Provenance: The Bridge Between Experiments and Data

Scientific applications are often structured as workflows compiled from abstract experiment designs. However, the automation of the compilation step often hides details about how results were produced. The authors investigate how provenance helps scientists connect their results with the original experiment's formulation.

More than ever before, managing data today is increasingly difficult in the scientific and business domains. Many projects such as the Laser Interferometer Gravitational Wave Observatory[1,2] and the Earth System Grid[3] are collecting petabyte-scale data sets. For this data's users, however, interpretation issues aren't related only to raw data descriptions but also to the description of the derived data products and the processes that created them. An additional complexity is introduced by execution systems, which can be very heterogeneous and distributed across many locations. To understand data, scientists must therefore be able to interpret metadata about it as well as its provenance (how the data came to be).

The latest trend is to use workflow technologies to manage data processing. Scientific workflows describe computations, their parameters, I/O data, and data or control dependencies between them, and software systems manage these workflows by following dependencies and executing computations on the desired data. Workflow technologies not only help automate complex scientific analyses, but they also provide the opportunity to capture transformations performed on the data.

To illustrate the complexity of today's analyses, we examine a popular astronomy application, called Montage,[4] which produces science-grade mosaics of the sky on demand. We can structure this application as a workflow that takes several images, projects them, adjusts their backgrounds, and adds the images together. A mosaic of 6-degrees square would involve processing 1,444 input images, require 8,586 computational steps, and generate 22,850 intermediate data products. To verify the final mosaic's quality, a scientist might need to check that, for the workflow that produced the mosaic, a particular input image was retrieved from a specific archive, that the parameters for the reprojections were correctly set, that the execution platforms didn't include processors with a known floating-point processing error, and so on.

Given the complexity of workflows with thousands of computational steps, possibly executing across multiple distributed resources, it's infeasi-

1521-9615/08/$25.00 © 2008 IEEE
Copublished by the IEEE CS and the AIP

Simon Miles
King's College London
Paul Groth, Ewa Deelman, Karan Vahi, and Gaurang Mehta
University of Southern California
Luc Moreau
University of Southampton

ble for users to directly define an executable workflow. Resources are often shared with other users and might become suddenly unavailable because of network failures or policy changes. Thus, researchers often use *workflow compilers* such as Pegasus[5,6] to generate an executable workflow based on a high-level, resource-independent description of the end-to-end computation (a so-called *abstract workflow*). This approach gives scientists a computation description that's portable across execution platforms and that can be mapped to any number of resources. However, additional workflow mapping can also increase the gap between what users define and what the system actually executes, thus complicating result interpretation by dropping the connection between the results and the original experiment.

We present a solution that not only describes how computations were executed but also how they were generated and how they relate to the scientist-provided description. In particular, we focus here on the broad principles behind our approach and its benefits for scientist users; a preliminary paper with a prototype-based performance evaluation appears elsewhere.[7] Specifically, we augment the compilation of workflows by Pegasus and their subsequent enactment by Condor DAGMan by recording the steps taken and the connections between them. This augmentation, using technology from the Provenance-Aware Service Oriented Architecture (PASOA; www.pasoa.org) project, allows scientists to determine, for a given data item, what process produced it, what abstract workflow led to its execution, and every stage in between.

Abstract vs. Executable Workflows

The abstract workflow provided by the user, portal, or some other workflow composition system[8] is resource independent, so it specifies the computations, their input and output data, and the interdependencies between them without indicating where the computations occur or where the data is located. A very simple workflow description, for example, could define computing function F on input x, generating output Y, and storing it on storage system resource S (see Figure 1). The basic process of generating the workflow involves the following steps:

- Find x, which can be located at 0 or more storage system resources $\{S_1, S_2, \ldots\}$.
- Find where to compute F, given that the computing site resources are $\{C_1, C_2, \ldots\}$.
- Choose a computation site c and a storage system s subject to constraints (performance, space availability, and so on).

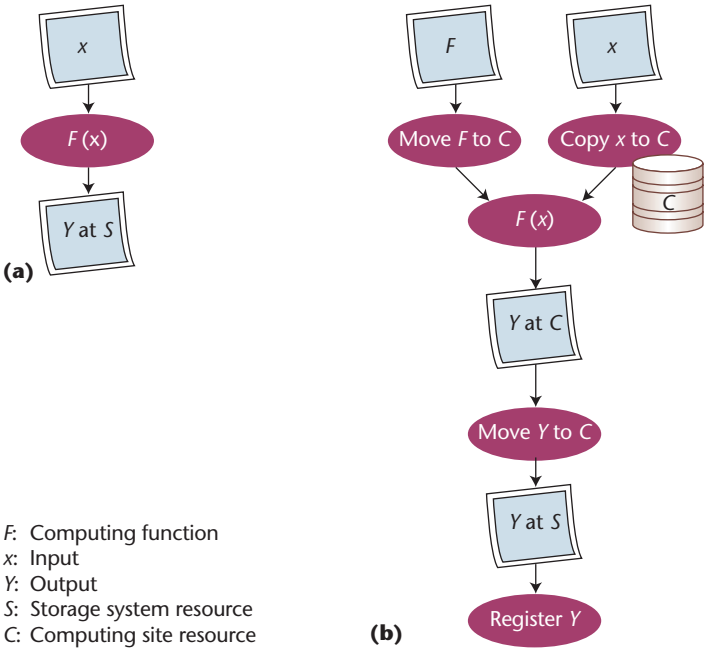

Figure 1. Workflows. (a) The abstract workflow excludes a lot of details found in (b) the concrete version. Ovals denote workflow components, document icons denote data, and silos denote resources.

As a result, we construct the following executable workflow:

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 S.
5. Register Y in data registry.

Once the workflow is generated, the descriptions in the workflow-generation process give us the provenance for the executable workflow, and once the workflow is executed, the descriptions and execution details in steps one through five give us the provenance of the workflow's output, Y.

Understanding the provenance of Y requires understanding its connection to the original workflow, especially when problems occur. Even with this simple workflow, things can go wrong—say, we didn't find x at s, $F(x)$ failed, c crashed, or there wasn't enough space at S. Given these four error messages, the user might only understand the second and last ones because they relate to the original request. But the fact that x wasn't at s is harder to interpret, especially if another copy of

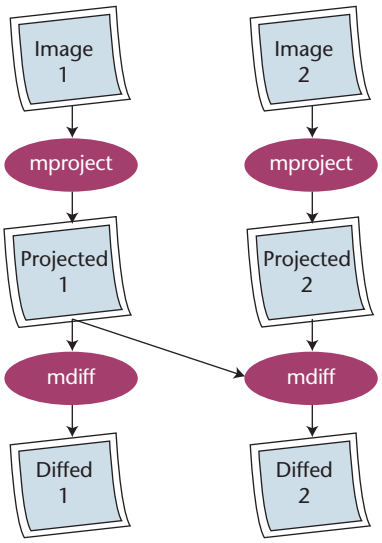

Figure 2. Montage workflow. Two images are reprojected using the mproject procedure, and then the differences between these images and others are found using mdiff (other image inputs to the mdiff operations are excluded for brevity).

x exists elsewhere. The "*c* crashed" message can be particularly hard to interpret. Obviously, the workflow mapping and execution system can try to shield the user from some of these failures and try to recover from them autonomously, but at some point this might not be possible, which is when understanding a data item's provenance becomes important.

Pegasus

We've just looked at an example of mapping from an abstract to an executable workflow, so let's look at the steps we must follow to compile an abstract workflow description into an executable workflow. This process is also known as *refinement*, because we're refining the information in the abstract workflow to the point of execution. Our approach uses the Pegasus workflow compiler system, which maps high-level, abstract workflow descriptions onto available distributed resources. To illustrate the changes made to the workflow as part of the refinement process, we use part of the Montage workflow in Figure 2, which reprojects images and takes their differences, and then refine it in Figure 3. Although, we use Pegasus as an illustrative example of workflow compilation, other workflow systems, such as Askalon,[9] also perform workflow restructuring.

Pegasus has five primary refinement steps:

- *Reduction* eliminates processing steps when intermediate data products have already been generated in another workflow or can be reused from a previous execution. In Figure 3a, the files "Projected 1" and "Diffed 1" already exist in a storage system, so we don't need to recompute them.
- *Site selection* chooses the computational resources on which to execute the jobs described in the workflow, including finding available resources and determining where the required executables are already installed or can be staged. Pegasus then annotates the workflow nodes with their target execution sites. In Figure 3b, resources *R*1 and *R*2 are available, so Pegasus uses them.
- *Data staging* selects sources of input data for computations after consulting a data registry and then adds nodes to the workflow to stage this data in or out of specific computation sites. Pegasus also adds nodes to transfer output data back to the storage sites. In Figure 3c, for example, it has added a node to transfer the intermediate result between execution sites *R*2 and *R*1 so that a workflow engine can invoke the computation at *R*1. We can also see that the intermediate result "Projected 1" from the workflow's first branch is staged for the mdiff computation.
- *Registration* adds registration nodes to the workflow to register final and intermediate data products in a data registry. Pegasus then adds data registration nodes to the workflow's final output data. Registration enables workflow-level check-pointing in case of failures, as well as for helping find data later.
- *Clustering* bundles workflow nodes together so that an execution site can handle them in one execution. This is because the granularity of computations (many short-run jobs) or the granularity of data staging (many small data transfers) is too fine in some cases, causing execution inefficiency. In Figure 3d, Projected 1 and Projected 2 are at the same site, thus Pegasus clusters the two transfers together to resource *R*1, as denoted by the box encasing those nodes.

The workflow is now ready for execution, so the scientist sends it to a workflow engine. In our approach, we use Condor DAGMan, which follows workflow dependencies and executes the workflow node directives.

Motivating Questions

A range of scientists and engineers have both

used and trialed the Pegasus system and PASOA project. Pegasus is being used for experiments that simulate earthquakes in the Southern California Earthquake Center,[13] in the Laser Interferometer Gravitational Wave Observatory[2] and Montage[4] physics projects, and in neuroscience experiments.[14] PASOA has been integrated into the aerospace simulation systems of Deutsches Zentrum fur Luft-und Ramfahrt e.V.[16] and is part of a system being developed for the Organ Transplant Authority in Catalunya to track decisions made regarding organ transplants.[15]

Once the workflow has executed, a scientist might want to ask several questions related to the provenance of the workflow execution's result. Based on our collaborations with the scientists working on the projects we just mentioned, we've found some basic questions to be important:

- Which data items generated a particular data product?
- What computations generated these data items?
- Where did the computations occur?

Scientists also have questions related to the workflow's evolution from abstract to executable:

- What happened to the node in my abstract workflow? Why isn't it in the executable workflow?
- Which intermediate data product was substituted for the actual computation?
- Why, given that the data was at *R*2, did the workflow use the data at *R*1?
- Which abstract node does a particular executable node correspond to?
- Why did the amount of disk at location *X* diminish so much?
- Why is this intermediate data not in the registry?

Many provenance systems can answer the first set of questions,[10,11] but to answer the second set, we must have information recorded during the refinement process in a form suitable for responding to those questions.

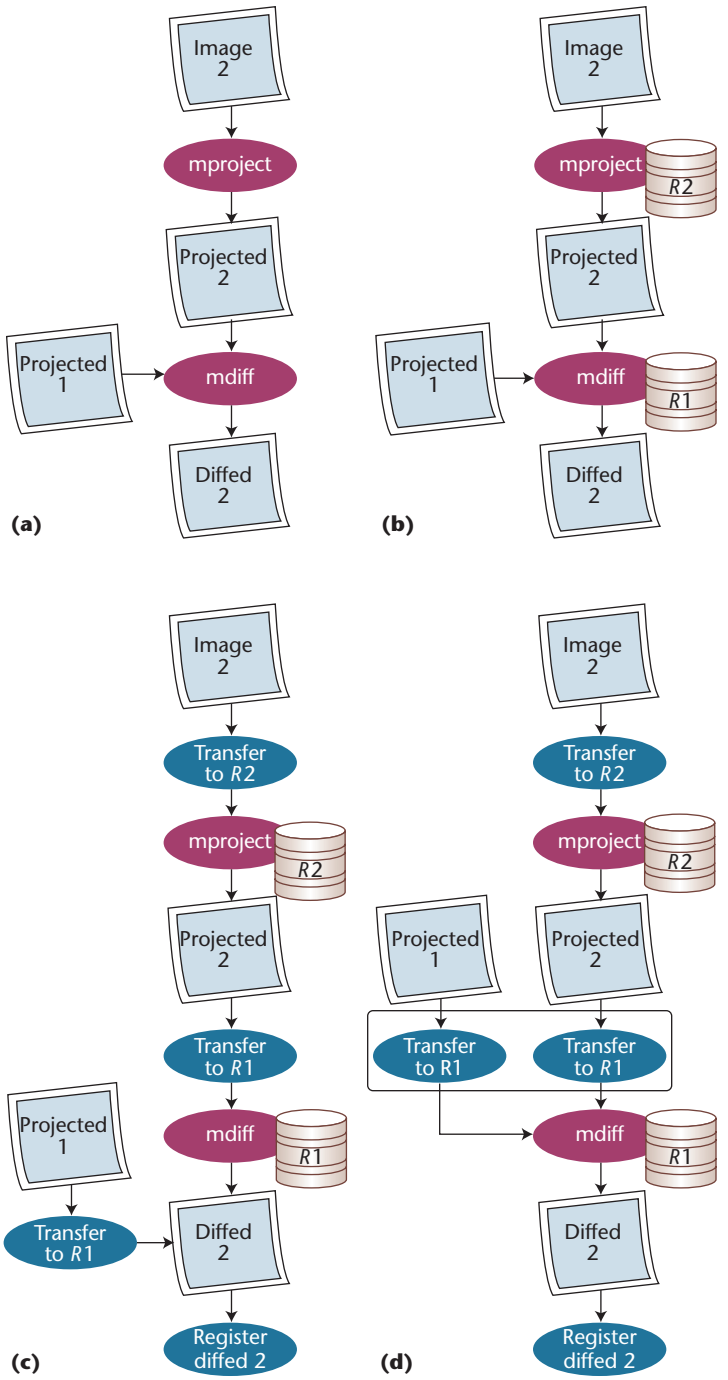

Figure 3. Montage workflow refinement. Moving from (a) reduction to (b) site selection to (c) data staging and registration to (d) clustering, we've produced a refinement of the original Montage workflow in Figure 2 that's now ready for execution.

Process Documentation

Because the resources used to execute a workflow can change between the actual execution time and when someone requests its provenance, it's often impossible to determine from the resources themselves what exactly occurred. Obviously, data provenance is important in a wide range of application areas,[12] but we can't predict all its potential in advance. We therefore need a generic software system for determining data provenance. We developed such a system in the PASOA project, which we describe in more detail later.

We can broadly split the functionality that such

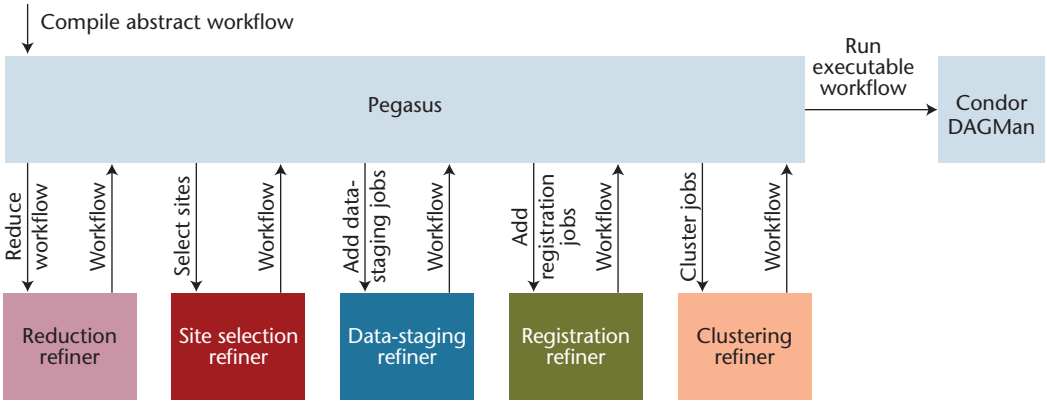

Figure 4. Pegasus. We modeled the workflow refinement process as a series of exchanges among interacting actors.

a system must support into three stages. First, as an application executes, it also creates a description of its execution (*process documentation*), which takes the form of a set of assertions about what's happening. Once the documentation is created, the application then records it in tailored persistent storage, called a *provenance store*. Finally, after an application produces a data item, users can obtain its provenance by querying the provenance store. To support querying, each independent source of process documentation must use the same data model. Workflows can bring together a range of resources that use a variety of technologies to tackle a given problem, so we need a data model that's independent of execution technology and application domain.

We can achieve this independence by viewing applications in terms of the service-oriented architecture (SOA) style adopted in many business and science applications. In technology-independent terms, a service is a component that takes inputs and produces outputs. In a SOA, clients invoke services, which may themselves act as clients for other services; *actor* denotes either a client or a service in a SOA. Actors communicate by exchanging messages, with the exchange of at least one message called an *interaction*. Thus, we can describe an application's execution as the exchange of messages between actors. In process documentation, each actor creates assertions about the data exchanged between itself and the other actor in an interaction, and about what processing was done on a service's inputs to produce the outputs. We call these *interaction* and *relationship assertions*, respectively, and taken together, they causally connect the data produced in an application. Thus, unlike other mechanisms for debugging applications, process documentation provides explicit causal connections between data items to help us track the process that led to an item's existence.

Provenance in Pegasus

Following the PASOA approach, we first considered the Pegasus system in terms of interacting actors. In this section, we separately address the refinement phase, in which Pegasus refines an abstract workflow into an executable one, and the enactment phase, in which Condor DAGMan enacts the executable workflow.

Refinement Process Documentation

In the refinement phase, we modeled Pegasus as a single actor interacting with five refiners (also actors) in turn. Figure 4 shows the model of Pegasus as actor: the interactions are the exchange of partially refined workflows between Pegasus and each refiner until the final refiner's output is an executable workflow passed to DAGMan.

For each refinement step, five recording actions occur. Figure 5 shows an expanded view of the site selection refinement step and Pegasus's invocation of it. Recording actions are labeled A to E in Figure 5b:

- Prior to each refinement, Pegasus records the current partially refined workflow that it's about to refine further (A).
- Pegasus then records relationship assertions linking this workflow to the previous refinement's output (B); these are identical because Pegasus itself doesn't alter the workflow.
- The refiner records the workflow it receives prior to its refinement (C).
- After refinement, the refiner records the refined workflow (D).

- The refiner also records relationship assertions from each node after refinement to the original nodes (E).

As mentioned in the last step, each refiner documents the relationships between nodes in the workflow as they were before and after refinement. The type of relationship gives anyone querying the provenance more information about how the refinement took place.

Refiners record the following types of relationship, corresponding to the refinement stages just described:

- *identicalTo* denotes that a workflow node hasn't changed in refinement. Its absence from a node in the pre-refinement workflow indicates that the node changed or was removed during refinement.
- *siteSelectionOf* denotes that, for the workflow node site, a compute job has been chosen and specified. This relationship is documented by recording action E in Figure 5b.
- *stagingIntroducedFor* denotes that the post-refinement workflow node is a data-staging operation introduced to stage data in/out for the job in the pre-refinement workflow.
- *registrationIntroducedFor* denotes that the post-refinement workflow node is a registration operation introduced to follow a stage-out in the pre-refinement workflow.
- *clusteringOf* denotes that the post-refinement workflow node is a cluster that combines several jobs in the pre-refinement workflow.

Figure 6 summarizes these relationships visually. We see the fragment through six states of refinement, from abstract to concrete; workflow nodes (ovals showing input and output data) at each stage are related (double-headed arrows) to those in the previous stage, with the relationship type (arrow label) describing the function the refiner performed to transform the workflow. By tracing relationships, a person querying this data can determine the provenance for each concrete job and how it relates to the original abstract workflow.

Enactment Process Documentation

The PASOA model for documenting enactment is the same as for documenting refinement. We modeled Condor DAGMan as an actor interacting with each job in the workflow. DAGMan sends invocation messages that contain command-line arguments, including input filenames, to executable jobs and receives completion messages returned from the jobs that contain the names of output files produced.

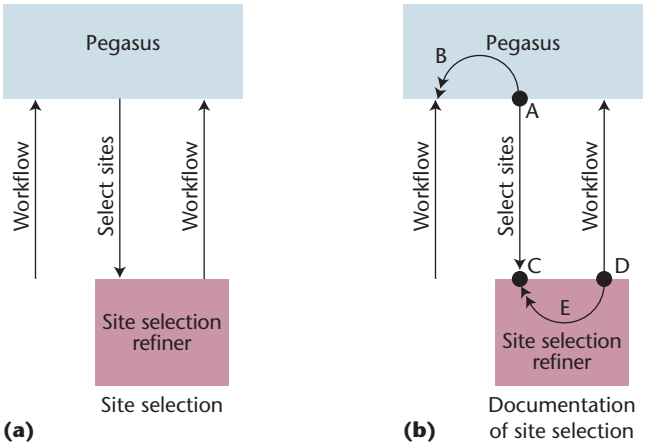

Figure 5. One refiner's documentation. (a) The interactions between Pegasus and site selection. (b) How these interactions are documented: each message is recorded when sent or received, and causal relationships connect the interactions.

As with refinement, relationships link together nodes from one step to the next, so that anyone can determine the provenance at a later point. However, the nodes here are the data items that the jobs process, referred to by filename, rather than the workflow's job nodes. The relationships between data items depend on the type of job enacted—for example, a job that invokes "gzip" would assert a relationship of type "gzip" between its output and input.

Refinement and Enactment Connected

The combination of the documentation for workflow refinement and enactment gives scientists detailed provenance for data items, including the executable workflow steps that produced it and any other data items that contributed to those steps, as well as helping them find the connection to abstract workflow jobs.

Let's revisit our initial questions and answer them:

- *Which data items generated a particular data product?* For the Montage workflow, a scientist could ask what data items generated Diffed 2 and find that Image 2 and Projected 1 were used.
- *What computation generated these data items?* mproject and mdiff.
- *Where did the computations occur?* The mproject computation occurred at computational resource *R*2 and the mdiff computation occurred at *R*1.

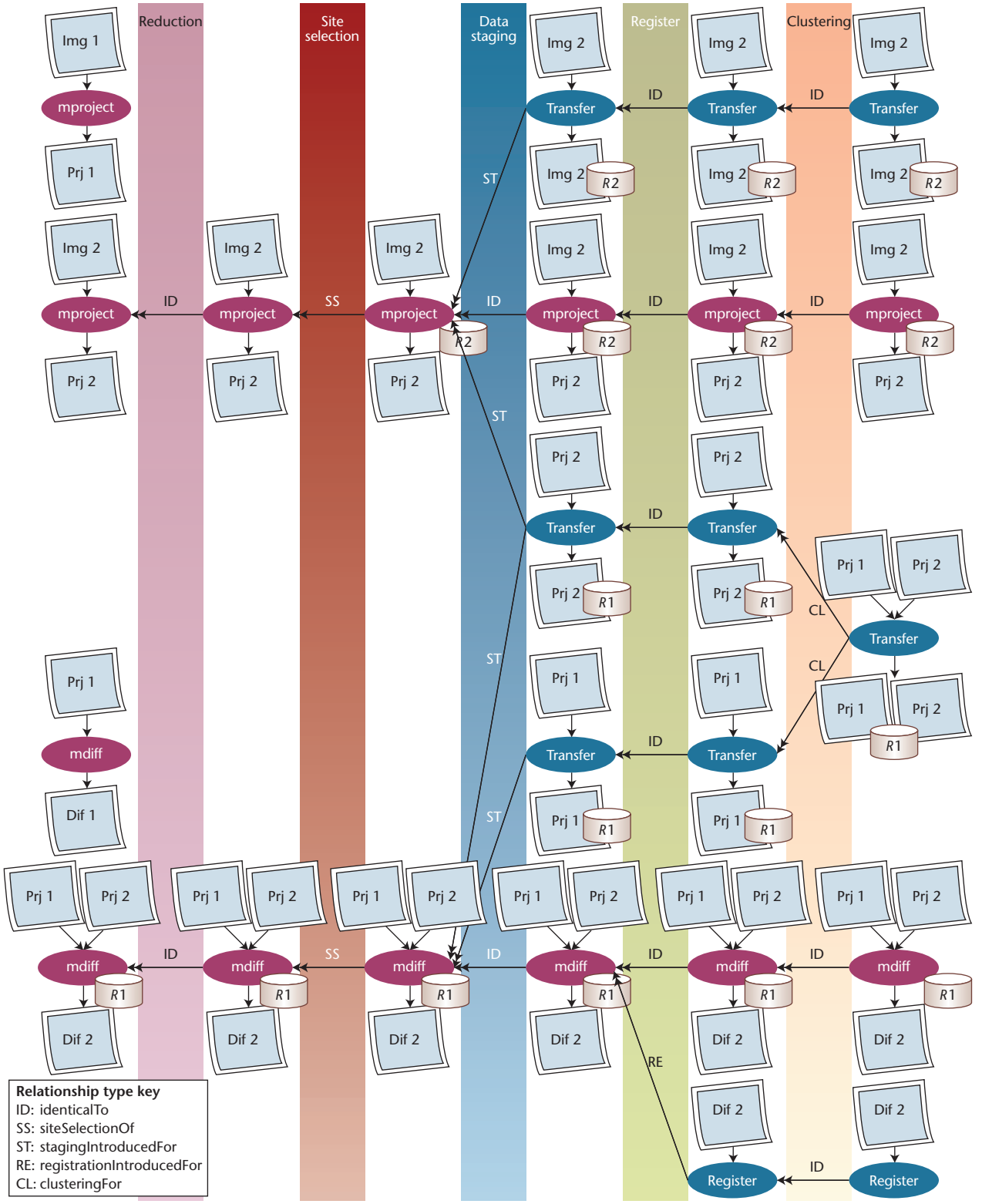

Figure 6. Relationships recorded during refinement. Each column depicts the example workflow at one stage of refinement. Moving left to right, the workflow has more refinements applied to it, and between workflows, relationships are recorded to denote exactly how the workflow nodes have changed.

We can also answer questions related to the workflow's evolution from abstract to executable:

- *What happened to this node in my abstract workflow? Why isn't it in the executable workflow*? In the Montage example, a scientist could specifically ask what happened to the mproject node that takes Image 1 as input and learn from the documentation that this node was eliminated during workflow reduction.
- *Which intermediate data product was substituted for the actual computation*? After the data-staging step in Figure 5, Pegasus added a transfer node that stages in Projected 1 rather than running mproject on Image 1.
- *Which abstract node does a particular executable node correspond to*? In Figure 5, for example, we can determine that the mproject node that takes Image 2 as input corresponds to the mproject node running on R2.
- *Why is this intermediate data not in the registry*? We can determine by analyzing the documentation that Projected 2 wasn't in the registry because Pegasus failed to add a corresponding data registration node.

The following two questions are left for future work, so we describe how extensions to our current system could help answer them:

- *Why did the amount of disk at location* X *diminish so much*? Answering this question requires knowledge of the causality between staging data to a resource and the amount of disk space available at a resource. We envision that models could answer these questions, but it's not clear how a user might formulate such a query.
- *Why, given that the data was at* R*2, did the workflow use the data at* R*1*? We intend to extend our system not only to capture the workflow system's decisions but also the reason behind those decisions.

We've shown that our system can answer questions about workflow refinement and execution, but it's important that the costs of using such a system don't outweigh the benefits. Recently, we've analyzed the performance costs of recording process documentation during execution and found them to be sufficiently minor.[7]

Understanding the process that ultimately produced a result is critical to correctly interpreting it, and it's particularly important when execution steps aren't apparent in the original process design. Provenance is a key ingredient of scientific reproducibility: colleagues can share this type of information and thereby reproduce and validate each other's results.

Future work in the area of provenance must address both low- and high-level issues. Our approach's scalability—although adequate for the problems tackled so far—requires additional techniques for coping with very large data sets, in which recording copies of all data passing through the system is infeasible. Other issues include the connection between a physical experiment's electronic data and physical components, each of which has its own interconnected provenance. By keeping the connection between experiment and data, even in complex distributed environments, our work provides a solid basis for future directions. The component for recording refinement documentation as described here is included as an option in the latest publicly available release of Pegasus (version 2.1.0); we're continuing to work on mechanisms to aid scientists in best exploiting the information recorded. CiSE

References

1. B.C. Barish and R. Weiss, "LIGO and the Detection of Gravitational Waves," *Physics Today*, vol. 52, no. 10, 1999, pp. 44–50.
2. D.A. Brown et al., "A Case Study on the Use of Workflow Technologies for Scientific Analysis: Gravitational Wave Data Analysis," *Workflows for e-Science*, I. Taylor et al., eds., Springer, 2006, pp. 39–59.
3. D. Bernholdt et al., "The Earth System Grid: Supporting the Next Generation of Climate Modeling Research," *Proc. IEEE*, vol. 93, no. 3, 2005, pp. 485–495.
4. G.B. Berriman et al., "Montage: A Grid Enabled Engine for Delivering Custom Science-Grade Mosaics On Demand," *Proc. SPIE Conf.*, SPIE, 2004; http://montage.ipac.caltech.edu/publications/2004SPIE/Montage_SPIE_2004_paper.pdf.
5. E. Deelman et al., "Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming J.*, vol. 13, no. 3, 2005, pp. 219–237.
6. E. Deelman et al., "Pegasus: Mapping Large-Scale Workflows to Distributed Resources," *Workflows in e-Science*, I. Taylor et al., eds., Springer, 2006, pp. 376–394.
7. S. Miles et al., "Connecting Scientific Data to Scientific Experiments with Provenance " *Proc. 3rd IEEE Int'l Conf. e-Science and Grid Computing*, IEEE CS Press, 2007, pp. 179–186.
8. Y. Gil et al., "Wings for Pegasus: A Semantic Approach to Creating Very Large Scientific Workflows," *Proc. OWL: Experiences and Directions* (OWL-ED), CEUR-WS.org, 2006; http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-216/submission_29.pdf.
9. M. Wieczorek, R. Prodan, and T. Fahringer, "Scheduling of Scientific Workflows in the ASKALON Grid Environment," *SIGMOD Record*, vol. 34, no. 3, 2005, pp. 56–62.
10. L. Moreau et al., "The First Provenance Challenge," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008, pp. 400–418.

11. Y. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance in e-Science," *SIGMOD Record*, vol. 34, no. 3, 2005, pp. 31–36.
12. S. Miles et al., "The Requirements of Using Provenance in e-Science Experiments," *J. Grid Computing*, vol. 5, no. 1, 2007, pp. 1–25.
13. E. Deelman et al. "Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example," *Proc. e-Science 2006*, IEEE Press, 2006, p. 14.
14. A. Lathers et al., "Enabling Parallel Scientific Applications with Workflow Tools," *Challenges of Large Applications in Distributed Environments* (CLADE), IEEE Press, 2006, pp. 55–60.
15. S. Álvarez et al., "Applying Provenance in Distributed Organ Transplant Management," *Proc. Int'l Provenance and Annotation Workshop* (IPAW 2006), Springer, 2006, pp. 28–36.
16. G. K. Kloss, and A. Schreiber, "Provenance Implementation in a Scientific Simulation Environment," *Proc. Int'l Provenance and Annotation Workshop* (IPAW 06), Springer, 2006, pp. 37–45.

AMERICAN INSTITUTE OF PHYSICS

The American Institute of Physics is a not-for-profit membership corporation chartered in New York State in 1931 for the purpose of promoting the advancement and diffusion of the knowledge of physics and its application to human welfare. Leading societies in the fields of physics, astronomy, and related sciences are its members.

In order to achieve its purpose, AIP serves physics and related fields of science and technology by serving its member societies, individual scientists, educators, students, R&D leaders, and the general public with programs, services, and publications—information that matters.

The Institute publishes its own scientific journals as well as those of its member societies; provides abstracting and indexing services; provides online database services; disseminates reliable information on physics to the public; collects and analyzes statistics on the profession and on physics education; encourages and assists in the documentation and study of the history and philosophy of physics; cooperates with other organizations on educational projects at all levels; and collects and analyzes information on federal programs and budgets.

The scientists represented by the Institute through its member societies number more than 134 000. In addition, approximately 6000 students in more than 700 colleges and universities are members of the Institute's Society of Physics Students, which includes the honor society Sigma Pi Sigma. Industry is represented through the membership of 37 Corporate Associates.

Governing Board: Louis J. Lanzerotti* (chair), Lila M. Adair, David E. Aspnes, Anthony Atchley*, Arthur Bienenstock, Charles W. Carter Jr*, Timothy A. Cohn*, Bruce H. Curran*, Morton M. Denn*, Alexander Dickison, Michael D. Duncan, H. Frederick Dylla* (ex officio), Janet Fender, Judith Flippen-Anderson, Judy R. Franz*, Brian J. Fraser, Jaime Fucugauchi, John A. Graham, Timothy Grove, Mark Hamilton, William Hendee, James Hollenhorst, Judy C. Holoviak, Leo Kadanoff, Angela R. Keyser, Timothy L. Killeen, Harvey Leff, Rudolf Ludeke*, Kevin B. Marvel*, Patricia Mooney, Cherry Murray, Elizabeth A. Rogan*, Bahaa E. A. Saleh, Charles E. Schmid, Joseph Serene, Benjamin B. Snavely* (ex officio), A. F. Spilhaus Jr, Gene Sprouse, Hervey (Peter) Stockman, Quinton L. Williams.

*Members of the Executive Committee.

Management Committee: H. Frederick Dylla, Executive Director and CEO; Richard Baccante, Treasurer and CFO; Theresa C. Braun, Vice President, Human Resources; James H. Stith, Vice President, Physics Resources; Darlene A. Walters, Senior Vice President, Publishing; Benjamin B. Snavely, Secretary.

www.aip.org

Simon Miles *is a lecturer in computer science at King's College London. His research interests include e-science, agent-oriented software engineering, electronic contracting, and distributed systems. He co-led the international provenance challenges, bringing together researchers from 20 disparate teams in two six-month exercises to compare their systems using a single, medical application. Miles has a PhD in computer science from the University of Warwick, UK. Contact him at simon.miles@kcl.ac.uk.*

Paul Groth *is postdoctoral research associate in the Information Sciences Institute at the University of Southern California. His research focuses on provenance, multi-institutional and distributed systems, and e-science. Groth has a PhD in computer science from the University of Southampton. Contact him at pgroth@isi.edu.*

Ewa Deelman *is an assistant research professor in the University of Southern California's computer science department and a project leader in its Information Sciences Institute. Her research interests include the design and exploration of collaborative, distributed scientific environments, with particular emphasis on workflow management. Deelman has a PhD in computer science from Rensselaer Polytechnic Institute. Contact her at deelman@isi.edu.*

Karan Vahi *is a research programmer at the University of Southern California's Information Sciences Institute. His research interests include large-scale distributed computing, workflows, and scheduling. Vahi has an MS in computer science from the University of Southern California. Contact him at vahi@isi.edu.*

Gaurang Mehta *is a research programmer in the University of Southern California's Center for Grid Technologies. His research interests include high-performance systems, workflows, and scheduling. Mehta has an MS in electrical engineering from the University of Southern California. He is a member of the IEEE. Contact him at gmehta@isi.edu.*

Luc Moreau *is a professor at the University of Southampton. His research interests include distributed computing, service-oriented computing, and provenance. Moreau has a PhD in computer science from the University of Liège, Belgium. Contact him at l.moreau@ecs.soton.ac.uk.*