

A Framework for Collecting Provenance in Data-Centric Scientific Workflows

Yogesh L. Simmhan, Beth Plale, Dennis Gannon

Indiana University, Bloomington IN 47405, USA

E-mail: {ysimmhan, plale, gannon}@cs.indiana.edu

Abstract

The increasing ability for the earth sciences to sense the world around us is resulting in a growing need for data-driven applications that are under the control of data-centric workflows composed of grid- and web- services. The focus of our work is on provenance collection for these workflows, necessary to validate the workflow and to determine quality of generated data products. The challenge we address is to record uniform and usable provenance metadata that meets the domain needs while minimizing the modification burden on the service authors and the performance overhead on the workflow engine and the services. The framework, based on a loosely-coupled publish-subscribe architecture for propagating provenance activities, satisfies the needs of detailed provenance collection while a performance evaluation of a prototype finds a minimal performance overhead (in the range of 1% for an eight service workflow using 271 data products).

1. Introduction

The proliferation of inexpensive specialized instruments and sensor networks that feed data into *data-driven applications* is enabling broader scientific understanding of the world around us [26]. Such data-driven scientific investigations are often modeled as *dataflow applications* in which data passes from one process to another as it is transformed, filtered, fused, and used in complex models. Grid- and web-services provide an abstraction to access these processes through well-defined interfaces and allow applications to be designed as workflows that capture the invocation logic. Such scientific computing applications can be composed of hundreds of interconnected services and reach gigabytes to terabytes of data in a single run. At this scale of processing, not only must the process be automated, but it should also be largely carried out away from the scientist's workstation – on services deployed on shared large-scale computational resources such as the Teragrid [4]. Operations like restarts, partial runs, and run-diagnosis of workflows are no longer as easy for a scientist to control. As a result, a line of re-

search that has developed alongside research into workflow systems is that of provenance collection and representation. *Provenance* on a workflow execution describes the workflow's service invocations during its lifecycle [20]. This helps track service and resource usage patterns, and forms metadata for service and workflow discovery. In data-driven applications, however, provenance about the data is central to understanding and recreating earlier runs. In *data-centric workflows*, data products are first-class parameters to services that consume and transform the input data to generate derived data products. These derived data products are ingested by other services in the same or a different workflow, forming a data derivation and data usage trail. *Data provenance* provides this derivation history of data that includes information about services and input data that contributed to the creation of a data product. This is extremely valuable not only for diagnosing problems with services and understanding the performance of a particular workflow run, but also to determine the origin and quality of a particular piece of derived information [22].

Current methods of collecting provenance are from workflow engines logs [1] or by instrumenting the services [29, 28]. In the former case, the logs from the workflow engine are at the message level and have insufficient semantic information to decipher provenance about the data products, while instrumenting services introduces a burden on the service author to modify their service to generate provenance metadata. The challenge we address in our work is to record uniform and usable provenance metadata that meets the domain needs while minimizing the modification burden on the service authors and the performance overhead on the workflow engine and the services. We collect two forms of provenance: *Workflow provenance*, also known as workflow log or process provenance [20], is metadata describing the workflow's execution and associated service invocations; and *Data provenance*, provides complementary metadata about the derivation history of the data product, including services used and input data sources transformed to generate it. Provenance is important because it allows scientists to monitor workflow progress at runtime [9], which is crucial for dynamic and adaptive workflows found in e-

Science. Post-execution, provenance can be mined to locate sources of errors [24], determine data quality [22], and validate the results through repetition or simulation of the workflow execution [20]. It also assists resource providers to review resource usage for purposes of auditing and provisioning [5].

The specific contributions of this paper are as follows:

1. A model for provenance collection that is independent of the workflow implementation and leverages notification-based approaches that underly many large-scale web-service implementations [18, 8, 23].
2. A performance evaluation of a prototype of the above provenance collection model to determine the overhead in generating and recording provenance.

The remainder of the paper is organized as follows. In Section 2, we motivate the discussion of provenance collection with an example from mesoscale meteorology. In Section 3, we define a provenance model based on a generalized data-centric workflow model. In Section 4, we show how the framework enables automatic construction of provenance graphs. Section 5 discusses a prototype implementation. Section 6 contains the performance evaluation of provenance collection overhead. Related work appears in Section 7 and future work in Section 8.

2. Motivation

Scientific workflows [27] often use a Service Oriented Architecture (SOA) to solve advanced scientific problems in a Grid environment [9]. Workflows are composed from services that provide a well-defined functionality with open interfaces to access them. Workflows model the interactions between the services as directed cyclic graphs (or occasionally as directed acyclic graphs) whose edges represent dependencies and dataflows, and may incorporate decision-making logic for runtime selection of execution paths [2]. Workflow execution is typically orchestrated by a central workflow engine that interprets the workflow document and invokes the services according to the dependencies, ensuring dataflow between connected services.

For instance, in the Linked Environments for Atmospheric Discovery (LEAD) [17] project, regional-scale numerical weather forecasting is done using dynamically adaptive workflows, such as in Fig. 1, that run end-to-end forecast models. These computationally intense workflows may be launched on-demand in response to a severe weather event and may ingest new data products at any point during execution directly from real-time observational sources. The model data products generated by the services may be reused in the same or in a different workflow, forming a data processing pipeline [13].

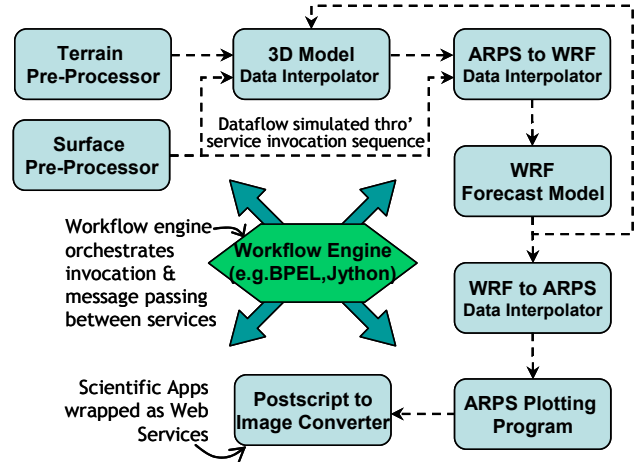


Figure 1. Weather Forecasting Workflow.

A typical meteorological experiment shown in Fig. 1 is a weather prediction using the Weather Research and Forecasting (WRF) Model. Pre-processing, interpolation, post-processing, and visualization are done by the Advanced Regional Prediction System (ARPS) Model. The workflow can be broadly divided into pre-processing and grid interpolation stage (Services 1 - 4 in Fig. 1), numerical weather prediction stage (Service 5), and post-processing and visualization stage (Services 6 - 8). Each of these services produce and consume between 4 and 100 data products in the form of files [16]. This workflow, applied to data from Hurricane Katrina's landfall on the Gulf Coast in 2005, is used for the performance evaluation in Section 6.

3. Provenance Model

Before proposing the provenance model, we provide a brief description of the workflow model, which defines the context upon which the provenance model is developed.

Workflow Model. The generalized data-centric workflow model is based on a service-oriented architecture, with four principal components: *workflow engine*, *services*, *applications* that provide the functionality for the services, and *data products* that are produced and consumed within the workflow. Services are black-box tasks accessed through a well defined interface. The workflow is a *directed cyclic graph* of services, with the services forming nodes in the graph and its edges constituting dataflow between services or applications wrapped by the service.

A central workflow execution engine executes the workflow by invoking each service in the workflow and passing messages between those that are connected, while maintaining the dependencies imposed by the graph. Outputs from one service can be used as input to other services and cycles are permitted. The workflow execution terminates successfully when all service invocations in the workflow have

completed successfully or if any of them fail.

Data products, often logical or physical files identified by a globally unique ID, passed as input to a service invocation are said to be *consumed* by that service invocation, while those newly created by that invocation are said to be *produced* by it. The data products consumed and produced within the workflow execution form their own dataflow graph since they are passed as parameters between services.

Execution occurs at three levels in the workflow model. The highest level comprises of the *workflow execution* where the workflow engine invokes the services. At the next level are the *services* that receive invocations and initiate the action specified by calling a method or launching an application. At the bottom is the *application level* that involves the actual scientific task that is performed by the invocation. The data products are consumed or produced at this level, and hence, the applications are the components best suited to generate the data provenance for the workflow.

Naming. Each running or completed workflow instance is assigned a globally unique identifier (GUID), referred to as the *Workflow ID*, that acts as a namespace for all activities that take place within that workflow. The format of the GUID does not matter as long as its uniqueness is guaranteed. Every service instance in the workflow is given a GUID, called the *Service ID*, which it retains throughout its lifetime. It is sufficient for applications to have a unique ID within the scope of a service instance (a Service ID) since it is unlikely that applications are shared across services. We refer to this as the *Application Name*. Data products require a GUID, called the *Data Product ID*, and this refers to the logical data product.

Provenance Model. The generalized workflow model provides a basis for the provenance model. Provenance is collected as a set of discrete *activities* distributed over time, space, level, and dataflow, and takes place during the lifetime of the workflow. The activities sufficiently describe the workflow and enable the construction of the workflow and data provenance graph for the workflow's execution.

The workflow model executes as a series of activities over *time* that result in devolvement of control between the different levels of the workflow execution, and the execution is distributed in *space* across various services, and in the process consuming and producing data products. An example of an activity is the invocation of a service by the workflow engine, or the production of a data product at the application level. By keeping track of these various activities that take place in the different workflow components, it is possible to derive the workflow provenance and data provenance for a workflow execution.

The provenance activities can be ordered along four dimensions: the execution *level* of the activity, the *location* of the workflow component involved (space), the *time* at which

it took place, and the *dataflow* - data products produced and consumed - during the activity. Each of these attributes is necessary to construct the provenance graph from the activities. Fig. 2 illustrates these provenance activities with a simple workflow and the sequence diagram for the activities that take place during its execution. A total of 11 activities transpire at different levels, namely $\{Workflow, Service, Application\} \times \{-Started, -Finished, -Failed\}$, *Data-Produced*, and *-Consumed*, along with the different attributes that they carry. The sample workflow *WF0* in Fig. 2 consists of two services *S1* and *S2* that expose methods (applications) *A1* and *A2*, each taking one data product as input and producing one data product as output. Key activities during a workflow's execution occur at the boundaries between two execution levels that appear at the start and finish of the workflow, the service, and the application.

The dimension of *level* helps track the control flow between these execution levels. In Fig. 2, the workflow execution starts at the workflow level and oscillates along the Level axis, flowing from workflow, to service, to application, and back to service level - corresponding to the invocation of application *A1* in *S1*, and this repeats for the invocation of *A2* in *S2*, before control returns back to the workflow. The activities that take place during the execution are listed along the Time axis and they correspond to the list of 11 activities. The activity names implicitly refer to the level of the activity, such as *Workflow-Started* occurring at the workflow level and *Service-Finished* occurring at the service level.

The next dimension of *location* (or space) helps determine the specific workflow component at which the activity took place. The location attribute of the activity is represented using the uniform naming scheme for workflow components discussed earlier. In the example, the activities have the location attribute set to the identifiers *WF0*, *S1*, *S2*, *A1*, and *A2* which can uniquely locate these components. In addition, all activities carry the Workflow ID as an attribute to provide the execution context for the activity, and application activities in addition have the Service ID within which they appear.

The dimension of time allows the temporal ordering of activities as they occur, and makes provenance construction independent of the order in which the activities are propagated. In the absence of a single globally synchronized clock, we determine the time dimension by associating a logical timestamp, called *Workflow Timestep*, as an activity attribute. The central workflow engine maintains this timestep and assigns it to activities when it orchestrates service invocations. This sequences the service invocations in the workflow example of Section 2 as *WF0-S1-A1-S2-A2* as seen along the time axis. It may be possible for two activities with identical *timesteps* to be properly ordered if they occur at different activity levels in a single invocation chain.

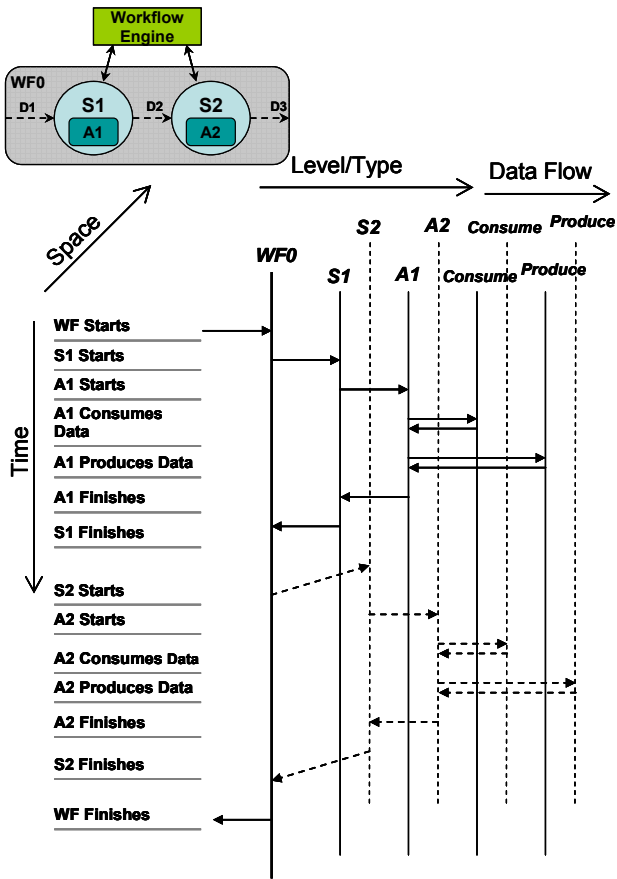


Figure 2. Sequence diagram for the activities that take place during the execution of the workflow. The workflow is at the top. The 4 activity dimensions are represented along the 3 axes; level and dataflow share the X axis. Horizontal arrows, labeled on the left, show activities that take place as the workflow progresses.

For example, *Service-Started* activity always appears before its corresponding *Application-Started* activity. Similarly, *Workflow-Started* and *Workflow-Finished* activities implicitly mark the beginning and end boundaries of the activity sequence, and do not require a timestep.

While the previous three dimensions helped identify the participating workflow components and arrange them in time, the *dataflow* dimension is necessary to connect the activities together through the data products they share. This helps construe the data provenance for the data products involved in the workflow's execution. *Data -Produced* and *-Consumed* activities occur at the application level when the application uses or creates a data product, and these data products are identified using their Data Product ID attribute present in the dataflow activities. In our example, *A1* consumes data product *D1* and produces *D2*, while *A2* consumes *D2* and produces *D3*.

4. Provenance Graph Construction

Constructing a graph of activities after their occurrence is important because it allows a user to reason about the high level behavior of the workflow and the interplay between services and data products within and across a workflow. In our approach, the provenance graph can be constructed natively unlike other provenance collection approaches that require external means for doing so. In this section we describe the algorithm by which provenance graphs can be constructed from a series of provenance activities.

The provenance service is a central service that records all provenance activities generated for workflows executing within the virtual organization. This service records the discrete activities in a repository that can be queried based on the attributes of the activity. Given access to all the 11 types of activities for a workflow execution, it is possible for the provenance service to construct the workflow and data provenance for the execution. The workflow and data provenance are represented as directed cyclic graphs. The nodes of the workflow provenance graph are services and data products form the edges representing dataflow and invocation. In a data provenance graph for derived data, the derived data is the leaf node, the service generating it is its parent node, and the input data product to that service are input nodes that connect to the service node. The edges, according to their direction, represent data consumed or produced by that service. Additional information present in the activity are represented as attributes of the nodes and edges.

The algorithm to construct workflow and data provenance using the activities is similar to an algorithm used to construct a graph given a set of node-edge pairs, which the activities form. Given the *Workflow ID*, the workflow provenance is constructed by retrieving all activities for that workflow based on that ID since all activities carry it as an attribute. For each *Workflow-*, *Service-*, and *Application-Started* activities, a node is defined in the provenance graph with its activity attributes such as service and application IDs and the invocation time. The data produced and consumed activities for the services define edges between two nodes if the source node produced a data product and the sink node consumed it. Since data-centric workflows pass at least one data product between services connected in the workflow, we are assured of building a connected workflow provenance graph.

Data provenance graphs can potentially span across activities from multiple workflows if the service deriving a data product consumed data products generated by another workflow. Given a *Data Product ID* for a derived data, the data provenance graph algorithm retrieves the service invocation that produced this data by looking for a *Data-Produced* activity matching the data. From the IDs for this invocation activity, all *Data-Consumed* activities that repre-

sent inputs to this invocation can be retrieved. Once these data product nodes, invoking service node, and dataflow edges are available, building the data provenance graph merely involves connecting them together. We can "drill-down" this provenance graph by recursively applying the above algorithm to all input data products to get the complete data derivation history for a data product. An additional nugget of information that can be inferred from the activities is the *data usage graph* that shows all services that consume a given data product, and this is useful for tracking resource usage and for source attribution.

5. Implementation

The *Karma Provenance Service* is used as a focal web service to collect and query over data and workflow provenance within the LEAD domain. Provenance activities are represented as XML notifications and the Karma service acts as a notification subscriber on all provenance topics to receive provenance activities from workflows executing within LEAD. These activities are stored in a relational database backend and the Karma service has a querying interface that presently provides the essential query primitives to retrieve: (1) the workflow provenance graph given the Workflow ID, (2) the data provenance graph given the Data Product ID, and (3) a data usage graph given the Data Product ID. Support for richer queries, such as to filter upon Service IDs and execution times, is planned. When a query is received by the provenance service, the algorithm outlined in Section 4 is used to query the database and construct the appropriate provenance graph just-in-time. The graph is represented as an XML document that is returned as the result of the query. We're also exploring ways to disseminate the provenance results in a schema that is interoperable with other evolving provenance standards [10]. Since the provenance activities are static once they arrive at the database, we can cache the provenance results for the queries to reduce graph construction overhead (pursuant to a study of the space-time trade offs).

Notification Protocol. Publish-Subscribe (or Pub-Sub) notification protocol is a uni-directional, asynchronous communication between publishers and consumers who communicate by means of "channels" or "topics". Consumers, decoupled in space from the publisher, receive events by subscribing to one or more topics [7].

A notification protocol is a good choice for provenance activities since provenance gathering can be done without perturbing the publishers and can be collected independent of the provenance service location. *Time decoupling* [7], whereby the listeners need not be active when the notification is published, puts less stringent requirements on the availability of the provenance service. Archival or message-box facilities present in most notification brokers remove the onus from workflow components in ensuring reliable de-

livery of activity messages. Since provenance is more often used for mining and analysis than for runtime monitoring of the workflow, users are tolerant to delayed receipt. Pub-sub is a mature field with several open-source and commercial implementations, and many open standards, such as JMS and WS-Eventing, which aid interoperability when collecting provenance across different domains, as is common in Grid systems.

Use of Notifications in Karma. A representative notification service is WS-Messenger [12], that uses a topic based publish-subscribe system based on the *WS-Eventing* standard. WS-Messenger contains four main components relevant to Karma: the Notification Publisher, the Notification Consumer, the Notification Broker, and the Message Box. The Notification Publisher generates notifications that are published to a topic at the Notification Broker. The broker, a web service, provides a topic-based subscription interface and routes notifications published to a certain topic to the registered listeners. Notifications Consumers are web services that subscribe with the broker for notifications and receive them via a standard web service interface they implement. For listeners that lie behind a firewall or that would like to have notifications buffered for them, a Message Box service can act as a proxy and subscribe to the broker on the listener's behalf. The listeners periodically retrieve the notification from the Message Box and this provides a means to persist the notification for reliable delivery.

Provenance activities are mapped to an XML Schema in order to provide a standard and open format for exchanging provenance. XML documents that follow this schema form the payload for provenance notifications that are generated by the workflow components. A sample XML document for the *Application-Started* activity is given in Fig. 3. Client libraries that simplify creating and publishing of notifications from the workflow, web services, and application scripts are provided. These libraries encapsulate the workflow activity at each level and when possible, automatically populate fields in the activity, such as Workflow ID, Service ID, and Workflow Timestep, from the execution context to ease provenance generation. The libraries also define other notification types which enable workflow components to generate generic logging information in addition to provenance notifications.

For each workflow execution, a unique notification topic, identified by the Workflow's ID, is created with the LEAD notification broker. All components in the workflow use the same topic to publish their notifications. This allows listeners interested in only notifications from a certain workflow execution, such as a workflow monitor, to listen to just that topic. The broker allows wildcard or "*" subscriptions and the provenance service uses this feature to subscribe to all notifications that are generated from all workflows that use that broker.

```

<applicationStart
  topic="iu/drlead/testproj/myexperiment-014">
  <workflowID> iu/lead/myproj/myWrfWorkflow-014
  </workflowID>
  <nodeID> 4 </nodeID>
  <wfTimeStep> 4 </wfTimeStep>
  <serviceID>{http://leadproject.org/}WRF792501
  </serviceID>
  <timestamp> 2005-10-29T10:05:40.720-05:00
  </timestamp>
  <name> WRF Application Script </name>
  <host> lead1.ou.edu </host>
</applicationStart>

```

Figure 3. Application-Started Activity in XML.

Provenance Visualization. The Karma provenance service’s query interface can be invoked by any web service client to retrieve data and workflow provenance. The *Karma Provenance Browser* is a tool that uses the provenance service API to integrate provenance metadata with additional metadata on the services, workflows, and data products available from the different catalogs within LEAD. The browser can retrieve and visualize the provenance graph for a given Workflow ID. Users can select services and data products within that workflow and retrieve metadata about them from the external catalogs. The browser also allows seamless navigation from a workflow graph to a data provenance graph or data usage graph for data products in that workflow, allowing users to jump across provenances from different workflow executions.

The *XBaya* Monitor GUI is another tool that allows users with access to the original workflow document to monitor the progress of the workflow execution by listening to the provenance notifications. The monitor is built into the workflow composer interface; so users can compose a workflow visually and later use the same workflow graph to monitor it at runtime. The workflow monitor subscribes to the notification topic for that workflow and updates the status of the different components in the workflow as it receives the provenance activity. Since notifications are directly delivered to the monitor by the notification broker, this obviates the need to access the provenance service for the workflow status and makes the monitoring more realtime. We are additionally exploring the use of workflow provenance available with the provenance service to perform visual replays of the workflow execution in the Workflow Monitor GUI.

6. Performance Evaluation

Provenance collection incurs a cost. We quantify the overhead using the workflow example given in Section 2; an example that exemplifies the data-driven applications in LEAD. A more detailed, comparative performance analysis of Karma is available in a separate publication [21]. The workflow of Section 2 performs weather prediction for

the New Orleans region, initialized with observational data from August 28th 2005. It simulates the weather for the next 37 hours, tracking Hurricane Katrina’s land fall on the Gulf Coast. The output of the simulation model is analyzed and visualized by generating animations.

The eight applications in the workflow are invoked sequentially to collect execution times for each application invocation. The application service script’s execution time includes the staging times for input and output files, the wall clock time of the application execution, and the provenance overhead. The first four preprocessing and interpolation applications, and the last two post processing and visualization applications are executed on a 3GHz Linux workstation, while the WRF forecasting model and WRF2ARPS post-processor are executed on a 32-node 1.3GHz Linux cluster. The Notification Broker runs on a 1.2 GHz Solaris workstation while the Karma provenance service runs on a 2GHz Windows XP workstation. All hosts are connected through a Gigabit Ethernet network.

The bar graph in Fig. 4 shows the cumulative time taken by the applications in the workflow, with and without generating provenance notifications. As seen from the bars on the far right, the time taken for the entire workflow to finish is 2834 secs when the applications generate provenance notifications and 2809 secs when they do not. This translates to a total overhead of about 0.8% of execution time for the forecasting workflow, and this is a very small overhead. Currently, the overhead percentage depends upon the number of notifications generated in each script, which translates to the number of data products produced and consumed. We have initial results from batching data -produced and -consumed activities that circumvents this problem to achieve near constant time overhead. Each of the application produces and consumes between 11 and 61 files each, ranging in size between 55MB and 3.5GB, for a total of total of 147 files consumed and 124 files produced. This corresponds to a total of 271 dataflow notifications, and the small number of started/finished notifications from the workflow, services, and applications which can be ignored for non-trivial workflows. These figures are typical of data-centric scientific workflows in the meteorology domain, and the provenance overhead is bound to be similarly small.

The line graph in Fig. 4 shows the additional time taken by each application due to publishing provenance notifications. The vertical lines mark the standard error of the mean time at each data point due to variations in the measurement caused by I/O wait times and system processes. Ignoring data points with large standard errors, we see that the overhead time for each application is in the range of 1 to 6 seconds. This is quite low considering that the normal execution time for LEAD applications is between 10’s of seconds to several hours.

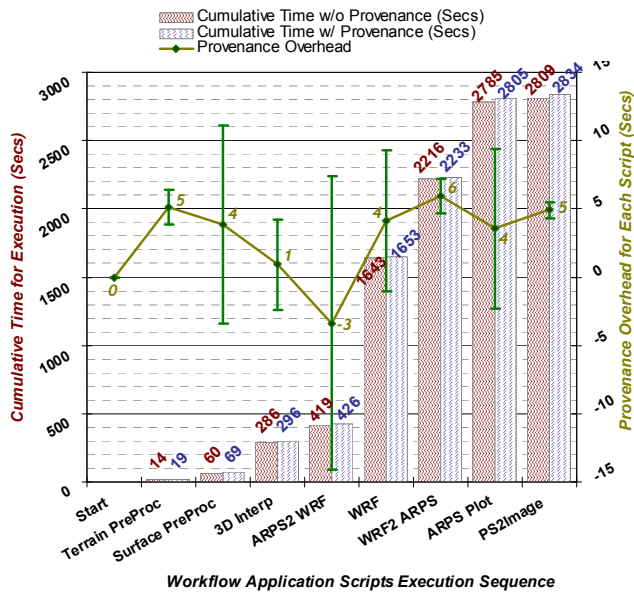


Figure 4. [Bar Graph, Left Y-axis] Cumulative execution time, with and without provenance, for applications in workflow. [Line Graph, Right Y-axis] Provenance overhead for individual applications in workflow.

7. Related Work

A synthesis of requirements for an effective and usable provenance system for scientific workflows should, in broad terms, require it to (1) provide an open and interoperable interface to collect provenance, (2) track workflow and data provenance within a virtual organization independent of workflow engine implementation and data formats, and (3) minimize performance overheads and modifications required of workflow components. While systems and techniques to collect provenance in scientific workflows are gaining popularity, they fall short of meeting these needs. Workflow environments such as Virtual Data System [29] and myGrid [28] provide the capability to record provenance but are tightly integrated with their workflow execution environment and do not provide interoperable means for collecting and using provenance. PReServ [10] has a more general approach by defining an open protocol for recording provenance through a set of messages exchanged between service invocation actors and a provenance server. Recording the in-wire service requests and responses from both the client's and service's views allow it to track workflow provenance in a non-repudiable manner. However, users have to extend the provided schema to track data products used in the invocations and native support to build provenance graphs is absent. The current system is also constrained by performance overheads [21].

Related to provenance collection are workflow and pro-

cess mining tools, such as IBM Data Collector [1] and PRoM [25], that are used in commercial workflow systems to log the audit trail for a workflow's service calls. They correlate the log to regenerate, visualize, and validate the workflow model, and are akin to workflow provenance. However, they do not adequately support data provenance, which is of vital importance in the scientific domain. Distributed logging tools like Net Logger [11] that provide generic logging support for distributed applications lack the higher level abstractions necessary for a service oriented architecture. They can potentially be used in place of the notification broker in our system. Instrumentation and performance analysis systems like svPablo [6] and AutoPilot [19] are oriented towards realtime monitoring of distributed applications to tune application performance and optimize resource allocation, but do not help with holistic tracking of the dataflow and workflow execution.

Surveys on provenance systems include a meta-model for a systems architecture for lineage retrieval [3], and present a taxonomy of approaches taken towards building provenance systems [20], and exemplify use cases for a provenance system in biology [14].

8. Conclusion and Future Work

In this paper, we have identified challenges involved with collecting provenance – both data and workflow provenance – for data-driven applications. The Karma provenance framework we propose provides a generic solution for collecting provenance for heterogeneous workflow environments. The activity model is adapted for publish-subscribe systems that provide ubiquitous means for interoperable collection of provenance, and we leverage this to implement provenance collection with low perturbation. Use of the WS-Eventing standard allows choice of the most suitable pub-sub implementation for the domain. The user involvement is limited to instrumenting the workflow components to generate the provenance notifications. This is similar to logging information that workflows usually generate for debugging and the libraries we provide for automated provenance publishing alleviates the burden on the service and application providers. Current work to automate the generation of application scripts through higher level abstractions will reduce this effort even further. Our evaluation of the framework implementation also shows that provenance collection can be done with minimal performance overhead on the workflow, on the order of 1% of the execution time for 271 files used. Initial results from incorporating batch publishing shows promising results with a sub-linear performance overhead as the number of files used increases.

The framework described here is the basis for our ongoing research in the use of provenance to automatically determine data product quality based on user guided met-

rics and through collaborative feedback on the products and services participating in the data provenance [22]. In addition, we are examining a number of open issues. Provenance collection from hierarchically composed workflows, i.e. workflows execution invoking other workflows as service, poses a challenge because the provenance activity must occur at both workflow and service levels simultaneously. Long running workflows increase the probability of incomplete provenance. Using this information effectively is an area under investigation, as is handling missing notifications, possibly using WS-Reliable Messaging [15]. Also of interest are data products that appear in collections that are themselves data products. Provenance for the collection data products and for individual data products within them needs to be tracked without breaking the transparency offered by uniform naming.

References

- [1] IBM web services data collector <http://www.alphaworks.ibm.com/tech/wsdatacollector/>, 2005.
- [2] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services Version 1.1*. BEA Systems and International Business Machines Corporation and Microsoft Corporation and SAP AG and Siebel Systems, May 2003.
- [3] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv.*, 37(1):1–28, 2005.
- [4] C. Catlett. *The TeraGrid: A Primer*. TeraGrid, September 2002.
- [5] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang. Programming scientific and distributed workflow with triana services. *Concurrency and Computation: Practice and Experience*, 2005. In Press.
- [6] L. A. de Rose and D. A. Reed. Svpablo: A multi-language architecture-independent performance analysis system. In *International Conference on Parallel Processing*, page 311, Washington, DC, USA, 1999. IEEE Computer Society.
- [7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.
- [8] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6), 2002.
- [9] D. Gannon, B. Plale, M. Christie, L. Fang, Y. Huang, S. Jensen, G. Kandaswamy, S. Marru, S. L. Pallickara, S. Shirasuna, Y. Simmhan, A. Slominski, and Y. Sun. Service oriented architectures for science gateways on grid systems. In *ICSOC*, 2005.
- [10] P. Groth, S. Miles, W. Fang, S. C. Wong, K.-P. Zauner, and L. Moreau. Recording and using provenance in a protein compressibility experiment. In *HPDC*, 2005.
- [11] D. Gunter, B. Tierney, K. Jackson, J. Lee, and M. Stoufer. Dynamic monitoring of high-performance distributed applications. In *HPDC*, pages 163–170, 2002.
- [12] Y. Huang, A. Slominski, C. Herath, and D. Gannon. Wsmessenger: A web services-based messaging system for service-oriented grid computing. In *CCGrid*, 2006.
- [13] B. Ludscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice & Experience*, 2005. To Appear.
- [14] S. Miles, P. Groth, M. Branco, and L. Moreau. The requirements of recording and using provenance in e-science experiments. Technical report, Electronics and Computer Science, University of Southampton, 2005.
- [15] S. Pallickara, G. Fox, B. Yildiz, S. L. Pallickara, S. Patel, and D. Yemme. On the costs for reliable messaging in web/grid service environments. In *e-Science*, 2005.
- [16] B. Plale. Resource requirements study for lead storage repository. Technical Report 001, Linked Environments for Atmospheric Discovery, 2005.
- [17] B. Plale, J. Alameda, B. Wilhelmson, D. Gannon, S. Hampton, A. Rossi, and K. Droegemeier. Active management of scientific data. *Internet Computing*, 09(1):27–34, 2005.
- [18] B. Plale, D. Gannon, Y. Huang, G. Kandaswamy, S. L. Pallickara, and A. Slominski. Cooperating services for data-driven computational experimentation. *Computing in Science and Engineering*, 7(5):34–43, 2005.
- [19] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed. Autopilot: Adaptive control of distributed applications. In *HPDC*, page 172. IEEE Computer Society, 1998.
- [20] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3), 2005.
- [21] Y. L. Simmhan, B. Plale, and D. Gannon. Performance evaluation of the karma provenance framework for scientific workflows. *LNCS*, 2006. To Appear.
- [22] Y. L. Simmhan, B. Plale, and D. Gannon. Towards a quality model for effective data selection in laboratories. In *IEEE Workshop on Workflow and Data Flow for Scientific Applications*, April 2006.
- [23] R. D. Stevens, A. J. Robinson, and C. A. Goble. mygrid: Personalised bioinformatics on the information grid. *Bioinformatics*, 19 Suppl 1:i302–i304, 2003.
- [24] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
- [25] B. van Dongen, A. de Medeiros, H. Verbeek, A. Weijters, and W. van der Aalst. The prom framework: A new era in process mining tool support. *LNCS*, 3536:444–454, 2005.
- [26] K. West. Scoping out the planet. *Scientific American*, 5:40–42, November 2005.
- [27] J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3), 2005.
- [28] J. Zhao, C. Goble, and R. Stevens. An identity crisis in the life sciences. In *IPAW*, 2006.
- [29] Y. Zhao, M. Wilde, and I. T. Foster. Applying the virtual data provenance model. In *IPAW*, 2006.