CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2007; **VOLUME**:1–9999 *Prepared using* **cpeauth.cls** *[Version: 2002/09/19 v2.02]*

Tracking Provenance in a Virtual Data Grid

Ben Clifford[1], Ian Foster[1,2], Jens-S. Voeckler[3], Michael Wilde[1,2], Yong Zhao[1]

[1] *Computation Institute, University of Chicago, Chicago, IL 60637, USA*
[2] *Math & Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA*
[3] *USC Information Sciences Institute*

SUMMARY

The virtual data model allows data sets to be described prior to, and separately from, their physical materialization. We have implemented this model in a Virtual Data Language (VDL) and associated supporting tools, which provide for both the storage, query, and retrieval of virtual data set descriptions, and the automated, on-demand materialization of virtual data sets. We use a standardized data provenance challenge exercise to illustrate the powerful queries that can be performed on the data maintained by these tools, which for a single virtual data set can include three elements: the computational procedure(s) that must be executed to materialize the data set, the runtime log(s) produced by the execution of the computation(s), and optional metadata annotation(s) that associate application semantics with data and procedures.

1. INTRODUCTION

The *virtual data system* (VDS) [1] was developed within the Grid Physics Network (GriPhyN) project [2] with the goal of allowing data sets to be described prior to, and separately from, their physical materialization. A *virtual data language* (VDL) is used to describe how data files are computed from input data files and parameters. A runtime system provides for the on-demand derivation of files in a variety of execution environments, including distributed Grids. A *virtual data catalog* (VDC) stores descriptions of both virtual data sets and computational procedures, together with logs of process executions and metadata *annotations* on both procedures and data, provided by users or applications. This integration of three types of provenance data—program structure, runtime logs, and annotation—into a unified relational schema [3] enables powerful discovery and analysis operations.

In the sections that follow, we first review VDS briefly, with a focus on how it records provenance information. Then, we use a set of provenance challenge queries [4] to show how VDS can support provenance queries.

2. THE VIRTUAL DATA SYSTEM

The VDC contains descriptions of five types of objects. First, *files* are data objects, referred to by logical names to allow for the creation of multiple physical replicas. Second, *transformations* define functions that invoke other transformations (within *compound transformations*) or executables (*simple transformations*) to produce output files from input files and parameters. Transformations are identified by tuples of the form *(namespace, name, version)*. Third, *derivations* (as we term VDL function calls) describe calls to transformations. They define how a named transformation is invoked with named input files and parameter values to produce named output files. Derivations are also stored in the VDC, identified by namespace, name, and version. Fourth, *annotations* can be associated with transformations, derivations, and files. An annotation is a name-value pair, where the value may be of type text, integer, float, or date. Finally, *runtime execution records* reported by the *kickstart* [5] application-launching utility during program execution provide detailed provenance information about both application behavior at runtime and the runtime environment.

The *Virtual Data Language (VDL)* is used to define transformations and derivations [1]. For example, in the code below, the TR statement defines a simple transformation, which takes as parameters the name of a PGM-format input file and the name of a GIF-format output filename. This transformation calls the image processing application `convert`. The two DV statements each describe a call to this transformation and thus define a virtual data product. The first such statement specifies how `atlas-x.gif` can be created by calling `convert` with `atlas-x.pgm` as input, and the second specifies the same for `atlas-y.pgm`.

```
TR unix::convert( in inpgm, out outgif ) {
    argument = ${inpgm};
    argument = ${outgif};
}
DV ipaw.pc1.wf01::convert.13->unix::convert(
    inpgm = @{in:"Data/Derived/atlas-x.pgm"},
    outgif = @{out:"Data/Derived/atlas-x.gif"}
);
DV ipaw.pc1.wf01::convert.14->unix::convert(
    inpgm = @{in:"Data/Derived/atlas-y.pgm"},
    outgif = @{out:"Data/Derived/atlas-y.gif"}
);
```

VDL statements are compiled to create both an abstract workflow in XML (referred to as an "abstract DAG in XML" or *DAX* document) and VDC entries for the transformation and derivation definitions.

A DAX contains the *logical* information about the workflow necessary to materialize the defined data files—defining, for example, which jobs must be run, which files are involved, and the partial ordering of those jobs (i.e., the workflow graph). (The input and output labels on transformation arguments determine execution ordering: output files must be created before they are required for subsequent use as inputs, and thus a set of calls will typically form a directed, acyclic task graph.) *Physical* decisions such as where to execute each job and from where to acquire particular input files are deferred at this stage and are not specified in the DAX.

A planner makes decisions that were deferred from compilation time, and then executes the necessary file transfers and remote jobs. One planner, Pegasus [6], can be used to execute jobs on the Grid through Condor-G [7] and then Globus [8]. Another, the shell planner, can run the same workflows in a simpler host-local sequential execution environment. We employed the shell planner in the examples described here. The same workflow definitions are fully executable in a distributed Grid, simply by using an alternative planner, but our focus here is on describing the provenance recording conventions of VDS, and these are independent of the runtime environment.

When a job is run on a computing resource (e.g., a grid site or local host), we initiate its POSIX process execution using an application-launching tool called *kickstart* [5]. Kickstart serves two purposes–it provides a uniform way for VDS to pass arguments to an execution site, and it captures detailed information about the actual execution environment and the applications behavior. This information is returned after execution to the submitting host as an *invocation document* and is stored in the VDC. (This process is depicted in Figure 1. The invocation document contains environmental details (such as host name, IP address, current working directory and the complete set of environment variables), information about the behavior of the application (including its exit code and signals), and the application's resource usage (including its system time and user time, paging and swapping activity, and OS context switches). Virtually all information available about the process through standard POSIX/Linux end-user interfaces is captured. Such information has proven invaluable in debugging and monitoring distributed applications, and in recording the provenance of the data derived by VDS applications.

3. THE CHALLENGE QUERIES

We now describe how we implemented the challenge queries using the Virtual Data System.

In general, our provenance queries for the challenge are performed by extracting relevant data from the VDC using SQL. For many of the queries we then further process the SQL results with textual filter scripts. Workflow queries, which interrogate the VDC to determine the steps needed (or executed) to derive a given data object, are performed via a VDS API method (and its associated command) called *gendax* (for "generate abstract DAG in XML"). The gendax method is implemented using SQL, and encapsulates the complexities of the VDC graph traversals needed to generate workflows.

Our virtual data provenance model is represented in a relational schema that integrates the three key dimensions introduced above: i.e., workflow lineage graphs, runtime provenance, and

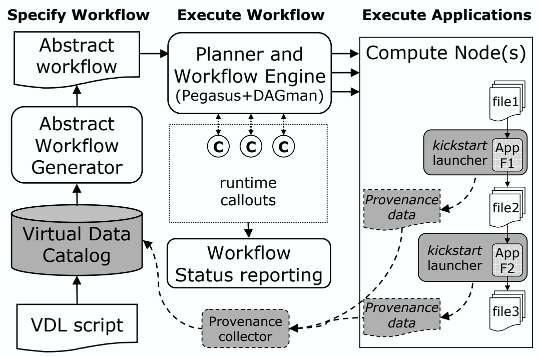

Figure 1. Kickstart captures runtime provenance

metadata. This approach maps the model naturally into a relational database representation, and permits flexible and powerful queries to be posed in scripts structured around SQL statements. While the full processing query could be expressed in SQL using procedures and/or recursion, the hybrid approach described here enables significant flexibility through scripting, and avoids constructing overly complex SQL procedures. The full code of these hybrid SQL/shell scripts and their result sets for Provenance Challenge 1 are available from our challenge entry web page [9].

For each challenge query, we present an overview of the query's expression using the VDS, and a discussion of the VDS approach.

Query 1: This query is a straightforward data-lineage query: given a data object (in this case a terminal data product derived by the workflow) it requests the "process"—the steps—by which the specified target data product was derived. This process is a subset of the complete task execution graph. It also asks for "everything" that causes the product to be "as it is." The first and foremost question we would like to ask about a derived file is "show me the history of the file", i.e., from which files was it derived, and what application programs were used to process those files?

With VDS, we addressed this query by producing an abstract DAX document to describe the process as an abstract DAG (XML DAX document) of application invocations (in various formats), which, together with the information recorded in the VDC from invocations records, constitutes "everything that causes the product to be as it is." The process is described as both the complete XML DAX document, as well as visual renderings of the process, both with and without file exchange information. Visual renderings of the same process graph are shown in Figure 2.

The following information is provided to answer the "everything we know" part of Query 1:

- From the DAX:

  - The namespace, name and version of each application invoked
  - The input files, input parameters used by each application
  - The explicitly set environment variables for each application
  - The output files to be generated by each application

- From the invocation records, describing each invocation:
  - The full set of POSIX environment variable names and values
  - The full command line
  - Detailed file status information from the POSIX *stat* system call
  - Return code and signal information from the POSIX *wait* system call
  - Resource consumption from the POSIX *getrusage* system call

The DAX, the complete application invocation records, and formatted summaries of them are presented on the Web [9], along with the commands that produced them. We used the VDS *gendax* utility to create a workflow graph in DAX format for requested process, and the scripts *prdax* and *drawwf* to format that DAX into human-readable text and graphical representation, augmented with runtime invocation provenance information from the VDC. Since the computation had already been executed, the VDC contains all transformation and derivation information as well as invocation records.

This query illustrates a few points specific to the VDS provenance model and its (then) current implementation. The scope of the process is constrained by "how far back" the process of file derivation is expressed. Files for which the VDC has no derivation history are considered to be "raw input files" and form the leaves of the derivation process graph, at which gendax terminates its traversal of the process lineage. Being targeted primarily at a distributed environment, file names listed in the VDC are typically *logical* names. (Translation mechanisms used by the VDS planners and run time callouts map logical to physical names.) Lastly, we have experimented with various schemas for the VDC in order to achieve suitable processing speeds to handle workflows of many hundreds of thousands of files. Currently, the best performance was achieved by representing the details of VDL transformations and derivations as XML documents in SQL text fields. This approach prevents us from expressing the queries entirely in SQL, and requires scripting to achieve various select and join operations.

Query 2: This query addresses the problem of finding a subgraph within a workflow graph (by "excluding everything prior to the averaging of images with softmean"). In scientific discovery and collaboration, it is often important to identify and understand a subset of a processing workflow, and then adapt and revise that part to derive new analysis protocols. We approach this problem by identifying the subgraph prior to softmean, and then removing that subgraph from the complete derivation graph.

This query requires the filtering of a workflow graph produced by gendax, for which we perform a graph traversal, starting from the specified node (softmean) and proceeding backwards through the workflow graph (opposite to the direction of execution, from the sink to the source) eliminating all predecessor tasks of the softmean node. The script exclude_prior performs this pruning on the XML workflow graph document (the genatlas.dax file generated by Query 1), producing a new workflow document, in the file exuptosoftmean.dax. We then run the *prdax* script on this graph to display the provenance.

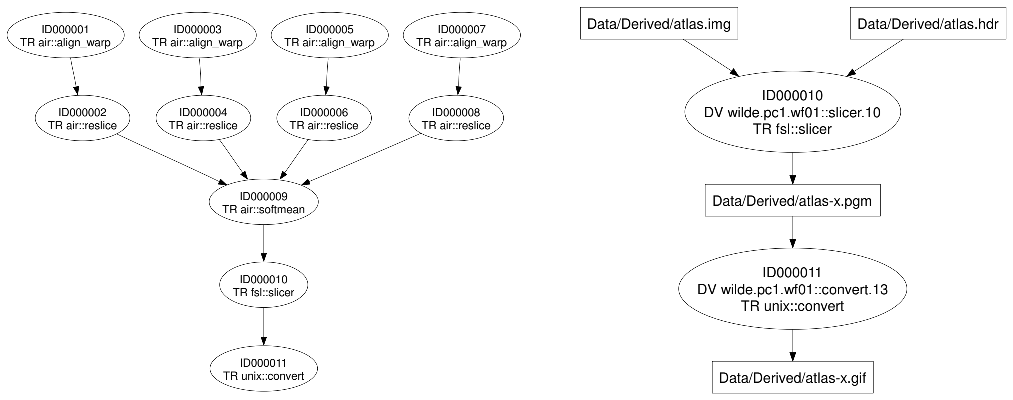
ID000001 TR air::align_warp
ID000003 TR air::align_warp
ID000005 TR air::align_warp
ID000007 TR air::align_warp
ID000002 TR air::reslice
ID000004 TR air::reslice
ID000006 TR air::reslice
ID000008 TR air::reslice
ID000009 TR air::softmean
ID000010 TR fsl::slicer
ID000011 TR unix::convert
Data/Derived/atlas.img
Data/Derived/atlas.hdr
ID000010 DV wilde.pc1.wf01::slicer.10 TR fsl::slicer
Data/Derived/atlas-x.pgm
ID000011 DV wilde.pc1.wf01::convert.13 TR unix::convert
Data/Derived/atlas-x.gif

Figure 2. Graphical output for query 1: Workflow (left), file detail of last two stages (right)

The exclude_prior script operates by reading the DAX file, constructing its workflow graph in memory as a successor list in associative array objects, and using the recursive function *elim* (shown on the Challenge web entry) to remove graph elements that we wish to prune.

Query 3: This query is similar to Query 2 but is based on stages of the workflow graph rather than on graph paths. This query requires the labeling of the stages according to their depth in the overall processing graph (and only makes sense for workflows with regular stage-like patterns, which are indeed very common). Each derivation in a VDS DAX is annotated with a level attribute that specifies its level within the breadth-first graph traversal of the VDC that is performed to go from a requested data product backwards through the VDC to find all the dependent data products and the derivations needed to produce them. These VDS levels are the complement of the stage numbers specified in the challenge problem.

This query is performed by the script *select_stage*, which applies a function maxlevel() to determine the depth of the workflow graph, and uses this value to convert the requested stage numbers to levels. Then, *select_stage* proceeds in a manner similar to that used by *exclude_prior* in Query 2, pruning out all levels of the workflow graph outside of a specific range of stages. It illustrates again the concept of querying by filtering the XML DAX document to select workflow process nodes that meet a specific criteria.

Query 4: This query is concerned with independent application invocations, rather than their relationships within a workflow or to each other. It uses two selection criteria to return a list of invocations: 1) applications of a specific name and parameter values and 2) applications that ran at specific points in time ("on a Monday"–essentially a pattern match / modularity test on a time value).

Provenance queries of this form combine what we call *prospective provenance*–the specification of the workflows procedure calls and data dependencies—and *retrospective* *provenance* information—the recordings of when and where each procedure ran, and how each invocation behaved [3]. Prospective provenance typically takes the form of "recipes" that can be used to derive a file, such as programs and parameters, while retrospective provenance captures information about the runtime environment in which the derivation was performed.

To perform this query in VDS, we ran SQL queries on the VDC database directly, performing a join of the invocation tables with the parameters of the DV (as given by the *darg* table). For days of the week, we use a SQL expression on the DV invocation date: `SELECT EXTRACT(DOW FROM start)` to pick a specific date. (Note that we queried for Tuesday (2) rather than Monday (1) in order to match data from our test examples which ran on Tuesday). The query and its output are shown on the Challenge entry page. Its output is a dump of all the fields of the invocation record as stored in the VDC database.

Query 5: This query looks for the output products of workflows whose inputs meet a specific metadata annotation criterion. This query treats an entire workflow as a "black box". The query assumes an environment in which many workflows have been run, and in which our task is then to select workflows whose input images meet a specific annotation criteria and list the atlas images derived by those workflows.

Query 5 requires a join to be performed between metadata annotations and workflow graphs. It selects the derived files of workflows whose input files have a specific annotation–a search criterion that we expect is common in scientific data analysis: metadata annotations on scientific datasets provide application-specific knowledge that supplement function signatures and workflow lineage graphs. Such annotations often play an important role in understanding the semantic meaning of datasets and programs.

To create the data environment for this query, we ran a script to take metadata contained in the header files of the 3D input images, and to enter this data into the VDC as metadata annotations associated with the workflow input files. This enabled the annotations to be joined in queries with objects in the VDC describing the workflow.

We then performed Query 5 using a script which use SQL queries on the VDC to find anatomy files with the VDC annotation "maxlevel=4095", and to find all derived atlas graphics files. Then for each output atlas file, the script queried the VDC (using gendax) to find the workflow that derived that file, and listed the workflow as a query match if it contained the target input file.

We must point out a few weaknesses in our implementation of this query. First, the subqueries made assumptions about file naming conventions to find the correct file types. A more thorough approach would have been to apply further metadata annotations to positively specify each file's data type. Second, the query was somewhat brute-forced using sequential searches that will not scale well. A richer schema can readily address this. Lastly, in this writing we discovered a minor error in the query: we searched for the correct metadata value but omitted the metadata tag name "maxlevel" from the annotation query. While this was a very prototypical implementation of this kind of query, the approach helped to verify that the VDC contains the necessary data to effectively answer such queries.

Query 6: This query probes the provenance relationships that help scientists determine if their data has been processed in a specific manner. In this case, the query looks for image data that has been processed by workflow paths of the form:

```
[align-warp using model m12] -> [any other processing] -> [softmean]
```

In other words, find all the outputs of a given procedure where its inputs had the lineage of being processed in a specific manner.

To perform this query using the VDS we used an approach similar to that of Query 5, in the sense that we examined the inputs and outputs of specific workflows. The algorithm used to perform this query was as follows:

```
Find all align_warp derivations invoked with argument model=12
    (Using SQL >alignwarpDVs)
Find all softmean derivations
    (Using SQL >softmeanDVs)
FOREACH softmean derivation smDV in softmeanDVs
       Generate smw, the workflow for smDV
       IF smw contains any alignwarpDVs
       THEN print the output image filenames of smw
```

Query 7: This query seeks to present the differences between two workflow graphs. Graph difference detection and presentation is a complex problem, and a prudent approach to attack it rigorously requires building on a capable graph library and/or query engine, as well as visualization capabilities. While we did not perform query 7, it is within the scope of the queries that VDS should be able to answer, and we sketch here a possible implementation of a rudimentary solution.

For many classes of workflows and provenance problems, the differences between two workflows can be readily assessed by a scientific user using simple textual difference tools, once the graphs have been expressed in a suitable textual representation and ordering.

For example, if for each derivation in a workflow we create (e.g., from the DAX document) one text line of the form *outputfile = f(input files and arguments)* and then sort this file by *outputfile* (assuming each output file is immutable and thus occurs at most once as an output in a workflow), then a simple textual comparison (using, e.g. the UNIX *diff* command) will tell us if two workflows are logically identical or not.

In examples such as Query 7, the textual differences will indicate to the user that the second workflow had replaced the routine *convert* with the two routines *pgmtoppm* and *pnmtojpeg*. Other simple deltas would also be readily apparent: for example, a simple change in argument value or argument strings would show up clearly and comprehensibly. By providing filters that expose or hide various levels of difference between similar derivations, the user could probe multiple workflows and perform useful comparisons and audits.

Query 8: This query is a straightforward query that searches for a commonly sought pattern: files produced by a given transformation whose inputs have specific annotation attributes. In

VDS, this query is done entirely in SQL using nested SELECT statements, as only single transformations (rather than workflow graphs) are involved, and the VDC schema contains all the necessary attributes to express the query directly. The inner SELECT finds the files which meet the annotation criteria, and the outer query selects the output files of derivations which process the files selected in the inner query as inputs.

Query 9: The final query is concerned entirely with metadata annotations, and illustrates the common case where a scientist wants to narrow a search to a specific set of datasets that meet some annotation criteria, and then discover the complete annotations of those datasets. In this case, it's convenient to assume that metadata annotations are also used to tag datasets as being of particular types, but such a type attribute could also be more deeply embedded into a workflow system or scripting language for data analysis.

For this query we created several file annotation records in the VDC. Like Query 8, the request is coded entirely in SQL. Due to the manner in which the VDC stores metadata (there exists one metadata "value table" for each of the types of metadata values supported in the model—Boolean, integer, float, text and date) we use one query to return textual annotations, and a second query, virtually identical, to return numeric annotations.

4. DISCUSSION AND COMPARISONS

We compare and contrast our approach to provenance with that taken in other systems, with the goal of both surveying related work and providing further insights into the VDS approach.

Execution information. Our use of a user-space application wrapper to capture the system-level environment is portable but limited: for example, it cannot capture information about system calls performed by the application. Other systems, such as PASS [10] and ES3 [11], capture provenance information at the operating system level—PASS in the kernel, at the file system interface, and ES3 via the user-level *strace* interface. The additional information collected by these systems can be useful, allowing them to detect (for example) dependencies that were (erroneously) not declared by the VDL programmer. On the other hand, as illustrated in the PASS provenance challenge entry, the amount of data generated can be prohibitive. In any case, PASS and ES3 could easily be integrated into VDS, by having them return information via kickstart or insert information directly into the VDC.

Prospective provenance. Other provenance-enabled systems (e.g., PASS, Pegasus, ES3) focus only on execution (retrospective) provenance: they do not maintain direct knowledge about functions and their signatures and parameter values. This issue is displayed in Query 6, which concerns execution patterns. In VDS, we could answer this question directly, while most other workflow challenge participants had to perform ad hoc textual searches through runtime execution logs.

Multi-site interactions. VDS integrates mechanisms for keeping track of multi-site interactions, an important concern for distributed workflow systems. So do some systems, such as PASOA, based on OPA, the Open Provenance Architecture [12]. PASS, on the other hand, does not.

Semantic model. It is instructive to compare and contrast VDS and the OPA general purpose provenance architecture. Our VDC maintains TR, DV, and related records that have specific (and considerable) semantics within the context of the VDS functional computation model. In contrast, PASOA allows any entity involved in a computation to generate low-level "p-assertions," at any desired level of detail. The OPA model is more general, but also more verbose when dealing with functional computations, as in the provenance challenge. We expect that this makes queries more complex to express and debug.

Workflow evolution. It is interesting to compare and contrast VDS and VisTrails[13], which provides explicit support for tracking changes to workflows over time. VDS can also accommodate changing workflows, through versioning of transformations and derivations, with a version number built into the name of each transformation and derivation. VisTrails adopts a similar relational schema to the VDC to represent provenance information such as *vt* for versions, *wf* for functions and workflows, and *log* for execution logs, and defines a query language for compact representations of provenance queries, which can be combined with its graphical interface to show query results.

Query languages. A range of query languages were used in the provenance challenge. We do not see any one language as markedly superior to the others, but believe that our use of an integrated schema has advantages in terms of expressiveness and ease of use. VDS does not yet provide good support for finding subgraphs or graph patterns, or for addressing yet more complicated cases such as graph isomorphism and graph differences (see Query 7).

We note that while not entirely trivial as an end-user operation, queries such as query 6, which was expressed in 28 non-commentary lines of SQL and shell script, provide a glimpse into a model for manipulating provenance information in scientific collaborations of many scales. With some further modularization of, for example, fully-qualified derivation name patterns, and encapsulation of XML tools to process the DAX documents, we expect that queries such as this could be performed simply by users with modest scripting skills, giving them great power and flexibility in the ability to query the provenance of large records of ongoing scientific analysis processes.

The hybrid combination of relational query and textual filter has the benefit of allowing us to leverage tools of the Virtual Data System, and affords the user great flexibility in the manipulation of provenance data, in a manner that fits well with the standard script-based approach to executing e-Science workflows in the highly-distributed Grid environment. However, this approach still requires users to have knowledge of the virtual data schema and SQL query syntax. A more user-friendly and expressive query language could further reduce this complexity.

5. SUMMARY

Our virtual data provenance model enables us to represent and query provenance information using an integrated relational schema, and answer efficiently the queries proposed in the first provenance challenge. We continue to study both user requirements for provenance and the approaches taken by other participating teams, with a view to providing an improved workflow and provenance management system. Participation in the provenance challenge has shown us improvements to make in the VDS schema and graph traversal API that will enhance the power and reduce the complexity of VDS provenance queries.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under grants ITR-800864 and PHY-0636265, the U.S. Department of Energy, under contract DE-AC02-06CH11357, and the NIH National Institute on Deafness and Other Communication Disorders under grant DC008638-01. We thank Jed Dobson of the Dartmouth Brain Research Institute for providing us with the fMRI atlas-creation workflow that this challenge exercise was based on, and for its translation into VDL.

REFERENCES

1. Foster I, Voeckler J, Wilde M, and Zhao Y. Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In *14th Conference on Scientific and Statistical Database Management*, 2002.
2. Avery P and Foster I. The GriPhyN Project: Towards Petascale Virtual Data Grids. **http://www.griphyn.org**, 2001.
3. Zhao Y, Wilde M, and Foster I. Applying the Virtual Data Provenance Model. In *International Provenance and Annotation Workshop (IPAW)*, 2006.
4. Luc Moreau, Bertram Ludäscher, Ilkay Altintas, Roger S. Barga, Shawn Bowers, Steven Callahan, George Chin Jr., Ben Clifford, Shirley Cohen, Sarah Cohen-Boulakia, Susan Davidson, Ewa Deelman, Luciano Digiampietri, Ian Foster, Juliana Freire, James Frew, Joe Futrelle, Tara Gibson, Yolanda Gil, Carole Goble, Jennifer Golbeck, Paul Groth, David A. Holland, Sheng Jiang, Jihie Kim, David Koop, Ales Krenek, Timothy McPhillips, Gaurang Mehta, Simon Miles, Dominic Metzger, Steve Munroe, Jim Myers, Beth Plale, Norbert Podhorszki, Varun Ratnakar, Emanuele Santos, Carlos Scheidegger, Karen Schuchardt, Margo Seltzer, Yogesh L. Simmhan, Claudio Silva, Peter Slaughter, Eric Stephan, Robert Stevens, Daniele Turi, Huy Vo, Mike Wilde, Jun Zhao, and Yong Zhao. The First Provenance Challenge. *Concurrency and Computation: Practice and Experience*, 2007.
5. Voeckler J, Mehta G, Zhao Y, Deelman E, and Wilde M. Kickstarting remote applications. In *Grid Computing Environments Workshop*, 2006.
6. Deelman E et al. Pegasus: Mapping Scientific Workflows onto the Grid. In *2nd EU Across Grids Conference*, 2004.
7. Frey J, Tannenbaum T, Foster I, Livny M, and Tuecke S. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–247, 2002.
8. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Int. Journal of Supercomputing Applications*, 11(2):115–128, 1997.
9. IPAW 2006 Provenance Challenge 1 - UChicago Entry Page. **http://twiki.ipaw.info/bin/view/Challenge/UChicago**, 2006.
10. Margo Seltzer, David A. Holland, Uri Braun, and Kiran-Kumar Muniswamy-Reddy. Pass-ing the provenance challenge. *Concurrency and Computation: Practice and Experience*, 2007.
11. James Frew, Dominic Metzger, and Peter Slaughter. Automatic capture and reconstruction of computational provenance. *Concurrency and Computation: Practice and Experience*, 2007.
12. Simon Miles, Paul Groth, Steve Munroe, Sheng Jiang, Thibaut Assandri, and Luc Moreau. Extracting Causal Graphs from an Open Provenance Data Model. *Concurrency and Computation: Practice and Experience*, 2007.
13. Carlos Scheidegger, David Koop, Emanuele Santos, Huy Vo, Steven Callahan, Juliana Freire, and Claudio Silva. Tackling the provenance challenge one layer at a time. *Concurrency and Computation: Practice and Experience*, 2007.