C}$
    Create table $\$c$(i)
    Create table $\$c$Subject(i,p,o)
    Create table $\$c$Object(s,p,i)
  End For
  For each property $\$p \in \mathcal{P}$, Create table $\$p$(s,o) End For
End Algorithm
```

Figure 3. Algorithms SchemaMapping-V **and** SchemaMapping-T

$\$c \in \mathcal{C}$, and (5) a view $\$p$*(s,o)* that captures all instances of property $\$p \in \mathcal{P}$. SchemaMapping-T behaves similarly, but materializes all views as tables. The total number of generated relations (tables and views) is $1 + 3 \times |\mathcal{C}| + |\mathcal{P}|$.

In addition, we make extensive use of database indexes. For the SchemaMapping-V schema, we create indexes on columns (s,p,o), (s,o), (o,p), and (p) of table *Triple(s,p,o)*. For the SchemaMapping-T schema, we create similar indexes on columns of *Triple(s,p,o)*, $\$c$*Subject(i,p,o)*, and $\$c$*Object(s,p,i)*, indexes on columns (s,o) and (o) of $\$p$*(s,o)*, and single-column indexes on tables $\$c$*(i)*. Finally, each table has the uniqueness constraint on the tuples it stores.

3.3 Experimental study

All the experiments reported in this paper were conducted on a PC with 2.4 GHz Pentium IV CPU and 1024 MB of main memory operated by MS Windows XP Professional. All algorithms were implemented in C/C++ and MySQL 5.0 Community Edition was employed as the RDBMS. Our developed provenance server communicated with MySQL using the MySQL C API.

The performance of our schema mapping algorithms on PO is as follows.

Characteristic	SchemaMapping-V	SchemaMapping-T
# of tables created	1	135
# of views created	134	0
# of indexes created	4	365
Time (s)	4.437	53.641

The reported times include the time required to process the ontology. In our approach, the schema mapping is only required to be performed once to store multiple provenance datasets.

4 Provenance metadata to relational data mapping

4.1 Data mapping algorithms

In this section, we present two data mapping algorithms that insert a new provenance dataset D, either of a workflow definition, a workflow run, or a workflow provenance dependency graph, into the database. The DataMapping-V algorithm that corresponds to SchemaMapping-V is trivial as all we need to do is to insert D into table *Triple*. For the database schema created by SchemaMapping-T, table *Triple* can be populated similarly, i.e., by simply inserting D into *Triple*. Let *Triple*$'$ be a temporary table for storing the triples of D. New tuples for $\$c$ can be calculated by $\$c'$*(i)* $\leftarrow$ `Select s From` *Triple*$'$ `Where p='`*rdf:type*`' And o='`$\$c$`'`, and new tuples for $\$p$ can be calculated by $\$p'$*(s,o)* $\leftarrow$ `Select s,o From` *Triple*$'$ `Where p='`$\$p$`'`. The question is how we can calculate new tuples for tables $\$c$*Subject* and $\$c$*Object* for each class $\$c \in \mathcal{C}$.

One strategy, called *brute-force*, is to calculate *Triple*$'$, $\$c'$, and $\$p'$ and insert them into tables *Triple*, $\$c$, and $\$p$, respectively. Then, delete contents of $\$c$*Subject* and $\$c$*Object* and rematerialize these two tables: $\$c$*Subject(i,p,o)* $\leftarrow$ `Select s,p,o From` *Triple,* $\$c$ `Where s=i` and $\$c$*Object(s,p,i)* $\leftarrow$ `Select s,p,o From` *Triple,* $\$c$ `Where o=i`. However, this strategy is expensive since we have to recompute joins of *Triple* and $\$c$, whose sizes are growing over time.

A better strategy, called *incremental*, calculates new tuples in $\$c$*Subject*$'$ and $\$c$*Object*$'$ and then inserts them into $\$c$*Subject* and $\$c$*Object*, respectively. $\$c$*Subject*$'$ and $\$c$*Object*$'$ are calculated as follows: $\$c$*Subject*$'$*(i,p,o)* $\leftarrow$ `(Select s,p,o From` *Triple*$'$, $\$c'$ `Where s=i) Union (Select s,p,o From` *Triple,* $\$c'$ `Where s=i) Union (Select s,p,o From` *Triple*$'$, $\$c$ `Where s=i)` and $\$c$*Object*$'$*(s,p,i)* $\leftarrow$ `(Select s,p,o From` *Triple*$'$, $\$c'$ `Where o=i) Union (Select s,p,o From` *Triple,* $\$c'$ `Where o=i) Union (Select s,p,o From` *Triple*$'$, $\$c$ `Where o=i)`. In other words, we need to compute unions of three joins: (1) *Triple*$' \bowtie \$c'$, (2) *Triple* $\bowtie \$c'$, and (3) *Triple*$' \bowtie \$c$. In contrast to the brute-force strategy, the incremental strategy requires to compute joins of smaller tables.

Finally, our *optimized incremental* strategy is similar to the incremental one, except that we do not need to compute the join *Triple* $\bowtie \$c'$ when populating $\$c$*Subject*$'$ and $\$c$*Object*$'$. This simplification is possible because provenance datasets are stored in *order*, such that a workflow definition is stored at first, its workflow run is stored at second

```
Algorithm DataMapping-T
Input: RDF dataset $D$, class-set $\mathcal{C}$, and property-set $\mathcal{P}$
Output: database populated with relational tuples
Begin
  Let Triple′(s,p,o) be a temporary table
  Parse $D$ and bulkload triples into Triple′(s,p,o)
  Insert into Triple(s,p,o) ← Select s,p,o From Triple′(s,p,o)
  For each class $\$c \in \mathcal{C}$
    Insert into $c(i) ← Select s From Triple′ Where p=‘rdf:type’ And o=‘$c’
    Insert into $cSubject(i,p,o) ← (Select s,p,o From Triple′, $c Where s=i)
    Insert into $cObject(s,p,i) ← (Select s,p,o From Triple′, $c Where o=i)
  End For
  For each property $\$p \in \mathcal{P}$
    Insert into $p(s,o) ← Select s,o From Triple′ Where p=‘$p’
  End For
  Delete all tuples from Triple′
End Algorithm
```

Figure 4. Algorithm DataMapping-T

and its provenance graphs are stored last. As a result, a to-be-stored dataset D may have an instance X whose type (X *rdf:type class*) is not defined in D, but is defined in the triple-set stored in the database; the other way around can never be true. Therefore, the join of *Triple* and $\$c'$ can return only tuples that are already in the database in $\$c$*Subject* or in $\$c$*Object*. Furthermore, since we left with only two required joins *Triple*$' \bowtie \$c'$ and *Triple*$' \bowtie \$c$, we can replace them by *Triple*$' \bowtie (\$c' \cup \$c)$ or by inserting $\$c'$ into $\$c$ and computing *Triple*$' \bowtie \$c$. This strategy substantially reduces the number of join operations.

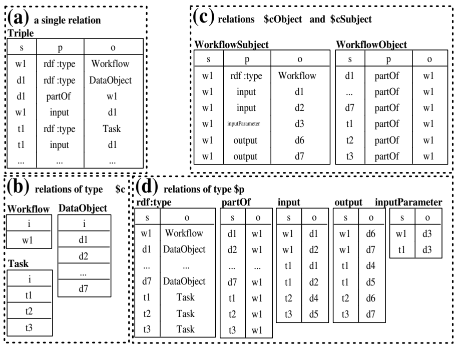
Figure 4 formally defines algorithm DataMapping-T that implements the optimized incremental strategy.

Note that the following three properties regarding the cardinalities of relations always hold: (1) $|Triple| \geq |\$cSubject| \geq |\$c|$, (2) $|Triple| \geq |\$cObject| \geq |\$c|$, and (3) $|Triple| \geq |\$p|$. This information can be used to select the smallest relation for query optimization. In addition, we cache cardinality statistics for relations $\$p$, $\$c$*Subject*, and $\$c$*Object*.

Figure 5 shows some of the relations generated and loaded with tuples that correspond to the workflow RDF graph in Figure 2(b).

4.2 Experimental study

Algorithms DataMapping-V and DataMapping-T were evaluated to store a single workflow run into the database for varying number of workflow runs already stored in the database. In particular, we stored five workflow definitions into the database and measured the times to store 1st, 3rd, 21st, 201st, 2001st, and 20001st workflow runs. Each workflow run provenance contained 500 triples in the N-Triples format. The reported times (see Figure 6) correspond to algorithm evaluations over MySQL with "warm cache" (see [7] for "cold" runs). Both algorithms showed good performance and scalability, in particular, DataMapping-V showed stable performance of about



(a) a single relation

Triple

s	p	o
w1	rdf :type	Workflow
d1	rdf :type	DataObject
d1	partOf	w1
w1	input	d1
t1	rdf :type	Task
t1	input	d1
...	...	...

(b) relations of type $c

Workflow

i
w1

DataObject

i
d1
d2
...
d7

Task

i
t1
t2
t3

(c) relations $cObject and $cSubject

WorkflowSubject

s	p	o
w1	rdf :type	Workflow
w1	input	d1
w1	input	d2
w1	inputParameter	d3
w1	output	d6
w1	output	d7

WorkflowObject

s	p	o
d1	partOf	w1
...	partOf	w1
d7	partOf	w1
t1	partOf	w1
t2	partOf	w1
t3	partOf	w1

(d) relations of type $p

rdf:type

s	o
w1	Workflow
d1	DataObject
...	...
d7	DataObject
t1	Task
t2	Task
t3	Task

partOf

s	o
d1	w1
d2	w1
...	...
d7	w1
t1	w1
t2	w1
t3	w1

input

s	o
w1	d1
w1	d2
t1	d1
t1	d2
t2	d4
t3	d5

output

s	o
w1	d6
w1	d7
t1	d4
t1	d5
t2	d6
t3	d7

inputParameter

s	o
w1	d3
t1	d3

Figure 5. A workflow RDF graph stored into a relational database

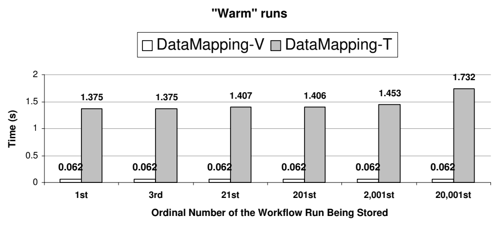


Figure 6. Performance of DataMapping-V **and** DataMapping-T

0.1s per workflow run, sequentially storing 20,000 different workflow runs in 2,375.625s, and DataMapping-T – about 1.3 – 1.7s per workflow run, sequentially storing 20,000 different workflow runs in 34,576.156s.

5 SPARQL-to-SQL query translation

5.1 Query translation algorithm

In our approach, SPARQL is primary language for provenance querying. SPARQL is used to specify queries with basic, group, optional, and alternative graph patterns that are matched over RDF provenance graphs. For example, the following SPARQL query returns information that describes workflows that require user input (have input parameters):

```
Select ?w ?p ?o
Where {?w rdf:type :Workflow .  ?w ?p ?o .
?w :inputParameter ?x}
```

In the `Select` clause, it specifies three variables `?w` for the workflow, `?p` for the predicate, and `?o` for the object, whose instantiations must be returned. In the `Where` clause, the query has the basic graph pattern consisting of three triple patterns: `?w rdf:type :Workflow` to match instances of the *Workflow* type, `?w ?p ?o` to match related instances, and `?w :inputParameter ?x` to ensure that a workflow has at least one input parameter. To evaluate such a query over our relational provenance database, we design algorithms to translate a SPARQL query into an equivalent SQL query.

We first consider the problem of matching a triple pattern tp ($tp.sub$, $tp.pre$, $tp.obj$) against the database. Two questions need to be answered: (1) Which relation should be used for tp? (2) Which relational attributes should be used for $tp.sub$, $tp.pre$, and $tp.obj$? These questions are formulated in terms of two mapping functions, ρ and α, such that $\rho(tp)$ returns the relation that stores all the triples matching tp, while $\alpha(tp, sub)$, $\alpha(tp, pre)$, and $\alpha(tp, obj)$ return the corresponding relational attributes. The two mapping functions provide a foundation for schema-independent SPARQL-to-SQL translation, such that the relational schema design, which concerns about ρ and α, is fully separated from the translation algorithm that is parameterized by ρ and α. In addition, such an abstraction enables schema design optimization and query optimization. We will not pursue the schema design optimization issue in this paper, but illustrate two query optimization techniques below by using the type information of an instance.

The first optimization is based on the following two questions: (1) Given an instance or variable X in a triple pattern tp from a SPARQL query q, can we determine the type of X based on q? (2) If the answer to the first question is yes, let $\tau(X)$ be the type of X, then which relation should be chosen for $\rho(tp)$ among $\tau(X)$, $\tau(X)$*Subject*, $\tau(X)$*Object*, or *Triple*? Intuitively, we like to choose the relation with the smallest number of tuples.

For the first question, given an instance or variable X in a triple pattern tp from a SPARQL query q and our provenance ontology PO, $\tau(X)$ can be decided as follows:

$$\tau(X) = \begin{cases} \textit{c} & \text{If } tp \text{ is of the form } X \textit{ rdf:type :c}\text{, where } c \in \mathcal{C} \text{ is a class in PO;} \\ \textit{p} & \text{If } tp.pre = \textit{:p} \text{ and } X = \textit{:p}\text{, where } p \in \mathcal{P} \text{ is a property in PO;} \\ \textit{c} & \text{If } (\bigcap_{\forall p_d \in q} domain(p_d)) \cap (\bigcap_{\forall p_r \in q} range(p_r)) = \{\textit{:c}\}, \\ & \text{where } p_d \text{ and } p_r \text{ are property instances in some triple patterns} \\ & tp_d \text{ and } tp_r \text{ from } q\text{, such that } tp_d.sub = X, tp_d.pre = p_d, \\ & tp_r.obj = X, tp_r.pre = p_r, \text{ and } p_r \text{ is of type} \\ & \textit{owl:ObjectProperty}; \\ \textit{undef} & \text{otherwise.} \end{cases}$$

In other words, $\tau(X)$ is defined if the type of X is explicitly stated in q via the *rdf:type* property, or X is a property instance, or it is computed as the intersection of the domain and range sets of all the properties that are predicates in q's triple patterns and X is a subject and object, respectively, in these triple patterns. If the result of the intersection is a set with one ontology class, then $\tau(X)$ is defined; otherwise, $\tau(X)$ is undefined.

For our sample query, $\tau(?w)$ = *Workflow* as stated in the first triple pattern, τ(*rdf:type*) = *type*, and τ(*:inputParameter*) = *inputParameter*. The other instances and variables have undefined value of τ; in particular, $\tau(?x)$ can not be computed because the range of *inputParameter* contains several classes.

The answer to the second question of choosing the best relation for $\rho(tp)$ is described in Figure 7. Algorithm Calculate-ρ-α is used to compute ρ and α for each tp in a SPARQL query, such that $\rho(tp)$ is assigned the smallest relation and α values are directly decided by the relation schema of $\rho(tp)$. The smallest relation is identified using the min function (line 08). When $\tau(tp.sub)$, $\tau(tp.obj)$, or $\tau(tp.pre)$ is undefined, we assign $|\tau(tp.sub)$*Subject*$|$=$+\infty$, $|\tau(tp.obj)$*Object*$|$=$+\infty$, or $|\tau(tp.pre)|$=$+\infty$, respectively.

For our sample query, we have ρ(`?w rdf:type :Workflow`) = *Workflow* (Case 1), ρ(`?w ?p ?o`) = *WorkflowSubject* (Case 2) because $\tau(?p)$ and $\tau(?o)$ are undefined and $|$*WorkflowSubject*$|\leq|$*Triple*$|$, and ρ(`?w :inputParameter ?x`) = *inputParameter* (Case 4), assuming that $|$*inputParameter*$| \leq |$*WorkflowSubject*$|$. α is assigned accordingly.

The second optimization is the elimination of some redundant triple patterns from a basic graph pattern bgp in a SPARQL query. In particular, we eliminate tp of the form X *rdf:type :c*, if X also appears in another triple pattern tp', and $\rho(tp')$ =*cSubject* or $\rho(tp')$ =*cObject*. This is based on the fact that X *rdf:type :c* restricts X to match only instances of type *:c*, however the same restriction is already in place when we match X over relational attribute $\alpha(tp'.sub)$ of *cSubject* or $\alpha(tp'.obj)$ of *cObject*.

In our sample query, `?w rdf:type :Workflow` should be eliminated, because ρ(`?w rdf:type :Workflow`) = *Workflow* and ρ(`?w ?p ?o`) = *WorkflowSubject*.

Finally, we are ready to present our schema-independent basic graph pattern translation algorithm BGPtoSQL in Figure 8. BGPtoSQL takes bgp, ρ, and α and outputs an equivalent SQL query to evaluate bgp against the relational RDF database. Its main idea is to retrieve all possible variable instantiations from relations that correspond (based on ρ) to triple patterns in bgp, restricting (1) relational attributes that correspond (based on α) to the same variables in different triple patterns to match the same values and (2) relational attributes that correspond (based on α) to instances or literals to match the values of those instances or literals, respectively.

For our optimized query with the basic graph pattern consisting of `?w ?p ?o` and `?w :inputParameter ?x`, the SQL `From` clause contains two relations (lines 05-06 in the algorithm) *WorkflowSubject* and *inputParameter* with aliases $t1$ and $t2$, respectively. The hash h (lines 07-09) contains only one relational attribute for every variable, except for $?w$, $h(?w) = \{t1.i, t2.s\}$, and therefore, the SQL `Where` clause should ensure their equality $t1.i = t2.s$ (lines 10-12). The bgp contains one instance (*:inputPara-*

```
01 Algorithm Calculate-$\rho$-$\alpha$
02 Input: $tp$, $\tau$
03 Output: $\rho$, $\alpha$
04 Begin
05   If $tp.pre$ = rdf:type and $tp.obj$ is an ontology class $c$ then
06     /*Case 1*/ $\rho(tp) = c$, $\alpha(tp, sub) = i$, $\alpha(tp, pre)$ =undef, $\alpha(tp, obj)$ =undef
07   Else
08     Switch [$min(|\tau(tp.sub)Subject|, |\tau(tp.obj)Object|, |\tau(tp.pre)|, |Triple|)$]
09       Case 2 $|\tau(tp.sub)Subject|$: $\rho(tp)$=$\tau(tp.sub)Subject$, $\alpha(tp, sub)$=$i$, $\alpha(tp, pre)$=$p$, $\alpha(tp, obj)$=$o$
10       Case 3 $|\tau(tp.obj)Object|$: $\rho(tp)$=$\tau(tp.obj)Object$, $\alpha(tp, sub)$=$s$, $\alpha(tp, pre)$=$p$, $\alpha(tp, obj)$=$i$
11       Case 4 $|\tau(tp.pre)|$: $\rho(tp)$=$\tau(tp.pre)$, $\alpha(tp, sub)$=$s$, $\alpha(tp, pre)$=undef, $\alpha(tp, obj)$=$o$
12       Case 5 $|Triple|$: $\rho(tp)$=Triple, $\alpha(tp, sub)$=$s$, $\alpha(tp, pre)$=$p$, $\alpha(tp, obj)$=$o$
13     End Switch
14   End If
15   Return $\rho$, $\alpha$
16 End Algorithm
```

Figure 7. Algorithm Calculate-ρ-α

```
01 Algorithm BGPtoSQL
02 Input: $bgp$, $\rho$- and $\alpha$- mappings
03 Output: SQL
04 Begin
05       Assign a unique alias $a_i$ to each $tp_i \in bgp$
06       Construct the SQL From clause: for each $tp_i \in bgp$, $from$ += "\$$\rho(tp_i)$ \$$a_i$",
07       Construct an inverted index (hash) $h$ on variables in $bgp$:
08             for each $tp_i \in bgp$, for each variable $?v \in tp_i$, $h(?v) \cup =$ "\$$a_i$.\$$\alpha(tp_i, pos(?v))$",
09             where $pos(?v) \in \{sub, pre, obj\}$ encodes the position of $?v$ in $tp_i$
10       Construct the SQL Where clause:
11             for each distinct variable $?v \in bgp$ and $|h(?v)| > 1$,
12                   let $x \in h(?v)$, for each $y \in h(?v)$ and $y \neq x$, $where$ += "\$$x$ = \$$y$ And"
13             for each $tp_i \in bgp$, for each literal or instance $l \in tp_i$ and $\alpha(tp_i, pos(l)) \neq undef$,
14                   $where$ += "\$$a_i$.\$$\alpha(tp_i, pos(l))$ = '\$$l$' And"
15       Construct the SQL Select clause:
16             for each distinct variable $?v \in bgp$, let $?v \in tp_i$, $select$ += "\$$a_i$.\$$\alpha(tp_i, pos(?v))$ As $v$,"
17       Return "Select $select$ From $from$ Where $where$"
18 End Algorithm
```

Figure 8. Algorithm BGPtoSQL

meter), however its α value is undefined (lines 13-14). The SQL `Select` clause projects each distinct variable in *bgp* (lines 15-16), resulting in the translated query (line 17):

```
Select t1.i As w, t1.p As p, t1.o As
o, t2.o As x From WorkflowSubject t1,
inputParameter t2 Where t1.i=t2.s
```

Finally, the SQL equivalent of our SPARQL query is

```
Select w, p, o From (
Select t1.i As w, t1.p As p, t1.o As o,
t2.o As x From WorkflowSubject t1,
inputParameter t2 Where t1.i=t2.s ) t3
```

Note that our system supports the translation and evaluation of arbitrary complex optional graph patterns [8], group patterns, value constraints, which is not presented here due to space limit.

5.2 Experimental study

A wide range of scientific queries can be answered based on our provenance ontology model enhanced with reasoning capabilities. In [7], we presented several such queries that our e-scientist collaborators are interested in. The SPARQL-to-SQL translation showed to be very efficient, returning SQL equivalents of the sample provenance queries in less than 0.01s. The "cold" run times of the generated SQL queries for our two database schemas are reported in Figure 9. Both schemas showed to be efficient to support provenance queries. With the growth of the database size, the SchemaMapping-T schema showed better scalability and substantially outperformed the SchemaMapping-V schema for most queries as all views are materialized. "Warm" runs of the queries (second time, third time, and so on) showed nearly 0.000s time for most queries.

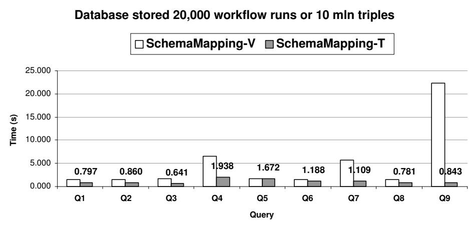


Figure 9. Performance of provenance queries

6 Conclusions and Future Work

In this work, we designed an RDBMS-based provenance management system for storing and querying scientific workflow provenance metadata. Our approach seamlessly integrates the interoperability, extensibility, and reasoning advantages of Semantic Web technologies with the storage and querying power of an RDBMS. Our schema mapping, data mapping, and SPARQL-to-SQL query translation algorithms are optimized to efficiently support common provenance queries, incremental data loading that employs the ordering of inserting various provenance metadata, and schema-independent query translation that is optimized on-the-fly by using the type information of an instance and the statistics of the sizes of the tables in the database. Experimental results showed that our algorithms are efficient and scalable.

In future, we will continue to seek for further optimizations of database schema design, data loading, and querying, with the main focus on ontology-based query optimization.

References

[1] R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In *VLDB*, pages 149–158, 2001.

[2] S. Alexaki, V. Christophides, G. Karvounarakis, and D. Plexousakis. On storing voluminous RDF descriptions: The case of Web portal catalogs. In *WebDB*, 2001.

[3] I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the Kepler scientific workflow system. In *IPAW*, pages 118–132, 2006.

[4] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.

[5] S. Bowers, T. M. McPhillips, B. Ludäscher, S. Cohen, and S. B. Davidson. A model for user-oriented data provenance in pipelined scientific workflows. In *IPAW*, 2006.

[6] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying RDF and RDF Schema. In *ISWC*, 2002.

[7] A. Chebotko, X. Fei, C. Lin, S. Lu, and F. Fotouhi. Scientific workflow provenance metadata management using an RDBMS. Tech. Rep. TR-DB-052007-CFLLF. May 2007. `http://www.cs.wayne.edu/~artem/main/research/TR-DB-052007-CFLLF.pdf`.

[8] A. Chebotko, S. Lu, H. M. Jamil, and F. Fotouhi. Semantics preserving SPARQL-to-SQL query translation for optional graph patterns. Tech. Rep. TR-DB-052006-CLJF. May 2006. `http://www.cs.wayne.edu/~artem/main/research/TR-DB-052006-CLJF.pdf`.

[9] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with Triana services. *Concurrency and Computation: Practice and Experience*, 18(10):1021–1037, 2006.

[10] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming Journal*, 13(3):219–237, 2005.

[11] I. Foster, J. Vöckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM*, 2002.

[12] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, 2006.

[13] P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. Moreau. Recording and using provenance in a protein compressibility experiment. In *International Symposium on High Performance Distributed Computing (HPDC'05)*, 2005.

[14] S. Harris and N. Shadbolt. SPARQL query processing with conventional relational database systems. In *SSWS*, 2005.

[15] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[16] S. Miles, P. Groth, M. Branco, and L. Moreau. The requirements of recording and using provenance in e-science experiments. *Journal of Grid Computing*, 2006.

[17] T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A.Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[18] Z. Pan and J. Heflin. DLDB: Extending relational databases to support Semantic Web queries. In *PSSS*, pages 109–113, 2003.

[19] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, 2005.

[20] Y. Simmhan, B. Plale, and D. Gannon. A framework for collecting provenance in data-centric scientific workflows. In *ICWS*, pages 427–436, 2006.

[21] Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of RDF/S stores. In *ISWC*, pages 685–701, 2005.

[22] R. Volz, D. Oberle, B. Motik, and S. Staab. KAON SERVER - A Semantic Web management system. In *WWW, Alternate Tracks - Practice and Experience*, 2003.

[23] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF storage and retrieval in Jena2. In *SWDB*, 2003.

[24] J. Zhao, C. Wroe, C. A. Goble, R. Stevens, D. Quan, and R. M. Greenwood. Using Semantic Web technologies for representing e-science provenance. In *ISWC*, 2004.

[25] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, I. Raicu, T. Stef-Praun, and M. Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *Int'l Workshop on Scientific Workflows (SWF), IEEE SCC Workshops (SCW)*, 2007.

[26] Z. Zhao, A. Belloum, C. de Laat, P. Adriaans, and B. Hertzberger. Distributed execution of aggregated multi domain workflows using an agent framework. In *Int'l Workshop on Scientific Workflows (SWF), IEEE SCC Workshops (SCW)*, 2007.