Connecting Scientific Data to Scientific Experiments with Provenance

Simon Miles[1], Ewa Deelman[2], Paul Groth[3], Karan Vahi[2], Gaurang Mehta[2], Luc Moreau[3]
[1] *Department of Computer Science, King's College London, UK*
[2] *Information Sciences Institute, University of Southern California, US*
[3] *Electronics and Computer Science, University of Southampton, UK*

Abstract

As scientific workflows and the data they operate on, grow in size and complexity, the task of defining how those workflows should execute (which resources to use, where the resources must be in readiness for processing etc.) becomes proportionally more difficult. While "workflow compilers", such as Pegasus, reduce this burden, a further problem arises: since specifying details of execution is now automatic, a workflow's results are harder to interpret, as they are partly due to specifics of execution. By automating steps between the experiment design and its results, we lose the connection between them, hindering interpretation of results. To reconnect the scientific data with the original experiment, we argue that scientists should have access to the full provenance of their data, including not only parameters, inputs and intermediary data, but also the abstract experiment, refined into a concrete execution by the "workflow compiler". In this paper, we describe preliminary work on adapting Pegasus to capture the process of workflow refinement in the PASOA provenance system.

1. Introduction

Today, workflows are used by many researchers in a range of sciences [15]. Workflow execution systems provide a means to coordinate the execution of thousands of tasks accessing Petabytes of data. Given the size of analyses, a major challenge for scientists is the interpretation of results produced by workflows, as potentially many steps are involved in generating a particular data product and the computations are often executed in a distributed environment. For example, a popular astronomy application, Montage [2] produces science-grade mosaics of the sky on demand. This application can be structured as a workflow that takes images, projects them, adjusts their backgrounds, and adds them together. A mosaic of 6 degrees square involves processing 1,444 input images, requires 8,586 computational steps and generates 22,850 intermediate data items. For a scientist to verify the quality of the final mosaic, he/she may need to check that an input image was retrieved from a specific archive, the reprojections' parameters were set correctly, the execution platforms used did not have processors with a known floating point processing error, etc.

Given the complexity of workflows with thousands of computational steps executing across multiple distributed resources, it is infeasible for users to directly define the executable workflow. Often, researchers use "workflow compilers" such as Pegasus [6, 7] to generate the executable workflow from a high-level, resource-independent description of the end-to-end computation (an *abstract workflow)*. However, the additional workflow mapping also increases the gap between what the user defines and what is executed by the system and so complicates interpretation of results: the connection between scientific results and the original experiment is lost.

In this paper, we present a proof-of-concept solution for this issue, based on technology for determining the provenance of data, i.e. the process by which they were produced.

1.1 Provenance systems

Many workflow systems now support some mechanism by which execution can be tracked [16] so that the *provenance* of results can later be determined. Most such mechanisms focus on documenting how execution steps, parameters, and intermediate data produce the final results.
Therefore, while these provenance systems allow the connection between data items to be made more evident, they do not provide the connection between results and steps in the original, abstract, high-level workflow description; the provenance merely describes the process of the compiled (*executable*) workflow.

1.2 Connecting data to experiments

The key contribution of this paper is to recognize that scientific workflows, whether abstract or concrete, are

also first-class data, and that provenance mechanisms designed for tracking data can also be used to track processing applied to workflows. Combining provenance of workflows with that of data they produce, gives a comprehensive solution that allows scientists to relate scientific data to scientific workflows.

This paper describes preliminary work in recording workflow refinements (transformations) made by Pegasus so that this information can help users understand the relationship between the executed workflow and its abstract precursor defined by the user. Such a record contributes to the *full provenance* of the scientific data. We provide the ability to produce and access this information via a provenance management system. The technical contributions of this paper are:

- A model for defining workflow transformations conducted by workflow compilers such as Pegasus.
- A mapping of that model to that of the PASOA provenance system, and a description of its implementation.
- An evaluation of approach's cost in an astronomy application, measured by the overhead to the workflow compiler in documenting its execution.
- An illustration of uses to which scientists can put the provenance in answering questions.

2. Workflow Compilation/Refinement

Our work is based on the Pegasus workflow compiler, which maps high-level, abstract workflow descriptions onto available distributed resources. The abstract workflow provided by a user, portal, or another workflow composition system [10] is resource independent. It specifies the computations, their input and output data, and interdependencies between them without indicating where the computations take place, or where data is located. A simple workflow description could define computing the function *F* on an input *x*, generating the output *Y* and placing it at *L*. The process of generating an executable workflow involves the following steps:

a) Find where *x* is from $\{S_1, S_2, \ldots\}$, where S_i is a storage system.
b) Find where F can be computed from $\{C_1, C_2, \ldots\}$, where C_i is a computational site.
c) Choose a site *c* and a storage system *s* subject to constraints (performance, space availability etc.)

As a result, the following executable workflow will be constructed.

1. Copy *x* from *s* to *c*
2. Move *F* to *c*
3. Compute *F(x)* at *c,* obtaining *Y* at *c*.
4. Move *Y* from *c* to *L*
5. Register *Y* in data registry

A description of steps a-c is the executable workflow's provenance while a description of steps 1-5 is Y's provenance within the workflow.

Understanding Y's full provenance requires knowing its connection to the original workflow. Even with this simple workflow, things can go wrong: *x* was not found at *s*, *F(x)* failed, *c* crashed, or there was not enough space at *L*. Given these four types of error messages, the user may only understand the second and last. That *x* was not at *s* is hard to interpret, especially if there are copies of *x* elsewhere, because *s* was not chosen by the user. A workflow system may shield users from some failures, but for others this is impossible.

2.1 Pegasus and workflow refinement

Above, we provided an example of the mapping from abstract to executable workflow. In this section, we present the *refinement* process that Pegasus goes through as it refines information in the abstract workflow towards execution. The main steps are:

- **Reduction** eliminates processing steps when intermediate data products have already been generated (by another workflow or previous execution of this workflow) and can be reused.
- **Site Selection** chooses computational resources on which to execute jobs described in the workflow. This means finding available resources and determining where required executables are already installed or can be staged in. The workflow nodes are annotated with their target execution sites.
- **Data Staging** selects sources of input data for computations, and adds nodes to the workflow to stage this data in and out of the computation sites.
- **Registration** causes final and intermediate data to be registered in a registry, by adding registration nodes to the workflow.
- **Clustering**: The granularity of computations (many short run jobs) or the granularity of data staging (many small data transfers) can be too fine to be efficient. In such cases, Pegasus clusters workflow nodes together to be handled as one in execution.

The refinements result in an executable workflow that looks very different than the abstract one defined by the user.

2.2 Example of a Workflow Refinement

To illustrate how a workflow changes in refinement, we use part of the Montage workflow which reprojects images and takes their differences (Figure 1).

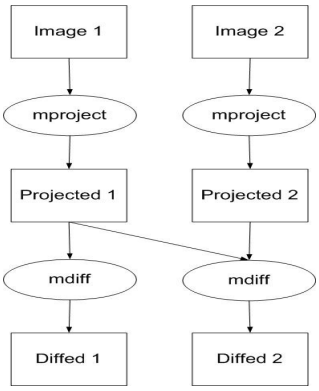

Figure 1. Part of Montage Workflow. Ovals denote computations, rectangles denote data.

Reduction: Based on available data products, Pegasus reduces the workflow: we assume that files "Projected 1" and "Diffed 1" already exist on storage system S1, so do not need to be recomputed. The refined workflow is in Figure 2(a).

Site Selection: Sites for job execution are selected: resources R1 and R2 are chosen (Figure 2(b)).

Data Staging: After consulting a registry, Pegasus adds nodes to transfer the data from their storage sites to where the computations occur. Nodes are also added to transfer outputs back to the storage sites. Here, a node ("Projected 2 R2 > R1" in Figure 3) is added to transfer the intermediate data between sites R2 and R1 so that the computation can be invoked at R1. The intermediate result "Projected 1" from the workflow's first branch is staged in for the mdiff computation.

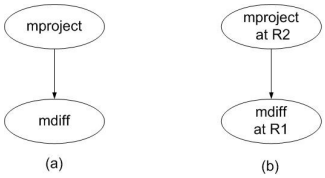

Figure 2: Workflow (a) After Reduction (b) After Site Selection.

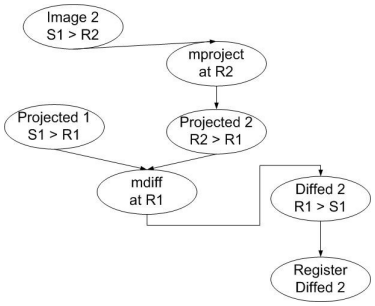

Figure 3: Workflow with data transfer and registration nodes.

Registration: Data registration nodes are added for the workflow's outputs. Registration enables workflow-level checkpointing in case of failures and helps find the data later. Figure 3 shows the workflow after the data transfer and registration nodes are added.

Clustering: Data staging and computations on the same sites can be clustered to improve overall performance. The cluster is shaded in Figure 4.

The workflow is now ready for execution and is sent to a workflow engine (we use Condor DAGMan [9]). After execution, data products requested by the original workflow are available on S1.

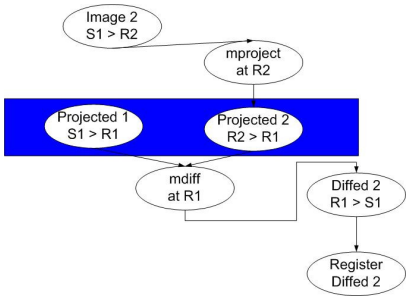

Figure 4: Workflow with node clustering of data transfer tasks.

3. Motivating Questions

From our collaborations with domain scientists who use workflows for analyses, some basic questions are known to be important:

- Which data items were used to generate a particular data product?
- What computations where conducted to generate these data items?
- Where did the computations occur?
- Which version of the software was used?

There are also questions related to the evolution from the abstract to the executable workflow:

- Why is this node in my abstract workflow not in the executable workflow?

- Which intermediate data product was substituted for the actual computation?
- Why, given that the data was at S1, did the workflow use the data at S2?
- Which abstract node does a particular executable node correspond to?
- Why did disk space at location X diminish so much?
- Why is this intermediate data not in the registry?

The first set of questions can be answered by many provenance systems but, to answer the second set, information must be known about refinement, and in a form suitable for answering those questions.

4. Documentation for Provenance

To address the needs of a range of e-Science applications [14], the PASOA and Provenance projects developed an architecture for determining of the provenance of data [12]. The architecture proposes the following lifecycle.

1. **Create**: As an application executes, it also creates a description of its execution, called *process documentation,* comprised of *p-assertions,* assertions about individual process steps.
2. **Record**: Once documentation has been created, it is recorded into a *provenance store*.
3. **Query**: After a data item is produced by an application, users (or applications) obtain the provenance of this data item by querying the store. A query retrieves the p-assertions that describe process by which the data item was produced.

An application that produces and stores process documentation is *provenance-aware*, and in Section 5, we describe how Pegasus has been made provenance-aware.

E-science applications are often composed of multiple independent components that may execute in a variety of environments, so a generic data model for process documentation was developed. Such a data model, shared by all components, allows users to query documentation without having to know which components created it, and for creators to create documentation understood by future unknown queries.

The provenance store is organized to enable the provenance to be determined from documentation by independent distributed sources. Applications are viewed as service-oriented architectures (SOA). In this style, a service is a component that takes inputs and produces outputs. Clients invoke services, which may themselves act as clients for other services; the term *actor* denotes either a client or a service. Communication between actors is by exchanging *messages*, and exchange of one message between actors is an *interaction*. An application's execution is described as the exchange of messages between actors and transformations that actors perform on messages they receive in order to generate new messages.

After mapping an application to an SOA, its execution is captured using three types of p-assertion.

- An **interaction p-assertion** documents an interaction between two actors, including the content of the message exchanged.
- A **relationship p-assertion** documents the function applied to data within an actor's incoming messages to create an outgoing message, i.e. processing done by an actor for an invocation.
- An **actor state p-assertion** documents internal states of an actor that may be important in the execution, e.g. configuration information

Using p-assertions, each actor in the application documents the interactions it participates in and how those interactions are related.

5. Implementation

Following PASOA's approach, we modeled Pegasus in terms of interacting actors. We consider the *refinement phase*, where Pegasus refines an abstract workflow is refined to be executable, and the *enactment phase*, where Condor DAGMan enacts it.

5.1 Refinement Process Documentation

In refinement, Pegasus is modeled as an actor, interacting with five *refiners*, also actors. The flow of this process is in Figure 5. Each arrow is a documented interaction or relationship. The interactions are exchanges of partially refined workflows between Pegasus and a refiner, until the final refiner's output is an executable workflow passed to DAGMan.

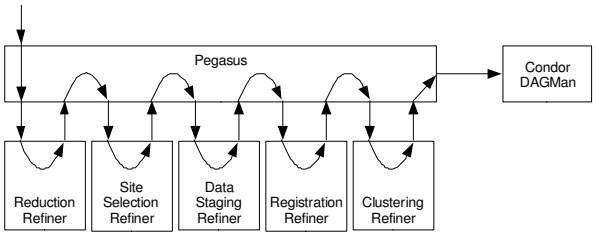

Figure 5: Pegasus workflow refinement modeled as interacting actors.

For each refinement step, five recording actions take place. For explanation, Figure 6 depicts a refinement step and Pegasus' invocation of it. Recording actions, labeled A to E, and described below.

- Prior to each refinement, Pegasus records the current, partially refined workflow that is about to be refined further (A).
- It records relationship p-assertions linking this workflow to the output of the previous refinement (B): these are identical as Pegasus itself does not alter the workflow.
- The refiner records the workflow received prior to its refinement (C).
- It then records the workflow after refinement (D).
- It also records relationships from each workflow node after refinement to nodes that caused it to be as it is in the pre-refinement workflow (E).

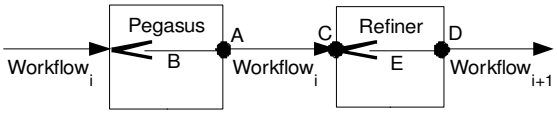

Figure 6: One refiner's documentation.

5.2 Refinement Relationships

As mentioned in the final step above, each refiner documents the relationships between nodes in the workflow as it was before and after refinement. Each relationship's type gives queriers more information on the refinement that occurred.

- **identicalTo:** Denotes that a node has not changed in refinement. The absence of the relationship for a node in the pre-refinement workflow indicates that the node was changed removed in refinement.
- **siteSelectionOf:** Denotes that the post-refinement workflow node is a compute job in the pre-refinement workflow for which the target execution site has been chosen and specified.
- **stagingIntroducedFor** Denotes that the post-refinement node is a data staging operation introduced to stage data in/out for the computation present in the pre-refinement workflow.
- **registrationIntroducedFor:** Denotes that the post-refinement node is a registration operation introduced to follow a stage-out node.
- **clusteringOf:** Denotes that the post-refinement node is a cluster of jobs combining several jobs present in the pre-refinement workflow.

Relationships documented for Section 3's example are summarized in Figure 7. We show the workflow fragment through six stages of refinement, from abstract to executable. The workflow nodes (ovals) at each stage are related (large arrows) to those in the previous stage, with the relationship type (arrow label) stating the function performed by the refiner that transformed the workflow. By tracing relationships backwards, a querier determines the provenance of each concrete job, and how it relates to the original abstract workflow.

5.3 Enactment Process Documentation

The new model for documenting enactment is very similar to that for refinement. DAGMan is modelled as an actor, interacting with each workflow job. DAGMan sends invocation messages, containing command-line arguments including input file names to executable jobs, and completion messages are returned from the jobs containing the names of output files produced.

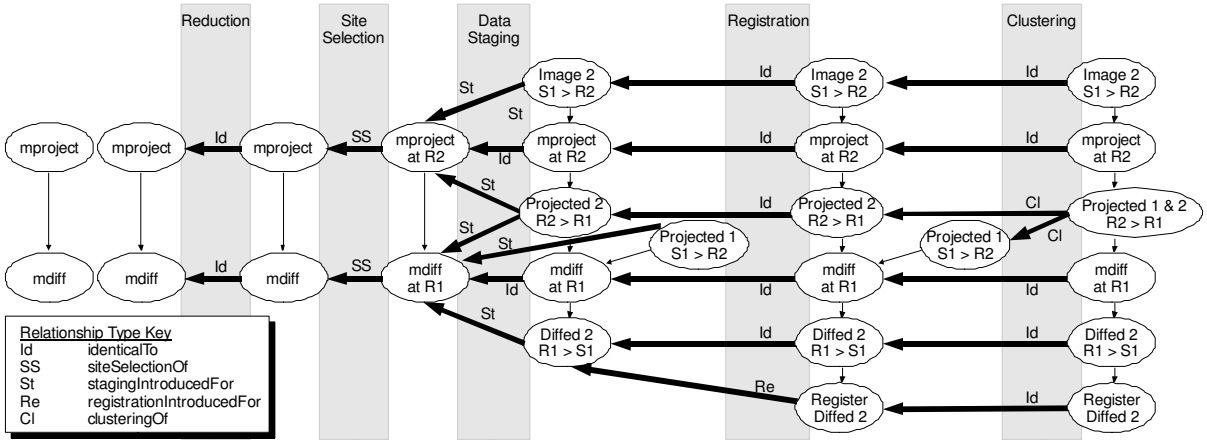

Figure 7: Relationships recorded during refinement.

As with refinement, relationships link nodes from one step to the next, so that provenance can later be determined, as in Figure 7. However, here the nodes are the data items processed by the jobs, referred to by filename, rather than job nodes of the workflow.

Relationships between data items depend on the type of job enacted, e.g. a job invoking 'gzip' would assert a relationship of type 'gzip' between its output and input.

We did not adapt DAGMan itself to record documentation; we automatically added wrapper code to each job, which recorded the appropriate documentation for that job and, by consulting the abstract workflow, the connections between the inputs to that job and the outputs of its parents in the executable workflow.

5.4 Refinement and Enactment Connected

The combination of the documentation for workflow refinement and enactment, allows detailed provenance of a data item to be found:

- For each data item, we can find the concrete workflow steps that produced it and other data items that contributed to those steps.
- For each workflow step, we can find its connection to the abstract workflow jobs from which it was refined.

The full set of documented connections is depicted in Figure 8, an extension of Figure 5 including the documentation of the enactment's interactions and relationships. Jobs in the concrete workflow produced from the refinement lead to the data being produced, so the refinement process is part of the data's provenance.

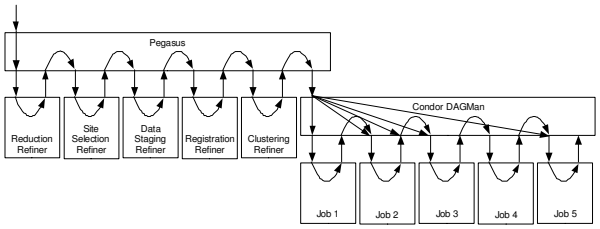

Figure 8: Combined workflow and data process documentation.

5.5 Performance Evaluation

To be realistically usable, the overhead cost of recording process documentation during enactment must be insignificant. While the work presented here is merely a test of the feasibility of the approach, our preliminary results show that this is achievable. We take different sizes of the large-scale Montage workflow, with 0.5, 1, and 2 degrees square for our evaluation. These have 65, 232 and 1444 nodes in their abstract workflows respectively.

Prior to using PASOA, Pegasus recorded some documentation about the enactment of jobs in a database, the Provenance Tracking Catalog (PTC) [24]. The causal connection between data items and jobs were not captured, and the workflow refinement phase was not documented at all. In our experiments, we compare the performance of refining and enacting the workflows for Pegasus with its previous setup and with the new setup, recording all documentation to a PASOA's Web Service provenance store [11]. For brevity, we will refer to these setups as PTC and PASOA respectively.

The workflows were run on a cluster of 7 dual processor 2.4Ghz XEON nodes. A separate host (2GhZ Pentium machine with 1Gb of memory) was used for workflow submission. The Web Service provenance store and the PTC were on the submit host.

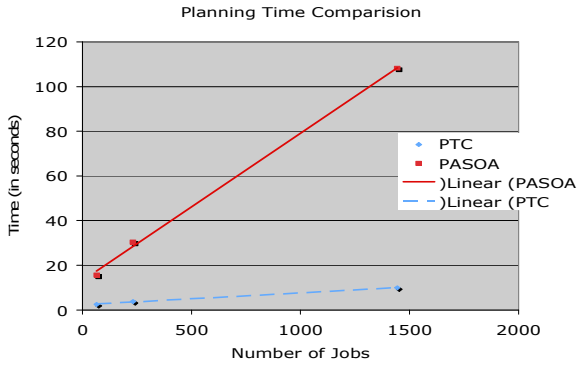

Figure 9: Time to refine workflow with and without recording as workflow size increases

In Figure 9, we show the difference in execution time for Pegasus with and without process documentation recording to PASOA. As can be seen, while there is an overhead to recording, this is linear to the size of the workflow. While future work will much reduce this overhead, through asynchronous recording for example, the increase is already manageable. In Figure 10, we show the total time for refinement and enactment, with recording to PTC (enactment only) and PASOA. The times recorded were comparable, and increased linearly. Additionally, one needs to put these times into the context of the time that Pegasus takes for planning. For example, for the 2 degree Montage workflow, the overhead of recording the refinement provenance is 0.05% of the running time of Pegasus. The biggest overhead was seen for the smallest workflow and was on the order of 0.9%.

We also quantified the amount of disk space used to store both the enactment and refinement provenance information in PASOA. For the three workflow sizes the enactment provenance was 4.7MB, 11.6MB, and 39MB for the 0.5, 1, and 2 degree square mosaics. The corresponding size of the refinement provenance was 3.3MB, 11.6MB, and 23MB.

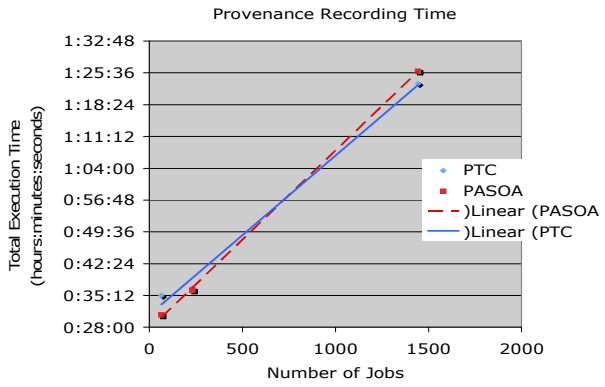

Figure 10: Time for refinement and enactment with PTC and PASOA systems

6. Related Work

Within computer science, Bose and Frew give an overview of provenance related systems [3], Simmhan et al. survey its application to e-Science [18], and further compilations of the state-of-the-art exist [1, 14, 16]. Here, we focus on provenance in workflow-based environments.

Deriving from the myGrid e-Science project, the Taverna workflow enactment engine captures provenance-related data in an RDF based data model [22]. A query application programming interface (API), ProQA, enables a variety of provenance queries to be executed over the provenance RDF graph [23]. As the documentation is stored as RDF, it enables both workflow annotations and provenance information to be queried over simultaneously [21]. Barga and Digiampietri modified Windows Workflow Foundation [5] to support the collection of process documentation. A multi-layered model allowed the size of data stored to be significantly decreased [16]. The Kepler workflow enactment engine aims to support multiple kinds of workflows from bioinformatics experiments using complex, high-level tools to processes for job control and data movement in Grids [13]. With modifications to support explicit dependencies and metadata, the execution of workflows in Kepler can be captured with the Kepler Provenance Recorder [4]. None of the above-mentioned systems support tracking the planning of workflows, and many do not support execution on computational Grids.

While the above systems capture process documentation only from the workflow enactment engine, the Karma Provenance Framework [19, 20] supports the capture of this information both from the workflow enactment engine and from the services used, via a notification model. The provenance model previously used by Pegasus as part of the Virtual Data System (VDS) also captured provenance-related information from both the workflow enactment engine and executing applications [24]. VDS does not, however, store explicit relationships between input and output data, so determining provenance of data relied on access to the same workflow definition as was executed at the time.

As with the above tools, VisTrails provides a graphical user interface for building workflows, but, instead of just capturing the execution of a workflow, VisTrails also captures how workflows are created and edited by the user [17]. As the user modifies a workflow the various changes in the workflow are kept using a mechanism similar to that of versioning systems. When a workflow is run, the user can track back to the particular workflow that was run and more importantly see the evolution of that workflow [8]. VisTrails has similarities to our approach in that it tracks how a workflow is changed before execution. However, it captures only how users change the executable workflow and not how automated compilation processes affect the execution of the workflow. VisTrails is then complementary to our approach, as it tracks creation and modification of executable workflows.

7. Conclusions and Future Work

Understanding the process that ultimately produced a result is critical to interpreting it, and is particularly important when execution steps are not apparent in the original process design. In this paper, we have described our approach to capturing and giving access to a workflow's provenance: the details chosen for its execution by a Pegasus workflow compiler. We connected this to captured documentation of the workflow execution by Condor DAGMan, allowing scientists to determine, for a given result, by what process it was produced, what abstract workflow led to its execution, and every stage between. The connection between an experiment's results and the original experiment steps is thus made evident, even when a scientist delegates execution details to Pegasus. To ensure that the cost of this approach did not excessively slow the workflow's execution, we evaluated our approach against several large-scale workflows.

Future work will examine queries suited to answer common questions regarding workflow provenance, e.g. those in Section 4. We will also tackle questions that relate to failures in workflow execution. The work presented here demonstrates the feasibility of our approach in re-connecting scientific results with the

original experiments from which they are derived, even when execution is large-scale and highly distributed.

Acknowledgments

This work was supported by the National Science Foundation under Cooperative Agreement OCI-0438712, the SoCA and PASOA projects (EPSRC references EP/C528131/1 GR/S67623/01).

8. References

[1] "Provenance and Annotation of Data -- International Provenance and Annotation Workshop, IPAW 2006," 2006.

[2] G. B. Berriman, et al., "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand," in *SPIE Conference 5487: Astronomical Telescopes*, 2004.

[3] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: a survey," *ACM Computing Surveys,* vol. 37, pp. 1-28, 2005.

[4] S. Bowers, et al., "A Provenance Model for Collection-Oriented Scientific Workflows," *CCPE,* 2007.

[5] D. Box and D. Shukla, "WinFX Workflow: Simplify Development With The Declarative Model Of Windows Workflow Foundation," *MSDN Magazine,* vol. 21, January 2006.

[6] E. Deelman, et al., "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming Journal,* vol. 13, pp. 219-237, 2005.

[7] E. Deelman, et al., "Pegasus: Mapping Large-Scale Workflows to Distributed Resources," in *Workflows in e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds.: Springer, 2006.

[8] J. Freire, et al., "Managing Rapidly-Evolving Scientific Workflows.," *IPAW,* vol. 4145, 2006.

[9] J. Frey, et al., "Condor-G: A Computation Management Agent for Multi-Institutional Grids.," *Cluster Computing,* vol. 5, pp. 237-246, 2002.

[10] Y. Gil, et al., "Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows," in *OWL: Experiences and Directions (OWL-ED)*, Athens, GA, 2006.

[11] P. Groth, et al., "PReServ: Provenance Recording for Services," *Proceedings of the UK OST e-Science Fourth All Hands Meeting (AHM05),* September 2005.

[12] P. Groth, et al., "An Architecture for Provenance Systems," Univ. of Southampton October 2006.

[13] B. Ludäscher, et al., "Scientific Workflow Management and the Kepler System," *Concurrency and Computation: Practice & Experience,* 2005.

[14] S. Miles, et al., "The Requirements of Using Provenance in e-Science Experiments," *Journal of Grid Computing,* 2006.

[15] L. Moreau, et al., "The First Provenance Challenge," *Concurrency and Computation: Practice and Experience,* 2007.

[16] L. Moreau, et al., "The First Provenance Challenge," *Concurrency and Computation: Practice and Experience,* 2007.

[17] C. Scheidegger, et al., "Tackling the Provenance Challenge One Layer at a Time,"*Concurrency and Computation: Practice and Experience,* 2007.

[18] Y. L. Simmhan, et al., "A survey of data provenance in e-science," *SIGMOD Record,* vol. 34, pp. 31-36, 2005.

[19] Y. L. Simmhan, et al., "Performance Evaluation of the Karma Provenance Framework for Scientific Workflows," *In.l Provenance and Annotation Workshop, IPAW 2006,* vol. 4145, 2006.

[20] Y. L. Simmhan, et al., "Querying Capabilities of the Karma Provenance Framework," *Con. and Computation: Practice and Experience,* 2007.

[21] J. Zhao, et al., "Annotating, linking and browsing provenance logs for e-Science," *Proc. of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data,* 2003.

[22] J. Zhao, et al., "Using Semantic Web Technologies for Representing e-Science Provenance," *Proceedings of the 3rd International Semantic Web Conference,* vol. 3298, pp. 92-106, 2004.

[23] J. Zhao, et al., "Mining Taverna's Semantic Web of Provenance," *CCPE,* 2007.

[24] Y. Zhao, et al., "Virtual data Grid middleware services for data-intensive science," *CCPE,* vol. 18, pp. 595-608, May 2006.