Provenance Browser: Displaying and Querying Scientific Workflow Provenance Graphs

Manish Kumar Anand[1], Shawn Bowers[2], Bertram Ludäscher[1,3]

[1]*Dept. of Computer Science, University of California, Davis*
[2]*Dept. of Computer Science, Gonzaga University*
[3]*Genome Center, University of California, Davis*
{maanand, ludaesch}@ucdavis.edu, bowers@gonzaga.edu

Abstract— **This demonstration presents an interactive provenance browser for visualizing and querying data dependency (lineage) graphs produced by scientific workflow runs. The browser allows users to explore different views of provenance as well as to express complex and recursive graph queries through a high-level query language (QLP). Answers to QLP queries are lineage preserving in that queries return sets of lineage dependencies (denoting provenance graphs), which can be further queried and visually displayed (as graphs) in the browser. By combining provenance visualization, navigation, and query, the provenance browser can enable scientists to more easily access and explore scientific workflow provenance information.**

I. Introduction

A key advantage of scientific workflow systems over traditional scripting approaches is their ability to automatically record data and process dependencies introduced during workflow runs. Scientific workflow provenance is typically represented using data and process *dependency* (i.e., *causal*) *graphs* [1], [2], which can be used by scientists to better understand, reproduce, and verify scientific results. Provenance graphs may be large due to the complexity of the workflow, the size of input data sets, and the number of intermediate data sets produced, which can make it difficult for users to effectively find and view relevant information; and the ability to effectively store, query, and visualize provenance graphs has been identified as a significant challenge [2], [3], [4].

We address these challenges in this demonstration through a *provenance browser* that combines visualization, navigation, and high-level graph queries. Typical provenance queries are exploratory and involve finding some or all of the data and process dependencies that led to the creation of one or more data products [5]. Posed over dependency relations, these queries often require transitive closures and selection conditions on lineage paths. Current approaches for querying provenance information are largely based on physical data representations [2] (e.g., relational, XML, or RDF schemas) in which users express provenance queries through corresponding query languages (SQL, XQuery, or SPARQL). For most users, expressing these queries against physical provenance schemas requires considerable expertise and is cumbersome even for simple queries [2]. The complexity of these queries also creates challenges for efficient query evaluation [6].

Our Approach and Contributions. We present a provenance browser that can display different provenance views for scientific workflow traces and that supports provenance queries through a novel *Query Language for Provenance* (QLP) [7]. QLP provides specialized operators for querying provenance graphs that, unlike other languages which return sets of nodes (e.g., [8], [9]), are closed under lineage relationships (i.e., QLP lineage queries return sets of lineage dependencies forming provenance subgraphs). The browser provides an integrated environment where users can issue QLP queries against workflow traces, and query results can be displayed, navigated, and further queried. We also employ novel query and storage optimization techniques that makes querying real-world provenance information within the browser practical and efficient.

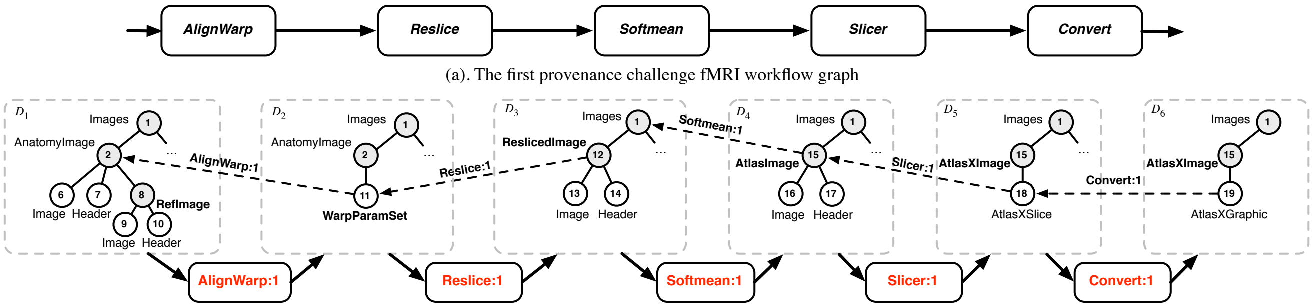
II. Provenance Model

Consider the example workflow in Fig. 1(a) showing a straightforward implementation of the fMRI image processing pipeline of the First Provenance Challenge [1]. We refer to steps in the workflow as *actors* that are *invoked* over input data supplied by previous steps. This workflow takes a set of anatomy images representing 3D brain scans and a reference image, and applies the following actors.

1) *AlignWarp* is invoked over each anatomy image to produce a set of "warping" parameters;
2) *Reslice* is invoked over each set of warping parameters to transform the associated anatomy image;
3) *Softmean* averages transformed images into an atlas image;
4) *Slicer* produces three different 2D slices of the atlas; and
5) *Convert* creates a graphical image for each 2D slice.

In the implementation of the workflow each invocation of an actor receives an XML data structure, performs an update on a portion of that structure, and then sends the updated version of the structure to downstream actors (see Fig. 1(b)) [7]. Here we assume that each XML structure consists of workflow data products in which each tree node has a unique identifier, and tree nodes represent either *collection tokens* or *data tokens* (wrapping complex objects or referencing external data, e.g., stored within a file). A collection token may be an internal node (for non-empty collections) or a leaf node (for empty collections), whereas data tokens are leaf nodes only.

978-1-4244-5446-4/10/$26.00 © 2010 IEEE



(a). The first provenance challenge fMRI workflow graph

(b). Flow graph with fine-grained node dependencies for the first invocation of each actor

Fig. 1. Example fMRI image analysis workflow [1] (a) and the flow graph showing the first invocation of each actor for a typical run (b).

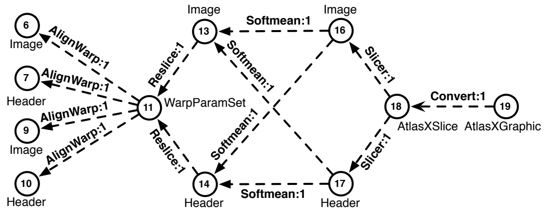


Fig. 2. The implied data dependency graph for data items in Fig. 1(b).

Fig. 1(b) shows the first invocation of each actor for a typical run of the workflow. The invocation of the AlignWarp actor (i.e., *AlignWarp:1*) modifies the first AnatomyImage collection (node 2), and replaces its contents with a WarpParamSet data token (node 11). Similarly, the invocation of the Reslice actor uses this WarpParamSet to generate a new Image and Header data token (nodes 13 and 14, respectively). Since only a part of an XML data structure D may be modified by an invocation, we also represent explicit ("fine-grained") data dependencies as part of a run. For example, the dashed arrow from node 11 to node 2 in Fig. 1(b) states that the WarpParamSet was created from the AnatomyImage collection by the first invocation of AlignWarp. Note that implicitly, node 11 depends on each of the descendents of node 2 (nodes 6-10 in the figure). Each descendent of a collection also implicitly inherits the dependencies of its ancestors. In the example, node 13 is a descendent of node 12, and thus implicitly depends on node 11. Taken together, Fig. 1(b) denotes a portion of the *flow graph* (or *trace*) for a run of Fig. 1(a), in this case corresponding to the first invocation of each workflow actor.

III. Expressing Provenance Queries in QLP

QLP queries are posed against flow graphs using the constructs of Fig. 3. Different constructs are used to query distinct *dimensions* of the flow graph representing: (i) *lineage relations* among nodes and invocations; (ii) *flow relations* among input and output structures of invocations; and (iii) *structural relations* among nodes within and across data structures.

Queries over Lineage Relations. A lineage relation is of the form $\langle n_1, i, n_2\rangle$ for nodes n_1 and n_2 and invocations i, e.g., $\langle 2, \text{AlignWarp:}1, 11\rangle$ is a lineage relation of Fig. 1(b) stating node 2 was used by the first invocation of the AlignWarp actor to produce node 11 (i.e., node 11 *depends on* node 2). Sets of lineage relations define lineage graphs (see Fig. 2), and QLP lineage queries act as filters over lineage relations. For example, the following QLP lineage queries

* derived 19 (1)
6 derived * (2)
#Softmean through #Convert derived * (3)

return (1) lineage relations denoting the set of paths starting from any node and ending at node 19, (2) lineage relations denoting the set of paths starting at node 6 and ending at any node, and (3) lineage relations denoting the set of paths starting at invocations of Softmean and go through invocations of Convert.

Queries over Flow Relations. QLP also allows lineage graphs to be filtered based on specific versions (or *occurrences*) of nodes within a flow graph using the @in and @out operators. For example, the following queries

* @in derived 19 (4)
18 @out Slicer:1 derived * (5)

return (4) lineage edges denoting paths that start at a node in the input of the workflow run and end at node 19, and (10) lineage relations denoting paths that start at the occurrence of node 18 in the output of the first invocation of Slicer.

Queries over Structural Relations. QLP queries can also contain XPath expressions for accessing nodes based on their type (i.e., tag name) and parent-child relationships. For example, the following query

* derived //AtlasXGraphic (6)

returns (6) lineage relations denoting paths that end at AtlasXGraphic nodes. Each of the above dimensions can also be combined to form more complex query expressions [7].

IV. The Provenance Browser

The provenance browser provides users with an interactive application for accessing provenance information generated by scientific workflow runs. We describe the main features of the browser below.

Provenance Browser Architecture. The basic architecture of the provenance browser is shown in Fig. 4, and the basic user interface is shown in Fig. 5. The provenance browser has been integrated with the Kepler Scientific Workflow System [10], [11], and can also be run as a standalone application. The browser works over workflow traces represented in a generic XML format. Given a trace file, a set of pre-processing

Construct	*Descriptive Form*	*Result*
Node and invocation expressions		
n, x, *	n, x, *	Node expressions e_n as a single node n, XPath expression x, or set of trace nodes *.
#i, #a	i, a	Invocation expressions e_i as an invocation i or actor a (denoting the set of a invocations).
e_n @in e_i	e_n @in e_i	Nodes of e_n input to invocations of e_i. If e_i not given, then nodes of e_n input to the workflow run.
e_n @out e_i	e_n @out e_i	Nodes of e_n output by invocations of e_i. If e_i not given, then nodes of e_n output by the workflow run.
Lineage-preserving path queries (examples)		
$* .. e_n$	* derived e_n	Lineage graph for nodes in e_n.
$e_n .. *$	e_n derived *	Lineage graph for nodes derived from nodes in e_n.
$e_{n_1} .. e_{n_2}$	e_{n_1} derived e_{n_2}	Lineage graph with paths from nodes in e_{n_1} to nodes in e_{n_2}.
$e_{n_1} .. e_i .. e_{n_2}$	e_{n_1} through e_i derived e_{n_2}	Lineage graph with paths from nodes in e_{n_1} to nodes in e_{n_2} that pass through an invocation in e_i.

Fig. 3. Subset of basic QLP constructs with descriptive notations.

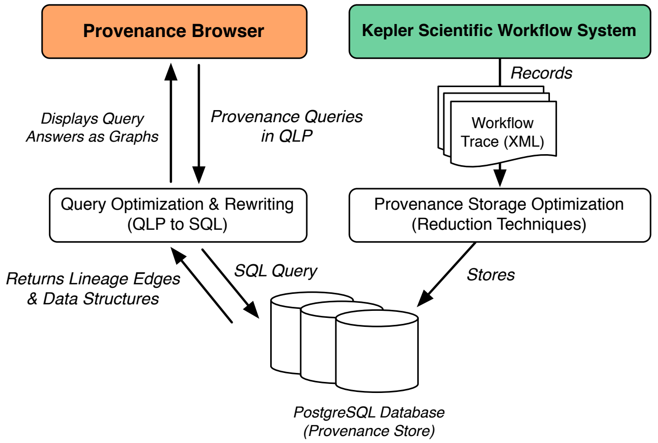
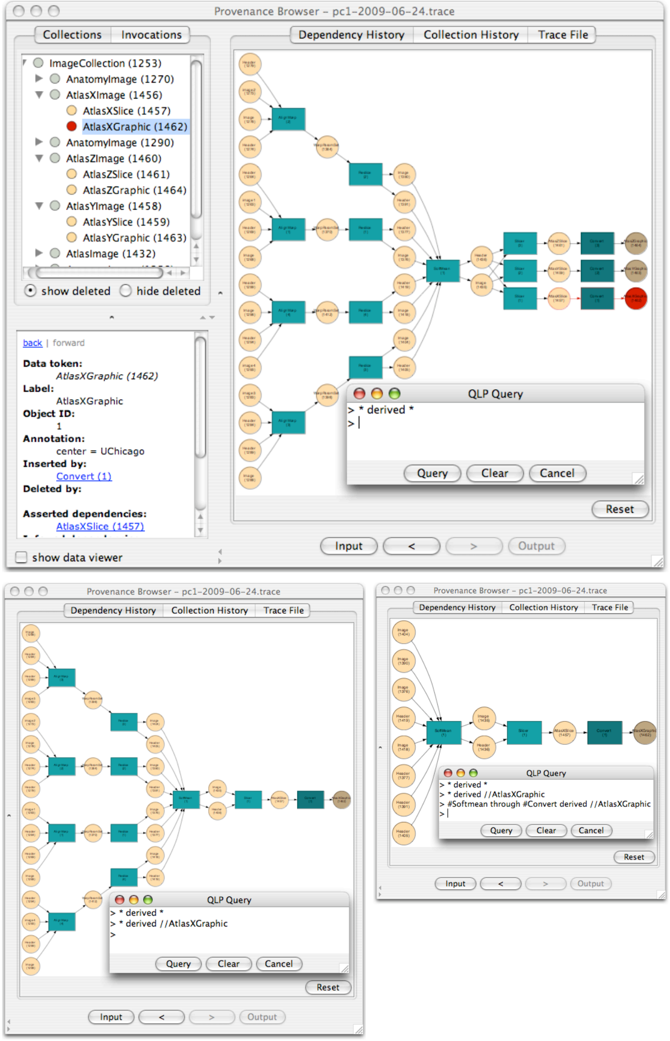

Provenance Browser
Kepler Scientific Workflow System
Records
Workflow Trace (XML)
Displays Query Answers as Graphs
Provenance Queries in QLP
Query Optimization & Rewriting (QLP to SQL)
Provenance Storage Optimization (Reduction Techniques)
Returns Lineage Edges & Data Structures
SQL Query
Stores
PostgreSQL Database (Provenance Store)

Fig. 4. Basic provenance browser architecture.

steps are applied to the trace prior to storage in a relational database (in our current implementation, PostgreSQL). These pre-processing steps perform storage reduction techniques (based on factorization) over the data lineage graph of the workflow trace as described in [4]. Using the provenance browser, a user can connect to a provenance store to select traces to view, issue QLP queries against the trace, and then display, navigate, and further query these results. As shown in Fig. 4, QLP queries are parsed, optimized, and rewritten to corresponding SQL queries expressed against the provenance database. Optimized and translated SQL queries return sets of lineage edges as query results from which the browser constructs and displays the corresponding lineage graph.



ImageCollection (1253)
AnatomyImage (1270)
AtlasXImage (1456)
AtlasXSlice (1457)
AtlasXGraphic (1462)
AnatomyImage (1290)
AtlasZImage (1460)
AtlasZSlice (1461)
AtlasZGraphic (1464)
AtlasYImage (1458)
AtlasYSlice (1459)
AtlasYGraphic (1463)
AtlasImage (1432)

back | forward

Data token: *AtlasXGraphic (1462)*
Label: AtlasXGraphic
Object ID: 1
Annotation: center = UChicago
Inserted by: Convert (1)
Deleted by:
Asserted dependencies: AtlasXSlice (1457)

```
> * derived *
> |
```

```
> * derived *
> * derived //AtlasXGraphic
>
```

```
> * derived *
> * derived //AtlasXGraphic
> #Softmean through #Convert derived //AtlasXGraphic
> |
```

Fig. 5. The data-dependency view and navigation panels of the provenance browser (top), and QLP query results displayed in the browser (bottom).

Displaying and Navigating Provenance Views. As shown at the top of Fig. 5, the left-side of the provenance browser displays the XML collection structure together with the details of actor invocations. Much like a web browser, this information can be navigated (e.g., to select among different data items and invocations). The browser also displays various provenance views of the execution trace: the *dependency history* view (Fig. 5) combines data dependency and process invocation graphs (where data nodes are denoted as circles and invocations as squares); the *collection history* view (top of Fig. 6) shows the data structures input and output by invocations; and the invocation graph view (bottom of Fig. 6) shows process dependencies. Each of these views are synchronized, e.g., selection of a data item in the dependency history view also selects the corresponding item in the collection history view. Within a view, users can also step forward and backward ("VCR-style") through the execution history to display corresponding portions of the XML structures and data dependencies.

Incremental Querying. The provenance browser allows users to issue QLP queries against lineage graphs as in Fig. 5. For example, the top of Fig. 5 shows the result of evaluating query Q1 "* derived *" returning the set of lineage relations shared between all nodes. Users can further execute queries via the QLP shell, which are executed over the current portion of the graph (alternatively, prior queries can be modified and rerun). For example, the bottom left of Fig. 5 shows

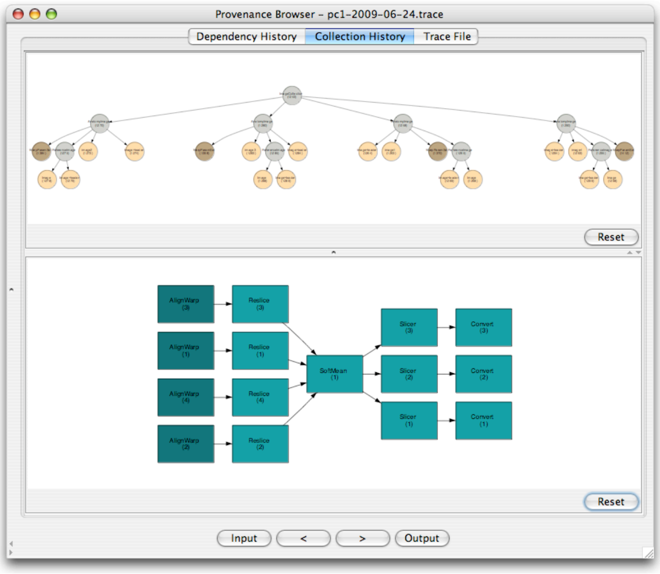


Fig. 6. Collection history (*top*) and invocation graph (*bottom*).

the result of query Q2 "* derived //AtlasXGraphic" issued over the result of query Q1. Similarly, the result of Q2 is further queried in Q3 "#Softmean through #Convert derived //AtlasXGraphic" returning the subgraph in the bottom-right of Fig. 5. By incrementally querying provenance graphs in this way, users can more easily inspect and explore relevant portions of large provenance graphs—which contrasts with more static approaches that display entire lineage graphs containing, e.g., hundreds or thousands of nodes and edges.

V. Storage and Query Optimization

The left side of Fig. 7 shows the sizes of actual provenance traces generated from metagenomic (STP, STM, and CYC), phylogenetic (WAT), and astronomy (PC3) workflows. As shown, the provenance graphs are relatively large, containing between ∼5–20K lineage edges, as shown by the number of tuples when only immediate edges (*I*) are stored. Even for simple QLP lineage queries, e.g., involving single-step derivation path expressions "$s_1 .. s_2$', standard evaluation techniques result in query execution times that are impractical. For instance, by storing both immediate and transitive dependencies (*IC*), these simple queries can take upwards of 1000 s (these times are worse if only immediate edges are stored, since recursion is required). Thus, to make the provenance browser feasible, we have developed novel storage [4] and query optimization techniques [12], which are exploited by the provenance browser.

Efficient evaluation of QLP path queries is closely tied to how lineage relations are stored. In the *ICP* approach, immediate edges and their transitive closure are stored using "pointer-based" reduction techniques [4]. As shown in Fig. 7, the space required for storing lineage relations in *ICP* is considerably less than for *IC*, and in many cases even less than *I* (due to the reduction techniques). By reducing storage size, evaluating lineage queries in two steps (computing nodes and then edges), and through pruning and temporary materialization of views and subquery results [12], query execution time can also be significantly reduced. As shown in Fig. 7, employing these techniques reduces query execution time to less than 100 ms, making common (but relatively complex) graph queries practical within the provenance browser.

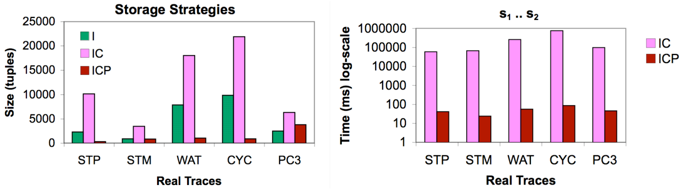


Fig. 7. Example sizes (left) and query time (right) for real provenance traces.

VI. Demonstration

Our demonstration highlights the following novel, end-to-end features of the provenance browser: (i) we show the use of the browser over a variety of real world traces produced by Kepler (i.e., those of Fig. 7); (ii) we demonstrate the ability of the browser to support incremental querying of these traces using QLP (demonstrating support of common types of provenance queries); (iii) we demonstrate the ability of the browser to navigate and display query results (e.g., for exploring data products, invocation parameters, and workflow execution history); and (iv) we show that our query optimization techniques improve query performance by comparing optimized versus non-optimized query evaluation (as in Fig. 7). To the best of our knowledge, the provenance browser is the only application of its kind that combines multiple views of provenance information, support for navigation, and a high-level, closed graph query language.

Acknowledgment

This work supported in part through NSF grants IIS-0630033, OCI-0722079, IIS-0612326, DBI-0533368, ATM-0619139, and DOE grant DE-FC02-07ER25811.

References

[1] L. Moreau, *et al*, "The open provenance model," ECS, Univ. of Southampton, Tech. Rep. 14979, 2007.
[2] S. B. Davidson and J. Freire, "Provenance and scientific workflows: challenges and opportunities," *SIGMOD*, 2008.
[3] T. Heinis and G. Alonso, "Efficient lineage tracking for scientific workflows," *SIGMOD*, 2008.
[4] M. K. Anand, S. Bowers, T. McPhilips, and B. Ludäscher, "Efficient provenance storage over nested data collections," *EDBT*, 2009.
[5] L. Moreau, *et al.*, "The first provenance challenge," *CCPE* 20(5):409-418, 2008.
[6] H. He and A. K. Singh, "Graphs-at-a-time: query language and access methods for graph databases," *SIGMOD*, 2008.
[7] M. K. Anand, S. Bowers, T. McPhilips, and B. Ludäscher, "Exploring scientific workflow provenance using hybrid queries over nested data and lineage graphs," *SSDBM*, 2009.
[8] C. Scheidegger, *et al.*, "Tackling the provenance challenge one layer at a time," *CCPE* 20(5):473-483, 2008.
[9] D. Holland, *et al.*, "A data model and query language suitable for provenance," *IPAW*, 2008.
[10] B. Ludäscher, *et al.*, "Scientific workflow management and the Kepler system," *CCPE* 18(10):1039-1065, 2006.
[11] S. Bowers, *et al.*, "Kepler/pPOD: Scientific workflow and provenance support for assembling the tree of life," *IPAW*, 2008.
[12] M. K. Anand, S. Bowers, and B. Ludäscher, "Techniques for efficiently querying scientific workflow provenance graphs," Submitted, 2009.