

Karma2: Provenance Management for Data Driven Workflows *Extended and invited from ICWS 2006 with id 217*

Yogesh L. Simmhan, Beth Plale, Dennis Gannon
Indiana University, Bloomington, IN 47405 USA
{ysimmhan, plale, gannon}@cs.indiana.edu

ABSTRACT:

The increasing ability for the sciences to sense the world around us is resulting in a growing need for data driven applications that are under the control of workflows composed of services on the Grid. 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 is based on generating discrete provenance activities during the lifecycle of a workflow execution that can be aggregated to form complex data and process provenance graphs that can span across workflows. The implementation uses a loosely-coupled publish-subscribe architecture for propagating these activities and the capabilities of the system satisfies the needs of detailed provenance collection. 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).

KEY WORDS:

Provenance, Workflows, Metadata, Process Mining, Data Management

INTRODUCTION

The need to access and share large scale computational and data resources to support dynamic computational science and agile enterprises is driving the growth of Grids (Foster, 2002). In the realm of e-Science, *science gateways* are archetypes for accessing, managing and sharing virtualized resources to solve large collaborative challenges (Gannon, 2005; Catlett, 2002). Science gateways are built as a *service oriented architecture* with Grid resources virtualized as services. These resources, including physical resources such as sensors, computational clusters, and mass storage devices, and software resources like scientific tasks and models, are available as services that provide an abstraction to access the resources through well defined interfaces.

A significant constituent of applications that make use of the science gateways are *data driven applications* (Simmhan, 2006a). The proliferation of wireless networking and inexpensive sensor technology is allowing the sciences an increasing ability to sense the world around (West, 2005). This is specifically resulting in a growing need for data driven applications, that is, applications that can be computation intense and are usually either dataflow applications, in which data flows from one process to another, or demand driven, in which computations are triggered in response to events occurring in the world around us. Data driven scientific experiments are designed as workflows composed of services on the Grid and data flows from one service to another, being transformed, filtered, fused, and used in complex models. These workflows capture the invocation logic for the scientific investigation and may be composed of hundreds of services connected as complex graphs. Data driven workflow executions also see the participation of thousands of data products that reach terabytes in size. At this scale of processing, users need to

ability to automatically track the execution of their experiments and the multitude of data products created and consumed by the services in the workflow. Provenance collection and management, also called process mining, workflow tracing, or lineage collection, is a new line of research on the execution of workflows and the derivation and usage trail of data products that are involved in the workflows (Simmhan, 2005; Bose, 2005; Moreau, 2007).

Provenance collected about the tasks of a workflow describes the workflow's service invocations during its execution (Simmhan, 2005). This helps track service and resource usage patterns, and forms metadata for service and workflow discovery. In data driven applications, however, it is provenance about the data that is central to understanding and recreating earlier runs. In *data driven 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 kind of information is extremely valuable not only for diagnosing problems and understanding performance of a particular workflow run, but also to determine the origin and quality of a particular piece of derived information (Goble, 2002; Simmhan, 2006b).

Current methods of collecting provenance are from workflow engines logs (IBM, 2005) or by instrumenting the services (Bose, 2004; Zhao, 2004). In the former case, the logs from the workflow engine are at the message level and insufficient for deciphering provenance about the data products, while instrumenting services introduces a burden on the service author to modify their service to generate provenance metadata. They also tend to be specific to the workflow framework and are not interoperable with heterogeneous workflow models that are likely to be present in a Grid environment. Work is also emerging on more general models for provenance collection (Moreau, 2007).

The challenge we address in our work is to record uniform and usable provenance metadata independent of the workflow or service framework used, while minimizing the modification burden on the service authors and the performance overhead on the workflow engine and the services. Karma collects two forms of provenance: *Process provenance*, also known as workflow trace (Simmhan, 2005), is metadata describing the workflow's execution and associated service invocations; and *Data provenance*, which provides similar metadata about the derivation history of the data product, including services used and input data sources transformed to generate it. These forms of provenance allow scientists to monitor workflow progress at runtime (Gannon, 2005) and, post-execution, mine the provenance to locate sources of errors, determine data quality (Simmhan, 2006b), and validate the results through repetition or simulated replay of the workflow execution (Szomszor, 2003). It also assists resource providers to review resource usage for purposes of auditing (Greenwood, 2003) and provisioning (Churches, 2006).

This article details the Karma provenance framework for collecting and managing provenance for data driven workflows. The current Karma system, which we describe in this article, builds upon previous exploratory work we undertook on provenance management; hence the allusion to the present Karma version 2 or Karma2. The remainder of the paper is organized as follows. In Section 2, we motivate the need for provenance collection with an example from mesoscale meteorology, which is an exemplar domain for data driven applications. In Section 3, we define a generalized data driven workflow model for which the provenance is collected. In section 4, we discuss provenance activities that occur during the lifetime of the workflow that go towards building provenance. Section 5 describes the provenance model of Karma and the different forms of provenance that are available. Section 6 grounds the provenance model in an implementation

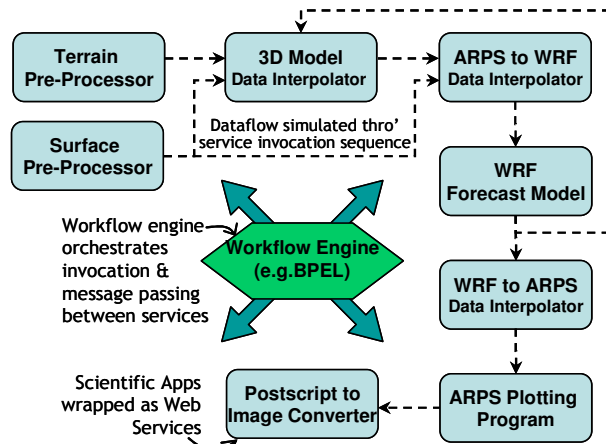


Figure 1. Sample LEAD Weather Forecasting Workflow

of the framework that we discuss. Section 7 presents details on performance evaluation of provenance collection and querying. Related work appears in Section 8 and future work in Section 9.

MOTIVATION

Scientific workflows (Yu, 2005) often use a service oriented architecture to solve advanced scientific problems in a Grid environment (Gannon, 2005). 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 graphs whose edges represent dependencies and dataflows, and incorporate decision-making logic for runtime selection of execution paths (BEA, 2003). There are different models for workflow execution. Orchestration can be done by a central workflow engine that interprets the workflow document and invokes the services according to the dependencies, ensuring dataflow between connected services. Execution may also be handled in a distributed manner by the services themselves.

In the Linked Environments for Atmospheric Discovery (LEAD) (Droegemeier, 2005) project, regional-scale numerical weather forecasting is done using dynamically adaptive workflows, such as shown in Figure 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 (Liu, 2005; Ludäscher, 2006). The dynamic nature of the workflows mean that the workflow execution path is not static and can change due to external events, while adaptability implies the ability to bind to an appropriate and available service instance at the time of workflow execution.

A typical meteorological experiment shown in Figure 1 is a weather prediction workflow 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 is composed in the Business Process Execution Language (BPEL) and orchestrated by a central BPEL engine (Slominski, 2006). The workflow can be broadly divided into pre-processing and grid interpolation stage (Services 1 – 4 in Figure 1, starting with Surface Pre-Processor and moving clockwise), 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 transparently referenced through URIs (Plale, 2005). This workflow, applied to data from Hurricane Katrina's landfall on the Gulf Coast in 2005, is used for the performance evaluation in Section 7. More complex workflows with high degrees of parallelism and complex conditional logic are also seen in LEAD when running ensemble simulations (Droegemeier, 2005).

Provenance is collected from such workflows in order to verify that a simulation run was successful and to validate its results. This may be through visual inspection of the provenance graphs or through automated processes. This is important, given the dynamic and adaptive nature of the workflows, to determine what really happened. Data provenance also helps to keep track of data products used by and generated from the experiments and discover them for reuse at a later time. This can even avoid running the workflow if an identical run made by another user has generated the required output data and the workflow is deterministic. Data quality models have been developed to mine historic provenance information for patterns of service execution that can assist in making quality predictions about data products generated by services as a function of their input data and parameters (Simmhan, 2006b).

WORKFLOW MODEL

An abstract workflow model, for which provenance is collected, is necessary to ensure that the provenance model is independent of the workflow implementation or design. This generalized data driven workflow model is based on a service oriented architecture, with four principal components: *workflows*, *services*, and *service clients*, and *data products* that are produced and consumed within the workflow. Services are black-box tasks accessed through a well defined interface. They frequently are but not limited to web services by the model. The workflow is a *directed graph* of services, with the services forming nodes in the graph and its edges constituting dataflow between services or applications wrapped by the service. The graph can have cycles, such as iterative loops. Workflows themselves are considered as services, allowing them to participate in other workflows. This allows for a hierarchical pattern of constructing workflows by reusing existing workflows and services.

Data products, often logical or physical files, are passed as inputs to and generated as outputs from a service invocation. The data products are identified by a globally unique ID in the request and response messages, and they are said to be *consumed* by that service invocation if used as input, while those newly created by that invocation are said to be *produced* by it. Every service invocation in a data driven workflow consumes or produces data products, and these form their own dataflow graph which is captured as part of the provenance.

Service Invocation and Workflow Execution Paradigms

There are several invocation paradigms for running workflows (Yu, 2005). In a *centralized model*, such as used by BPEL, a central workflow engine that has the workflow dependency logic invokes the services in the workflow in succession, passing messages between the services and acting as their client. Alternatively, the control could be distributed, with different engines being aware of a portion of the workflow graph (e.g. one engine for each workflow in a nested workflow) and acting as clients at different times of the workflow lifecycle. In a *component model* (Armstrong, 1999; OMG, 2006), each service in the workflow would be aware of the subsequent service(s) "connected" to it and invoke the next service(s) once its invocation completes, thus acting as a client to the next service(s).

An invocation of a service by a client, be it a workflow engine, another service, or just a standalone client, can either be *synchronous* or *asynchronous*. In the former, the invocation request message is sent to the service and the client blocks till the response message – which may be a successful result or a fault – is returned by the service. In the latter, a request by the client returns an acknowledgement message synchronously (if the request was received successfully, an exception otherwise), while the actual response from for the invocation arrives at a later time, say, in the form of a callback to the client.

All these paradigms are possible in the generalized workflow model and hence the provenance model we describe is compatible these too. In the interest of space, we restrict our examples in this article to a centralized workflow engine that acts as an asynchronous client to the services.

Resource Naming and Unique IDs

One fundamental requirement of provenance is the ability to uniquely identify the data products whose provenance is being described, as also the services that use or generate these data products. This is driven by the fact that the service and data products are distributed across organizational and geographical boundaries, and also by that the provenance needs to be preserved for a period longer than the lifetime of the services or the workflows themselves. All resources that participate on our workflow model, including running or completed workflow instances, service instances, and data products are assigned a globally unique identifier (GUID). These are referred to as *Workflow ID*, *Service ID*, and *Data Product ID*, respectively. It is possible to transparently map from these logical IDs to the actual physical resources, which may be an endpoint reference or a data URL, through a naming service. Use of such logical handles to resources instead of physical addresses makes the provenance independent of the data product representation since they are identified by just an ID, which could equally well point to a remote file, a database tuple, or a virtual collection. The format of the GUID does not matter as long as its uniqueness is guaranteed. Another identifier that is assigned to services in the context of a composed workflow is the *Workflow Node ID*. This ID is distinct for every service (node) in an abstract workflow. This differs from the Service ID in that late binding of workflows allows them to use different service instances for the same node when invoked in different iterations.

PROVENANCE ACTIVITIES

Provenance is collected during the lifecycle of the generalized workflow model in the form of discrete activities that are expressed using an activity object. These activity objects are generated by the services and clients that participate in the workflow during a service invocation, and are distributed over *time*, *space*, *depth*, and *type of operation*. Some of these activities define the boundaries of the invocation while others describe the operation being performed. These provenance activity objects sufficiently describe the workflow execution and the dataflow to enable reconstruction of the workflow and data provenance graph exclusively from this runtime information without requiring access to the original workflow graph that was composed.

Activity Dimensions

The workflow model executes as a series of invocations over time that result in devolvement of control to different depths in the workflow execution, and within each invocation context, operations defined in the service implementation are performed. Activity objects can be distributed over several dimensions in the activity space and certain attributes present in the activity object will help place the activity at the exact point in this space. This allows the

activities to identify the invocation state and the operations that take place as part of the invocation unambiguously.

Starting with a client that initiates the workflow, control passes to the services in the workflow, and deeper still into nested workflows that these services may represent. The dimension of *depth* gives the distance of a service invocation from the root of the workflow graph, and holds for all activities generated by that service invocation. The depth comes out through the parent-child relationship between the workflow and the invoked service that is part of that workflow. The activity object contains the globally unique *Workflow ID* for the workflow that this service invocation is part of as well as the *Service ID* for the service. If the service represents another workflow, its service invocations would also contain the *Workflow ID* of child workflow as well as their own service IDs. These *Workflow ID*–*Service ID* pairs makes it possible to determine the depth of an invocation in the workflow graph in a recursive manner.

Activities take place as discrete events ordered over time, and the logical time forms an additional dimension to distinguish between activities. A *logical timestamp* attribute present in the activity helps with temporal ordering of the activities, in addition to causal ordering that may be implicit. For example, during an invocation, the invocation started activity would have to appear before the invocation finished activity, and can hence be implicitly ordered, while activities that described several data products being generated by the invocation may have appeared in any order unless explicitly sequenced using a logical timestamp. Logical time is also necessary to order across service invocations if there is no causality between them in the form of a dataflow from one to another. While a dataflow between all services is the norm in data driven workflows, we presuppose a globally synchronized logical time to support non-dataflow workflows too.

The services and clients are distributed in *space* and the workflow execution takes places across different services. The ability to identify the service instance through a globally unique ID helps distinguish between instances in space. Finally, the different *operations* that take place as part of the invocation, including dataflow operations that produce or consume a data product, are a separate dimension of the activity object. Depending on the type of operation, there may be additional attributes to fully describe them.

Activity Types

Bounding activities describe the interaction between two entities, the client invoking a service and the service being invoked. These activities are generated at the boundaries of the invocation, when an invocation request or response is being exchanged between the service and the client. For each of the request and response message, three activities are generated – two by the initiator of the message before and after sending it and one by the receiver of the message upon receiving it. In the case of a invocation request message from the client to service, this includes the *InvokingService* activity before sending the request message to the service and the *ServiceInvoked* activity after the request message has been sent to the service. The service generates one *ServiceInvoked* activity upon receiving the request message. Similarly for the response sent by the service to the client, three activities are generated as shown in Table 1.

Activity	Generated By	Attributes Timestamp, Description, and Annotation <i>are common to all activities</i>
ServiceInitialized	Service	<u>Service ID</u> , isWorkflow
ServiceTerminated	Service	<u>Service ID</u>
InvokingService	Client	<i>Client Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, Request Message,
ServiceInvoked	Service	<i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Client Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, Request Message
InvokingService [Succeeded Failed]	Client	<i>Client Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, [— Failure Trace]
DataTransfer	Service	<i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <u>Data ID</u> , <u>Source URL</u> , <u>Target URL</u> , <u>Size</u> , <u>Duration</u>
Computation	Service	<i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, Application, <u>Duration</u>
DataProduced	Service	<i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Data Product</i> + { <u>Data ID</u> , URL+, Size, Timestamp}
DataConsumed	Service	<i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Data Product</i> + { <u>Data ID</u> , URL+, Size, Timestamp}
SendingResponse	Service	<i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Client Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, Response Message {Result Fault}
ReceivedResponse	Client	<i>Client Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, Response Message {Result Fault}
SendingResponse [Succeeded Failed]	Service	<i>Service Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, <i>Client Entity ID</i> { <u>Service ID</u> , Workflow ID, Workflow Node ID, Logical Timestamp}, [— Failure Trace]

Table XX. List of provenance activities, the source of the activity, and the attributes present in each. Attributes that are underlined are required. Those in italics are complex attributes composed of those in curly braces. A '+' after an attribute represents an array, while a choice is denoted by a '|'.

The activity objects that are generated contain several attributes that describe the activity's dimensions and uniquely identify the entities that participate in the interaction. The state of the client and service entities at the time of invocation is given by a set of attributes collectively referred to as the *entity ID*. The entity ID for a service requires the Service ID. If this service invocation is as part of a workflow run, the entity ID additionally has the Workflow ID of that workflow, the Workflow Node ID of this service in that workflow, and logical timestamp of this invocation. If the client were another service, then it would have similar attributes (Service ID and, optionally, its Workflow ID, Workflow Node ID, and logical timestamp) but reflecting its own state when initiating the invocation. Else, the client would just identify itself through the Service ID attribute. Having both the client and service entity IDs present in the bounding activities serves two purposes: one, it allows the client and service to present their individual views of the invocation information so that any conflicting views can be detected, and two, in the absence of activities generated by either the client or the service, it provides the minimum, though one sided, information required to reconstruct the invocation chain.

Operation activities describe the tasks that take place within a service invocation and occur between the bounding activities. The operations that take place depend on the type of service. However, for a data driven workflow, two operations that are guaranteed to take place and are of interest to provenance are the consumption and generation of data products. When a service uses a data product in an invocation, the service is said to have consumed the data product and it generates a **DataConsumed** activity. When a service creates a new data product during the course of an invocation, it generates a **DataProduced** activity. These consumed and produced data products may or may not have been passed as parameters present in the request or response messages sent to/from the service. Other than the entity ID of the service, these activities contain the Data Product ID of the data that was produced or consumed as attribute. In addition to these *dataflow* operations, two other operations that usually take place in scientific workflows are data transfer and computation. We define these two additional activities given the nature of the e-science project that Karma was designed for. The operation activities and their attributes are listed in Table 1.

Example

The set of activities generated by a service invocation or a workflow changes with the pattern of execution. For example, a service could be invoked synchronously or asynchronously, and a workflow can be invoked by a workflow engine or enact as components in a dataflow. We present two examples of service and workflow execution with corresponding provenance activities generated for them and plotted on a sequence diagram.

Figure 2(a) shows a client invoking a service, passing data product D1 as input and receiving D2 as output. This invocation can take place synchronously or asynchronously, and the sequence diagram for each of these styles of execution is shown in Figure 2(b) and 2(c) respectively. The diagrams show the transfer of control from the client to the service at different points in the invocation over *time*, shown in the Y Axis. The activities produced at each state are marked with a red circle and the type of activity is on the left column. The center column shows the distribution of the client and service entities over *space* and their *depth*. Since this case has just 2 levels of depth – the client and the service – the depth shown on the top X axis moves from the solid line to the dotted line. The spatial location of the entity, shown on the lower X axis, varies with each unique entity. The vertical lines in the right column represent different *operations* that the service performs during its invocation.

In the synchronous invocation shown in Figure 2(b), once the service is started and generates the *ServiceInitialized* activity, the client invokes the service. The client generates an *InvokeService* activity and the calls the service, and control passes to the service. The service generates the *ServiceInvoked* activity and starts the execution. This involves transferring the input data D1 (*DataTransfer*), using the data product (*DataConsumed*) in a computation or application (*Computation*), producing the output data product D2 (*DataProduced*), and staging the output to a remote location (*DataTransfer*). When the service operations are complete, the service generates the *SendingResponse* activity before returning control and the result to the client. The client, upon receipt of the response, generates the *ReceivedResponse* activity. The service can then terminate at a later point in time after generating the *ServiceTerminate* activity.

The asynchronous call shown in Figure 2(c) is similar but has two variations. After receiving the asynchronous request, the client does not block and receives control in parallel to the service execution (as shown by the dotted arrow). When the request has been sent by the client successfully, it generates the *InvokingServiceSucceeded* activity. Likewise, after the service returns the response to the client, it gets back control in parallel to the client and generates a *SendingResponseSuccess* activity. This three phase protocol is necessary to track the fact that the invocation cycle was successful since either of the sending request or receiving response steps could potentially fail.

A more complex scenario is shown in Figure 2(a) where a nested workflow executes. Workflow 1 consists of two services S1 and WF2, where service WF2 is another workflow. WF2 in turn comprises of two services S2 and S3 that are connected. All the services take in one data product as input and generate another data product as output, which is used by the subsequent service. The sequence diagram in Figure 3(b) shows an asynchronous execution of the workflow when orchestrated by a workflow engine for each workflow. Before the workflow executes, services (including the workflows) are initialized, generating *ServiceInitialized* activities. A client (not shown in the figure) initiates the workflow and control passed to the workflow (engine) WF1. The sequence diagram starts from this stage. The workflow WF1 starts executing by invoking the

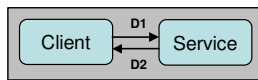


Figure 2 (a) Client invoking a service, passing data product D1 as input and receiving D2 as output

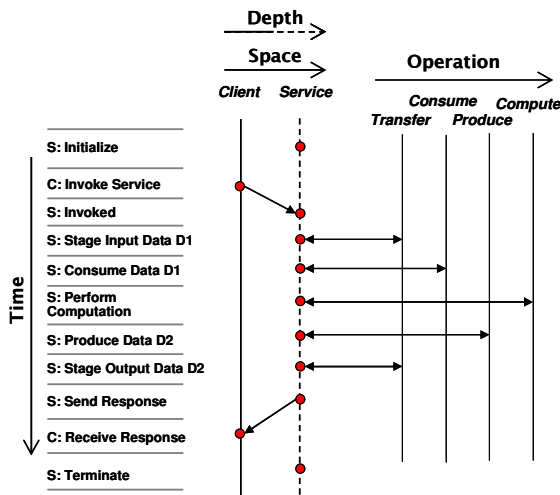


Figure 2 (b) Sequence diagram of a synchronous invocation of Figure 2(a)

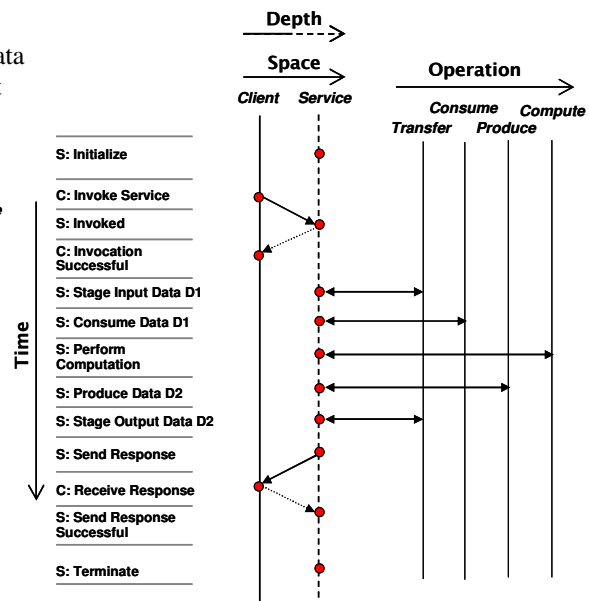


Figure 2 (c) Sequence diagram of an asynchronous invocation of Figure 2(a)

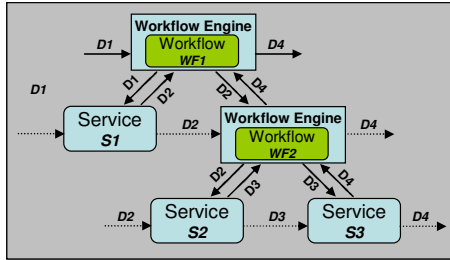


Figure 3 (a). Nested workflow. The root workflow WF1 contains two services S1 and WF2, where the latter is another workflow that is nested. WF2 in turn has two services, S2 and S3. All services and workflows produce and consume one data product each.

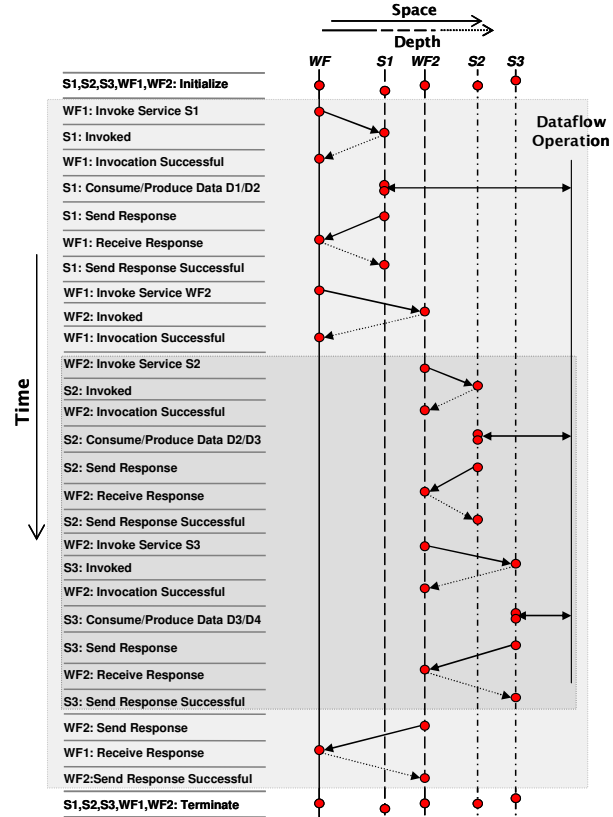


Figure 3 (b). Sequence diagram for nested workflow in Figure 3(a).

first service in the workflow, S1, with input parameter data product D1. Since WF1 acts as the client to S1, it generates `InvokingService` and `InvokingServiceSuccess` activities before and after calling S1. Service S1 generates `ServiceInvoked` activity after being invoked by WF1. The service performs operations such as consuming and producing data products (`DataConsumed` and `DataProduced`), computation and file transfers (not shown). When execution completes, S1 asynchronously returns the response message to the workflow WF1, in the process, generating `SendingResponse` and `SendingResponseSuccess` activities. WF1 receives the response for the invocation and generates the `ReceivedResponse` activity.

WF1 then invokes service WF2 that is next in the workflow sequence. Though WF2 is actually a workflow, WF1 interacts with it just as with any other service, generating the same activities as a service client, viz. `InvokingService`, `InvokingServiceSuccess`, and `ReceivedResponse`. WF2 upon receiving the invocation executes the workflow as if it received an invocation from a client and proceeds to invoke services S2 and S3. These generate the usual activities as seen before. When these services return, the final response from S3 is returned by workflow WF2 to workflow WF1, its client. Workflow WF1 then continues to completion.

PROVENANCE MODEL

The provenance model describes the structure of the provenance metadata exposed by Karma (Simmhan, 2007). It is an aggregation of the various facets portrayed by the provenance activities and this information model forms the basis for querying over and exporting of provenance. The model describes two primary types of provenance, one centered on service invocations, called

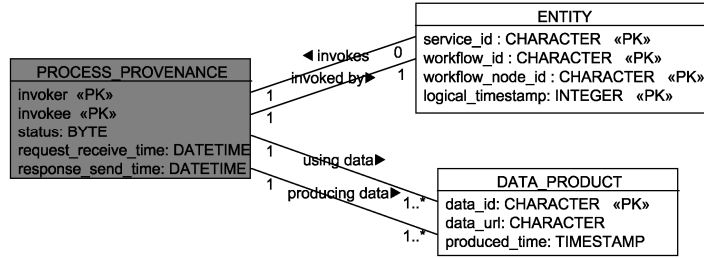


Figure 4. Process Provenance Model

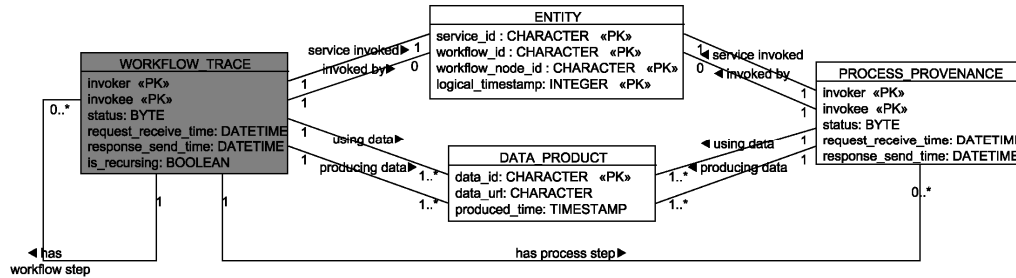


Figure 5. Workflow Trace Model

process provenance, and the other relating to data usage and generation, termed *data provenance*. These two concepts are complementary in that process provenance describes service invocations and makes a reference to the input and output data products to the invocation, while data provenance describes the data product and makes references to services that use it and the invocation that created it. They provide two different views over the global provenance information available from the activities.

Process Provenance

Process provenance describes the execution of a single process, which translates to an invocation of a service instance by a client. The process provenance information model, shown in Figure 4, identifies the service that was invoked and the client that invoked the service using the entity IDs described in the activities. These comprise of the relevant attributes from Service ID, the Workflow ID, Workflow Node ID, and the Logical Timestamp of the service and the client at the time of invocation. It also contains attributes such as the actual timestamp of the invocation, the status of the invocation, and optionally, the messages that were exchanged by the entities. The process provenance also references the data products that were used by the invocation and that the invocation created. The data products are identified by their unique data product ID along with optional attributes describing the location of the data and the timestamp of their generation or use. When this model is seen against the activities listed in Table 1 and the database model that will be described (Figure 10), it is apparent that the activities act as building blocks for constructing the provenance model.

While process provenance describes a single service invocation, the complete workflow run is captured by the *workflow trace* shown in Figure 5. The workflow trace is a coarser degree of abstraction of the provenance and includes zero or more records of process provenance relating to service invocations that were part of the workflow execution. In addition to the service invocation steps, workflow traces can also recursively refer to other workflow traces in the case of hierarchically composed workflows. The depth of recursion is configurable, allowing an arbitrary level of granularity at which process provenance can be viewed.

Data Provenance

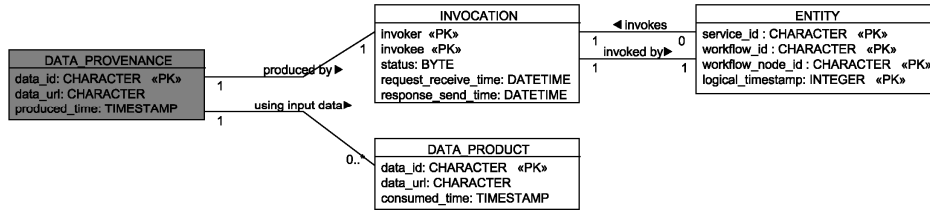


Figure 6. Data Provenance Model

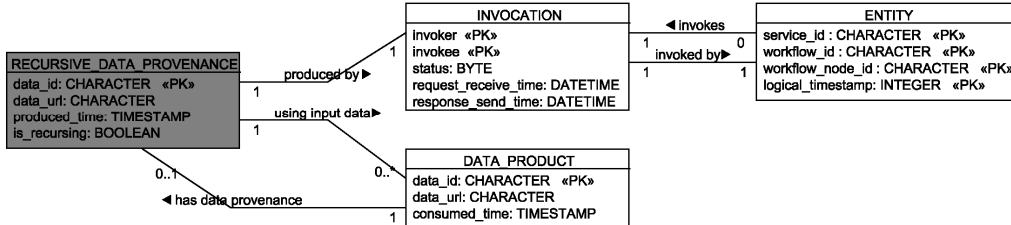


Figure 7. Recursive Data Provenance Model

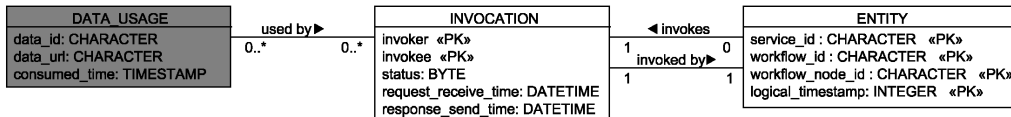


Figure 8. Data Usage Model

Data provenance describes the derivation path of a data product as the service invocation that created the data product and the inputs to the invocation. Figure 6 shows the data provenance model where the data is identified by the unique data product ID and produced by an invocation. The invocation comprises of a service entity being invoked by a client entity using zero or more data products as inputs to the data. The invocation attributes are similar to those for the process provenance record seen earlier; likewise the attributes for the data products.

Data provenance is, by default, the immediate data derivation history – the service invocation that directly created this data. A recursive form of data provenance, analogous to the workflow trace, is the deep or *recursive data provenance* (Cohen-Boulakia, 2007), shown in Figure 7. Deep data provenance recursively tracks the data provenance for the inputs to the invocation up to an arbitrary depth. This can be used to build the complete historical record of the ancestral data and invocations that caused this data to be derived.

Data provenance looks back in time on the derivation chain. A similar model can be constructed when moving forward in time and tracking the usage of a data product in different invocations. The *data usage* model, shown in Figure 8, describes the service invocations that use a certain data product as their input. The invocations are described by the service invoked and its client, while the data is identified by the data product ID and optional locations and timestamp of when it was used.

The different forms of process and data provenance are graph structures. The workflow trace consists of service invocation nodes and data product edges, with the process provenance records forming an edge-node-edge sub-graph. Similarly, the data provenance and data usage have data products as nodes and service invocations as edges.

IMPLEMENTATION

The *Karma framework* implementation the above provenance model consists of a focal web service to collect provenance activities and query over the provenance model, client libraries to generate the provenance activities, and graphical interfaces to visualize the provenance graphs and monitor workflow execution (Figure 9). Provenance activities are modeled as XML documents whose schema maps to activity attributes. The clients and services participating in a service or workflow execution can use the provided *workflow tracking* client library to generate the provenance activity documents shown in Table 1. The Java library maintains the state information for the service or client during the invocation and fills in the relevant entity ID fields for all activities once they are initially specified. It provides a simple API, akin to logging tools, for clients and services to populate the activity attributes for the bounding and operation activities. When the services being executed are web services, Karma also provides a SOAP library that can encode the entity ID information of the client and service in the SOAP header of the request and response messages, to allow transparent exchange of client and service state information that each need to fill in the activities they produce. This allows existing web services to generate provenance activities without having to change the WSDL definition for the web service.

The provenance service provides a web service interface for services and clients in the workflow to submit the provenance activities they generate. In addition to a synchronous submission of the activities, the implementation also allows an asynchronous publishing of activities as notifications that the provenance service subscribes to. The provenance service shreds the XML provenance activities and stores them in a relational database. This can then be queried through the web service API that provides methods to retrieve each of the five provenance models described earlier through their IDs.

Publishing Activities as Notifications

A *Publish-Subscribe* (or Pub-Sub) notification protocol is a uni-directional, asynchronous communication between publishers and consumers that communicate by means of “channels” or

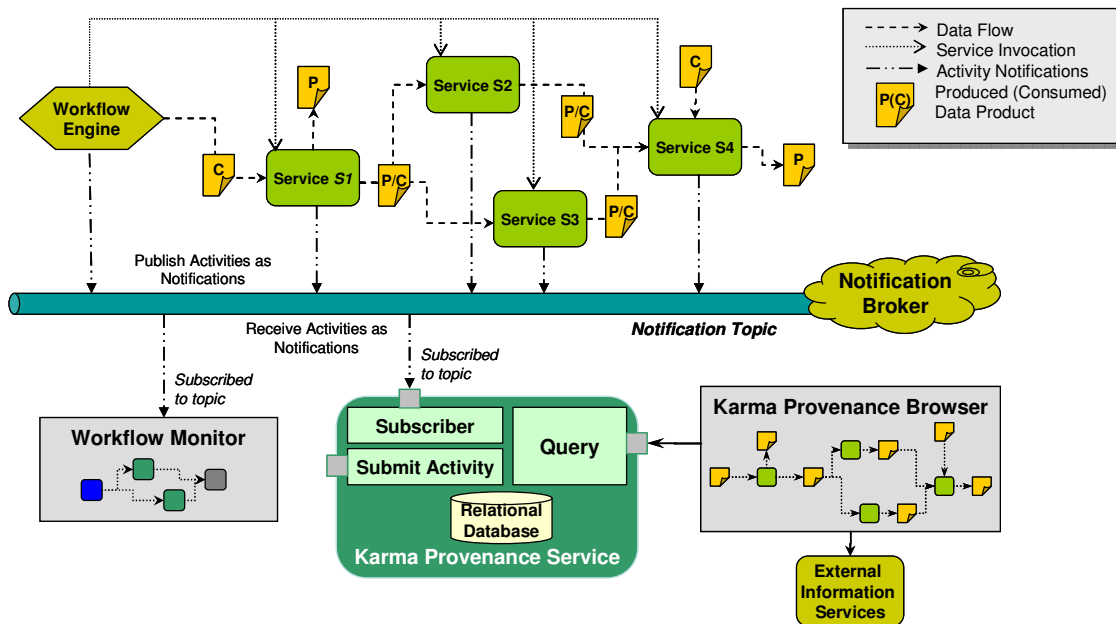


Figure 9. Interaction between Karma Provenance Service and Workflow

“topics”. Consumers, *decoupled in space* from the publisher, receive events by subscribing to one or more topics (Eugster, 2003).

A notification protocol is a good choice for publishing provenance activities from the workflows, services, and clients since provenance gathering can be done without perturbing the publishers and can be collected independent of the provenance service location. *Time decoupling* (Eugster, 2003), 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 many notification brokers remove the onus from workflow components in ensuring reliable delivery of activity messages. Since provenance is more often used for mining and analysis post workflow execution than for runtime monitoring of the workflow, users are tolerant to delayed receipt of activities at the provenance service. Pub-sub is a mature field with several open-source and commercial implementations, and many open standards, such as JMS and WS-Eventing (Box, 2004), which aid interoperability when collecting provenance across different domains, as is common in Grid systems.

A representative notification service is *WS-Messenger* (Huang, 2006), that uses a topic based publish-subscribe system based on the WS-Eventing standard (Box, 2004). 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.

For each workflow execution, a unique notification topic is created with the notification broker. All components in the workflow and its clients use the same broker and 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 provenance activity notifications that are generated from all workflows that use that broker.

Database Model

The Karma provenance service uses a relational database to store the activities that it receives for a workflow execution, as notifications or directly submitted through the web service API. The database schema is shown in Figure 10 (Simmhan, 2007). As XML activities arrive, the provenance service extracts the relevant attributes from them and creates or updates the tables in the database. The tables closely resemble the provenance data model with entities described in the *entity table* and the *invocation table* referencing client and service entity pairs along with the state of the invocation. Data products present in the *data product table* are consumed and produced by the invocations as captured by the *data produced* and *consumed tables* that relate the two. A separate *service table* is present to record information about the service instances, such as their initialization and termination times, and possibly their WSDL in the case of web services. An additional *activity table* keeps a log of all activities that were used to populate the tables. This

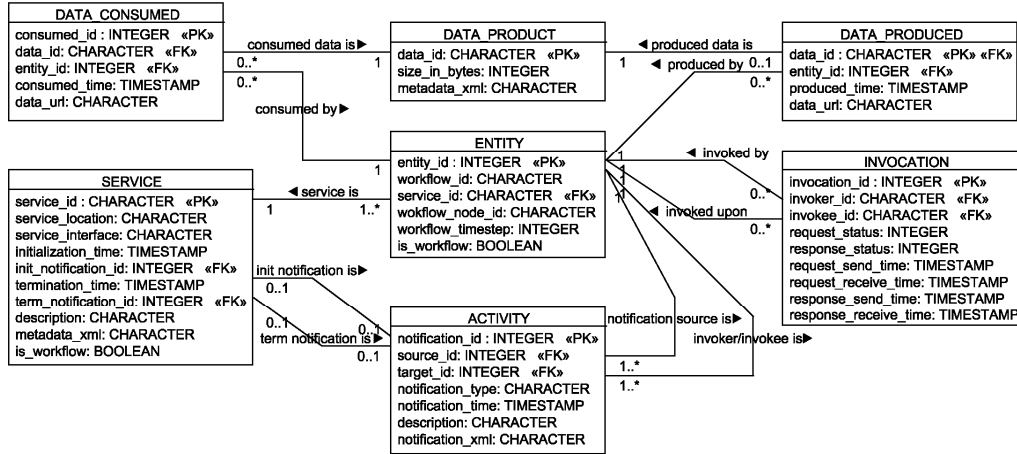


Figure 10. Relational Database model for storing provenance (Simmhan, 2007)

forms a provenance or audit trail about the provenance itself in case the provenance needs to be verified in future. It should also be noted that provenance is immutable and is only appended to over time as new activities take place, say when a service invocation changes from an invoked state to invocation complete state. Once all activities relating to an invocation have been received, its process provenance and the data provenance for data products created by the invocation are static.

Mapping the provenance model to a relational database helps to keep the implementation compatible with different representation of the activities or of the provenance graphs. Any other transport mechanism or metadata representation besides XML can be used to gather the activities as long as all attributes for each activity are present. These will require simple conversions from the activity’s format to the database/provenance model. Similarly, the provenance graphs can be exported in formats other than in the present XML schema, as may be required by visualization applications (Brandes, 2002). Efforts are emerging to standardize provenance representation (Moreau, 2007) and our approach allows us to interoperate with any future provenance standard that follows the broad principles of provenance being recorded as a causal graph. The availability of mature relational databases also allows for an efficient implementation of the provenance service that gives superior performance for activity updates and provenance queries as is further described in the ensuing section.

Provenance Queries

The Karma provenance service provides APIs to query over the provenance model and returns the various provenance graphs using an XML representation. The XML schema for each of the five types of provenance graphs maps to the information model for each provenance graph as shown in Figures 4 – 8. The query API has a method for each of the five different provenance graphs, namely process provenance, workflow trace, data provenance, recursive data provenance, and data usage. In order to retrieve the process provenance or workflow trace, the entity ID (i.e. the set of Service ID, Workflow ID, Workflow Node ID, Logical Timestamp) of the client invoking the service or workflow, along with the entity ID of the service or workflow being invoked is passed as parameter. In the case of workflow trace, an additional parameter is specified on how deep the workflow trace should recurse in case the workflow contains nested workflows. The methods return the process provenance and workflow trace graphs as XML documents. The immediate and deep data provenance and data usage methods take a data product ID as input parameter. The recursive data provenance also takes the depth to which the ancestral provenance

needs to be recursed into. The immediate data provenance is equivalent to invoking the recursive data provenance with the recursion factor set to 0. These return the data provenance and usage graphs as XML documents. These basic query APIs can be used as building blocks to construct and perform more complex queries as was shown in our entry at the First Provenance Workshop (Moreau, 2007).

In Karma, the provenance graph is constructed natively in the service unlike other provenance collection approaches that require external means for doing so (Groth, 2004). The algorithm for constructing the provenance graphs from a series of provenance activities is similar to an algorithm used to construct a graph given the set of node–edge pairs that comprise the graph. For example, in the case of process provenance, the *entity table* is looked up for the client and the service entity IDs that correspond to the provided Service ID, Workflow ID, Workflow Node ID, and Logical Timestamp for the client and the service. From these two entity IDs, the invocation in the *invocation table* is identified. The invocation ID also leads to the data produced and consumed by that invocation, available through the *data produced* and *consumed tables*. Further details about the data products are retrieved from the *data product table*, while further information about the service is present in the *service table*. These fields are sufficient to populate the process provenance model in the result. In the case of workflow trace, this step is repeated for all service invocations whose Workflow IDs match the workflow. In the case of data provenance or data usage, the procedure is similar, but the starting point is the *data produced* or *consumed tables* instead of the *entity table*. The immutability property of provenance can also be used to cache the results of building the provenance graph and reuse it, provided additional activities have not been appended to the provenance document since its construction.

Provenance Dissemination

The *Karma* provenance service's query interface can be invoked by any web service client to retrieve data and workflow provenance in XML. The *Karma Provenance Browser* is a graphical tool that uses the provenance service API to integrate provenance metadata with additional metadata on the services, workflows, and data products available from external information catalogs such as service registries or metadata catalogs found in the LEAD Project (Simmhan, 2006a). Provenance acts as a glue to relate the services with the data products but does not include complete metadata describing the service or the data product. The browser can retrieve and visualize the provenance graph for, say, a given Workflow ID, and 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 or view the graph from a data or service perspectives.

The XBay workflow monitor GUI (Shirasuna, 2006) 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 activities sent as notifications. The monitor supports BPEL workflows and also acts as a workflow composer interface. So users can compose a workflow visually using XBay, launch the workflow, and later use the same workflow graph to monitor it within XBay at runtime. The workflow monitor subscribes to the notification topic for that workflow and updates the status of the different components in the composed workflow graph as it receives the provenance activity. Since activities are directly delivered as notifications 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 from the provenance service to perform visual replays of the workflow execution in the workflow monitor GUI.

PERFORMANCE EVALUATION

Provenance collection incurs a cost. We quantify the overhead using the example given in Section 2; an example that exemplifies the data driven applications in LEAD. More detailed experiments on the provenance system, both for collecting provenance as well as to query the provenance records, have been performed and are available through a separate publication (Simmhan, 2006c). The workflow of Section 2 performs weather prediction for a 183x163 spatial domain with 53 vertical levels and 9 km grid spacing around the center at 28.5°N and -87.0°E. The experiment is initialized with observational data at 1200 Hours UTC on August 28th 2005 and 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 services in the workflow are invoked sequentially to collect execution times for each application invocation. The service's execution time includes the staging times for input and output files, the wall clock time of the computation, and the provenance overhead. The first four preprocessing and interpolation applications, and the last two post processing and visualization applications are executed on a Linux workstation with dual-Xeon 3 GHz processors, 2GB memory, and a Gbps Ethernet network. The WRF forecasting model and WRF2ARPS postprocessor are executed using 32 and 8 processors, respectively, on an Itanium2 1.3GHz Linux cluster with 6GB memory linked by Myrinet network and connected to a 10 Gigabit external network. The WS-Messenger notification broker runs on a Solaris workstation with dual-Sparc 1.2 GHz processor, 4GB memory, and a 1 Gbps Ethernet network. The *Karma* provenance service runs on a Windows XP workstation with P4 2GHz processor, 1.5GB memory, and 100Mbps Ethernet.

The bar graph in Figure 11 shows the cumulative time taken by the services 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 services 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 activities generated in each script. Since the number of bounding activities for a service invocation is constant, the number of activities generated is closely tied to the number of operation activities, in particular the number of data produced and consumed activities are generated since those are liable to number higher. We have initial results from batching data –produced and –consumed activities in the XML activity representation 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 activities, and the small number of bounding activities from the workflow, and services. These figures are typical of data driven scientific workflows in the meteorology domain, and the provenance overhead is bound to be similarly small.

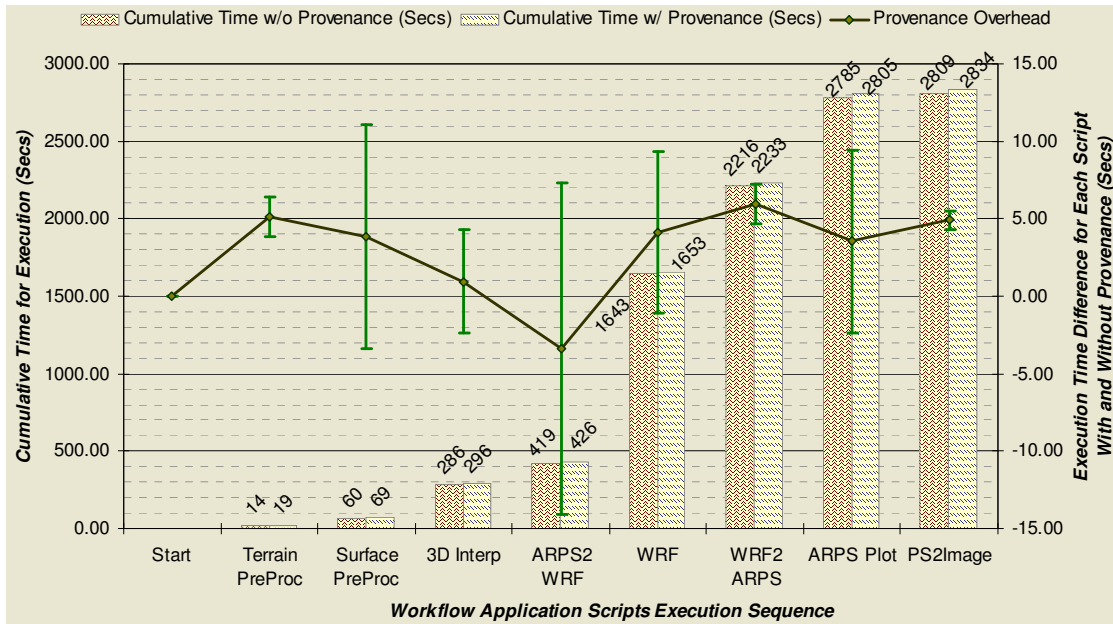


Figure 11. (a) [Bar Graph, Left Y Axis] Cumulative execution time, with and without provenance, for applications in workflow (b) [Line Graph, Right Y Axis] Provenance overhead for individual applications in workflow. $\Sigma(\text{script provenance overhead}) = 26s \cong (\text{total time w/ provenance} - \text{total time w/o provenance}) = 26s$

The line graph on the right Y axis in Figure 11 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.

A separate and more detailed study (Simghan, 2006c) has shown comparable results. A simulated ensemble workflow with 9 services, which included a 4-way parallel service invocation that iterated 10 times, has an overhead of about 70 seconds for collecting the provenance activities. That workflow has 63 service invocations and involved 2400 data products and a real execution of the workflow would have taken several hours to complete. Hence the relative overhead of provenance collection is minor. Those tests also show that the provenance system scaled with the number of parallel workflows, taking linear or sub-linear time as the number of simultaneous workflows increase. The response time for querying provenance was equally low in that study, taking under 1 second for querying individual provenance graphs and showing linear increase in query response time as the query selectivity or the number of parallel clients increase.

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 models and data formats, and (3) minimize performance overheads and modifications required of workflow components. While systems and techniques to collect provenance in scientific workflows exist, they fall short of these needs. Workflow environments such as the Virtual Data Grid (Foster, 2003), myGrid (Zhao, 2004), and GridDB (Liu, 2005)

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. Earth System Sciences Workbench (ESSW) (Bose, 2004) collects lineage as simple parent-child links between services and files, but recording provenance is tightly coupled to its lineage database, and lack of support for logical data products hinders tracking data across the virtual organization. Provenance Aware Service Oriented Architecture (PASOA) (Groth, 2004; Groth, 2005) defines an open protocol for recording provenance through a set of messages exchanged between service invocation actors and a provenance server. While recording the in-wire service requests and responses allows PASOA to track workflow provenance in a non-repudiable manner, users have to extend the provided schema to track data products used in the invocations, and native support to build provenance graphs is absent. Workflow tracing tools such as IBM Data Collector (IBM, 2005) log request-response messages for web service calls that can be correlated and visualized, but this is limited to workflow provenance and not data provenance. Distributed logging tools like Net Logger (Grunter, 2002) that provide generic logging support for distributed applications lack the higher level abstractions necessary for a service oriented architecture. Instrumentation and performance analysis systems like svPablo (de Rose, 1999) and AutoPilot (Ribler, 1998) 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 (Bose, 2005), and present a taxonomy of approaches taken towards building provenance systems (Simmhan, 2005), and exemplify use cases for a provenance system in biology (Miles, 2005). Several workshops on this topic (Bose, 2006; Moreau, 2006) including a recent provenance challenge workshop (Moreau, 2007) have shown an active interest emerging in this field in the e-Science domain.

CONCLUSION AND FUTURE WORK

In this article, 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 and independent for the workflow orchestration model or service invocation style. Indeed, it is not specific to web service based workflows alone and we have examples of it being used by workflows composed of Jython scripts. The activity model is compatible with a publish-subscribe paradigm that provides a 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. In addition, users also have the capability to directly submit provenance activities without requiring to publish them as notifications. 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 alleviate the burden on the service providers. We also have toolkits to automatically create services from an XML description of an application that already come instrumented to collect provenance, completely doing away with user overhead (Kandaswamy, 2006). This toolkit is widely used in the LEAD project. 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. More comprehensive tests support these numbers and provide equally favorable performance figures for querying for provenance (Simmhan, 2006c).

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 metrics and through collaborative feedback on the products and services participating in the data provenance (Simmhan, 2006b). Long running workflows increase the probability of incomplete provenance. Using this information effectively is an area under investigation, as is handling missing activities and undelivered notifications, possibly using WS-Reliable Messaging (Pallickara, 2005). 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. Provenance collected over time serves as a promising resource for mining and detecting patterns that can assist in anything from workflow completion tools to automated resource scheduling.

ACKNOWLEDGMENT

This work is supported in part by NSF cooperative agreement ATM-0331480 and NSF grant EIA-0202048. The authors would like to thank the members of the LEAD project for their active support in this work, in particular, Marcus Christie, Yi Huang, Suresh Marru, Srinath Perera, Satoshi Shirasuna, and Alek Slominski.

REFERENCES

- Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., and Smolinski, B. (1999), Toward a Common Component Architecture for High-Performance Scientific Computing, in *HPDC*.
- BEA Systems, IBM Corporation, Microsoft Corporation, SAP AG, Siebel Systems (2003), Business Process Execution Language for Web Services (BPEL 1.1), in *Technical Report*.
- Bose, R. and J. Frew (2004), Composing Lineage Metadata with XML for Custom Satellite-Derived Data Products, in *SSDBM*.
- Bose, R. and J. Frew (2005), Lineage retrieval for scientific data processing: A survey, *ACM Computing Surveys*, 37, 1-28.
- Bose, R., Foster, I., and Moreau, L. (2006), Report on the international provenance and annotation workshop, *SIGMOD Record*, 35(3), 51-53.
- Box, D., et al. (2004), Web Services Eventing (WS-Eventing), in *Technical Report*.
- Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., and Marshall, M.S. (2001), GraphML Progress Report: Structural Layer Proposal, *LNCS*, 2265, 501-512.
- Catlett, C. (2002), The TeraGrid: A Primer, www.teragrid.org.
- Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., and Wang, I. (2006), Programming Scientific and Distributed Workflow with Triana Services, *Concurrency and Computation: Practice and Experience*, 18(10), 1021-1037.
- Cohen-Boulakia, S., Cohen, S., and Davidson, S. (2007), Addressing the provenance challenge using zoom, *Concurrency and Control: Practice and Experience*.

de Rose, L. A. and Reed, D. A. (1999), SvPablo: A Multi-Language Architecture-Independent Performance Analysis System, in *ICPP*.

Droegemeier, K. K., et al. (2005), Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather, *Computing in Science and Engineering*, 7(6), 12-29.

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.M. (2003), The many faces of publish/subscribe, *ACM Computing Surveys*, 35, 114-131.

Foster, I., Kesselman, C., Nick, J., and Tuecke S. (2002), The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, in *Global Grid Forum*.

Foster, I. T., Vöckler, J., Wilde, M., and Zhao, Y. (2003), The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration, in *CIDR*.

Gannon, D., B. Plale, M. Christie, L. Fang, Yi-Huang, S. Jensen, G. Kandaswamy, S. Marru, S. L. Pallickara, S. Shirasuna, Y. Simmhan, A. Slominski, and Y. Sun (2005), Service Oriented Architectures for Science Gateways on Grid Systems, in *ICSOC*.

Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L., and Oinn, T. (2003), Provenance of e-Science Experiments - experience from Bioinformatics, in *UK OST e-Science 2nd AHM*.

Goble, C. (2002), Position Statement: Musings on Provenance, Workflow and (Semantic Web) Annotations for Bioinformatics, in *Workshop on Data Derivation and Provenance*.

Groth, P., Luck, M., and Moreau, L. (2004), A protocol for recording provenance in service-oriented Grids, in *OPODIS*.

Groth, P., Miles, S., Fang, W., Wong, S. C., Zauner, K.-P., and Moreau, L. (2005), Recording and Using Provenance in a Protein Compressibility Experiment, in *HPDC*.

Gunter, D., Tierney, B., Jackson, K., Lee, J., and Stoufer, M. (2002), Dynamic Monitoring of High-Performance Distributed Applications, in *HPDC*.

Huang, Y., Slominski, A., Herath, C., and Gannon, D. (2006), WS-Messenger: A Web Services based Messaging System for Service-Oriented Grid Computing, in *CCGrid*.

IBM (2005), Web Services Data Collector, www.alphaworks.ibm.com/tech/wsdatacollector.

Kandaswamy, G., and Fang, L., Huang, Y., Shirasuna, S., Marru, S., and Gannon, D. (2006), Building Web Services for Scientific Grid Applications, *IBM Journal of Research and Development*, 50(2/3), 249-260.

Liu, D. T., Franklin, M. J., Garlick, J., and Abdulla, G. M. (2005), Scaling Up Data-Centric Middleware on a Cluster Computer, in *Technical Report, Lawrence Livermore National Laboratory*.

Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., and Zhao, Y. (2006), Scientific Workflow Management and the Kepler System, *Concurrency and Computation: Practice and Experience*, 18(10), 1039-1065.

Miles, S., Groth, P., Branco, M., and Moreau, L. (2005), The requirements of recording and using provenance in e-Science experiments, in *Technical Report, Electronics and Computer Science, University of Southampton*.

Moreau, L. and Foster, I. (2006), Provenance and Annotation of Data – International Provenance and Annotation Workshop, *LNCS 4145*.

Moreau, L. and Ludascher, B. (2007), The First Provenance Challenge, *Concurrency and Control: Practice and Experience*.

OMG (2006), CORBA Component Model v4.0, in *Technical Report, OMG*.

Pallickara, S., Fox, G., Yildiz, B., Pallickara, S. L., Patel, S., and Yemme, D. (2005), On the Costs for Reliable Messaging in Web/Grid Service Environments, in *e-Science*.

Plale, B. (2005), Resource Requirements Study for LEAD Storage Repository, in *Technical Report 001, Linked Environments for Atmospheric Discovery*.

Ribler, R. L., Vetter, J. S., Simitci, H., and Reed, D. A. (1998), Autopilot: Adaptive Control of Distributed Applications, in *HPDC*.

Shirasuna, S. and Gannon, D. (2006), XBay: A Graphical Workflow Composer for the Web Services Architecture, *Technical Report 004, Linked Environments for Atmospheric Discovery*.

Slominski, A. (2006), Adapting BPEL to Scientific Workflows, *Workflows for e-Science*.

Simmhan, Y., B. Plale, and D. Gannon (2005), A survey of data provenance in e-science, *SIGMOD Record*, 34(3), 31-36.

Simmhan, Y. L., Pallickara, S. L., Vijayakumar, N. N. and Plale, B. (2006a), Data Management in Dynamic Environment-driven Computational Science, in *IFIP Working Conference on Grid-Based Problem Solving Environments (WoCo9)*.

Simmhan, Y. L., B. Plale, and D. Gannon (2006b), Towards a Quality Model for Effective Data Selection in Collaboratories, in *Workflow and Data Flow for Scientific Applications (SciFlow)*.

Simmhan, Y. L., B. Plale, and D. Gannon (2006c), Performance Evaluation of the Karma Provenance Framework for Scientific Workflows, *LNCS 4145*.

Simmhan, Y. L., B. Plale, and D. Gannon (2007), Query capabilities of the Karma provenance framework, *Concurrency and Control: Practice and Experience*.

Szomszor, M. and Moreau, L. (2003), Recording and Reasoning over Data Provenance in Web and Grid Services, in *ODBASE*.

West, K. (2005), Scoping Out the Planet, *Scientific American*.

Yu, J. and Buyya, R. (2005), A taxonomy of scientific workflow systems for grid computing, *SIGMOD Record*, 34 (3), 44-49.

Zhao, J., Wroe, C., Goble, C. A., Stevens, R., Quan, D., and Greenwood, R. M. (2004), Using Semantic Web Technologies for Representing E-science Provenance, in *ISWC*.

ABOUT THE AUTHOR

Yogesh L. Simmhan is a doctoral candidate at Indiana University, working on data management and information services for distributed applications found in e-science. His research interest lies in investigating various aspects of provenance in the context of scientific workflows and applying provenance to novel uses such as data quality metrics, workflow enhancement tools, and long term preservation of data. IEEE member.

Beth Plale is an Associate Professor of Computer Science and Director of the Data Search Institute at Indiana University. Her research interest is in the broad area of large-scale data management, specifically stream mining and event processing, distributed metadata and integration, provenance, grid and service-oriented architectures, and petascale databases. IEEE member.

Dennis Gannon is a Professor of Computer Science and Science Director of Pervasive Technology Labs at Indiana University. His research interests include programming systems and tools, distributed computing, computer networks, parallel programming, computational science, problem solving environments and performance analysis of Grid and MPP systems. IEEE member.