Performance Evaluation of the Karma Provenance Framework for Scientific Workflows

Yogesh L. Simmhan, Beth Plale, Dennis Gannon, and Suresh Marru

Indiana University, Bloomington IN 47405, USA
{ysimmhan, plale, gannon, smarru}@cs.indiana.edu

Abstract. Provenance about workflow executions and data derivations in scientific applications help estimate data quality, track resources, and validate *in silico* experiments. The *Karma provenance framework* provides a means to collect workflow, process, and data provenance from data-driven scientific workflows and is used in the Linked Environments for Atmospheric Discovery (LEAD) project. This article presents a performance analysis of the *Karma* service as compared against the contemporary *PReServ* provenance service. Our study finds that Karma scales exceedingly well for collecting and querying provenance records, showing linear or sub-linear scaling with increasing number of provenance records and clients when tested against workloads in the order of 10,000 application-service invocations and over 36 concurrent clients.

1 Introduction

Data-driven scientific investigations often follow a dataflow pattern where data progresses through a number of processes as they are transformed, fused, and used in complex models. Services provide an abstraction to access these processes through well-defined interfaces and allow applications to be modeled as workflows that capture the invocation logic. *Process provenance*, collected about the workflow, describes the service invocations during a workflow's execution and enables tracking of workflows and services in collaboratory environments [4,15]. In data-driven applications, provenance about the data involved in the workflow is critical to understanding its results. *Data provenance*, the derivation history of derived data, includes the service and its parameters that contributed to the data creation, and is valuable to determine the origin and quality of a particular derived data, and for its discovery and reuse in other workflows [15].

The *Karma provenance framework* [16] records uniform and usable provenance metadata for scientific workflows that meets the domain needs of the Linked Environments for Atmospheric Discovery (LEAD) project [14] while minimizing the performance overhead on the workflow engine and the services. It collects two forms of provenance: *process provenance*, also known as process-oriented provenance or workflow trace, describes the workflow's execution and associated service invocations, and is used to monitor the workflow progress and mine it for results validation; and *data provenance*, which provides complementary metadata about the derivation history of data products in the workflow, including

L. Moreau and I. Foster (Eds.): IPAW 2006, LNCS 4145, pp. 222–236, 2006.

the services that create and use it, and the input data transformed to generate it, and forms the basis for quality-oriented data product discovery [17].

Karma is used to collect provenance from meteorology workflows in LEAD, where hundreds of simultaneous users are expected to run workflows and query for provenance at any given time. This article describes our empirical evaluation of the Karma provenance framework in meeting the needs of the LEAD project to record and retrieve provenance documents for different workflow loads and with concurrent clients. We present our experimental results alongside the performance results for *PReServ* [8,9], a comparable service for recording provenance assertions, for equivalent workloads.

Several provenance frameworks have emerged in the past several years [19,18,12,9,7,3,6] that have defined the provenance needs for e-Science and have applied the systems to different scientific domains. Surveys on provenance have compared these systems based on a meta-model for provenance [4] and through the use of a taxonomy [15]. While some of the provenance systems have presented a preliminary evaluation of the performance of their systems [8], none have undertaken a detailed comparative performance study of their provenance framework as we have in this article. Such a study, based on multiple workflow and query loads, is necessary to determine the overhead of collecting provenance, and the costs for querying and using provenance metadata under different application scenarios. A comparative evaluation also provides additional context to interpret the results. PReServ was selected as the provenance system compared against due to several reasons. PReServ is similar to Karma in that it is a standalone provenance service independent of the workflow or service environment, and it is motivated by the provenance requirements for scientific experiments. It also provides the ability to store metadata annotations as part of provenance, that allows us to record provenance akin to that captured by Karma. Lastly, it is a contemporary system that is being actively developed and used.

The rest of the article is organized as follows: in Section 2, we briefly describe the architecture of Karma; in Section 3, we discuss the hardware and software deployment used in the experiments; Section 4 describes the experiments for collecting provenance and their results, and Section 5 does likewise for querying provenance; Section 6 discusses the results of the experiments, and, finally, Section 7 presents our conclusion and future work.

2 Architecture

Karma collects provenance for data-centric workflows in a service oriented architecture. Such workflows are composed of services connected as directed graphs, with each service passing one or more data product as input to the service it is connected to, thus forming a dataflow. The workflow is orchestrated by an execution engine, and execution takes place at three levels. At the *workflow level*, the workflow engine invokes services with appropriate parameters, in the order prescribed in the workflow graph. At the *service level*, the service that receives the invocation from the workflow engine initiates action by executing a method

or launching an application. At the *application level*, the actual task corresponding to the service invocation is performed. Each workflow, service, application, and data product used in the workflow is identified using a globally unique ID.

Karma uses the notion of activities [16] that take place at different levels of a workflow's execution, in space and in time, to collect provenance (Fig. 1). The key activities are Workflow-Started and -Finished, generated by the workflow identified by its *Workflow ID*; Application-Started and -Finished, generated by the invoked service (or application) identified by its *Service ID*; and Data-Produced and -Consumed, submitted by the end-application to list the *Data Product ID* of the data it transforms. Based on these activities, Karma builds three forms of provenance documents that can be retrieved: *workflow trace* describes all activities for a workflow's execution, *process provenance* captures activities for a single invocation of a service or application within the workflow, including its input and output data, and *data provenance* returns the application that created the data product and those that use it, all potentially from different workflows.

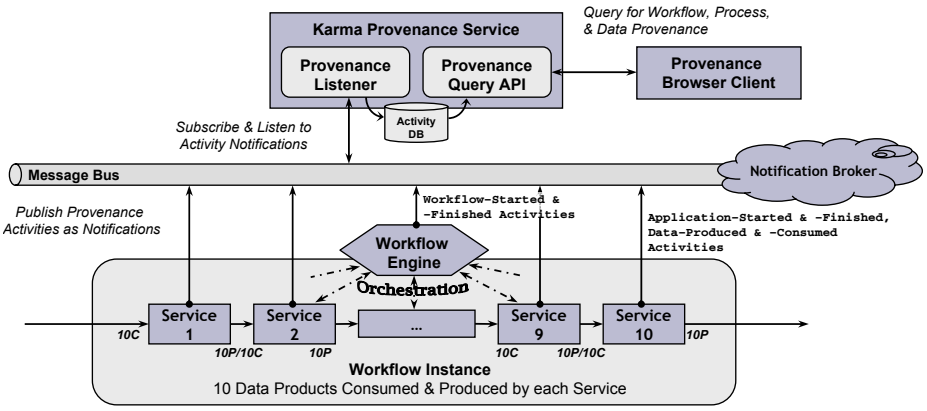

Fig. 1. Architecture of Karma provenance framework. At the bottom, a linear workflow is orchestrated by a workflow engine and publishes provenance activities on the workflow, services, and data products as notifications. A listener in the Karma service subscribes to and receives these notification from the notification broker, and persists them in a database. Upon query by a provenance client, the Karma service constructs and returns the workflow, process, or data provenance graph from the activities.

Provenance is submitted by the workflow components, namely the workflow engine, services, and applications, either synchronously through a web-service call to the Karma service or asynchronously. In the latter case, provenance activities are modeled as XML notifications that are published to a notification broker that the Karma service subscribes to, making it easier for messaging-aware applications to submit provenance. For our experiments, we use the asynchronous approach, with notifications published using the *WS-Eventing* standard [5] through the *WS-Messenger* [10] notification broker implementation. The Karma service persists the activities it receives to a relational database and provides

a web-service API to query for the workflow trace, process provenance, and data provenance. The provenance graphs are constructed on-the-fly from the activities recorded in the database and represented as XML documents with a defined schema. A graphical interface is available to visualize and navigate the provenance graphs.

3 Experimental Setup

The components of both Karma and PReServ services run on identical nodes in a Linux cluster, each node consisting of dual 2.0 GHz 64 bit Opteron processors with 16 GB RAM. Karma service v0.3, WS-Messenger notification broker, and a mySQL database used by Karma run on separate nodes. The PReServ service v0.2.3 is deployed on a single node within a Tomcat 5.0 web-server container and uses the default embedded Java database. Clients that generate and retrieve provenance run on a separate 128-node Linux compute-cluster formed of dual 2.0 GHz Opteron processors with 4 GB RAM. Parallel executions of client programs is managed by a SLURM job manager [2]. All nodes in both clusters are connected by Gigabit Ethernet and use local IDE disks for persistent storage. The services and test applications are written in Java and Jython, and use Sun Java 1.5 JVM as the Java runtime.

4 Collecting Provenance

Workflows, services, and applications in LEAD are written in Java or as Jython scripts which invoke FORTRAN binaries. They are instrumented using the *Notifier* Java library to publish Karma provenance activities as asynchronous notifications. The services in the workflow are usually auto-generated web-services wrappers for applications, to enable their use in a service-oriented architecture [11]. The provenance instrumentation in the case of workflows and services is automated. Applications written by the service providers are manually instrumented using the Notifier library to generate the provenance activities.

In the experiments for collecting provenance, the standard Notifier library is used by all workflow components to generate the Karma activities asynchronously. For evaluating PReServ, the default Notifier implementation is replaced by a thin client that synchronously records provenance with the PReServ web-service using its client library. The following experiments measure the provenance generation overhead for different workflow loads shown in Fig. 2.

In both Karma and PReServ, a single invocation of a service in a workflow generates a set of provenance records. For Karma, the provenance recorded is as follows: one each of Application-Started and -Finished activities marking the start and end of the invocation, and identifying the workflow, service, and application scope in which the invocation took place; and one Data-Consumed or -Produced activity for each data product used or created during the invocation, containing the *Data Product ID*, its location, usage or creation time, and its size [16]. In the

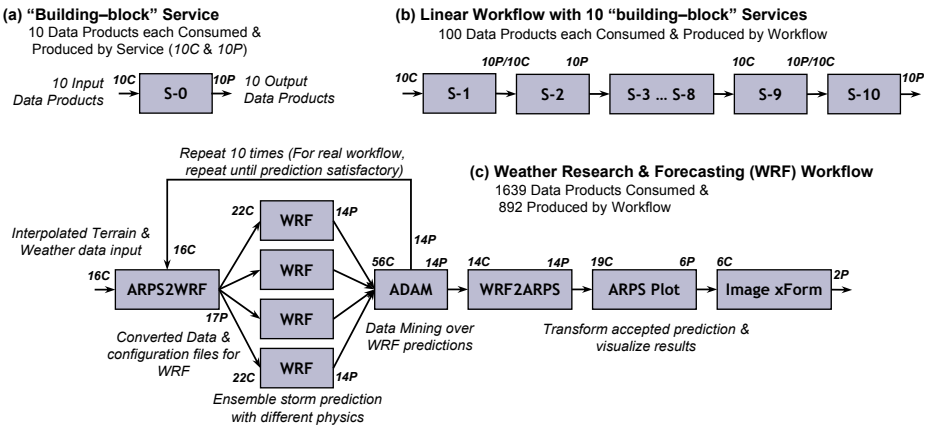

Fig. 2. Types of workflows and services used to generate provenance for experiments. The rectangles denote the service in the workflow and the arrows are the invocation order or data flow. The arrows are labeled with the number of data products that flow between services.

case of PReServ, a service invocation generates three types of provenance assertions: one Interaction assertion that establishes the occurrence of the invocation by providing the source and sink of the service invocation along with a unique ID for that interaction; one Actor State assertion with the list of data products produced and consumed during that interaction, identified by their *Data Product ID*s; and two Relationship assertions that respectively associate the Interaction assertion with the produced and consumed data present in the Actor State assertion [8]. PReServ optionally allows for recording the SOAP message for the service invocation in the assertions but this is not used in any of the experiments.

4.1 Single Service

The first experiment evaluates the performance of recording provenance for a single service as the number of service invocations and the number of provenance records already present in the provenance service increase. The simple "building-block" service shown in Fig. 2(a) acts as a client that generates provenance about its invocation. It is modeled as a Java application that takes 10 data products as input and generates as many as output, doing no computation or I/O operations other than record provenance. This building-block service is repeatedly invoked from a test harness running in the same JVM, and, for each invocation, the service records a set of provenance records describing its invocation.

As noted earlier, for this service invocation, Karma generates two activities marking the start and end of the application and 20 activities on the data products used and created. These map to four notifications that are published. While the Application-Started and -Finished activities are published as two individual notifications, the set of 10 Data-Consumed and 10 Data-Produced activities are batched as two notifications. PReServ generates one Interaction assertion, one

Actor State assertion, and two Relationship assertions, for a total of 4 provenance assertions. Karma and PReServ represent their provenance activities and assertions as XML documents, and for this single service invocation, the total size of all XML provenance documents recorded in both cases is about 10 KB.

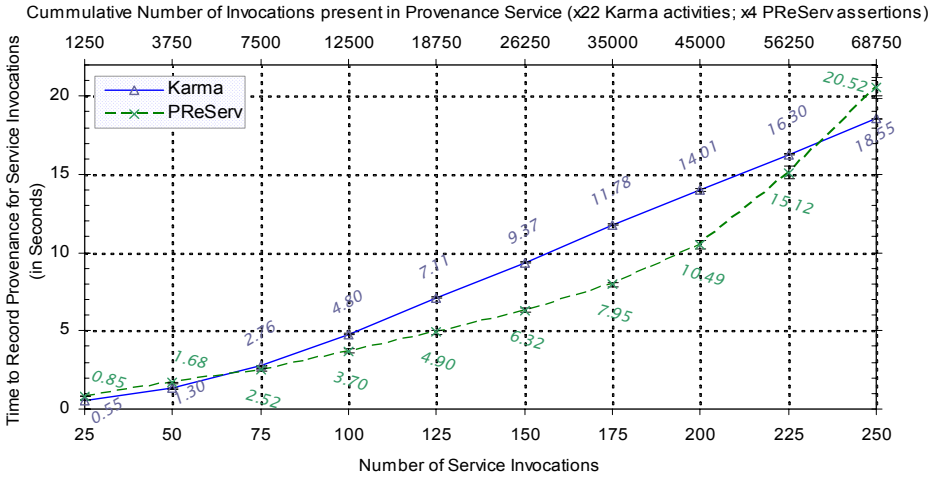

Fig. 3. Total time to record provenance (Y axis) for "building block" service as the number of service invocations per trial (bottom X axis) and the number of provenance records present with the provenance service (top X axis) increase. Each trial is averaged over 50 iterations. Barring two points, the standard error for all averages are under 5%; it is under 10% for all points.

Figure 3 shows the total time taken to record the provenance for all service invocations in a trial as the number of invocations in a trial increase from 25 to 250 along the bottom X axis. As the trials progress, the records accumulating with the provenance services are shown in the top X axis. As the number of invocations rise, Karma shows a linear behavior in provenance recording time, averaging 74 ms per invocation for 250 invocations when there are over 1.5 million provenance activities (68,750 invocations × 22 activities per invocation) in the Karma service. This also exhibits a sub-linear recording time against the number of records present in the service. Publishing provenance activities asynchronously as notifications insulates the clients from potential fluctuations in the backend store, although for the workloads that are used, synchronous recording of activities shows similar results too.

PReServ shows super-linear trend as the number of invocations increases from 25 to 175. While taking lesser time at 7.95 seconds for 175 invocations compared to Karma's 11.78 seconds, beyond that the recording time rises quadratically, with 250 invocations taking 20.52 seconds. However, the time increases almost linearly against the number of records present in the service. This indicates that the performance limitation of PReServ is imposed by its backend store used

to persist provenance records. PReServ shreds and stores the XML provenance records into an embedded Java database that allows for easy portability and deployment, but this experiment shows the limitations of such an approach in scaling beyond 140,000 records (after 175 invocations).

4.2 Simultaneous Linear Workflows

This experiment evaluates the scalability of collecting provenance as the number of concurrent clients generating provenance grows. A simple linear workflow composed of 10 building-block services connected linearly is used as shown in Fig. 2(b). This workflow provides a uniform load under which the provenance systems can be compared, with each workflow run generating 220 Karma activities (40 notifications) and 40 PReServ assertions from the 10 services. Each workflow is started simultaneously on multiple hosts as parallel jobs and a workflow controller program invokes the 10 services in sequence within the same JVM and iterates over 50 trials.

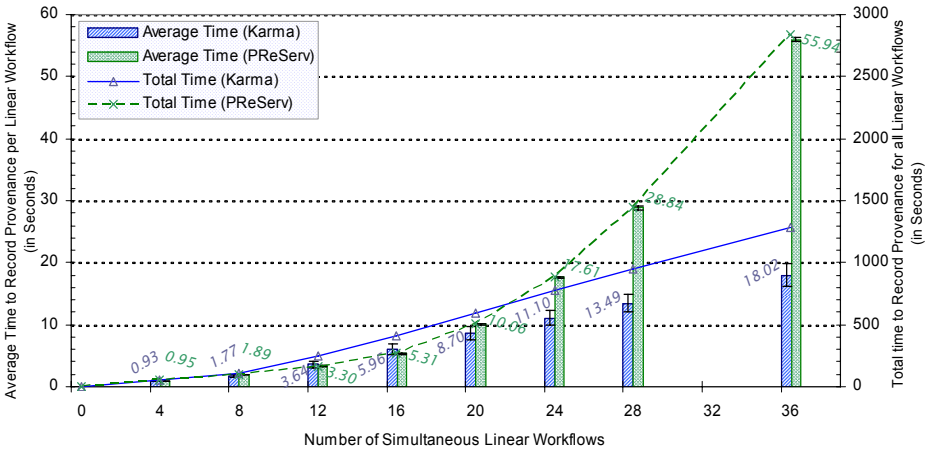

Fig. 4. Times to record provenance for linear workflows with the number of workflows running simultaneously. The left Y axis bar graph shows the average time (over 50 iterations) taken by each workflow to complete as the number of parallel workflows increase along X axis. The data points are labeled with the average time. The right Y axis line graph has the total time for all 50 iterations of each trial to complete and is scaled by 50 times the left Y axis. The standard error for the average workflow time is between 5–15% for Karma and under 3% for PReServ.

Figure 4 shows the time for each workflow run (bar graph on left Y axis) averaged over 50 iterations and the total time for the 50 iterations (line graph on right Y axis) as the number of concurrent workflows increases from 4 to 36 along the X axis. Karma and PReServ show similar performance until 8 concurrent workflow clients, taking an average of less than two seconds per workflow with 8 workflows. As the parallelism increases from 12 to 20, PReServ outperforms

Karma but begins to display super-linear behavior. Karma shows good scalability by maintaining a linear trend throughout, averaging 18 seconds per workflow with 36 parallel clients, compared to 56 seconds for PReServ, which is at best a quadratic trend. This may partially be attributed to the increase in the number of records stored in PReServ, as observed earlier. The standard error for the average workflow time for Karma is between 5–15% while PReServ has more uniform provenance recording time with standard error under 3%.

4.3 Simultaneous Complex Workflows

In order to measure provenance collection performance under realistic workloads, a mesoscale storm prediction workflow from the LEAD project is used in this experiment. Figure 2(c) shows a synthetic workflow involving four parallel Weather Research & Forecasting (WRF) applications that repeat 10 times. The services perform no computation or I/O but generate the same provenance as a real WRF workflow would – 2657 Karma activities (252 notifications) and 252 PReServ assertions from the nine services in the workflow. Such a simulated workflow accelerates gathering of performance results; a real WRF workflow for this experiment configuration requires the use of 320 compute nodes for 1 week. Also, a sample run of the WRF workflow showed high error margins due to I/O contention by the applications, precluding accurate determination of the provenance overhead. In an ideal situation, the computation and I/O time for the real applications should remain constant across concurrent runs, and the synthetic workflow that is used in its stead duplicates such a scenario.

Figure 5 shows the time taken to run the WRF workflow as the number of concurrent workflows increase from 1 to 20. Such a workload is typical in the LEAD system where numerous users simultaneously run such complex workflows [13]. The average time each workflow takes over 25 iterations is shown in the bar graph on the left Y axis and the total time for all workflows to complete 25 iterations is the line graph on the right Y axis while the X axis shows increasing parallelism. As in the previous experiment, Karma achieves linearity while PReServ shows super-linear behavior as concurrency increases. The unsynchronized forks and joins of the four ensemble WRF applications from different workflow instances causes the graph to be relatively flat up to 8 concurrent workflows. In this experiment and the previous, there is a marked increase in the average workflow run time for both Karma and PReServ beyond about 8 parallel workflows. This is likely due to the provenance services reaching a hardware or OS imposed limit, such as network socket availability or bandwidth. The trial for 20 concurrent workflows could not be completed for PReServ since the local disk used by it ran out of disk space. PReServ takes 8 GB to store the provenance records for 16 concurrent workflows while Karma's mySQL database uses 700 MB – an order of magnitude difference in the storage overhead for the two systems.

4.4 Number of Data Products

Karma stores fine-grained information about the data provenance that requires an activity for each data product consumed or produced by a service. Each data

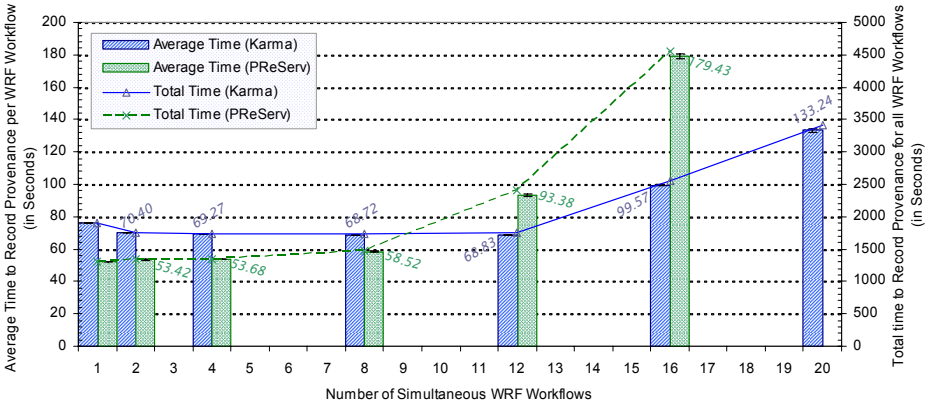

Fig. 5. Average and total time to record provenance for synthetic WRF workflows running in parallel. Left Y axis bar graph represents the average time for each WRF run to complete when averaged over 25 iterations as the number of parallel WRF workflows increases along X axis. The data points are marked with this average time. The right Y axis line graph shows the total time for all 25 iterations to complete, and hence is scaled by 25 times the left Y axis. The trial for 20 concurrent workflows using PReServ could not be completed. The standard error for all averages is below 1%.

product activity contains information such as the *Data Product ID*, creation time, and size, in addition to information about the workflow component that generated it. These allow Karma to natively build the data provenance for any data product involved in various workflows. PReServ does not prescribe any particular metadata to be provided for data provenance and allows any metadata to be submitted as part of its Actor State assertion. For the experiments, the Actor State assertion for a service carries the essential information about all data products involved in its invocation, namely their *Data Product ID*s.

This experiment estimates the provenance collection overhead as the number of data products involved in a service invocation increases. The single linear workflow shown in Fig. 2(b) is used to generate provenance, but suitably modified so that the constituent services consume and produce progressively increasing number of data products – from 25 to 250 each, for a total of 250 to 2500 data products per workflow.

In Fig. 6, the Y axis shows the average time to record provenance for each workflow as the total number of data products involved in a workflow increase along the X axis. There is negligible difference between the performance of Karma and PReServ for 250 data products used in the workflow. Beyond this, the average time to record provenance using Karma increases linearly with the number of data products, taking 10.77 seconds for the workflow involving 2000 data products. This rise correlates with the increase in the size and complexity of the data product batch XML notifications that contain the Data-Produced and -Consumed activities for each service invocation. PReServ shows a near-constant time for recording provenance as the number of data products increase and this can be attributed to the fact that the minimum information it stores about the

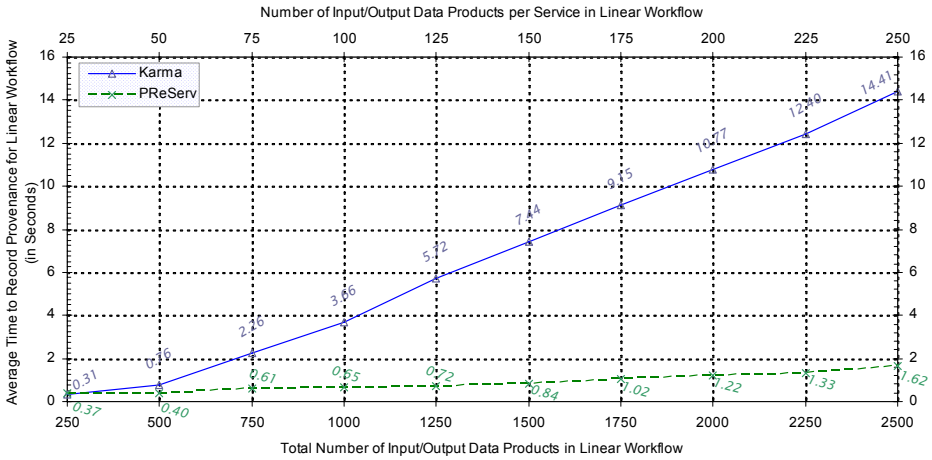

Fig. 6. Average time to record provenance from linear workflow with increasing data products involved in each workflow. The average time over 100 iterations is along the Y axis and labeled at the data points. The lower X axis has the total number of data products involved in the workflow. The upper X axis has the number of data products produced and consumed by each service in the workflow, and is scaled to 10 times the lower X axis since there are 10 services in a workflow.

data products in the Actor State assertion is represented using a simpler XML document. In the LEAD project, 95% of workflows involve less than 250 data products [13] and the performance of Karma for those workflows is comparable to PReServ's. The advantage of being able to record and query data provenance natively offsets the increase in overhead for the other 5% of workflows.

5 Querying for Provenance

Karma builds three types of provenance documents from the activities, namely workflow trace, process provenance, and data provenance. It provides a web-service interface to retrieve these based on the *Workflow ID*, *Service ID*, and *Data Product ID* respectively. The PReServ web-service allows provenance retrieval using *XQueries*, and two simple *XPath* queries that return all Interaction assertions for a given *Workflow ID* and for a given *Service ID* form the equivalent of workflow trace and process provenance queries in Karma. Building data provenance using PReServ requires a more complex query over Interaction, Actor State, and Relationship assertions and is hence omitted for these experiments.

For the following tests, the provenance services are loaded with provenance records for workflow runs and queried for through their web-service APIs from Java clients using the respective client libraries provided by the systems. In the case of Karma, provenance for 1000 linear workflows like those in Fig. 2(b) are loaded, with each service consuming and producing 10 data products. This translates to 10,000 service invocations or 220,000 provenance activities present

in Karma. PReServ is loaded with only 100 linear workflows like those in Fig. 2(b) and corresponds to provenance for 1000 service invocations or 4000 provenance assertions. The factor of 10 difference in the number of workflows recorded with Karma (1000 workflows) and PReServ (100 workflows) is because PReServ is unable to complete queries over 1000 workflows, requiring memory in excess of the 4 GB assigned to it, thus imposing a bound on its query scalability.

5.1 Query Result Size

This experiment measures the query response time as the number of provenance records retrieved increases. For the each of the three provenance types, a single client queries by *Workflow ID*, by *Service ID*, and by *Data Product ID* respectively, making one call to the provenance service per provenance document. For Karma, the queries fetch between 0.01% and 10% of provenance documents, distributed uniformly to prevent any locality advantage. This translates to retrieving between 1–100 workflow traces out of 1000 available, 10–1000 process provenance out of 10,000 available, and 100–10,000 data provenance documents out of 100,000 available. For PReServ, the queries return between 1–10 workflow traces of 100 available, and 1–100 process provenance documents out of 1000 service invocations available. Each query is averaged over 50 trials and the response time plotted as a log–log graph shown in Fig. 7.

The X axis of the plot shows the resultset size from each type of query and the Y axis shows the average response time of the query, both these axes being in the logarithmic scale. All three query types for Karma and both types for PReServ are parallel to the central diagonal on the log–log plot, implying that they all have linear characteristics. But the slopes for the linear equations are markedly different. Karma conservatively takes a factor of 50 lesser time than PReServ for workflow trace queries (e.g. 0.49 seconds vs. 26.9 seconds for retrieving 10 workflow trace documents) and a factor of 200 lesser time for process provenance queries (e.g. 0.97 seconds vs. 209.5 seconds for querying 100 process provenance records). The query response time for data provenance is also low for Karma at 55.86 seconds for 10,000 data provenance records. The scalability of Karma in responding to queries can be attributed to mapping the provenance activities from an XML schema to a relational schema for storage, and the provenance queries translate to SQL queries that leverage indices present on key fields. PReServ provides an XQuery interface for querying over provenance records and its database does not utilize any indices [1]. This causes all provenance records to be accessed for resolving a query, which also leads to it running out of memory when 1000s of records are queried over.

5.2 Simultaneous Query Clients

This experiment evaluates the scalability of the provenance service as the number of parallel clients querying for provenance records increases from 1 to 48. This is performed only for Karma since PReServ is less optimized for querying and

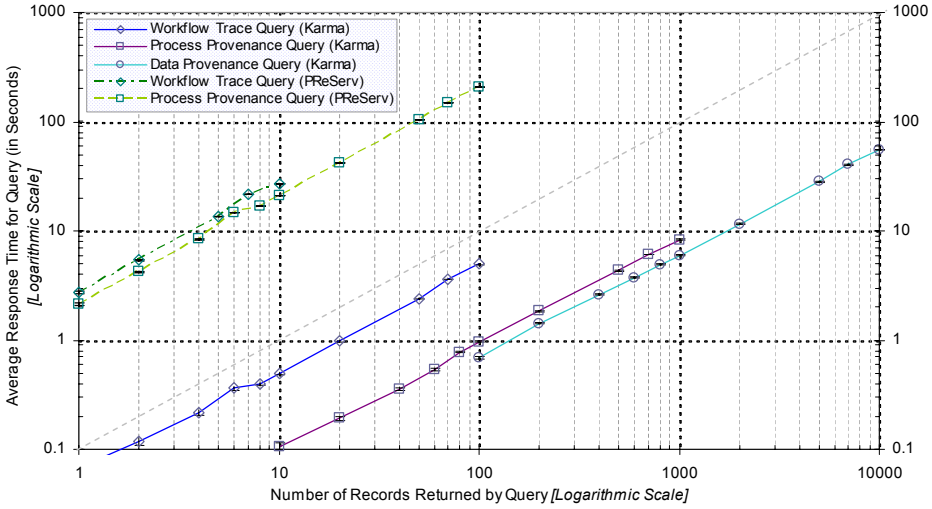

Fig. 7. Query response time to retrieve provenance with increasing number of records returned. The average time to respond to each type of query is along the Y axis and the number of records that the query fetches increases along the X axis. Both axes are in logarithmic scale. The averages times are over 50 iterations; the standard errors for the response times for Karma is under 6% for all but 3 data points, and for PReServ is under 3% for all data points.

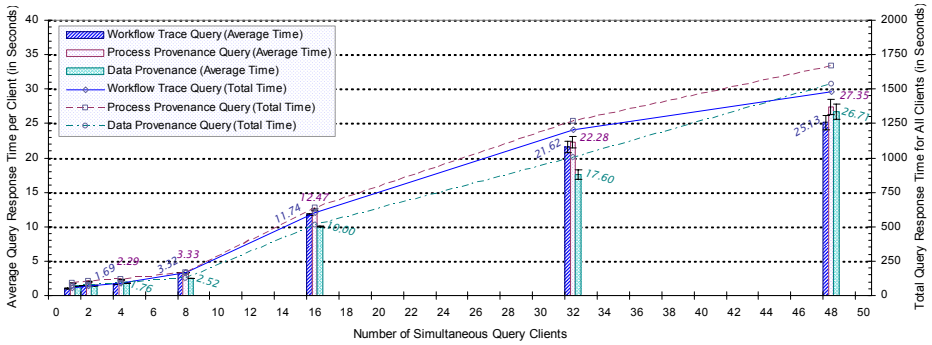

Fig. 8. Query response time to retrieve 20 workflow trace, 200 process provenance, and 200 data provenance from increasing number of concurrent query clients. The average query time over 50 iterations is shown on the left Y axis bar graph, the data points being labeled with this value. The total time to complete the 50 iterations is the line graph on the right Y axis, and is accordingly scaled by 50 times compared to the left Y axis. The X axis shows the increasing number of parallel query clients. The standard errors for all averages are under 4%.

hence results collected for it are less relevant. For the three types of provenance queries, each Karma client retrieves 20 workflow traces, 200 process provenance, and 200 data provenance documents respectively.

Figure 8 shows the average query response time for each query type along the left Y axis as the number of concurrent clients increases along the X axis. The times are averaged over 50 iterations with the total time for all clients to complete the 50 iterations shown on the right Y axis. Karma shows a sub-linear trend as the number of clients increases beyond 8, taking an average of 3.32 seconds and 25.13 seconds respectively to retrieve the workflow trace from 8 and 48 clients. The results for process and data provenance queries similarly exhibit good scalability. As seen when submitting provenance from concurrent clients, the slope increases when the number of clients goes beyond 8 and may be attributed to an OS or machine threshold being reached.

6 Discussion

The above experiments establish the scalability of Karma for collecting and querying provenance over hundreds of thousands of services and from numerous clients. The workloads used in the experiments are typical in large collaboratory scientific projects like LEAD [13]. Provenance collection for Karma shows a linear trend with a low slope for both increasing number of service invocations and for increasing levels of concurrent clients, and performs better than PReServ in these experiments. PReServ shows better characteristics with increasing number of data products, taking constant time unlike Karma which takes linearly time. However, it remains comparably low for Karma in 95% of the use cases for LEAD, that involve less than 250 data products per workflow. Querying Karma for workflow trace, process provenance, and data provenance increases in at most linear time as the number of results retrieved and the number of clients increase, and it uniformly performs better than PReServ.

The difference in performance of Karma and PReServ is due both to design choices and their implementations. Karma and PReServ share several features. They are both stand-alone provenance frameworks and define different types of messages to record provenance, synchronously or asynchronously, from a workflow's execution – Karma using activities and PReServ using assertions. However, PReServ takes an open ended approach to defining provenance assertions, requiring just a few fields to determine the workflow provenance and allowing additional user-defined annotations to be submitted as part of the assertions. While this flexibility may be required for certain applications, it tends to overlap with the functionality provided by existing information services like metadata catalogs and registries for data products and services. Such flexibility may also impose limitations on its implementation, contributing to reduced performance and scalability. Also lacking is inherent support for tracking data provenance, being left to the user to define it as annotations. Karma's activities are less expansive but contain sufficient information to recreate provenance about a workflow run, a service invocation, and the derivation and usage history of a data product. It provides a light-weight and scalable implementation to meet the core needs of recording and querying for these provenance graphs over hundreds of thousands of service invocations and data products. In its current implementation, PRe-

Serv is better suited to record provenance for a smaller number of workflows but with rich XML annotation capabilities, while Karma is effective in meeting more direct and scalable provenance needs in large collaboratories.

7 Conclusion and Future Work

This article evaluates the performance of the Karma provenance framework in collecting and querying for provenance from workflow executions, and finds it to scale well with the size of the workflows and the number of concurrent clients. The workloads used for the experiments are motivated by the requirements of the LEAD meteorology project [13] and is relevant to similar scientific projects. The workloads in themselves form a benchmark to compare and evaluate other provenance systems, and such a comparison is done with the PReServ service.

Karma is currently deployed and being used in the LEAD testbed. Our future work includes evaluating the performance of Karma for real workflow runs and getting usable results for them by suppressing the I/O variations we encountered in the data intensive applications – possibly by the use of local storage instead of network file systems. In addition to visually browsing provenance graphs, we are investigating further ways to apply provenance. Notable among these is on using data provenance as a factor in predicting the quality of data products to assist in data selection and ranking in collaboratory environments [17]. Provenance helps identify applications that produce good or poor quality data as a function of their inputs. In the quality model we propose, this provenance function is one of several metrics used to estimate a quality score for derived data products.

Acknowledgments. This work is supported in part by NSF cooperative agreement ATM-0331480 and NSF grant EIA-0202048. The authors would like to thank Paul Groth from the University of Southampton for helping us deploy the PReServ server, the members of the LEAD team for their support and feedback on our work, and Abhijit Mahabal and Ramyaa Ramyaa from Indiana University for their help in analyzing the empirical data.

References

1. Personal communication with Paul Groth, University of Southampton, 2006.
2. Simple Linux Utility for Resource Management (SLURM) Reference Manual. Technical Report UCRL-WEB-201386, Lawrence Livermore National Laboratory, 2006.
3. Ilkay Altintas, Oscar Barney, and Efrat Jaeger-Frank. Provenance Collection Support in the Kepler Scientific Workflow System. In *IPAW*, 2006.
4. Rajendra Bose and James Frew. Lineage Retrieval for Scientific Data Processing: A Survey. *ACM Computing Surveys*, 37(1):1–28, 2005.
5. Don Box, Luis Felipe Cabrera, Craig Critchley, Francisco Curbera, Donald Ferguson, Alan Geller, Steve Graham, David Hull, Gopal Kakivaya, Amelia Lewis, Brad Lovering, Matt Mihic, Peter Niblett, David Orchard, Junaid Saiyed, Shivajee Samdarshi, Jeffrey Schlimmer, Igor Sedukhin, John Shewchuk, Bill Smith, Sanjiva Weerawarana, and David Wortendyke. Web Services Eventing (WS-Eventing), August 2004.

6. Uri Braun, Simson Garfinkel, David A. Holland, Kiran-Kumar Muniswamy-Reddy, and Margo I. Seltzer. Issues in Automatic Provenance Collection. In *IPAW*, 2006.
7. Juliana Freire, Claudio T. Silva, Steven P. Callahan, Emanuele Santos, Carlos E. Scheidegger, and Huy T. Vo. Managing Rapidly-Evolving Scientific Workflows. In *IPAW*, 2006.
8. Paul Groth, Michael Luck, and Luc Moreau. A Protocol for Recording Provenance in Service-oriented Grids. In *OPODIS*, 2004.
9. Paul Groth, Simon Miles, Weijian Fang, Sylvia C. Wong, Klaus-Peter Zauner, and Luc Moreau. Recording and Using Provenance in a Protein Compressibility Experiment. In *HPDC*, 2005.
10. Yi Huang, Alek Slominski, Chatura Herath, and Dennis Gannon. WS-Messenger: A Web Services based Messaging System for Service-Oriented Grid Computing. In *CCGrid*, 2006.
11. Gopi Kandaswamy, Liang Fang, Yi Huang, Satoshi Shirasuna, Suresh Marru, and Dennis Gannon. Building Web Services for Scientific Grid Applications. *IBM Journal of Research and Development*, 50(2/3):249–260, 2006.
12. James D. Myers, Carmen Pancerella, Carina Lansing, Karen L. Schuchardt, and Bret Didier. Multi-Scale Science: Supporting Emerging Practice with Semantically Derived Provenance. In *Semantic Web Technologies for Searching and Retrieving Scientific Data Workshop*, 2003.
13. Beth Plale. Resource Requirements Study for LEAD Storage Repository. Technical Report 001, Linked Environments for Atmospheric Discovery, 2005.
14. Beth Plale, Dennis Gannon, Dan Reed, Sara Graves, Kelvin Droegemeier, Bob Wilhelmson, and Mohan Ramamurthy. Towards Dynamically Adaptive Weather Analysis and Forecasting in LEAD. *LNCS*, 3515:624–631, 2005.
15. Yogesh Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. *SIGMOD Record*, 34(3):31–36, 2005.
16. Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Framework for Collecting Provenance in Data-Centric Scientific Workflows. In *ICWS*, 2006.
17. Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. Towards a Quality Model for Effective Data Selection in Collaboratories. In *IEEE Workshop on Scientific Workflows and Dataflows (SciFlow)*, 2006.
18. Jun Zhao, Carole Goble, and Robert Stevens. An Identity Crisis in The Life Sciences. In *IPAW*, 2006.
19. Yong Zhao, Michael Wilde, and Ian T. Foster. Applying the Virtual Data Provenance Model. In *IPAW*, 2006.