# Provenance Query Patterns for Many-Task Scientific Computing

Luiz M. R. Gadelha Jr., Marta Mattoso
*Computer and Systems Engineering Program*
*Federal University of Rio de Janeiro*
*Rio de Janeiro - Brazil*

Michael Wilde, Ian Foster
*Computation Institute*
*University of Chicago / Argonne National Laboratory*
*Chicago - USA*

## Abstract

Provenance information enable the analysis of large scale many-task computations often specified as scientific workflows. They allow for one to determine how each resulting data set was derived from other data sets and applications. In this work, we survey queries used for exploring provenance information about many-task computations. We present a set of patterns that can be identified in these queries, which is being used as a basis for the design and implementation of a provenance management system for many-task scientific computations, integrated to the Swift parallel scripting system. It has a data model similar to the Open Provenance Model, with extensions that enrich core structural provenance data, represented as consumption and production relationships between applications and data sets, with information about the runtime behavior of each application, and domain-specific information such as the scientific parameters used by applications.

## 1 Introduction

Scientific computations can often be specified and automated as workflows [16] [5] that comprise many data sets and applications. Manual analysis of the results of a large scale many-task scientific computation is generally unfeasible. This involves, for instance, checking inputs and outputs of each component application of the workflow, verifying if jobs failed on remote computational resources, and checking all processes that contributed to the creation of a particular data set. Many of these activities can be done automatically by querying provenance information, since it describes how these scientific workflows were designed and sometimes how they evolved [7] (*prospective provenance*), and they were executed (*retrospective provenance*). Retrospective provenance describes, for instance, the input and output relationships between data sets and processes. Provenance information can also be used to verify the results of scientific workflows through re-execution. Other applications of provenance information include pre-publication review of results; learning research practices and protocols; and support for taking over someone else's work. To allow the exchange of retrospective provenance information between provenance management systems, the Open Provenance Model (OPM) [10] was proposed.

In this work, we survey queries used for exploring provenance information about many-task computations. We present a set of patterns that can be identified in these queries, which is being used as a basis for the design and implementation of a provenance management system for many-task scientific computations, integrated to the Swift [14] parallel scripting system. It extracts provenance information from log files generated by Swift after the execution of parallel scripts that specify many-task scientific computations, and stores it in a relational database. Instead of proposing an ad-hoc query language for provenance, our approach leverages the robustness of relational database management systems, with the implementation of functions and stored procedures that simplify query design. In the subsequent sections we present provenance query patterns commonly found in many-task scientific computations, and describe how they are supported in our provenance management system.

## 2 Provenance Query Patterns

Provenance management systems can gather consumption and production relationships between data sets and processes, hierarchical relationships between data sets, versioning information about scientific workflows (expressed in Swift as parallel scripts) and their component applications, runtime information about external applications invoked from within a scientific workflow, and allow for the users to enrich their provenance records with annotations about provenance entities. To explore all this information, a provenance management

system should provide a usable and useful query interface. In this section we survey patterns for querying provenance in scientific computations, which is an important step in determining solutions to issues such as indexing strategies, database partitioning, and query abstractions. Our survey is based on the queries proposed for the first three international Provenance Challenges [1]; on provenance queries found in the academic literature; and on provenance queries designed in collaboration with Swift's scientific users. The First Provenance Challenge (PC1), which focused on demonstrating provenance system features, and the Second Provenance Challenge (PC2), which focused on interoperability, share the same queries. The Third Provenance Challenge (PC3) focused on the use of OPM to exchange data between provenance systems. In our experience in PC3 [8], we observed that performing provenance queries in plain SQL is often cumbersome, due to the extensive use of relational joins, for instance, that are not easy for a domain scientist to master and write.

Our current provenance management system prototype is integrated into Swift [15] [14], a parallel scripting system that supports the specification, and execution of large-scale many-task computations on parallel and distributed systems. It is a successor of the Virtual Data System (VDS) [6]. In Swift, scientific workflows are specified using SwiftScript, a high-level language that supports constructs such as conditional branching, loops, data types, mapping of in-memory data structures to in-disk data, and compound procedures. Procedures that are independent of each other are executed in parallel. The data model we propose for provenance of many-task scientific computations is a refinement of the one used during the Third Provenance Challenge [8]. It is similar to OPM [10] having entities that correspond to its notions of artifact, process, and artifact usage (either being consumed or produced by a process). These are augmented with entities used to represent scientific workflows, and to allow for entity annotations. These annotations have the purpose of collecting variable information about entities, such as versioning and scientific parameters. This data model also takes into account characteristics of parallel and distributed computing environments, where processes compete for execution slots, or are not even able to execute due to failures. The following entities are part of this data model:

*Data set*. Corresponds to OPM's artifacts. Data set types can be atomic or composite. Atomic types are given by primitive types, such as integers and strings, or *mapped* types. Mapped types are used for declaring and accessing data that is stored in files. Composite types are given by structures and arrays. Containment relationships define a hierarchy where each data set may have child data sets (when it is a structure or an array), or a parent data set (when it is a member of a structure or of an array). A data set may have as attributes a value, when it is an in-memory variable; or a filename, when it is a file.

*Process*. Can take data sets as input, perform some computation, and produce data sets as output. In Swift, processes can be given by invocations of external applications, and internal procedures, built-in functions, and operators; each process is defined in the context of a scientific workflow, specified in a Swift parallel script.

*Application invocation*. A type of process that is given by an invocation of a component applications of a scientific workflow. In Swift, it is given by an invocation to an external application called from a Swift parallel script. These external applications are listed in an application catalog along with the computational resources where they can be executed.

*Application execution*. Are given by execution attempts of an external application. Each invocation of an external application triggers one or more execution attempts, where a particular computational resource will be selected to actually execute the application.

*Script run*. Refers to the execution (successful or unsuccessful) of a many-task scientific computation, which is specified in a Swift parallel script. A set of runs of the same Swift parallel script can be referred to as a *campaign*.

*Annotation*. A name-value pair associated with either a data set, process, or script run. This is generally used to gather context-specific information about the entities of the provenance data model.

The production and consumption relationships between processes and data sets define a lineage graph that can be traversed to determine ancestors or descendants of a particular entity. A process dependency and a data dependency graph can be derived from this lineage graph by transitivity. The provenance model presented in this work is an evolution of the one used by Swift during the Third Provenance Challenge [8], which was shown to be similar to the Open Provenance Model (OPM) [10]. Process has the same meaning as in OPM, however our model distinguishes different types of processes, such as external applications, and internal functions and operators. Since distributed systems are subject to failures, each external application invocation may have one or more execution attempts. Each of these attempts is captured by our model. Runtime information such as CPU and memory usage can be captured for each application execution. Data sets correspond to OPM's artifacts and can be given by files, in-memory values or a collection of other data sets. Currently there are no entity sets that would correspond to OPM's agents, however we plan to extend the current model to capture which user credentials were used to submit tasks to computa-

tional resources. There are tools in the current implementation that can export the provenance database into a provenance graph in OPM format, which enables better interoperability with other provenance systems that also support OPM.

In the context of many-task scientific computations, provenance queries can be application-independent, where the user is usually interested in information that is present in every run of a Swift script, such as production and consumption relationships between data sets and processes, and data set containment relationships. We identified the following patterns for application-independent provenance queries:

*Entity Attribute* (EA). Queries for attributes of an entity of the data model. Some of these queries may also be application-specific, as with the variable entity set, that stores values of atomic in-memory data sets that can be given by some scientific parameter of an application.

*One-step Relationship* (R). Queries for entities involved in a relationship of the data model, such as data set consumption and production, or data set containment.

*Multiple-step Relationship* ($R^*$). Queries for entities involved in the transitive closure of a relationship of the data model, e.g. for data set lineage and for data set containment hierarchy.

*Lineage Graph Matching* (LGM). Queries for determining similarity between lineage graphs. It can include a combination of EA, R, and $R^*$ queries, along with graph similarity algorithms to verify, for instance, common subgraphs, and graph difference. These problems are known to have high computational complexity in general.

One can observe that each application-independent query pattern is a generalization of the preceding one. Application-independent queries can be used to describe the structure of production and consumption provenance relationships. However, to understand and analyze the results of a computational experiment, a scientist often needs information that goes beyond structural provenance. Some query patterns depends more on which specific scientific computation was executed, since they involve queries about inherent attributes of a scientific application, such as values for input and output scientific parameters:

*Run summary* (RS). Given by application specific attributes of a script run or of the entities it contains (all its processes and their respective input/output data sets). Two special cases of this pattern are frequently found:

- *Run resource-level performance* (RRP). Given by information about the runtime behavior of the application executions of the script run. Some of this information is available as attributes of the process entity, therefore this pattern can be considered a special case of the EA pattern. In Swift, one can optionally monitor and record resource usage statistics such as memory allocation and the amount of data read or written to the file system by an application execution.
- *Run science-level performance* (RSP). Given by queries for input and output scientific parameters. These are gathered either as entity attributes, when given by scientific parameters that are eventually stored in in-memory variables during a script run; or as annotations, when this is information is not directly visible to Swift.

*Run comparisons* (RCp). These queries compare multiple script runs with respect to some attribute (scientific or runtime, for instance) to analyze how it varied across them. One might one to know what was the range of versions of a particular application, or which protein was modeled in a bioinformatics workflow, across multiple script runs.

*Run correlations* (RCr). Given by queries for correlating attributes from multiple script runs. One can, for instance, correlate the resulting accuracy of some computational model run (science-level performance) with the duration of its execution (resource-level performance).

*Campaign-level summaries* (CLS). These queries aggregate data from a *campaign*. Some examples of this pattern pattern would be querying for average computational model accuracy, or for total number of computational tasks in a campaign.

Table 1 describes which patterns are present in each query proposed in the Provenance Challenge series. One can observe that some patterns, such as LGM and RCr, are present in only a few queries of the Provenance Challenges. However some useful queries can be designed exploring these patterns. Users of our provenance management system, from different scientific domains, were frequently interested in queries that present application-specific patterns, in particular for correlations between scientific-level and resource-level performance attributes. The Open Protein Simulator (OOPS) [9] is a protein structure prediction application. It is used in conjunction with pre-processing and post-processing applications in a high-level workflow that is described in figure 1. The doLoopRound workflow activity is a compound procedure, described in figure 2. Annotations are gathered by an application-specific script executed after an OOPS run, and stored in the provenance database. Scientists were interested to know, for instance, what the correlation between *root mean square distance* (RMSD), which measures how similar the modeled protein is to the actual one, and the number of simulation (loopModel) steps was for a given protein. This type of query, which presents the RCr pattern, can help the scientist to esti-

Table 1: Provenance query patterns found in the Provenance Challenge series.

| Pattern | PC1/PC2 | | | | | | | | | PC3 | | | | PC3 (Optional Queries) | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 | 2 | 3 | 5 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| EA | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × | × |
| R | × | × | × | | × | × | × | × | × | × | × | × | × | | × | × | × | × | × | × | × | × | × | × | × | × | × | × |
| R* | × | × | × | | | × | × | | | × | × | × | × | | × | | × | | | | × | × | | | × | × | × | × |
| LGM | | | | | | | × | | | | | | | | × | | | | | | | | | | | | | |
| RS | × | × | × | | | | | | | × | × | × | × | × | | × | × | | | | | | × | × | × | × | × | |
| RCp | | | | | × | × | × | × | × | | | | | | × | | | × | × | × | × | × | | | | | | × |
| RCr | | | | × | | | | | | | | | | | | | | | | | | | | | | | | |

mate the required number of simulation steps for achieving the desired accuracy. This is useful since extra simulation steps are usually computationally expensive. The following query returns the desired answer for the protein TR567:

```
SELECT run_id, r.value as nSim, t.value as rmsd
FROM   compare_run_by_param('proteinId') as r
       INNER JOIN
       compare_run_by_param('nSim') as s USING (run_id)
       INNER JOIN
       compare_run_by_annot('rmsd') as t USING (run_id)
WHERE  r.value='TR567' and run.id LIKE 'psim.loops%';

             run_id                | nSim |  rmsd
-----------------------------------+------+---------
 psim.loops-20100604-2215-cdifsnb3 |  256 | 3.33123
 psim.loops-20100613-0125-keyyyc35 |  512 | 0.76274
 psim.loops-20100616-1512-h6q4g4ja | 1024 | 0.68426
 ...
```

Where `compare_run_by_param` and `compare_run_by_annot` are functions that abstract the RCp pattern. They compare how the value of a parameter and the value associated to an annotation key varies across different runs respectively. The composition of these functions allows for the design of queries that present the RCr pattern. Queries of the R* pattern are supported by the SQL function `ancestors`, that uses Common Table Expressions to define a recursive query.

```
CREATE OR REPLACE FUNCTION ancestors(varchar)
RETURNS SETOF varchar AS $$
  WITH RECURSIVE anc(ancestor,descendant) AS
  (
       SELECT parent AS ancestor, child AS descendant
       FROM   prov_graph
       WHERE child=$1
     UNION
       SELECT prov_graph.parent AS ancestor,
              anc.descendant AS descendant
       FROM   anc, prov_graph
       WHERE  anc.ancestor=prov_graph.child
  )
  SELECT ancestor FROM anc
$$ LANGUAGE SQL;
```

Where `prov_graph` is a database view that defines the edges of the provenance graphs stored in the database. An invocation of this procedure returns:

```
SELECT *
FROM   ancestors('dataset:20100618-0402-ia0bqb73:72000045');

                  ancestor
-----------------------------------------------
 execute:psim.loops-20100618-0402-qhm9ugg4:451006
 dataset:20100618-0402-ia0bqb73:72000039
 ...
```

The EA, R and RS query patterns are well supported using native SQL. Support for LGM queries is being developed in our current work.

## 3 Implementation Overview

In this section, we briefly describe describe our ongoing work for designing and implementing a provenance management system for many-task scientific computations, integrated to the Swift parallel scripting system. The provenance information it manages is extracted on a per script run basis from log files generated by Swift. This information is stored in a relational database using a database schema that implements our data model and contains abstractions implemented as functions and stored procedures for some of the patterns presented that are harder to express with relational database queries, such as the R* and RCr queries.

Relational database management systems are well known for their robustness and scalability, however some of their shortcomings for managing provenance are the use of fixed schemas, and weak support for recursive queries. Despite using a fixed schema, our data model allows for name-value pair annotations for each provenance entity, which gives it some flexibility to store information not explicitly defined in the schema. The SQL:1999 standard introduced native constructs for performing recursive queries, which only recently were implemented in major relational database systems. Ordonez [12] proposed recursive query optimizations that can enable transitive closure computation in linear time complexity on binary trees, and quadratic time complexity on sparse graphs. We leverage the constructs intro-

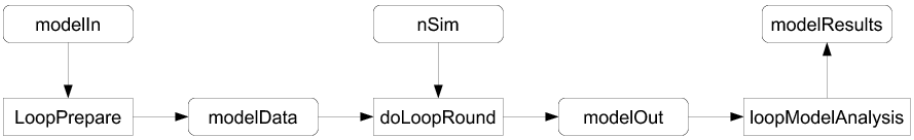


Figure 1: OOPS workflow.

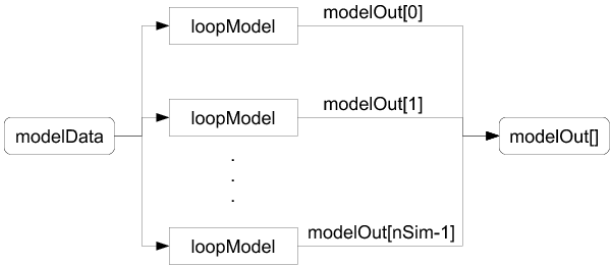


Figure 2: doLoopRound compound procedure.

duced for recursive queries by implementing a function that abstract queries matching the R* pattern. Relationship transitive closures, which are required by R* pattern queries, are well supported by graph-based data models, however many interesting queries require aggregation of entity attributes. These aggregations can be costly in graph-based data models since retrieving entity attributes require graph traversals, whereas in the relational data model they are straightforward.

Annotations can be gathered by writing application-specific annotation extraction scripts that are automatically run before each application execution. We did some experiments using annotations to store domain-specific parameters contained in data sets that were opaque to Swift, and with application versioning using annotations to store version of component applications that are available through SVN repositories.

## 4 Related Work

The Virtual Data System (VDS) [6] [17] [4] defines a data provenance model, to represent information about virtual data objects and the functional procedures that were used to produce them. A relational database schema is used to record relationships between datasets and procedures. Name-value pairs are used to annotate the various entities of the model with application specific information. One problem with this model is the use of different data models, relational and semi-structured, to query provenance information, which makes it difficult to compose them into more complex queries. The Vistrails [7] workflow management system supports exploratory computational scientific experiments, and therefore keeps track of how workflow specifications evolve, in addition to derivation relationships between data and processes. vtPQL [13] is a language for querying provenance information in Vistrails. The query system is augmented by useful constructs in the context of provenance and workflows, such as functions for provenance graph traversal. The underlying data models used for storage include both XML, for storing workflow evolution information, and relational databases, for storing execution information. Anand et al. [2] advocate the representation of provenance information as fine-grained relationships over nested collections of data. For this purpose they present a provenance model that also supports multiple invocations of the same process. This model allows for multiple processes operating on the same nested data collection. They present a Query Language for Provenance (QLP) that is independent from the underlying data model used for storage, independent of workflow management system, and closed under query composition. QLP operators for querying lineage act as filters over lineage relations, returning a subset of them. Lineage queries can be combined with queries over data structures. PASS' Path Query Language (PQL) [11] uses a query language that has a graph-oriented query model. Chebotko et al. [3] present RDFProv, a provenance management system that is based on semantic web techniques and that uses relational databases to store provenance data.

With respect to these related works, Swift's current provenance management system enables gathering additional runtime details that are important in the context of parallel and distributed systems. The use of annotations enriches provenance information with domain-specific

information. Using functions and stored procedures in the relational data model that abstract the query patterns presented and explore information from these different domains allows for the design of useful queries that can be more difficult to express in other systems.

## 5 Concluding Remarks

This work presents a survey about provenance query patterns for many-task scientific computations. The identification of these patterns is important for supporting the design and implementation of provenance management systems with respect to the choice of appropriate data models, storage strategies, and query interfaces. Our current implementation gathers provenance about processes and data sets manipulated by Swift during a script run. The initial implementation of our provenance management system in the relational data model used functions and stored procedures to abstract each query pattern and their respective joins to allow easier query design. We are currently designing a provenance query language similar to SQL that further simplifies query design by avoiding some of SQL's restrictions, such as static typing, while maintaining useful features such as aggregations.

Once provenance management systems become mature, useful and usable, it is likely that large scale distributed computing infrastructures, such as the Teragrid, will start deploying provenance stores to support their users. This will likely raise issues about data integration and scalability in provenance management systems, which are some of the problems we plan to investigate in our future work.

## Acknowledgments

This work was supported in part by CAPES, CNPq, and the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

## References

[1] Provenance Challenge Wiki. http://twiki.ipaw.info, 2011.

[2] ANAND, M., BOWERS, S., MCPHILLIPS, T., AND LUDÄSCHER, B. Exploring scientific workflow provenance using hybrid queries over nested data and lineage graphs. In *Scientific and Statistical Database Management*, vol. 5566 of *Lecture Notes in Computer Science*. Springer, 2009, pp. 237–254.

[3] CHEBOTKO, A., LU, S., FEI, X., AND FOTOUHI, F. RDFPROV: A relational RDF store for querying and managing scientic workow provenance. *Data & Knowledge Engineering 69*, 8 (2010), 836–865.

[4] CLIFFORD, B., FOSTER, I., VOECKLER, J., WILDE, M., AND ZHAO, Y. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience 20*, 5 (2008), 565–575.

[5] DEELMAN, E., GANNON, D., SHIELDS, M., AND TAYLOR, I. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems 25*, 5 (2009), 528 – 540.

[6] FOSTER, I., VÖCKLER, J., WILDE, M., AND ZHAO, Y. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. *International Conference on Scientific and Statistical Database Management* (2002).

[7] FREIRE, J., SILVA, C., CALLAHAN, S., SANTOS, E., SCHEIDEGGER, C., AND VO, H. Managing rapidly-evolving scientific workflows. In *Provenance and Annotation of Data*, vol. 4145 of *Lecture Notes in Computer Science*. Springer, 2006, pp. 10–18.

[8] GADELHA, L., CLIFFORD, B., MATTOSO, M., WILDE, M., AND FOSTER, I. Provenance management in Swift. *Future Generation Computer Systems 27*, 6 (2011), 775 – 780.

[9] HOCKY, G., WILDE, M., DEBARTOLO, J., HATEGAN, M., FOSTER, I., SOSNICK, T., AND FREED, K. Toward Petascale ab initio Protein Folding through Parallel Scripting. Tech. Rep. ANL/MCS-P1645- 0609, Argonne National Laboratory, 2009.

[10] MOREAU, L., CLIFFORD, B., FREIRE, J., FUTRELLE, J., GIL, Y., GROTH, P., KWASNIKOWSKA, N., MILES, S., MISSIER, P., MYERS, J., PLALE, B., SIMMHAN, Y., STEPHAN, E., AND DEN BUSSCHE, J. V. The open provenance model core specification (v1.1). *Future Generation Computer Systems 27*, 6 (2011), 743 – 756.

[11] MUNISWAMY-REDDY, K., BRAUN, U., HOLLAND, D., MACKO, P., MACLEAN, D., MARGO, D., SELTZER, M., AND SMOGOR, R. Layering in Provenance Systems. In *Proc. of the USENIX Annual Technical Conference* (2009).

[12] ORDONEZ, C. Optimizing Recursive Queries in SQL. In *Proc. ACM SIGMOD Conference* (2005), pp. 834–839.

[13] SCHEIDEGGER, C., KOOP, D., SANTOS, E., VO, H., CALLAHAN, S., FREIRE, J., AND SILVA, C. Tackling the Provenance Challenge One Layer at a Time. *Concurrency and Computation: Practice and Experience 20*, 5 (2008), 473–483.

[14] WILDE, M., FOSTER, I., ISKRA, K., BECKMAN, P., ZHANG, Z., ESPINOSA, A., HATEGAN, M., CLIFFORD, B., AND RAICU, I. Parallel scripting for applications at the petascale and beyond. *Computer 42*, 11 (2009), 50–60.

[15] ZHAO, Y., HATEGAN, M., CLIFFORD, B., FOSTER, I., LASZEWSKI, G., RAICU, I., STEF-PRAUN, T., AND WILDE, M. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In *Proc. 1st IEEE International Workshop on Scientific Workflows (SWF 2007)* (2007), pp. 199–206.

[16] ZHAO, Y., RAICU, I., AND FOSTER, I. Scientific Workflow Systems for 21st Century, New Bottle or New Wine? In *Proc. IEEE Congress on Services - Part I* (2008), pp. 467–471.

[17] ZHAO, Y., WILDE, M., AND FOSTER, I. Applying the Virtual Data Provenance Model. In *Proc. 1st International Provenance and Annotation Workshop (IPAW 2006)* (2006), vol. 4145 of *LNCS*, Springer, pp. 148–161.