Provenance Collection Support in the Kepler Scientific Workflow System

Ilkay Altintas[1], Oscar Barney[2], and Efrat Jaeger-Frank[1]

[1] San Diego Supercomputer Center, University of California, San Diego,
9500 Gilman Drive, San Diego, CA 92092-0505
{altintas, efrat}@sdsc.edu
[2] Scientific Computing and Imaging Institute, University of Utah
50 S. Central Campus Drive, Salt Lake City, UT 84112
oscar@sci.utah.edu

Abstract. In many data-driven applications, analysis needs to be performed on scientific information obtained from several sources and generated by computations on distributed resources. Systematic analysis of this scientific information unleashes a growing need for automated data-driven applications that also can keep track of the provenance of the data and processes with little user interaction and overhead. Such data analysis can be facilitated by the recent advancements in scientific workflow systems. A major profit when using scientific workflow systems is the ability to make provenance collection a part of the workflow. Specifically, provenance should include not only the standard data lineage information but also information about the context in which the workflow was used, execution that processed the data, and the evolution of the workflow design. In this paper we describe a complete framework for data and process provenance in the Kepler Scientific Workflow System. We outline the requirements and issues related to data and workflow provenance in a multidisciplinary workflow system and introduce how generic provenance capture can be facilitated in Kepler's actor-oriented workflow environment. We also describe the usage of the stored provenance information for efficient rerun of scientific workflows.

1 Introduction

Current technology significantly accelerates the scientific problem solving process by allowing scientists to access data remotely, distribute job execution across remote parallel resources, and efficiently manage data. Although an increasing amount of middleware to accomplish these tasks has emerged in the last couple of years, using different middleware technologies and orchestrating them with minimal overhead still remains difficult for scientists. Scientific workflow systems [1,2,3], aim to improve this situation by creating interfaces to a variety of technologies and providing tools with domain-independent customizable graphical user interfaces that combine different Cyberinfrastructure [4] technologies along with efficient methods for using them. Workflows enormously improve data analysis, especially when data is obtained from multiple sources and generated by computations on distributed resources and/or various analysis tools. These advances in systematic analysis of scientific information

L. Moreau and I. Foster (Eds.): IPAW 2006, LNCS 4145, pp. 118 – 132, 2006.

made possible by workflows have unleashed a growing need for automated data-driven applications that also collect and manage the provenance of the data and processes with little user interaction and overhead.

Requirements for Efficient Provenance Collection during Scientific Workflow Design and Execution. Scientific problem solving is an evolving process. Scientists start with a set of questions then observe phenomenon, gather data, develop hypotheses, perform tests, negate or modify hypotheses, reiterate the process with various data, and finally come up with a new set of questions, theories, or laws. Most of the time before this process can end in results, scientists will fine-tune the experiments, going through many iterations with different parameters [5]. This repeating process can reveal a lot about the nature of a specific scientific problem and, thus, provides information on the steps leading to the solution and end results. Making this information available requires efficient usage and collection of data and process provenance information. Another very important requirement for any scientific process is the ability to reproduce results and to validate the process that was followed to generate these results. Provenance tracking provides this functionality and also helps the user and publication reviewer/reader understand how the run happened and what parameters and inputs were associated with the workflow run.

A provenance collection system must have the ability to create and maintain associations between workflow inputs, workflow outputs, workflow definitions, and intermediate data products. Collecting intermediate data products in addition to other provenance data serves multiple purposes as the results of some processes in the workflow can vary from run to run and some workflow tests require manually stepping through some of the steps to verify and debug results. The intermediate results along with the other provenance data can be also used to perform "smart" reruns, which will be described in section 5.

These requirements illustrate the strong need to retain origin and derivation information for data and processes in a workflow, to associate results with customized and executable workflow version, and to track workflow evolution as described in [16]. This paper discusses an implementation that aims to keep track of all these aspects of provenance in scientific workflows.

In the rest of this paper, we review the related work and how our contribution differs from the rest of the previous work in this area, give a brief overview of the Kepler scientific workflow system, and introduce a generic provenance collection framework in Kepler. Finally, we explain the "smart" rerun feature that exhibits a usage of extracted provenance information for performing efficient re-runs for a slightly modified workflow in Kepler.

2 Related Work

The need for data provenance has been widely acknowledged and is evident in numerous applications and systems. Here, we give an overview of several research efforts in the field, some of which were also listed in a recent survey of data provenance techniques by Simmhan et al. [6]. We plan to extend the data, execution and workflow provenance capabilities of our provenance framework based on the past work explained below.

The Chimera [7] Virtual Data System uses a process model for tracking provenance in the form of data derivations. Using a Virtual Data Catalog (VDC), defined by a query-able Virtual Data Schema, Chimera represents executables as transformations, a particular execution as a derivation and inputs/outputs as data objects. Workflows in Chimera are defined as derivation graphs through a Virtual Data Language (VDL). The VDL is also used for querying the VDC independent of the catalog schema. Provenance in Chimera is used for tracking the derivation path that led to a data product, reproducibility of a derived product, and validation/verification of an experiment.

In the MyGrid project, provenance data is recorded for workflows in XScufl language that are executed using the Taverna workflow engine [8]. A provenance log is automatically recorded during the workflow execution in a framework differentiating between four provenance levels: The *process level* gathers information about the invoked processes, their inputs/outputs and processing times. The *data level*, inferred from the process level describes intermediate and final products derivation paths. The *organization level* stores the metadata for the experiment, and the *knowledge level* links the experiment's scientific findings/"knowledge" with the other provenance levels as supporting evidence. The stored information is used to infer the provenance of intermediate and final results and for quality verification of the data in terms of tracing the processing steps.

The above "provenance recording systems" are tightly coupled with their workflow execution environment. The Provenance Aware Service Oriented Architecture (PASOA) project aims to provide interoperable means for recording and using provenance data using an open provenance protocol [9]. In [18] PASOA identifies several requirements for a generic, application independent, provenance architecture for e-Science experiments. Among those requirements are recording of *interaction provenance*, *actor provenance* and *input provenance*, where interaction provenance is recording interactions between components and the data passed between them, actor provenance is recording processes information and the time of the execution, and input provenance is tracking the set of input data used to infer a data product. Other requirements include *reproducibility* of an experiment, *preservation* and *accountability* of provenance over time and *customizability* to support and integrate with diverse architectures.

Other applications of data provenance are evident in database and geographic information systems. Data lineage in database systems where data provenance refers to '*a description of the origins of a piece of data and the process by which it arrived in a database*' has been addressed by Bunman *et al.* [10, 19]. Bunman *et al.* define the data lineage problem as "why" and "where" provenance. *Why-provenance* refers to the set of tuples that contributed to a data item, where as *where-provenance* defines how a data item is being identified in the source data. In Geographic Information Systems (GIS) data lineage is used for dataset validation. Metadata is recorded for tracking the transformations applied to derive a data item [11].

The VisTrails system [15] was developed to facilitate interactive multiple-view visualizations by providing a general infrastructure, which can be used in conjunction with any existing visualization system, like Kitware's Visualization Toolkit [17], to create and maintain visualization workflows as well as to optimize their execution. Often, progress is made by comparing visualizations that are created by the same

basic workflow, but with slightly different parameters or components. Thus, the Vistrails system collects and maintains a detailed provenance record for each instance of a workflow as well as across different versions of a workflow thus tracking the evolution of the workflow. The Vistrails system is the first system to track a workflow's evolution [16], something that can be useful for anyone who wants to execute a workflow multiple times, store the results, and then compare multiple versions of the workflow in an organized fashion in order to find just the right set of components and parameters.

Provenance tracking is important for scientific computing. This paper discusses an implementation for the Kepler scientific workflow system, which aims to keep track of all aspects of provenance in scientific workflows: in workflow evolution, data and process provenance, and efficient management and usage of collected data. While there are similarities between aspects of the above-mentioned previous work and our own, to the best of our knowledge, this approach to designing a provenance collection framework that is highly configurable, comprehensive, model of computation independent, and includes facility for smart reruns is a unique contribution to provenance research in the scientific workflow community.

3 Kepler Scientific Workflow System

A scientific workflow is the automated process that combines data and processes in a structured set of steps to implement computational solutions to a scientific problem. Kepler [1] is a cross-project collaboration to develop a scientific workflow system for multiple disciplines that provides a workflow environment in which scientists can design and execute workflows.

Kepler builds on top of the mature Ptolemy II software [12], which is a Java-based system and a set of APIs for heterogeneous hierarchical modeling. The focus of Ptolemy II is to build models based on the composition of existing components, which are called 'Actors', and observe the behavior of these simulation models when executed using different computational semantics, which are implemented as components called 'Directors.'

Actors are the encapsulations of parameterized actions performed on input data to produce output data. Actors communicate between themselves by sending Tokens, which encapsulate data or messages, to other actors through ports. An actor can have multiple ports and can only send Tokens to an actor that it is connected to one of its output ports. The director specifies the model of computation under which the workflow will run. For example, in a workflow with a Process Network (PN) director, actors can be thought of as separate threads that asynchronously consume inputs and produce outputs. Under the Synchronous Dataflow (SDF) director, actors share a common thread, and the order of execution is statically determined because the number of tokens each actor will produce and consume is known ahead of time. Also, different domains control how the ports relay Tokens. For example, in PN each port behaves like a FIFO queue of unlimited size where as a port controlled by the SDF director acts like a FIFO queue with a size limited to the number of tokens an actor can produce or consume.

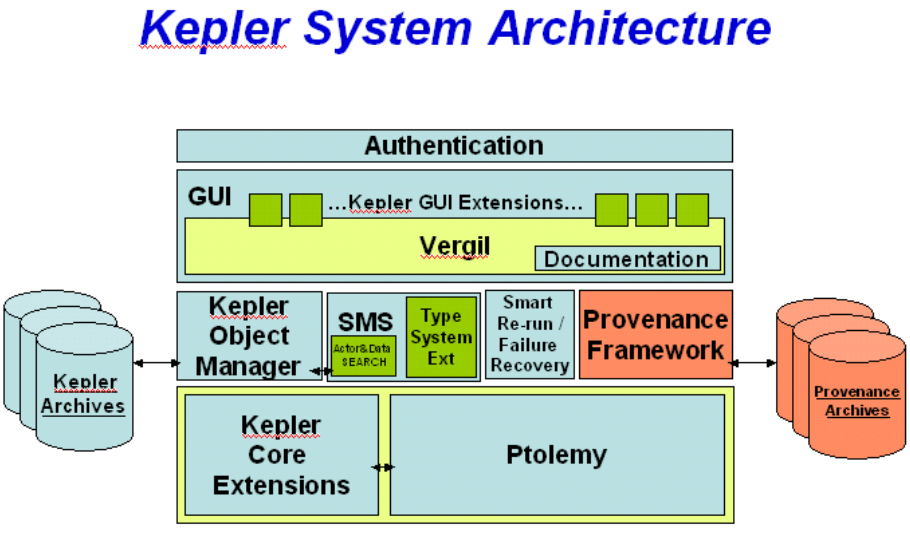

Fig. 1. The Kepler System Architecture

Kepler actors perform operations including data access, process execution, visualization, and domain specific functions. Kepler uses Ptolemy II's hierarchical actor oriented modeling paradigm to create workflows, where each step is performing some action on a piece of data. Workflows can be organized visually into sub-workflows. Each sub-workflow encapsulates a set of executable steps that conceptually represents a separate unit of work. The Kepler system can support different types of workflows ranging from local analytical pipelines to distributed, high–performance and high-throughput applications, which can be data- and compute-intensive. [13] Along with the workflow design and execution features, Kepler has ongoing research on a number of built-in system functionalities, as illustrated in Figure 1, including support for single sign-in GSI-based authentication and authorization; semantic annotation of actors, types, and workflows; creating, publishing, and loading plug-ins as archives using the Vergil user interface; and documenting entities of different granularities on-the-fly.

Ptolemy II separates the description of important aspects of a design such as behavior and architecture, or computation and communication. [14] Kepler inherits this concept of *separation of concerns* in design from the Ptolemy II. This provides significant advantages such as lower design time and better re-usability of the design because system designers can build a new component for the system and plug them in for testing without changing any of the underlying architecture. Also, workflow designers do not have to use ad hoc techniques to implement the workflow's design and execution of the workflow graph. The Ptolemy II system provides a general strategy for separating the workflow composition from the overall orchestration of the model by introducing the separate concerns for actors, their composition, and the implementation of computational domains that run the workflows. These 'separate concerns' are combined visually into a model on the screen, which provides an easy way for

system users to see what the exact behavior of the workflow will be without clicking on menu to find out things like what the model of computation will be, for example.

4 Generic Provenance Framework in Kepler

Given the collaborative and domain-independent nature of the Kepler project, our provenance framework needed to include plug-in interfaces for new data models, metadata formats and cache destinations. To accomplish this goal we have created a highly configurable provenance component that can be easily used with different models of computation using the separation of concerns design principle. Just like a director, provenance collection is modeled as a separate concern that is bound visually to the associated workflow. This way a user can easily see if provenance is being collected for a certain run of the workflow. Another advantage to this design is its compliance with Kepler's visual actor-oriented programming paradigm, and that it is consistent with the behavior of the Kepler user interface.

4.1 Design Objectives

A major objective when designing the provenance recording functionality was ease of use. We did not want the user to have to go through a complex configuration process or the actor designers to have to implement a complex API. Since Kepler is built on top of the Ptolemy II framework, we had to consider designs that would seamlessly integrate with existing code and work with any director.

When designing the provenance collection system, another major consideration was supporting the multi-disciplinary and multi-project nature of the Kepler project. To be more flexible we made our collection facility parametric and customizable. For example, a user may want to limit the granularity of the collected data, publish it in a specific data source, or only save certain results to verify the behavior of a specific workflow component during testing. To facilitate this, we allow the user to choose between various levels of detail, and even save all of the provenance data needed to recreate a workflow result when the workflow is used as a part of the scientific discovery.

A workflow run consists of several pieces of information that need to be recorded including the *context*, *the input data and its associated metadata*, the *workflow outputs*, *intermediate data products*, the *workflow definition,* and information about the *workflow evolution*. Context is the *who*, *what*, *where*, *when*, and *why* that is associated with the run. Workflow definition is a specification of what exists in the workflow and can have a context of its own. It includes information about the workflow's entities, their parameters and the connections between the actors. Workflow evolution, also known as a workflow trail [16], is a description of how the workflow definition has changed over time. This is an application of the ideas in [16]. By tracking the evolution of a workflow design, its runs, and its parameters over time, the scientist can efficiently manage the search of a parameter space and easily jump back to a previous version of the workflow that produced interesting results.

One of our side goals when designing the provenance recorder was its ability to help us debug a workflow during the implementation phase of workflow development. By mining and analyzing 'process provenance,' data related to the execution of the workflow, and intermediate data products that were processed at the time of an error, we may be able to figure out exactly what was happening at the time of an error in our prototype workflow.

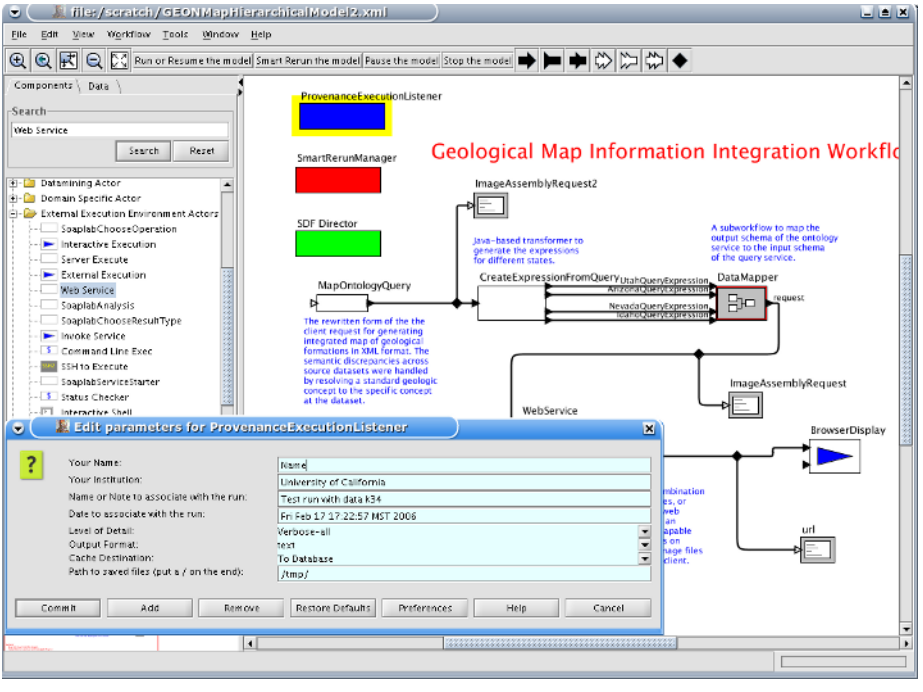

Fig. 2. A screenshot of Vergil that shows the different concerns for model of computation (green), provenance collection (blue), smart rerun (red) and actors

4.2 Implementation

To address ease of use, we designed the *Provenance Recorder (PR)* to be configured and represented in the same way as a Director in Kepler. To enable provenance collection in a workflow instance the user drags the PR from the toolbox and places it on the workspace with the Director and other workflow entities. Unlike using Directors, using the PR with a given workflow is optional, depending on the user's requirements for tracking provenance. Similar to the Director, it is configured with a standard configuration menu and becomes part of the workflow definition. (See Fig. 2.)

We had to pursue the following steps to provide automatic collection of provenance information in Kepler. We converted Kepler's internal XML workflow representation, MOML, into our internal format for provenance data. This format leaves out some of the unnecessary information MOML includes (i.e. actor coordinates and the custom actor icons in the user interface) and includes extra information (i.e. Token

production rate of the actors, etc.) critical to complete data provenance collection. Using the existing MOML generation capabilities in Kepler helped us to efficiently collect the provenance associated with the workflow definition. Also, by keeping track of all information associated with the workflow definition we are able to track workflow evolution and jump back to interesting workflows saved in our provenance store. We leave out the details of our internal format for provenance collection and caching here as it is out of the scope of this paper.

In order to collect information that is generated during a workflow run, the PR implements several event listener interfaces. Different 'concerns' in Ptolemy II and Kepler, such as the ports through which actors communicate, maintain a list of event listener objects that are registered with them. When something interesting happens, the event listeners registered with the specific 'concern' in question are notified, and take the appropriate action. For example, when the PR is notified that a data product is created, it can then associate the appropriate data lineage information with this data product and put it in the provenance store. Event listeners are also allowed to register and un-register with individual concerns so that we can easily control the amount of provenance data that is collected during any one run. This can be very important because some workflows create a massive number of intermediate data products, which are not always necessary to recreate the results of a certain workflow.

When the workflow is loaded, the PR will register with the appropriate 'concerns' in the workflow. When the workflow is executed, PR will process information received as events, and save it in provenance store. As we have mentioned before, the provenance recorder can save information that is useful for debugging the workflow. To accomplish this, we have the PR register with the appropriate concerns that send out notification of events related to the execution of the workflow and any errors that occur. In this way we can find out exactly what actor was executing, with what inputs when a certain error occurred.

Although we were able to automate much of the provenance collection, we had to extend the design to handle several other cases. For example, if an actor creates an external data product, it must register this product with the PR as well as reporting its internal actions. We developed a simple API allowing actors to notify the PR in these situations. Once the actors are extended using this API, the PR can collect and save these data products and actions in addition to any local data products that were automatically collected using the event listener interfaces.

5 Efficient Workflow Rerun Enabled by Provenance Data

In Kepler, we have added functionality to enable efficient reruns of a workflow by mining stored provenance data. The idea behind a "smart" rerun [1] is as follows. When a user changes a parameter of an actor then runs the workflow again, re-executing all the preceding, unchanged steps (actors) in the workflow may be redundant and time consuming. A "smart" rerun of the workflow will take data dependencies into account and only execute those parts of the workflow affected by the parameter change. The ability to store and mine provenance data is required to enable "smart" reruns since the intermediate data products generated in previous runs are used as the inputs to the actors that are about to be rerun.

We have created the Smart Rerun Manager (SRM) to handle all the tasks associated with the efficient rerun of a workflow. This includes data dependency analysis and provenance data management. The algorithm used by the SRM is derived from the Vistrail execution and cache management algorithm used in the Vistrails system [15]. The VisTrails cache management algorithm was developed to allow users of a visualization system to efficiently explore a parameter space. The premise is that we can extract intermediate results of the dataflow from a cache instead of recreating them in order to save time when rerunning the dataflow.

5.1 Implementation and Algorithm Description

The SRM is an event-based entity in the workflow, which is a 'separate concern' in the Kepler system just like the PR. They are both used and configured in a similar way. Once a SRM is placed on the workspace by dragging it from the toolbox, all provenance data needed to perform a smart rerun of the workflow will be collected. To allow users to choose whether or not they want to perform a smart rerun of the workflow, the SRM is activated by pressing a special "Smart Rerun" button in the user interface next to the standard "Run Workflow" button. In this section, we describe how the SRM system uses provenance data for optimized rerun and our changes to the underlying Vistrails algorithm, which the SRM builds upon.

The basic idea behind the Vistrails algorithm is to search a graph representation of the dataflow for sub-graphs that can be eliminated. The precondition for elimination of these sub-graphs is that the actors they contain have already been run with the current parameters and input data. The next step is to retrieve the intermediate data products produced by this eliminated sub-graph from the provenance store for use as input to the actors that need to be rerun. This part of the provenance store we will call the provenance cache. Each sub-graph is identified with a unique ID, which is an important concept that we borrow from the Vistrails system. A unique ID, which is used as a key when searching the provenance cache for information related to a particular sub-graph, represents a unique state of a component (a.k.a. actor), its parameters, and all the actor and parameters that come before it in the workflow. Each unique ID is associated with a specific actor and encapsulates the provenance information needed to uniquely identify and retrieve the intermediate data products produced by that actor.

When activated by pressing the special "Smart Rerun" button in Kepler's toolbar, the SRM builds a directed graph representing the data dependencies in the workflow. Each node in the graph represents an actor in the workflow and the edges represent the flow of data between actors. The SRM then analyzes this graph to detect sub-graphs that have been successfully computed before and the sub-graphs that must be rerun.

The analysis begins at the graph's sinks and recursively traverses all the input edges of each node in the direction opposite to the flow of data. At each node, a unique ID is generated and used as a key in the cache lookup. If this unique ID is associated with some data in the provenance cache, this means that the workflow as it exists from this node backward in the dependency graph has been executed successfully before. In this case the actors represented by the nodes sub-graph corresponding to the unique ID can be eliminated from the list of actors to be rerun. Conceptually, the intermediate data product retrieved from the provenance store using the unique ID replaces this sub-graph.

How this is done in practice will be clarified soon. If this unique ID is not associated with any data products in the provenance store, we keep traversing our graph until we do find a unique ID with associated data in the provenance cache or we reach the source nodes in the graph and they need to be rerun as well.

The last step of the "smart" rerun process is to replace the eliminated sub-graphs with components that can stream the data from the provenance cache. In Kepler-specific terms, we need to place the intermediate data products retrieved from the provenance cache on the appropriate actors input ports while the workflow is running. These intermediate data products are the tokens that flowed across this input edge from the eliminated actors in the previous runs of the workflow. We developed a special actor to replace the eliminated actors and replay the tokens that they would have produced. This special actor is called the *Stream Actor* because it streams data from the provenance store into the running workflow. This piece of the system is called the *vtkDataObjectPipe* in Vistrails. Figure 3 visually illustrates a simple example where the SRM retrieves the intermediate results of the previous execution and replaces the previously executed parts of the workflow with a streaming actor.

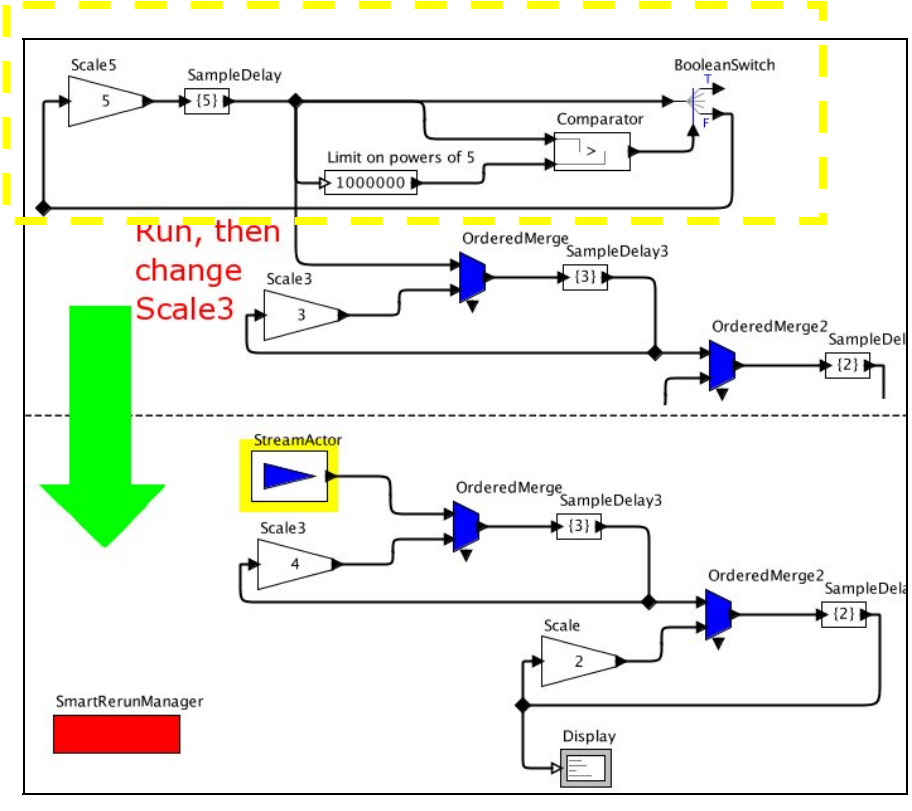

Fig. 3. Smart Rerun Manager retrieving the intermediate results of the previous execution and replacing the pre-executed parts of the workflow with a streaming actor. Bottom workflow shows *StreamActor* replacing actors whose computations would be redundant in the rerun.

It is important to note that the data associated with a successful lookup in the provenance cache can be associated with the preceding run of the workflow or any previous runs with the same components, connections, and parameters thus utilizing all past provenance information. Also, two sub-graphs of the same workflow could potentially be replaced by data from two distinct runs of the workflow. This is a major strength of the Vistrails algorithm, which ensures that no work is repeated.

For the most part, the SRM is able to use the Vistrails cache management algorithm, however we have made some noteworthy changes and additions. The Vistrails system was designed with a single model of computation in mind and actually executes portions of the workflow as a part of its graph analysis stage. Since the SRM must distinguish between different models of computation (Directors) in Kepler that expect different behaviors during the workflow run, we had to do the graph analysis step before handing off the execution of the workflow to the Director. This is an efficient design in Kepler because each Director has a distinct behavior and trying to encode these behaviors in the SRM would result in redundant code.

Also, our Stream Actor must take these separate models of computation into account. For example, when a workflow is executed using a Director, such as the Process Networks Director that requires each actor consume a stream of inputs and produce a stream of outputs, the SRM actor makes sure that the stream of tokens is replayed in the same order that it was collected when inserting data from the provenance store into the workflow. In a domain, such as Synchronous Dataflow (SDF), where each actor consumes a certain number of inputs and produces a certain number of outputs, the Stream Actor gives the tokens to input ports at the rate they are expected. What we mean by rate is clarified by the following example. In a model controlled by the SDF Director assume that actor A declares that it will consume x tokens each time it is activated. The SDF Director schedules the actors so that they will not run until they have the proper number of inputs. If the actor B creates x/2 inputs each time it executes and is connected to actor A, the SDF Director will schedule B to execute twice so that the actor A will have enough inputs when it executes. As it is illustrated in this example, the SRM must guarantee the production rate of the StreamActor to ensure that the rerun is performed correctly.

Another difference between the Vistrails algorithm and the algorithm that the SRM is using, is the way in which the SRM handles 'non-cacheable' actors in the workflow. Non-cacheable actors are the actors whose output depends on when the run occurs as well as what the inputs and parameters are. For example, an actor that queries a remote database is non-cacheable if the database modifiable because it may receive different results depending when the query is executed. In contrast, the Vistrails system views every component as a function, for a specified input you can predict the output. A non-cacheable actor in the Vistrails system does not have its outputs saved, and thus its unique ID will never be found in the provenance cache. Non-cacheable components in the context of Visualization workflows are those whose outputs cannot be saved or whose outputs are too large to be saved. Actors that depend on the non-cacheable actor are not rerun unless there is a new input or parameter change upstream. If it is not specified otherwise, the SRM will rerun all actors that depend on a non-cacheable actor since their results depend on non-deterministic nature of the non-cacheable actor.

The SRM's user interface allows the user to specify if an actor is cacheable or if it must be rerun every time. In some cases you may want to save the state of an actor that behaves in a non-deterministic way. This enables the user choose between doing a "smart" rerun of the workflow with saved provenance data to exactly recreate a past run or to rerun the workflow to get the most up-to-date results but still avoid redundant steps.

The SRM is an example to valuable usage of data from our provenance store. It has the potential to save scientists hours while they explore the parameter space of their workflows and is an important feature of the Kepler system.

6 Results and Conclusions

This paper discusses our generic provenance framework for use with scientific workflows. The framework is designed to support a wide range of workflow types and is extensible because of the modularity of its design and the flexibility of the event listener interfaces that it implements. Most of the discussed functionality has been implemented with the exception of a final data model design. This paper does not focus on the internal structure of the collected information to support provenance and caching, but mentions these to explain the PR and SRM. We plan to continue working on the data models and make it available in the near future. We have already had interest in the PR from a wide variety of users, some of which have used our initial version and given positive feedback.

Performance Evaluation. The PR has been designed to be as generic as possible and has met most of the design goals that we set out to achieve. The event based nature of the design has allowed us to collect the variety of information needed in order for the system to be useful in a wide range of application areas while at the same time having a minimal performance impact on the system. Specific performance measurements for workflows using the PR vary greatly depending on the amount of provenance data being saved and the ratio of data produced to time spent computing. For example, a computationally intensive workflow may produce the same amount of provenance data as a workflow that runs in a matter of seconds, but has less overhead as the PR takes much smaller percentage of total run time. We can safely say that we have accomplished our design goal of efficiency because in the majority of our test cases the increase in run time attributed to the PR is minimal and usually only a couple of seconds. Also, in some cases where a specific actor generates excessive amounts of data, our design allows us to specify that we are not interested in this actor's information by un-registering with its list of event listeners.

7 Future Work

We have developed several prototype relational and XML data models and plan to implement them in the Kepler Provenance Framework once we have design a suitable data model. The data model for storing provenance information in Kepler should accommodate the needs of different scientific domains as well as allow for efficient storage and retrieval of data. Another area of we are interested in researching is in

defining the policies for managing provenance data for different projects. This is an important problem that utilizes other functionality and system components in Kepler including the authentication and authorization framework, the data access API, and semantic annotations. Another planned usage of actor annotations is to annotate the actors so that the PR could automatically figure information related to what files they create during the run, what algorithm and data structures they use, etc. We also plan to implement querying and viewing system for collected provenance information.

There are multiple provenance activities within the Kepler collaboration including provenance tracking in collection-oriented workflows and integrating with RDF based provenance stores. In this paper, we have only mentioned the existing implementation, the algorithms it utilizes, and its functionality. We plan to bring the ongoing research by other Kepler developers and researchers together under the Kepler provenance framework once these research ideas are implemented and become available for public use. In particular we would like to experiment with the provenance model developed by Bowers et al. [20] to support a wide range of scientific use cases in phylogeny and the data model for the XMLSchema-based arbitrary textual and binary data format articulation capability by Talbott et al. [21].

This paper already described the usage of the Vistrails algorithm for smart re-runs of the same workflow. We plan to further incorporate the Vistrails system capabilities into our provenance framework for systematically capturing detailed provenance and workflow evolution information. [22] This will require customizing or extending the existing Vistrails action-based model by information on the Kepler workflow modeling language (MoML) and updating the core Kepler modeling components to record this information during the modeling and experimentation phase for a scientific workflow.

Acknowledgements

The authors would like to thank the rest of the Kepler team for their excellent collaboration, especially to Timothy McPhillips for the discussion on requirements for a provenance framework and future steps, Claudio Silva and Juliana Freire for their help on VisTrails and insight on workflow provenance, and Steven Parker for his support and guidance. This work was supported by DOE Sci-DAC DE-FC02-01ER25486 (SDM) and NSF/ITR 0225673 (GEON).

References

1. Ludäscher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger-Frank, E., Jones, M., Lee, E., Tao, J., Zhao, Y.: Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, to appear, 2005. http://kepler-project.org/
2. Oinn, T., Greenwood, M., Addis, M., Alpdemir, M.N., Ferris, J., Glover, K., Goble, C., Goderis, A., Hull, D., Marvin, D., Li, P., Lord, P., Pocock, M.R., Senger, M., Stevens, R., Wipat, A. Wroe, C.: Taverna: Lessons in creating a workflow environment for the life sciences". Accepted for publication in Concurrency and Computation: Practice and Experience Grid Workflow Special Issue

3. Churches, D., Gombas, G., Harrison, A., Maassen, J., Robinson, C., Shields, M., Taylor, I., Wang, I.: Programming Scientific and Distributed Workflow with Triana Services". In Grid Workflow 2004 Special Issue of Concurrency and Computation: Practice and Experience, to be published, 2005
4. Revolutionizing Science and Engineering Through Cyberinfrastructure: Report of the National Science Foundation Blue Ribbon Advisory Panel on Cyberinfrastructure
5. Lipps, J. H.: The Decline of Reason?. http://www.ucmp.berkeley.edu/fosrec/Lipps.html,
6. Simmhan, Y. L., Plale, B., Gannon, D., A survey of data provenance in e-science. In *SIGMOD Rec.* 34(3): 31-36, 2005
7. Foster, I., Voeckler, J., Wilde, M., Zhao, Y.: Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation. In Proceedings of the 14th Conference on Scientific and Statistical Database Management, 2002
8. Greenwood, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L., Oinn, T.: Provenance of e-Science Experiments - experience from Bioinformatics. In Proceedings of The UK OST e-Science second All Hands Meeting 2003 (AHM'03)
9. Groth, P., Luck, M., Moreau, L.: A protocol for recording provenance in service-oriented grids. In Proceedings of the 8th International Conference on Principles of Distributed Systems (OPODIS'04), 2004
10. Buneman, P., Khanna, S., Tan, W.C.: Why and Where: a characterization of data provenance. In Proc. ICDT 2001
11. Lanter, D.P., Design of a lineage-based meta-data base for GIS, In Cartography and Geographic Information Systems, 18(4):255-261, 1991
12. Ptolemy Project, See Website: http://ptolemy.eecs.berkeley.edu/ptolemyII/
13. Altintas, I., Birnbaum, A., Baldridge, K.K., Sudholt, W., Miller, M., Amoreira, C., Potier, Y., Ludaescher, B.: A Framework for the Design and Reuse of Grid Workflows. Lecture Notes in Computer Science, Scientific Applications of Grid Computing: First International Workshop, SAG 2004, Beijing, China, September 20-24, 2004, Volume 3458 (3), pp 119-132, ISBN3-540-25810-8.
14. Yang, G., Watanabe, Y., Balarin, F., Sangiovanni-Vincentelli, A.: Separation of Concerns: Overhead in Modeling and Efficient Simulation Techniques. Fourth ACM International Conference on Embedded Software (EMSOFT'04), September, 2004
15. Bavoil, L., Callahan, S., Crossno, P., Freire, J., Scheidegger, C., Silva, C., and Vo, H.: Vistrails: Enabling interactive multipleview visualizations. In IEEE Visualization 2005, pages 135–142, 2005
16. Callahan, S., Freire, J., Santos, E., Scheidegger, C., Silva, C., and Vo, H.: Managing the Evolution of Dataflows with VisTrails. In Proceedings of the IEEE Workshop on Workflow and Data Flow for Scientific Applications (SciFlow 2006)
17. The Visualization Toolkit (VTK), See Website: http://public.kitware.com/VTK/
18. Miles, S., Groth, P., Branco, M., Moreau, L.: The requirements of recording and using provenance in e-Science experiments. Technical Report, Electronics and Computer Science, University of Southampton, 2005
19. Buneman, P., Khanna, S., Tan, W. C.: Data Provenance: Some Basic Issues. In Proceedings of the 20th Conference on Foundations of Software Technology and theoretical Computer Science, 2000
20. Bowers, S., McPhillips, T., Ludaescher, B., Cohen, S., Davidson, S.B.:A Model for User-Oriented Data Provenance in Pipelined Scientific Workflows . In Proceedings of the IPAW'06 International Provenance and Annotation Workshop, Chicago, Illinois, USA May 3-5, 2006

21. Freire, J, Silva, C.T., Callahan, S.P., Santos, E., Scheidegger, C.E, Vo, H.T.: Managing Rapidly-Evolving Scientific Workflows. In Proceedings of the IPAW'06 International Provenance and Annotation Workshop, Chicago, Illinois, USA May 3-5, 2006
22. Talbott, T.D., Schuchardt, K.L., Stephan, E.G., Myers, J.D.: Mapping Physical Formats to Logical Models to Extract Data and Metadata: The Defuddle Parsing Engine. In Proceedings of the IPAW'06 International Provenance and Annotation Workshop, Chicago, Illinois, USA May 3-5, 2006