Provenance Algebra and Materialized View-based Provenance Management

Satya S. Sahoo[1,2], Roger S. Barga[2], Jonathan Goldstein[2], Amit P. Sheth[1]
[1]Kno.e.sis Center, CSE Dept., Wright State University, Dayton, OH 45435; [2]Microsoft Research, One Microsoft Way, Redmond, WA 98052
{sahoo.2, amit.sheth}@wright.edu, {barga,jongold}@microsoft.com

ABSTRACT

Provenance, from the French word 'provenir' meaning "to come from", describes the lineage of an entity. Provenance is critical information in eScience to accurately interpret scientific results. Though information provenance has been recognized as a hard problem in computing science (British Computing Society, 2004), many fundamental research issues in provenance have yet to be addressed.

A common provenance model with well-defined formal semantics to facilitate interoperability of provenance metadata from different sources has not been defined. Another important issue is the lack of a systematic study of provenance query characteristics across multiple applications. A classification or taxonomy of the provenance queries will not only help to better understand provenance metadata, but will also enable the definition of provenance query operators. Finally, while provenance for a user or an application is a specific view over all available provenance metadata, a provenance management system that supports provenance storage as views has not been implemented.

In this paper we propose a novel provenance algebra consisting of a common provenance model called *provenir*, defined in description logic based W3C Web Ontology Language (OWL-DL), along with a set of provenance query operators derived from the classification of provenance queries. We also introduce a practical provenance storage solution using materialized views over a generic relational database system. Our approach takes advantage of provenance query operators and well-defined indices to efficiently process complex provenance queries over very large datasets. To support our claims we present an evaluation of both performance and scalability aspects of our initial implementation. To the best of our knowledge this is the first provenance management system that supports the complete process from a formal provenance model and query operators to storage and efficient queries over provenance data.

1. INTRODUCTION

The eScience paradigm is enabling scientists in multiple domains to leverage distributed to achieve their objectives faster, more efficiently and on an industrial scale. The eScience application domains range from biology [1], oceanography [2] to astronomy [3], and include distributed resources such as remote sensors, computational tools, and data repositories. Provenance metadata, from the French word provenir meaning "to come from" (Wikipedia, retrieved on 06/24/08) represents the lineage or historical information about a piece of data. Provenance is critical information to accurately interpret scientific results, validate experimental processes, associate trust value, and verify quality of data. Provenance has been studied from multiple perspectives in computer science, such as database provenance [4], [5] [6], and scientific workflow provenance [7] [8], but many fundamental research issues have yet to be addressed.

eScience provenance metadata is typically generated in a distributed environment where each source may represent provenance differently. Thus, a common model is required to represent workflow provenance, database provenance, as well as domain-specific details in an integrated manner. Further, the scale of provenance metadata generated in high-throughput eScience experiments precludes manual interpretation and requires processing by software applications. Hence, a common provenance model should also allow both consistent interpretation and reasoning using entailment rules by software applications. The description logic based Web Ontology Language (OWL-DL) [9] represents the most expressive but decidable sub-language of the World Wide Web (W3C) recommended OWL standard. We propose a common semantic model of provenance called *provenir* defined in OWL-DL.

Many different examples of provenance queries addressing specific application requirements have been used in literature. But to date, there are no studies to identify common and distinct features of these queries. A classification or taxonomy of the provenance queries will not only help to better understand provenance metadata characteristics, but will also enable the definition of operators to support such queries. In this paper, we propose a classification scheme for provenance queries and use the classification to define query operators. We also define formal semantics for each operator. Together, the common provenance model and query operators form the provenance algebra to underpin provenance management system.

Provenance for a user or an application is a view over all available provenance metadata, but provenance views are complex and hence very expensive to compute repeatedly in response to queries. Further, provenance metadata is dynamic and changes to reflect modifications in the experimental environment, hence the views should support efficient view maintenance. To address these requirements, we propose the use of materialized views for storage management of provenance.

Contributions

The following are the contributions of this paper:

- A common provenance model called *provenir* ontology defined using the OWL-DL language. Provenir includes provenance classes and explicitly models the named relations between them. Modeling relations as first class entities enables the provenir ontology to capture provenance details that are closer to real world eScience experiments. The satisfiability of provenir is also discussed.

- A classification scheme for provenance queries in eScience is proposed for the first time, based on the classification a set of provenance query operators are defined. The query operators are defined in terms of the provenir ontology and mapped to existing work in both database and workflow provenance.
- A practical provenance storage solution is implemented on a commercial relational database system using a materialized views-based approach. This approach demonstrates that a provenance management system using a relational database system is feasible for complex queries over large datasets through implementation of well-defined provenance query operators and using materialized views. Our initial prototype implementation is evaluated for performance and scalability.

In the following section, we introduce a scientific workflow with additional simulated details in the oceanography domain from the Neptune project [2].

1.1 Oceanography eScience Scenario

The Neptune project, led by the University of Washington, [2] is an ongoing initiative to create network of instruments widely distributed across, above, and below the seafloor in the northeast Pacific Ocean. We consider a simulated scenario, illustrated in Figure 1.1, involving collection of data by ocean buoys (containing a temperature sensor and an ocean current sensor) which is then sent as input to a scientific workflow for creation of a visualization chart as output. We consider the following two scenarios and the associated provenance queries:

1. If an ocean buoy is found to be damaged through contamination with sea water, all visualization charts generated using data from the two sensors within this ocean buoy should be discarded. To accurately identify visualization charts that made use of data from either one or both the sensors in the damaged ocean buoy requires the analysis of provenance metadata.

   This category of provenance query uses provenance metadata beyond the perspective of a scientific workflow. The required provenance to answer this query includes domain-specific contextual information such as the explicit relationship "*contained in*" between the two types of sensors and the ocean buoy. This explicit modeling of relations or properties (we use relation and property interchangeably in this paper) with formal definition is one of the primary characteristics of both the Resource Description Framework (RDF) [10] and Web Ontology Language (OWL) standards.

2. Another typical provenance query involves the retrieval of all workflow-specific provenance metadata. For example, "given a "HyperCube" object with identifier "HyperCube85357162234026" (illustrated in Figure 1.1), retrieve all its provenance metadata. This query involves the traversal of the workflow links between computational tasks such as instances of "HyperCubeSchemaGeneratorActivity", input and output data, such as "NCFileName", that were involved in the creation of the given hypercube object. Though many current provenance systems can answer such queries, the scalability or performance of these systems has not been evaluated.

In the Experiments and Results Section (Section 5), we use the provenance data generated for this simulated oceanography scenario and the two provenance queries to evaluate our implementation.

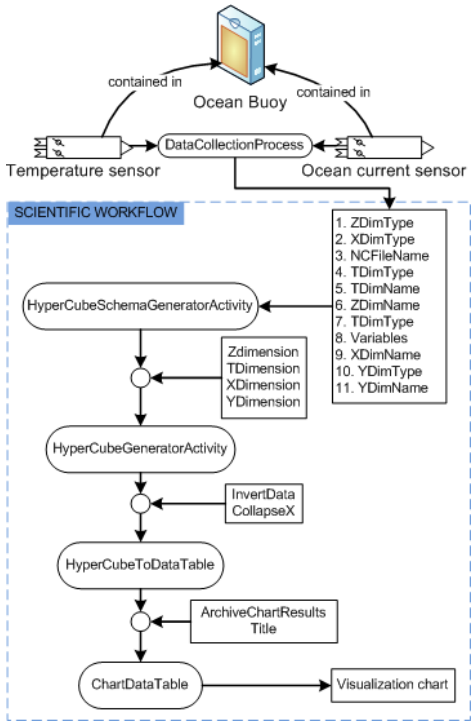

Figure 1.1: A simulated oceanography scenario from Neptune Project with data from sensors used to create chart visualizations

1.2 Outline of paper

In Section 2, we introduce the common provenance model called provenir, define its structure, and discuss its satisfiability. In Section 3, we present a taxonomy of provenance queries and define a set of provenance query operators. In Section 4, we introduce the use of indexed materialized views for storage of provenance. We discuss performance and scalability evaluations of our initial implementation in Section 5 which demonstrate the efficiency of our approach. We correlate the provenance query operators to existing provenance work in Section 6 and present closing remarks in Section 7.

2. SEMANTIC MODEL OF PROVENANCE

The provenir provenance model is defined in OWL-DL to formally represent the fundamental provenance classes and properties. An OWL-DL ontology consists of a set of classes $\{C_1, C_2, \ldots, C_n\}$, a set of properties $\{R_1, R_2, \ldots, R_n\}$, and a set of individuals $\{I_1, I_2, \ldots, I_n\}$ [11]. We use an abstract syntax specified in a version of Extended BNF [11] to construct axioms defining the classes and properties of provenir ontology.

The provenir ontology forms the core component of a modular approach for our eScience provenance framework. As illustrated by the first query in Section 1.1, domain-specific details are an important component of provenance metadata. But, a single monolithic provenance ontology that models all possible details from different domains is clearly not a feasible solution. Hence, our proposed modular provenance framework involves integrated

use of multiple ontologies, each modeling specific provenance metadata for a particular domain (for example, ProPreO ontology represents proteomics domain-specific provenance [12]). These multiple ontologies will use the provenir ontology as the common reference model, hence making it easier to interoperate with each other. This modular framework represents a scalable, flexible and maintainable approach that can be adapted to the specific requirement of different domains. In the next two Sections, we describe the classes and the properties in the provenir ontology (Figure 2.1).

2.1 Classes

In OWL, classes represent individuals with a set of common characteristics. To represent provenance metadata classes we use the two well defined, primitive concepts of "occurrent" and "continuant" from philosophical ontology [13]. Continuant is defined as "… *entities which endure, or continue to exist, through time while undergoing different sorts of changes, including changes of place*" [13]. Occurrent is defined as *"…entities that unfold themselves in successive temporal phases*" [13].

We define three base classes[1] in the provenir ontology representing the primary components of provenance, that is, "`data`", "`agent`" and "`process`".[2] The two base classes, "`data`" and "`agents`" are defined as specialization (sub-class) of continuant class. The third base class "`process`" is a synonym of occurrent. We present the axiomatic definition of each class:

1. **`data`**: This class models continuant entities that represent the starting material, intermediate material, end products of a scientific experiment, and parameters that affect the execution of a scientific process. `Data` inherit the properties of continuants such as enduring or existing while undergoing changes.

Axiom 1:
data::=' owl:Class', rdfs:subClassOf ('continuant')';

2. **`process`**: This class models the occurrent entities that affect (process, modify, create, delete among other dynamic activities) individuals of `data`.

Axiom 2:
process ::=' owl:Class', 'owl:equivalentClass('occurrent')';

3. **`agent`**: This class models the continuant entities that causally affect the individuals of `process`.

Axiom 3:
agent ::='owl:Class', 'rdfs:subClassOf ('continuant')';

In addition to the three base classes, five sub-classes (two direct and three indirect) of `data` are defined to help define properties in provenir ontology (defined in Section 2.2). The subclasses of `data` are:

a) **`data_collection`**: This class represents atomic or composite data entities that are acted upon during a scientific process

Axiom 4:
data_collection ::= ' owl:Class', 'rdfs:subClassOf (data)';

b) **`parameter`**: `parameter` is a class of individuals that affect the behavior of scientific process in the form of constraints and control input to `agent` and `process` classes.

Axiom 5:
parameter ::= ' owl:Class', 'rdfs:subClassOf (data)';

There are three subclasses of `parameter` defined along the three dimensions of spatial, temporal and thematic (domain-specific):

`temporal_parameter`: This class captures the temporal details associated with individuals of `data_collection` class (for example, the timestamp associated with a sensor reading), `process` (for example, the duration of a protein analysis process), and `agent` (for example, the time period during which a sensor was working correctly).

Axiom 6:
temporal_parameter ::= owl:Class', rdfs:subClassOf(parameter)';

`spatial_parameter`: The spatial metadata associated with individuals of `process` or `agent` or `data_collection` classes is represented by this class. For example, the geographical location of an ocean buoy is a spatial parameter.

Axiom 7:
spatial_parameter = ' owl:Class', 'rdfs:subClassOf (parameter)';

`domain_parameter`: One of the primary objectives of provenir ontology is to be extended to model different domain-specific provenance. The `domain_parameter` class will be used to model domain-specific parameters.

Axiom 8:
domain_parameter ::= ' owl:Class', 'rdfs:subClassOf (parameter)';

2.2 Properties

The explicit modeling of property as first class entities is an important characteristic of Semantic Web modeling languages. For example, the RDF data model [10] is defined by a triple <*subject*, *predicate*, *object*>, where the "subject" and "object" are individuals belonging to classes or in case of "object" may be literals (for example, a string value), and the "predicate" represents the property that links together the subject and object. In this Section, we define a set of foundational properties in the provenir ontology.

Instead of defining a new set of properties, we reuse and adapt properties defined in the Relation ontology (RO) from the Open Biomedical Ontologies (OBO) Foundry, which is part of the US National Institutes of Health (NIH) funded National Center for Biomedical Ontologies (NCBO). The OBO foundry has the stated

[1] The following namespaces are used in the text – (a) owl: http://www.w3.org/2002/07/owl#, (b) rdfs: http://www.w3.org/2000/01/rdf-schema#, (c) rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#

[2] We use the `courier` font to denote provenir ontology classes, relationships and individuals

goal of "establishing a set of principles for ontology development" and "interoperable reference ontologies".

The RO defines a set of ten primitive properties with clearly defined "domain" (classes of entities that can be "subject" of the property) and "range" (classes of entities that can be "object" of the property) values. We describe provenir ontology using model-theoretic semantics of OWL.

1. **ro[3]:part_of** – This property is defined for each of the three base classes of provenir ontology. The restriction for this relation is that the domain and range values belong to the same class and do not overlap. For example, if `data` is defined as the domain/range of the properties, the corresponding range/domain is also `data`. This restriction is derived from extending the notion of continuants and occurrents as non-overlapping categories [13] to the three base classes.

As defined in the RO [13], this property satisfies the standard axioms of mereology, that is, reflexivity, anti-symmetry, and transitivity.

Axiom 9:
ro:part_of::='owl:ObjectProperty',
'('rdfs:domain (data)','rdfs:range(data)')' |
'('rdfs:domain (agent)','rdfs:range(agent)')' |
'('rdfs:domain (process)','rdfs:range(process)')';

2. **ro:contained_in** – In provenir ontology, the `ro:contained_in` is defined with similar constraints as `ro:part_of`, that is, the domain and range values belong to same class and do not overlap. The property is defined for `data` and `agent` classes. Consistent with its definition in RO, the property is also defined to be non-transitive.

Axiom 10:
ro:contained_in::= 'owl:ObjectProperty',
'('rdfs:domain (data)','rdfs:range(data)')'|
'('rdfs:domain (agent)','rdfs:range(agent)')';

3. **ro:adjacent_to** – This property is defined for disjoint continuants in RO. In provenir ontology, it is defined only for `agent` class, where the adjacent spatial location of individuals of `agent` class may have an effect on scientific results. For example, presence of a sensor generating a magnetic field may affect the quality of observations made by another sensor that is adjacent to it.

We note that, similar to the mereotopological relations defined in RO [13] such as partial overlap, tangential proper part etc., corresponding properties can be added to extensions of provenir ontology.

Axiom 11:
ro:adjacent_to::='owl:ObjectProperty',
'rdfs:domain (agent)', 'rdfs:range(agent)';

4. **ro:transformation_of** – RO defines the `ro:transformation_of` as a property between two entities that preserve their identity between the two transformation stages.

[3] ro: represents the namespace for Relation ontology [13]

For example, if there is an individual that belongs to a sub-class of class `data`, say d_1 at time t_1; and the individual also belonged to another sub-class of class `data`, say d_2 at a time t_2 and $t_1 > t_2$. If at no time instant, the individual is member of both d_1 and d_2 then there is a `ro:transformation_of` property linking the two individuals.

Axiom 12:
ro:transformation_of::= 'owl:ObjectProperty',
'rdfs:domain (data)', 'rdfs:range(data)';

5. **ro:derives_from** – This RO property represents the derivation history of data entities as a chain or pathway. Unlike `ro:transformation_of` property which links identical entities, `ro:derives_from` links distinct individuals of `data`.

Axiom 13:
ro:derives_from::='owl:ObjectProperty',
'rdfs:domain (data)','rdfs:range(data)';

6. **ro:preceded_by** – This RO temporal property is defined for distinct individuals of `process` class. Similar to its interpretation in Smith et. al [13], in provenir ontology, more specific types of properties, such as "*immediately_preceded_by*" [13] which have more precise semantics and hence are more useful may be defined in extensions to provenir ontology.

Axiom 14:
ro:preceded_by::= 'owl:ObjectProperty',
'rdfs:domain (process)','rdfs:range(process)';

7. **ro:has_participant** – This is the primary property linking `data` to `process`, where the individual of `data` class participates in a `process`.

Axiom 15:
ro:has_participant::='owl:ObjectProperty',
'rdfs:domain (process)', 'rdfs:range(data)';

8. **ro:has_agent** – This is a causal property that links `agent` to `process` and is directly responsible for the change in state of the `process`. Similar to the description used in Smith et. al [13], the provenir ontology also allows the use of this property to "capture the directionality" of scientific experiments, for example which `agent` caused the activation of a `process`.

Axiom 16:
ro:has_agent::='owl:ObjectProperty',
'rdfs:domain (process)', 'rdfs:range(agent)';

9. **has_parameter** – This property links the spatial, temporal and, domain-specific parameters to an individual of a `data_collection`, `agent`, and `process`.

Axiom 17:
ro:has_parameter::= 'owl:ObjectProperty',
'rdfs:domain(data_collection, process, agent)',
'rdfs:range(parameter)';

Two specialized properties describing the temporal and spatial parameters are defined:

has_temporal_value – This is a specific property to assert temporal value for individuals of `data_collection`, `process`, and `agent` classes. For example, d_1 `has_temporal_value` t_1.

Axiom 18:
ro:has_temporal_parameter::='owl:ObjectProperty',
'rdfs:domain(data_collection, process, agent)',
'rdfs:range(temporal_parameter)';

ro:located_in – An instance of `data` or `agent` is associated with exactly one spatial region that is its exact location at given instance of time. In provenir ontology, this relation has two domain class `agent` and `data_collection` classes and has `spatial_parameter` as range class.

Axiom 19:
ro:located_in::='owl:ObjectProperty',
'rdfs:domain(data_collection, process, agent)',
'rdfs:range(spatial_parameter)';

2.3 Satisfiability

The provenir ontology, consistent with description logic, differentiates between the schema (also called as *Terminological* or *TBox* in description logic) and the provenance metadata instance values (also called as *Assertional* or *ABox* in description logic). In this Section, we discuss the satisfiability of the provenir ontology schema.

Using the satisfiability proving technique described in Lausen et. al [14], let $\mathcal{P}$ denote a provenance vocabulary and let $\mathcal{M}$ denote asset of constrained provenance assertions. Recall that in OWL 1.0, the property characteristics are transitivity, symmetric, functional, inverse, and inverse functional [15]. We say $\mathcal{P}_{\mathcal{M}} = (\mathcal{P}, \mathcal{M})$ to be provenance assertion constrained by the restrictions defined in provenir ontology such as class hierarchy, property hierarchy, domain and range values for properties etc. We describe the satisfiability of these constrained provenance assertions. We say that $\mathcal{P}_{\mathcal{M}}$ is satisfiable iff there exists an interpretation $\mathcal{I}$ of $\mathcal{P}$ which satisfies $\mathcal{M}$. We also define for class $\mathcal{E} \in \mathcal{P}$, $\mathcal{E}^{\mathcal{I}} \neq \emptyset$ and for property $\mathcal{R} \in \mathcal{P}$, $\mathcal{R}^{\mathcal{I}} \neq \emptyset$.

The satisfiability of constrained provenance assertions subscribing to provenir ontology is trivial, that is, it is always guaranteed. To illustrate, let $o \in \varphi_I$, and define $\mathcal{E}^{\mathcal{I}} = \{o\}$; and let $v_d, v_r \in \varphi_I$ and $v_d \in \text{domain}(\mathcal{R})$, $v_r \in \text{range}(\mathcal{R})$, and define $\mathcal{R}^{\mathcal{I}} = \{(v_d, v_r)\}$ for all provenance assertions.

LEMMA 1 Let $\mathcal{P}$ be a provenance vocabulary, $\mathcal{M}$ be provenance metadata assertions with constraints defined in provenir ontology. There exists an interpretation $\mathcal{I}$ of $\mathcal{P}$, such that $\mathcal{I} \models \mathcal{M}$.

The description logic (DL) expressivity of provenir ontology is $\mathcal{ALCH}$ and we know from [16] that for DL of expressivity $\mathcal{ALCHIQ}$, the satisfiability can be decided in exponential time. Hence, given that provenir ontology does not feature inverse roles and number restrictions, the complexity bound for provenance metadata assertions subscribing to provenir ontology is expected to be exponential or better. We plan to explore the complexity of algorithms for various DL inference problems over provenir ontology in more detail in the future.

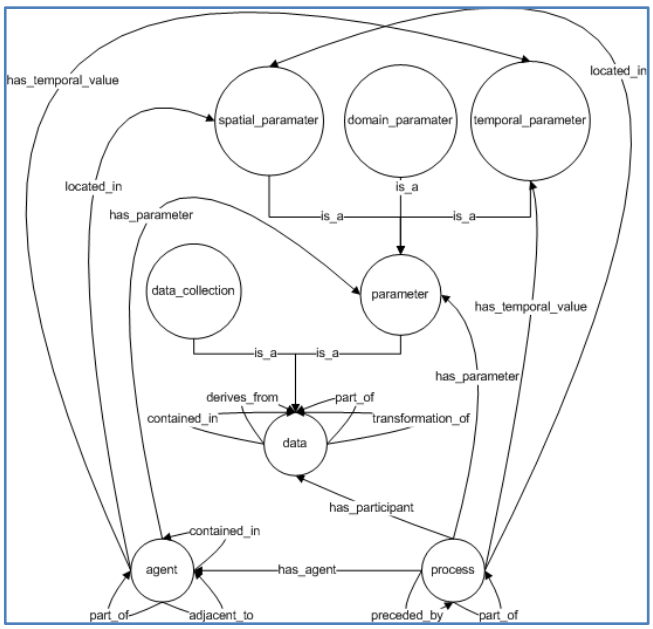

Figure 2.1: The provenir ontology schema

3. PROVENANCE QUERY CLASSIFICATION AND QUERY OPERATORS

The provenance literature features a large variety of queries, each addressing the specific requirements of an application under discussion. But without a systematic classification of provenance queries it is difficult to clearly identify the common and distinct characteristics of these queries, and more importantly, define query operators to support them. The provenance query operators can be implemented by provenance storage systems to more efficiently answer queries with query optimization techniques similar to generic database systems.

In this paper, we identify three categories of provenance queries:

1. **Querying for provenance metadata**: Given a data entity, this category of queries returns the complete set of provenance information that influenced the current state of the data entity. An example query, from the oceanography scenario described Section 1.1, is to find the provenance metadata for the *"HyperCube"* object with identifier *"HyperCube85357162234026"*;

2. **Querying for data values**: A diametrically opposite perspective to the first category of query is, given a set of constraints defined over both provenance metadata and data, expressed using formal context structure [17], retrieve a set of data entities satisfying the constraints. An example query from the oceanography scenario (described in Section 1.1) is to find chart visualization datasets from the ocean buoy with identifier "*oceanBuoy7044*" within the time period "*April 21, 2003 to May 2, 2003*";

   Depending on the quality of constraints specified by the context structure, this query operator can be used to implicitly reconcile heterogeneity among resultant datasets. For example, if the query constraint specifies data from temperature sensors with unit of measurement as degree Centigrade, the resulting data values are comparable and

arithmetic operations to compute average or sum can be applied to them without the need for semantic reconciliation;

3. **Modifying provenance metadata**: This category of queries is defined over the provenance metadata itself. Example operations include merging of provenance from different stages of an experiment and comparison of provenance for two datasets from different sources.

Using this query classification scheme, we introduce a set of provenance query operators.

Conventions for query operator definition

We define the provenance query operators in terms of the provenir ontology class and properties (introduced in Section 2) and use denotational semantics to define the formal semantics of each operator. We define the universal set of provenance metadata assertions "*PM*" in RDF, which is a directed edge and node labeled graph structure. If 'p' is an individual of provenir class `process` it is represented as p ∈ `process`. Corresponding to RDF notation, *a→x→b* represent that '*a*' is the subject and '*b*' is the object of the property '*x*'.

Definition 1:
PM = graph (N, E)
where,
- *N* ∈ *provenir*:`data` ∪ *provenir*:`process` ∪ *provenir*:`agent`
- *E ∈ R, R is set of properties defined in provenir ontology*

3.1 Querying for Provenance Metadata

provenance ( )- This is a closure operation on the provenance graph. That is, the operator returns the complete set of provenance metadata for a data entity. Let us again consider the query from Section 1.1, "*given a "HyperCube" object with identifier "HyperCube85357162234026", retrieve all its provenance metadata*". As Figure 3.1 illustrates, the query operator with input value *HyperCube85357162234026* expands the provenance metadata graph in a series of steps. The graph expansion steps are bounded by constraints defined in terms of valid classes and properties that can be traversed by the query operator. Formally,

Definition 2:
∀dc ∈ *provenir*: `data_collection` *provenance* (dc) = p_data (dc) ∪ p_process (dc) ∪ p_agent (dc) ∪ p_predicate (dc), holds ∈ DB
- p_process (dc) = {p | p→*ro*:`has_participant`→dc} ∪ {p | q→*ro*:`preceded_by`→p ∧ q ∈ p_process (dc)} ∪ {p | q→*ro*:`part_of`→p ∧ q ∈ p_process (dc)} ∪ {p | p→*ro*:`part_of`→q ∧ q ∈ p_process (dc)} ∪ {p | q→*ro*:`has_participant`→d ∧ d ∈ p_data (dc)} ∪ {p | p→*ro*:`has_agent`→a ∧ a∈ p_agent (dc)}.
- p_data (dc) = {d | p ∈ p_process (dc) ∧ p→*ro*:`has_participant`→d ∧ timestamp (d) ≤ timestamp (dc)} ∪ {d | dc→*ro*:`derives_from`→d} ∪ {d | dc→*ro*:`transformation_of`→d} ∪ {d | dc→*ro*:`part_of`→d} ∪ {d | d→*ro*:`part_of`→dc} ∪ {d | dc→*ro*:`contained_in`→d} ∪ {d | d→*ro*:`contained_in`→dc} ∪ {d | p ∈ p_process (dc) ∧ p→*ro*:`has_parameter`→d} ∪ {d | a ∈ p_agent (dc) ∧ a→*ro*:`has_parameter`→d}.
- p_agent (dc) = {a | p ∈ p_process (dc) ∧ p→*ro*:`has_agent`→a} ∪ {a | b ∈ p_agent (dc) ∧ b→*ro*:`part_of`→a} ∪ {a | b ∈ p_agent (dc) ∧ b→*ro*:`contained_in`→a} ∪ {a | b ∈ p_agent (dc) ∧ b→*ro*:`adjacent_to`→a} ∪ {a | b ∈ p_agent (dc) ∧ a→*ro*:`part_of`→b} ∪ {a | b ∈ p_agent (dc) ∧ a→*ro*:`contained_in`→b} ∪ {a | b ∈ p_agent (dc) ∧ a→*ro*:`adjacent_to`→b}.
- p_predicate (dc) = {r | X ∈ provenance (dc) ∧ dc→*r*→X ∧ X→*r*→dc}.

This query operator is expensive to implement and is not required in most scenarios. For example, provenance metadata generated by a scientific workflow represents a subset of result returned by this operator. Hence, we define a specialized form of the ***provenance ( )*** operator with additional constraints, called ***provenance_pathway ( )*** to retrieve workflow-specific provenance metadata.

I) ***provenance_pathway ( )***: This operator returns provenance information of a data entity consisting of only an individual belonging to `data_collection` and `process` classes. This operator reflects requirements of workflow provenance, where provenance consists of a series of "activities" (where an "activity" is any sub-class of `process`) and set of "data entities" (where a "data entity" is any subclass of `data_collection` class) that form the input and output of the processes. A set of constraints defined as an expression over individuals belonging to `data_collection`, `agent`, `process` and `parameter` can also be defined as part of the input. For example, query involving workflow provenance of ocean current values may have a time interval constraint as April 21, 2003 to May 2, 2003.

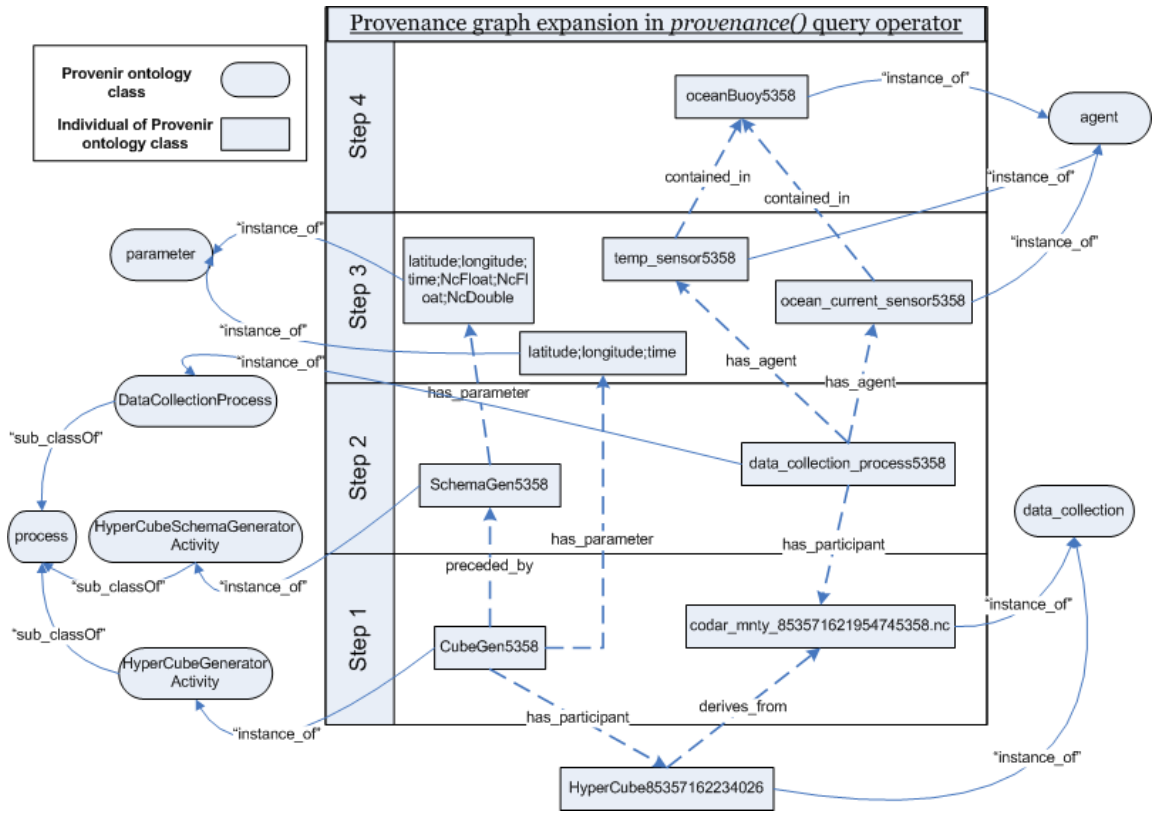

Figure 3.1: Provenance graph expansion steps, for input value "*HyperCube85357162234026*" to *provenance ()* query operator

Definition 3:
∀dc ∈ *provenir*: data_collection *provenance_pathway* (dc) = p_process (dc) ∪ p_data (dc) ∪ p_predicate (dc), holds ∈ DB

- p_process (dc) = {p | p→*ro*:has_participant→dc} ∪ {p | p→*ro*:has_participant→d ∧ d ∈ p_data(dc)} ∪ {p | p→*ro*:preceded_by→q ∧ q ∈ p_process(dc)}
- p_data (dc) = {d | p ∈ p_process(dc) ∧ p→*ro*:has_participant→d ∧ timestamp (d) ≤ timestamp(dc)} ∪ {d | dc→*ro*:derives_from→d} ∪ {d | dc→*ro*:transformation_of→d} ∪ {d | p ∈ p_process(dc) ∧ p→*ro*:has_parameter→d}
- p_predicate (dc) = {r | X ∈ provenance_pathway(dc) ∧ dc→r→X ∧ X→r→dc}

Correlation to "where-provenance" (defined in Buneman et. al [5]):

A restricted view of the "where provenance" is defined as, for each database query that returns a set of results, each piece of input data that contributes to given element of the result set is identified. This is termed as the "derivation basis" of the output value.

The semantics of the ***provenance ()*** is comparable to "derivation basis". Given the successful execution of a scientific process, where all components may be modeled as a workflow or may also include non-computational entities, the "where-provenance" of the data includes all the data entities and processes that contributed to the output data. The result of the ***provenance ()*** operator is a superset of the domain of the "where-provenance" defined in [5] in terms of not only databases involved in the scientific processes but also process and other data entities.

3.2 Querying for Data Value

provenance_context ( ) - This operator has an inverse-role to the ***provenance ( )*** operator, with more relaxed constraints that may be specified as input values.

Definition 4:
provenance_context (d_i, p_j, a_k, r_l, pa_m) ≡ {d_{res}}
where,

- d_i, d_{res} ∈ *provenir*:data_collection;
- p_j ∈ *provenir*:process;
- a_k ∈ *provenir*:agent;
- pa_m ∈ *provenir*:parameter;
- r_l ∈ R; and r_l R is set of provenir ontology properties linking individuals of z, for z ∈ {d_{res} ∪ d_i ∪ p_j ∪ a_k ∪ pa_m}

This operator uses any component of the provenance metadata graph *PM* to define a context structured subgraph. Considering our example query from the oceanography scenario (described in

Section 1.1), "*Find all datasets sourced from ocean buoy with identifier "oceanBuoy7044" that was in damaged state for the time period "April 21, 2003 to May 2, 2003*". Thus, given a set of the output of the operator is a set of individuals belonging to `data_collection` class that satisfy the constraints. This operator is closed with respect to `data_collection` class.

Similar to the ***provenance_context ( )*** operator that returns individuals of class `data_collection`, specialized operators can be defined for individuals of `process` and `agent` classes.

I) ***pc_process ()***: For given a set of constraints retrieve all the instances of process class that satisfy the set of constraints. An example query from the oceanography scenario (in Section 1.1) is "*Find all invocations of a computational process "HyperCubetoDataTable" with parameter "InverseData" set to value = false*".

Definition 5:
pc_process $(d_i, p_j, a_k, r_l, pa_m) \equiv \{p_{res}\}$
where,
- $d_i \in$ *provenir*:`data_collection`;
- $p_{res}, p_j \in$ *provenir*:`process`;
- $a_k \in$ *provenir*:`agent`;
- $pa_m \in$ *provenir*:`parameter`;
- $r_l \in R$; and R is set of provenir ontology properties linking individuals of z, for $z \in \{d_{res} \cup d_i \cup p_j \cup a_k \cup pa_m\}$

II) ***pc_agent ()***: For given a set of constraints retrieve all the instances of `agent` class that satisfy the constraints.

Definition 6:
pc_agent $(d_i, p_j, a_k, r_l, pa_m) \equiv \{a_{res} \mid a_{res} \in$ *provenir*:`agent`$\}$
where,
- $d_i \in$ *provenir*:`data_collection`;
- $p_j \in$ *provenir*:`process`;
- $a_k \in$ *provenir*:`agent`;
- $pa_m \in$ *provenir*:`parameter`;
- $r_l \in R$; and R is set of provenir ontology properties linking individuals of z, for $z \in \{d_{res} \cup d_i \cup p_j \cup a_k \cup pa_m\}$

A more detailed specification of these three query operators is given as **Appendix B** at the end of the paper.

3.3 Modifying Provenance Metadata Graph

In this Section we define a third category of operators to manipulate the provenance metadata graph itself. We note that RDF graph representing provenance metadata subscribes to provenir ontology (with a 1-1 mapping) hence it does not contain any blank nodes (a *ground* RDF graph [18]). These operators are adapted from the set of standard operators defined for a graph model:

I) ***provenance_compare ( )***: Accurate comparison of scientific results requires the comparison of the associated provenance information. For example, two ocean visualization charts are said to be comparable if their provenance information such as types of sensors that generated the source data, the parameters values used in the scientific workflow to compute over the source data etc., are identical or comparable.

We adapt the RDF graph equivalence definition [19] with the added functionality of "coloring" the nodes and labeling the edges using the provenir ontology class and properties to define equivalence between two provenance graphs. If a bijection function exists to map two provenance metadata graph elements, the two graphs are equivalent.

Definition 7:
provenance_compare $(pg_1, pg_2 \mid pg_1 = \text{graph}(d_1, p_1, a_1, r_1), pg_2 = \text{graph}(d_2, p_2, a_2, r_2)) \equiv \{true, false\}$
where, there exists a bijection M such that
- $\forall d_1\ M(d_1) = d_1$, and $d_1, d_2 \in$ *provenir*:`data`;
- $\forall p_1\ M(p_1) = p_1$; and $p_1, p_2 \in$ *provenir*:`process`;
- $\forall a_1\ M(a_1) = a_1$; and $a_1, a_2 \in$ *provenir*:`agent`;
- $\forall r_1\ M(r_1) = r_1$; and $r_1, r_2 \in R$ (set of provenir ontology properties);
- $\exists <s, p, o>$ in pg_1 iff $<M(s), p, M(o)>$ exists in pg_2, given $<s, p, o>$ is a RDF triple with *s*-subject, *p*-predicate, *o*-object

II) ***provenance_merge ( )***: Two sub-sections of the provenance metadata graph for given entities or set of entities may be combined using the RDF graph merge logic. The new merged graph does not include any duplicates individuals.

Definition 8:
provenance_merge $(pg_1, pg_2 \mid pg_1 = \text{graph}(d_1, p_1, a_1, r_1), pg2 = \text{graph}(d_2, p_2, a_2, r_2)) \equiv \{mg \mid mg = \text{graph}(d, p, a, r)\}$
where,
- $d = \{d_1 \cup d_2\}$, and $d_i\ d_2 \in$ *provenir*:`data`;
- $p = \{p_1 \cup p_2\}$, and $p_i\ p_2 \in$ *provenir*:`process`;
- $a = \{a_1 \cup a_2\}$, and $a_i\ a_2 \in$ *provenir*:`agent`;
- $r = \{r_1 \cup r_2 \cup r_3\}$, given $r_1, r_2 \in R$ (set of provenir ontology properties) and r_3 – set of properties linking elements of $\{d, p, a\}$

In the next Section, we implement the provenance algebra using materialized views defined over relational database.

4. MATERIALIZED VIEW IMPLEMENTATION

In the world of database systems for provenance support, the need to dynamically maintain large amounts of complex data conflicts with the demand for subsecond query response time. Our answer to this dilemma is materialized views and indices, both of which precompute aggregate information. A database can utilize materialized views to prejoin tables, presort solution sets, and integrate semantic information. The materialized view can be set up to automatically keep itself in synch with base data, updating itself at predetermined intervals. The precomputed index is created using one or more columns of the underlying database tables, providing the basis for both rapid random lookups and efficient access of records. Because this work is completed in advance, it gives end users the illusion of instantaneous response time. Materialized views are especially useful for provenance queries, where cross-tabulations and recursive queries could take several minutes to perform.

The following five tables were created to store provenance data. The first table stores information associated with the input objects used to seed the workflow:

```
create table DataItems
(  DataID nvarchar(50) PRIMARY KEY,
   TemperatureID nvarchar(50),
   CurrentID nvarchar(50),
   BouyID nvarchar(50)
);
```

The second table stores a row for each workflow entry:

```
create table Fact
(    ActivityID  nvarchar(50)  PRIMARY
KEY,
   WorkflowID nvarchar(50),
   ActivityType nvarchar(100),
   ActivityStatus nvarchar(50),
   ActivityTime nvarchar(50)
);
```

The third table stores the set of objects associated with each workflow entry in the fact table which processed base input objects. Note there is a foreign key from `ObjectName` to `DataItems.DataID`.

```
create table InputObjI
(  ActivityID nvarchar(50),
   ObjectType nvarchar(50),
   ObjectName nvarchar(50),
   PRIMARY      KEY      (ActivityID,
ObjectType)
);
```

The fourth table stores the set of input properties associated with each workflow entry in the fact table.

```
create table InputPropI
(  ActivityID nvarchar(50),
   Property nvarchar(50),
   Value nvarchar(50),
   PRIMARY KEY (ActivityID, Property)
);
```

The fifth and final table stores the set of output properties associated with each workflow entry in the fact table.

```
create table OutputPropI
(  ActivityID nvarchar(50),
   Property nvarchar(50),
   Value nvarchar(50),
   PRIMARY KEY (ActivityID, Property)
);
```

Representation of provenance data requires nested sets, and normalization results in five distinct tables to represent the associated properties. As stated earlier, there is one foreign key from `InputObjI.ObjectName` to `DataItems.DataID`. As we discuss later, an index on input and output properties allow us to efficiently recurse in both directions during query processing.

5. EXPERIMENTS AND RESULTS

The dataset for the evaluations was generated from the oceanography scenario described in Section 1.1. The scientific workflow is from the Neptune project and the simulated details describing data collection by temperature and ocean current sensors, both contained in an ocean buoy, were added by us. The provenance metadata for this scenario records the events (we assume one provenance record is generated per event) starting with the data collection by the sensors, followed by data processing in the scientific workflow and finally to the output in form of visualization charts.

The scientific workflow is currently run on the Windows Workflow Foundation-based Trident workbench [20]. We used the log file generated by the Trident workbench with additional details such as temperature sensors, ocean current sensors, and ocean buoys, to create a provenance template. Using this template we generated four provenance files corresponding to 10,000; 50,000; 100,000; 500,000 data collection (by sensors) and workflow execution processes representing realistic provenance metadata for our evaluations. As discussed in Section 4, the data is stored in the normalized form in five tables.

In terms of pre-computation, all primary keys and foreign keys had associated primary and secondary indices respectively. Note that this is a generic choice which could be automatically done for any provenance data cast into this provenance framework. These indices ensured that the corresponding lookups during query execution were performed efficiently. Note that more sophisticated forms of pre-computation, such as materialized views, could be used to improve performance in more complex queries such as rollups (aggregation).

The timings reported are mean result from the last five runs out of a total of ten runs; the variance was extremely low (<5% difference between highest and lowest score used in average for every case). The SQL queries for the experiments are listed in **Appendix A**.

5.1 Experimental Setup

The experiments were conducted using SQL Server 2008 on a windows server with 500MB cache, with all performance measurements collected on a warm cache. The goal of our experiments is to evaluate the performance and scalability gain from implementation of provenance query operators using well-defined indices for complex provenance queries.

5.2 Experiment I

This experiment evaluates the benefit of implementing the *provenance ()* query operator (described in Section 3.1), using indices, to answer the query that retrieves all provenance metadata associated with a data entity "*HyperCube85357162234026*".

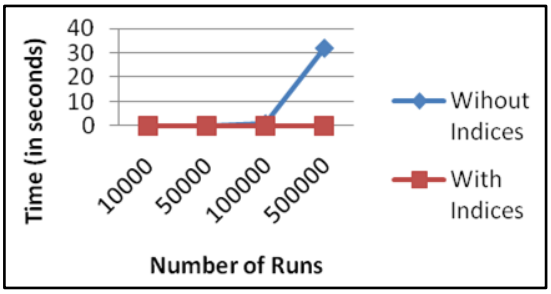

Figure 5.1: Performance values for provenance () query operator for various sized datasets

As expected, the implementation of the query using indices leads to a significant gain in performance as the dataset increases. For the 500K dataset, the system hit the memory limit of 500MB.

5.3 Experiment II

As discussed in Section 3.1, the *provenance ()* query is not required in most scenarios, hence the *provenance_pathway ()* query operator was defined with additional restriction focused on workflow provenance. In this experiment we run Experiment I query using *provenance_pathway()* operator.

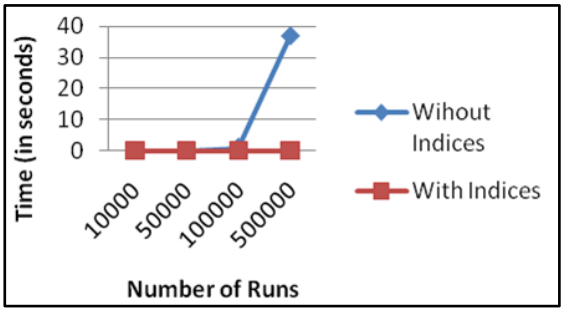

Figure 5.2: Performance values for provenance_pathway () query operator for various sized datasets

We used a recursive CTE (common table expression) called "ProvenancePathway" in this query. This is a recursive query with two parts. The first part is the seed of the recursion that selects a row from the `table 2` which has an associated output property with value "*HyperCube85357162234026*". The recursive part is then included after a union all. It takes existing rows from previous iterations, and joins them to the table **`Fact`** in a way which follows the provenir schema through one edge. The recursion is performed until no new rows are added. These results are similar to Experiment I, but the performance for dataset size of 500K is worse than Experiment I (37 seconds vs. 32 seconds for Experiment I) due to greater number of restrictions for this query as compared to *provenance ()* (discussed in Section 3.1). But, we clearly see that query implementation with indices scales easily with increasing dataset size.

5.4 Experiment III

This experiment evaluates the performance of *provenance_context()* query operator that returns the result data which used source data from a contaminated ocean buoy "*oceanBuoy7044*" (scenario described earlier in Section 1.1). The constraints define a provenance subgraph from which datasets belonging to provenir **`data_collection`** class (or one of its subclasses) are returned.

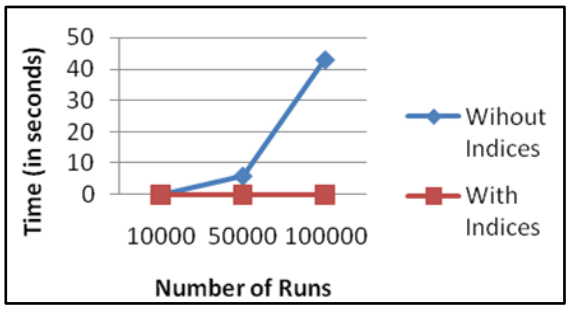

Figure 5.3: Performance values for provenance_context () query operator for various sized datasets

This result clearly illustrates the complexity of this query as compared to the first two queries and the benefit from implementing the query using indices. In addition to primary and foreign key indices we defined two additional indices on table **`DataItems`** columns for ocean buoy identifiers and input value column in table **`InputPropI`**.

6. RELATED WORK

Provenance metadata has been studied from multiple perspectives across a number of different domains. In this Section we discuss the provenance work that is divided into two categories in computer science literature, namely database provenance and workflow provenance [21]. We correlate the provenance query operators introduced in this paper to the different approaches for creation, representation and usage of provenance in these two categories, as illustrated in Figure 6.1.

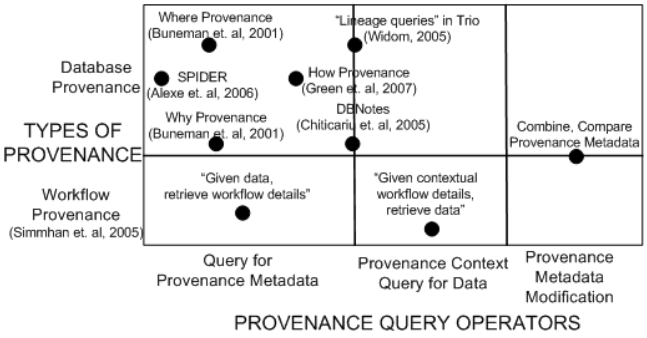

Figure 6.1: Correlation of provenance query operators with existing work in database and workflow provenance

6.1 Database Provenance

Database provenance or data provenance, often termed as "fine-grained" provenance [21], has been extensively studied in the database community. Early work includes the use of annotations to associate "data source" and "intermediate source" with data (polygen model) in a federated database environment to resolve conflicts by Wang et al [22], and use of "attribution" for data extracted from Web pages by Lee et al [23]. More recent work has defined database provenance in terms of "Why provenance", "Where provenance" (previously in **Section 3.1**, we correlated the *provenance ()* query operator with "Where provenance") by Buneman et al [5]; and "How provenance" by Green et al [6].

"Why provenance" was first described in [4], and in this paper we use the syntactic definition of "Why provenance" by Buneman et al [5] that defines a "proof" for a data entity. The proof consists of a query, representing a set of constraints, over a data source with "witness" values that result in a particular data output; hence, the semantics of *provenance ()* query operator closely relates to "Why provenance" [5]. To address the limitation of "Why provenance" that includes "…set of all contributing input tuples" leading to ambiguous provenance, Green et al [6] introduced semiring-based "How provenance". The *provenance ()* query operator over a "weighted" provenance model, to reflect the individual contribution of each component (for example process loops or repeated use of single source data), is comparable to "How provenance". The Trio project [24] considers three aspects of lineage information of a given tuple, namely how was a tuple in the database derived along with time value (when) and the data sources used. A subset of queries in Trio, "lineage queries",

discussed in [24] can be mapped to query for provenance and contextual query operators.

The SPIDER system [25] built on top of Clio [26] is an application that uses provenance information modeled as "routes" (schema mappings) between source and target data to capture aspects of both "Why provenance" and "How provenance".

6.2 Scientific Workflow Provenance

The rapid adoption of scientific workflows to automate scientific processes has catalyzed a large body of work in recording provenance information for the results generated by scientific workflows. As discussed earlier in Section 1, traditionally workflow provenance represented the execution sequence of computational processes, the input and output data values for these computational processes. Simmhan et al [7] survey different approaches for collection, representation and management of workflow provenance. The participants in the recent provenance challenge [8] collected provenance at different levels of granularities such as comprehensive workflow system trace in PASS [27], use of semantic annotations of services by Taverna [28], recording of data value details and service invocations in Karma [29], and workflow as well as data modeling in REDUX [30]. Further details about the provenance challenge are available at [8]. Recent work has also recognized the need for inclusion of domain semantics in form of domain-specific provenance metadata [31] with workflow provenance. The work presented in this paper forms the theoretical foundations of the provenance framework introduced in [31]. The Stanford Knowledge Provenance Infrastructure (KPI) [32] provides provenance information as explanations, including the source of data and any reasoning or inference processes applied to the data, related to Web data such as news feeds and other information sources. The Vistrails project [33] is a novel project to track the provenance of scientific workflows as they evolve through modifications to reflect changes in scientific processes. The provenance about workflows enables scientists to track and manage the processes represented as workflows.

Recent initiatives such as the Open Provenance Model (OPM) [34] are working to create a common model for provenance in the eScience community. The concepts proposed by OPM are equivalent to the three top-level concepts of the provenir ontology namely, artifact, process and agent. Unlike provenir ontology which is defined in OWL-DL, OPM has not yet decided on representation formalism beyond the generic graph model. Hence, the rules of reasoning proposed in OPM are easily contradicted [35]. A detailed discussion arguing for use of W3C Semantic Web languages formalism to represent OPM can be found in [31].

7. CONCLUSIONS

In this paper we introduce a common provenance model called *provenir* ontology defined in OWL-DL, and discussed the satisfiability of provenir. We propose a novel classification scheme for provenance queries, and based on this classification we define a set of provenance query operators. The semantics of each query operator is formally defined in terms of the provenir ontology and mapped to existing work in both database and workflow provenance. Finally, we introduce a practical provenance storage solution using materialized views over a generic relational database system and present an evaluation of performance and scalability of our initial implementation.

To the best of our knowledge this is the first provenance management system that defines both provenance model, consisting of algebra and operators, and efficient provenance storage and query support based on materialized views. Our approach supports the complete process from a formal provenance model to the storage and efficient queries over provenance data.

8. ACKNOWLEDGMENTS

We would like to thank T.K. Prasad for his valuable inputs and suggestions. The first author was partly funded by NIH RO1 Grant# 1R01HL087795-01A1 for this work.

9. REFERENCES

[1] "MyGrid." http://www.mygrid.org.uk/

[2] "Project Neptune." http://www.neptune.washington.edu/

[3] "The Panoramic Survey Telescope and Rapid Response System (Pan-STARRS)." http://pan-starrs.ifa.hawaii.edu/

[4] Y. Cui, Widom, J., "Practical Lineage Tracing in Data Warehouses," in *16th ICDE*, San Diego, California, 2000.

[5] P. Buneman, Khanna , S., Tan, W.C., "Why and Where: A Characterization of Data Provenance," in *8th International Conference on Database Theory*, 2001, pp. 316 - 330

[6] T. J. Green, Karvounarakis, G. ,Tannen, V., "Provenance Semirings," in *ACMSIGMOD-SIGACTSIGART Symposium on Principles of database systems (PODS)*, 2007, pp. 675–686.

[7] Y. L. Simmhan, Plale, A.B., Gannon, A. D., "A survey of data provenance in e-science " *SIGMOD Rec.* , vol. 34, pp. 31 - 36 September 2005.

[8] "IPAW 2008." http://www.sci.utah.edu/ipaw2008/overview.html

[9] "OWL Web Ontology Language Overview," in *W3C Recommendation*, D. L. McGuinness, et al., 2004.

[10] F. Manola, Miller, E.(Eds.), "RDF Primer," in *W3C Recommendation*, B. McBride, Ed., 2004.

[11] P. F. Patel-Schneider, Hayes, P., Horrocks, I., "OWL Web Ontology Language Semantics and Abstract Syntax " in *W3C Recommendation*, 2004

[12] S. S. Sahoo, Thomas, C., Sheth, A., York, W. S., and Tartir, S., "Knowledge modeling and its application in life sciences: a tale of two ontologies," in *Proceedings of the WWW '06* Edinburgh, Scotland, 2006, pp. 317-326.

[13] B. Smith, W. Ceusters, B. Klagges, J. Kohler, A. Kumar, J. Lomax, C. Mungall, F. Neuhaus, A. L. Rector, and C. Rosse, "Relations in biomedical ontologies," *Genome Biol*, vol. 6, 2005.

[14] G. Lausen, Meier, M., Schmidt, M., "SPARQLing constraints for RDF," in *EDBT'08*, 2008, pp. 499-509.

[15] M. K. Smith, Welty, C., McGuinness, D.L., "OWL Web Ontology Language Guide," in *W3C Recommendation*, 2004.

[16] S. Tobies, "Complexity Results and Practical Algorithms for Logics in Knowledge Representation." Ph.D. Thesis, RWTH Aachen, Germany, 2001.

[17] R. V. Guha, "Contexts: A Formalization and Some Applications ": PhD Thesis, Stanford University, 1991

[18] P. Hayes, "RDF Semantics," B. McBride, Ed., 2004.

[19] G. Klyne, Carroll, J. J., "Resource Description Framework (RDF): Concepts and Abstract Syntax," in *W3C Recommendation*, 2004.

[20] "Trident Workflow Workbench." http://www.microsoft.com/mscorp/tc/trident.mspx

[21] W. C. Tan, "Provenance in Databases: Past, Current, and Future," *IEEE Data Eng. Bull.*, vol. 30, pp. 3 -12 2007.

[22] Y. R. Wang, Madnick, S. E., "A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective," in *16th VLDB Conference*, 1990, pp. 519–538.
[23] T. Lee, Bressan, S, "Multimodal Integration of Disparate Information Sources with Attribution," in *Entity Relationship Workshop on Information Retrieval and Conceptual Modeling*, 1997.
[24] J. Widom, "Trio: A System for Integrated Management of Data, Accuracy, and Lineage," in *Second Biennial Conference on Innovative Data Systems Research (CIDR '05)*, 2005.
[25] B. Alexe, Chiticariu, L., Tan. W.-C., "SPIDER: a Schema mapPIng DEbuggeR. ," in *VLDB*, 2006, pp. 1179–1182.
[26] L. M. Haas, Hern´andez, M. A. , Ho, H. , Popa, L., Roth, M., "Clio Grows Up: From Research Prototype to Industrial Tool," in *ACM SIGMOD*, Baltimore, MD, 2005, pp. 805–810.
[27] D. A. Holland, Seltzer, M.I., Braun, U., Muniswamy-Reddy, K., "PASSing the provenance challenge," *Concurrency and Control: Practice and Experience*, vol. 20, pp. 531-540, 2008.
[28] J. Zhao, Goble, C., Stevens, R., Turi, D. , "Mining taverna's semantic web of provenance," *Journal of Concurrency and Computation:Practice and Experience*, 2007.
[29] Y. L. Simmhan, Plale, B., Gannon, D., " Karma2: Provenance management for data driven workflows," *International Journal of Web Services Research*, vol. 5, 2008.
[30] R. S. Barga, Digiampietri, L. A., "Automatic capture and efficient storage of escience experiment provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, pp. 419--429, 2008.
[31] S. S. Sahoo, Sheth, A., Henson, C., "Semantic Provenance for eScience: Managing the Deluge of Scientific Data," *IEEE Internet Computing*, vol. 12, pp. 46-54, 2008.
[32] D. L. McGuinness, Pinheiro da Silva, P., "Explaining Answers from the Semantic Web: The Inference Web Approach," *Journal of Web Semantics*, vol. 1, pp. 397-413, October 2004.
[33] "The VisTrails Project." http://www.vistrails.org
[34] "Open Provenance Model." http://twiki.ipaw.info/bin/view/Challenge/OPM
[35] Y. L. Simmhan, "FeedbackonOPM," 2008. http://twiki.ipaw.info/pub/Challenge/OpenProvenanceModelWorkshop/FeedbackonOPM.pptx

APPENDIX A – Provenance Queries in SQL

Query 1

```
WITH ProvenancePathway (ActivityID, WorkflowID, ActivityType, ActivityState, ActivityTime, Level)
AS (
select Fact.ActivityID, Fact.ActivityStatus, Fact.ActivityTime, Fact.ActivityType, Fact.WorkflowID, 0 as Level
from dbo.Fact, dbo.OutputProp
where (Fact.ActivityID = OutputProp.ActivityID) and
(OutputProp.Value='HyperCube85357162234026')
UNION ALL
select Fact1.ActivityID, Fact1.ActivityStatus, Fact1.ActivityTime, Fact1.ActivityType, Fact1.WorkflowID, Level+1
from dbo.Fact as Fact1, ProvenancePathway as Fact2
where EXISTS ((select IP.Value from dbo.InputProp as IP where IP.ActivityID = Fact2.ActivityID) intersect
(select OP.Value from dbo.OutputProp as OP where OP.ActivityID = Fact1.ActivityID)) )
select *
from ProvenancePathway
```

Query 2

```
WITH ProvenancePathway (ActivityID, WorkflowID, ActivityType, ActivityState, ActivityTime, Level)
AS (
select Fact.ActivityID, Fact.ActivityStatus, Fact.ActivityTime, Fact.ActivityType, Fact.WorkflowID, 0 as Level
from dbo.Fact, dbo.OutputProp
where (Fact.ActivityID = OutputProp.ActivityID) and
(OutputProp.Value='HyperCube85357162234026')
UNION ALL
select Fact1.ActivityID, Fact1.ActivityStatus, Fact1.ActivityTime, Fact1.ActivityType, Fact1.WorkflowID, Level+1
from dbo.Fact as Fact1, ProvenancePathway as Fact2
where EXISTS ((select IP.Value from dbo.InputProp as IP where IP.ActivityID = Fact2.ActivityID) intersect
(select OP.Value from dbo.OutputProp as OP where OP.ActivityID = Fact1.ActivityID)) )
select InputObj.ActivityID, InputObj.ObjectName, InputObj.ObjectType
from ProvenancePathway as PP, InputObj
where (PP.ActivityID = InputObj.ActivityID)
```

Query 3

```
WITH ProvenancePathway (ActivityID, WorkflowID, ActivityType, ActivityState, ActivityTime, Level)
AS (
select Fact.ActivityID, Fact.ActivityStatus, Fact.ActivityTime, Fact.ActivityType, Fact.WorkflowID, 0 as Level
from dbo.Fact, dbo.DataItems, dbo.InputObj
where (Fact.ActivityID = InputObj.ActivityID) and
(InputObj.ObjectName=DataItems.DataID) and
(DataItems.BouyID='oceanBuoy7044')
UNION ALL
Select Fact1.ActivityID, Fact1.ActivityStatus, Fact1.ActivityTime,
```

```
Fact1.ActivityType,          Fact1.WorkflowID,
Level+1
from dbo.Fact as Fact1, ProvenancePathway as
Fact2
where   EXISTS   ((select   IP.Value   from
dbo.InputProp as IP where IP.ActivityID =
Fact1.ActivityID) intersect
       (select OP.Value from dbo.OutputProp
as     OP     where     OP.ActivityID     =
Fact2.ActivityID)) )
select *
from ProvenancePathway as PP
where exists (select * from dbo.OutputProp
as OP where PP.ActivityID=OP.ActivityID and
OP.Value like 'ChartData%')
```

APPENDIX B – Provenance Context Query Operators

1. Provenance Context Query Operator

Definition 7:
∀d ∈ *provenir*: data, ∀p ∈ *provenir*: process, ∀a ∈ *provenir*: agent *provenance_context* (dc) = pc_data (d) ∩ pc_process (p) ∩ pc_agent (a), holds ∈ DB

- pc_data (d) = {dc | dc→*ro*:derives_from→d} ∪ {dc | dc→*ro*:transformation_of→d} ∪ {dc | dc→*ro*:part_of→d} ∪ {dc | d→*ro*:part_of→dc} ∪ {dc | dc→*ro*:contained_in→d}∪ {dc | d→*ro*:contained_in→dc}
- pc_process(p) = {dc | p→*ro*:has-participant→dc} ∪ {dc | p→*ro*:preceded_by→p_{res} ∧ p_{res}→*ro*:has_participant→dc } ∪ {dc | p→*ro*:has_part→p_{res} ∧ p_{res}→*ro*:has_participant→dc } ∪ {dc | p_{res}→*ro*:has_part→p ∧ pr→*ro*:has_participant→dc }
- pc_agent(a) = {dc | p→*ro*:has_agent→a ∧ p ∈ pc_process(p)} ∪ {dc | a→*ro*:contained_in→a_{res} ∧ p→*ro*:has_agent→a_{res} ∧ p ∈ pc_process(p)} ∪ {dc | a_{res}→*ro*:contained_in→a ∧ p→*ro*:has_agent→a_{res} ∧ p ∈ pc_process(p)} ∪ {dc | a→*ro*:part_of→a_{res} ∧ p→*ro*:has_agent→a_{res} ∧ p ∈ pc_process(p)} ∪ {dc | a_{res}→*ro*:part_of→a ∧ p→*ro*:has_agent→a_{res} ∧ p ∈ pc_process(p)} ∪ {dc | a→*ro*:adjacent_to→a_{res} ∧ p→*ro*:has_agent→a_{res} ∧ p ∈ pc_process(p)} ∪ {dc | a_{res}→*ro*:adjacent_to→a ∧ p→*ro*:has_agent→a_{res} ∧ p ∈ pc_process(p)}

2. Provenance Context Process Query Operator

Definition 7:
∀d ∈ *provenir*: data, ∀p ∈ *provenir*: process, ∀a ∈ *provenir*: agent *provenancecontext_process* (pc) = pc_data (d) ∩ pc_process (p) ∩ pc_agent (a), holds ∈ DB

- pc_data (d) = {pc | pc→*ro*:has_participant→d} ∪ {pc | pc→*ro*:has_parameter→d} ∪ {pc | d_{res}→*ro*:derives_from→d ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})} ∪ {pc | d→*ro*:derives_from→d_{res} ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})} ∪ {pc | d→*ro*:transformation_of→d_{res} ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})} ∪ {pc | d_{res}→*ro*:transformation_of→d ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})} ∪ {pc | d_{res}→*ro*:part_of→d ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})} ∪ {pc | d→*ro*:part_of→d_{res} ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})} ∪ {pc | d_{res}→*ro*:contained_in→d ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})} ∪ {pc | d→*ro*:contained_in→d_{res} ∧ (pc→*ro*:has_participant→d_{res}∨ pc→*ro*:has_parameter→d_{res})}
- pc_process(p) = {pc | pc→*ro*:preceded_by→p} ∪ {pc | p→*ro*:preceded_by→pc} ∪ {pc | p→*ro*:has_part→pc } ∪ {pc | pc→*ro*:has_part→p}
- pc_agent(a) = {pc | p→*ro*:has_agent→a} ∪ {pc | a→*ro*:contained_in→a_{res} ∧ pc→*ro*:has_agent→a_{res}} ∪ {pc | a_{res}→*ro*:contained_in→a ∧ pc→*ro*:has_agent→a_{res}} ∪ {pc | a→*ro*:part_of→a_{res} ∧ pc→*ro*:has_agent→a_{res}} ∪ {pc | a_{res}→*ro*:part_of→a ∧ pc→*ro*:has_agent→a_{res}} ∪ {pc | a→*ro*:adjacent_to→a_{res} ∧ pc→*ro*:has_agent→a_{res}} ∪ {pc | a_{res}→*ro*:adjacent_to→a ∧ pc→*ro*:has_agent→a_{res}}

3. Provenance Context Agent Query Operator

Definition 7:
∀d ∈ *provenir*: `data`, ∀p ∈ *provenir*: `process`, ∀a ∈ *provenir*: `agent` *provenancecontext_agent* (ac) = pc_data (d) ∩ pc_process (p) ∩ pc_agent (a), holds ∈ DB

- pc_data (d) = {ac | p→*ro*:`has_participant`→d ∧ pc_process(p, ac)} ∪ {ac | p→*ro*:`has_parameter`→d ∧ pc_process(p, ac)} ∪ {ac | d_{res}→*ro*:`derives_from`→d ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)} ∪ {ac | d→*ro*:`derives_from`→d_{res} ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)} ∪ {ac | d→*ro*:`transformation_of`→d_{res} ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)} ∪ {ac | d_{res}→*ro*:`transformation_of`→d ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)} ∪ {ac | d_{res}→*ro*:`part_of`→d ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)} ∪ {ac | d→*ro*:`part_of`→d_{res} ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)} ∪ {ac | d_{res}→*ro*:`contained_in`→d ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)} ∪ {ac | d→*ro*:`contained_in`→d_{res} ∧ (p→*ro*:`has_participant`→d_{res}∨ p→*ro*:`has_parameter`→d_{res}) ∧ pc_process(p, ac)}
- pc_process(p) = {ac | p→*ro*:`has_agent`→ac} ∪ {ac | p_{res}→*ro*:`preceded_by`→p ∧ p_{res}→*ro*:`has_agent`→ac} ∪ {ac | p→*ro*:`preceded_by`→p_{res} ∧ p_{res}→*ro*:`has_agent`→ac} ∪ {ac | p_{res}→*ro*:`part_of`→p ∧ p_{res}→*ro*:`has_agent`→ac} ∪ {ac | p→*ro*:`part_of`→p_{res} ∧ p_{res}→*ro*:`has_agent`→ac}
- pc_agent(a) = {ac | a→*ro*:`contained_in`→ac} ∪ {ac | ac→*ro*:`contained_in`→a} ∪ {ac | a→*ro*:`part_of`→ac} ∪ {ac | ac→*ro*:`part_of`→a} ∪ {ac | a→*ro*:`adjacent_to`→ac} ∪ {ac | ac→*ro*:`adjacent_to`→a}