Provenance Traces of the Swift Parallel Scripting System

Luiz M. R. Gadelha Jr.[1], Michael Wilde[2,3], Marta Mattoso[4], Ian Foster[2,3,5]
[1]National Laboratory for Scientific Computing, Brazil
[2]Mathematics and Computer Science Division, Argonne National Laboratory, USA
[3]Computation Institute, Argonne National Laboratory and University of Chicago, USA
[4]Computer and Systems Engineering Program, Federal University of Rio de Janeiro, Brazil
[5]Department of Computer Science, University of Chicago, USA
lgadelha@lncc.br, wilde@mcs.anl.gov, marta@cos.ufrj.br, foster@anl.gov

ABSTRACT

In this abstract, we describe provenance traces generated from executions of scientific workflows managed by the Swift parallel scripting system. They follow a provenance data model, used by MTCProv, the provenance management component of Swift. It is similar to PROV, representing most of its core concepts and including additional information about the scientific domain, computational resource consumption, and prospective provenance. We describe provenance queries that follow patterns commonly found in high performance computing and that are straightforward to support with MTCProv's built-in procedures. These queries often involve costly relational join operations and recursion, providing a relevant case for benchmarking.

1. INTRODUCTION

The Swift parallel scripting system [7] allows for specifying, executing and analyzing scientific workflows given by many computational tasks. Its execution engine supports environments commonly found in parallel and distributed systems. It is highly scalable and has been used for running large-scale scientific computations [6]. Swift optionally produces provenance information in its log files, and this information has been exported to relational databases using a data model [1] similar to OPM [4] and PROV [5]. MTCProv [3] extends provenance management in Swift by recording prospective provenance, annotations, and runtime resource consumption. It has a query interface which contains built-in procedures that abstract commonly used provenance queries [2] and automatically computes relational join expressions.

2. SUMMARY OF SUBMISSION

We have executed and generated provenance traces for the following scientific workflows: examples and tutorials that come in the standard distribution of Swift; demonstration scientific workflows for processing satellite images from MODIS and for ray tracing with C-Ray; and a scientific workflow for parallelizing BLAST sequence alignments through database partitioning. Table 1 has a summary of our provenance trace submission, which is basically given by SQL statements that create a relational database containing both prospective and retrospective provenance of the scientific workflows mentioned. We are using our own format for representing provenance, since it stores prospective provenance, which is not modeled by PROV.

Data format	SQL
Data model	Relational (MTCProv [3])
Size	10.4MB
Tools used for generating provenance	Swift [7], MTCProv [3]
Submission group	Swift
Contact	`swift-user@ci.uchicago.edu`
License	Creative Commons Attribution-Share Alike 3.0 Unported License

Table 1: Summary of submission.

3. EXPERIENCE STATEMENT

The basic process of generating the provenance traces consisted of enabling a configuration option in Swift for provenance logging, which adds lines containing provenance information to the log file associated to the execution of a scientific workflow. These log files were processed, *a posteriori*, by MTCProv to extract relevant provenance information and insert it into the provenance database. A dump of the provenance database containing the executions of the scientific workflows mentioned in subsection 2 was generated for submission.

4. APPLICATION

The provenance traces submitted can support the evaluation of provenance management systems for parallel and distributed scientific workflows. They capture core provenance information, such as prov:used and prov:wasGeneratedBy relationships, and additional information that scientists are usually interested in querying, such as science domain annotations and computational resource consumption by workflow activities. Other issues of parallelism and distribution are also captured, such as an activity invocation having possibly many execution attempts, due to failures; redundant submission of computational tasks to improve execution throughput. Some queries involve performing many relational join operations and recursion, providing a relevant case for benchmarking provenance management systems.

5. POSSIBLE PROVENANCE QUERIES

In this section we describe queries that can be performed using the provenance traces provided. Example queries implemented in either SQL or SPQL [3] can be found in [3]. In [2], we surveyed typical provenance queries. One typical provenance query involves determining transitive closures of causal relationships, such as the ones defined by the prov:used and prov:wasGeneratedBy. MTCProv supports such queries through procedures such as *ancestors*, which recursively determines the activities and entities that preceded a particular activity or entity through prov:used or prov:wasGeneratedBy relationships. Another query pattern, particularly common in high-performance scientific computations, is determining the execution behavior of particular activities of a scientific workflow and to correlate it to other information, such as scientific domain parameters. This query pattern is available through the *compare_run* procedure, which can be used to compare multiple workflow executions according to a number of parameters, resource consumption data, or annotations. Next we describe, in English, possible queries that can be performed on MTCProv's database: What are the values of scientific parameters of activities across scientific workflow executions? What amount of computational resources was consumed by these activities? What was the average number execution attempts per activity invocation? How can one correlate information from both of these domains (scientific parameters and runtime behavior)? Given a file produced as output by a scientific workflow, which activities and files lead to this file through consumption and production relationships? These queries illustrate common query patterns used by scientists that execute parallel and distributed scientific workflows. Despite not being considered core provenance information, resource consumption data about scientific workflow activities is relevant in this context since access to high performance facilities is usually under high demand, making it useful for profiling and optimization.

6. COVERAGE OF PROV

The provenance model used by MTCProv is straightforward to map to PROV [5]. prov:Activity and prov:Entity correspond, for instance, to the MTCProv's database entities *FunctionCall* and *Dataset* respectively. While we do not explicitly define an entity type equivalent to prov:Agent, this information can be stored as annotations for *FunctionCall* entries. The dependency relationships, prov:used and prov:wasGeneratedBy correspond to our *consumes* and *produces* relationships, respectively. Table 2 describes PROV-O terms covered by MTCProv. Our data model has additional entity sets to capture behavior that is specific to parallel and distributed systems, to distinguish, for instance, between application invocations and their execution attempts. Based on this concept correspondence, MTCProv provides tools for exporting retrospective provenance stored in the database to PROV, providing interoperability with other PROV-compatible provenance systems.

PROV-O terms	Covered	Comments
prov:Activity	Yes	
prov:Agent	No	Can be expressed through annotations.
prov:Entity	Yes	
prov:actedOnBehalfOf	No	
prov:endedAtTime	Yes	
prov:startedAtTime	Yes	
prov:used	Yes	
prov:wasAssociatedWith	No	
prov:wasAttributedTo	No	
prov:wasDerivedFrom	Yes	It is not present in the database schema but can be queried through a view.
prov:wasGeneratedBy	Yes	
prov:wasInformedBy	No	

Table 2: Coverage of PROV.

7. CONCLUDING REMARKS

Through MTCPRov, Swift users can collect, store, and query provenance data about parallel and distributed scientific workflows. The provenance traces provided illustrate the range of information and events captured by MTCProv, allowing for queries on core provenance information and on extended data that is relevant in the context of high performance scientific computations. They can also be useful in benchmarking provenance management systems based on relational databases, since typical queries usually involve costly join operations and recursion. The database schema used by MTCProv covers most of the core concepts of PROV, enabling straightforward exporting of provenance records to this standard. As future work, we plan to enable online recording of provenance to help in workflow execution steering and improving compatibility with PROV.

8. REFERENCES

[1] L. Gadelha, B. Clifford, M. Mattoso, M. Wilde, and I. Foster. Provenance management in swift. *Future Generation Computer Systems*, 27(6):775–780, 2011.

[2] L. Gadelha, M. Mattoso, M. Wilde, and I. Foster. Provenance query patterns for many-task scientific computing. In *Proceedings of the 3rd USENIX Workshop on Theory and Applications of Provenance (TaPP'11)*, 2011.

[3] L. Gadelha, M. Wilde, M. Mattoso, and I. Foster. MTCProv: a practical provenance query framework for many-task scientific computing. *Distributed and Parallel Databases*, 30(5-6):351–370, 2012.

[4] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. Van den Bussche. The open provenance model core specification (v1.1). *Future Generation Computer Systems*, 27(6):743–756, 2011.

[5] L. Moreau, P. Missier, K. Belhajjame, S. Cresswell, Y. Gil, R. Golden, P. Groth, G. Klyne, J. McCusker, S. Miles, J. Myers, and S. Sahoo. The PROV data model and abstract syntax notation. Technical report, World Wide Web Consortium (W3C), December 2011.

[6] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, and I. Raicu. Parallel scripting for applications at the petascale and beyond. *Computer*, 42(11):50–60, 2009.

[7] M. Wilde, M. Hategan, J. Wozniak, B. Clifford, D. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):634–652, 2011.