SOFTWARE METAPAPER

# Komadu: A Capture and Visualization System for Scientific Data Provenance

Isuru Suriarachchi[1], Quan (Gabriel) Zhou[1] and Beth Plale[1]

[1] School of Informatics and Computing, Indiana University, Bloomington, IN, 47405

Data provenance captured from scientific applications is a critical precursor to data sharing and reuse. For researchers wanting to repurpose data, it is a source of information about the lineage and attribution of the data and this is needed in order to establish trust in a data set. Komadu is a standalone provenance capture and visualization system for capturing, representing, and manipulating provenance coming from scientific tools, infrastructures, and repositories. It uses the W3C PROV standard [1] in representing data, and it is the successor of the Karma [2] provenance capture system which was based on Open Provenance Model (OPM) [3]. Komadu comes with two different interfaces: a Web Services interface based on Apache Axis2 [4] and a messaging interface based on RabbitMQ [5]. Komadu is completely open source and the source code is publicly available on GitHub [6]. Even though Komadu has been used most extensively in relation to scientific research, its interfaces are designed to collect and visualize provenance of any kind of application needing provenance.

**Keywords:** Data provenance; Data Lineage; Visualization; Komadu

## 1. Overview

### Introduction

Data provenance is information about the entities, activities and people who have effected some type of transformation on a data product through the product's lifecycle. Data provenance captured from scientific applications is a critical precursor to data sharing and reuse. For researchers wanting to repurpose and reuse data, it is a source of information about the lineage and attribution of the data and this is needed in order to establish trust in a data set. Data provenance has been shown useful in results validation, failure tracing, and reproducibility. The Komadu provenance capture system is standalone, meaning it is not coupled to or dependent upon any database management system, repository, or scientific workflow system. It provides an ingest API through which provenance notifications are fed into the system at high speeds, and a query API through which provenance information can be queried. The data model is both event oriented and graph oriented, in that graphs are pieced together in Komadu based on the events received from the environment.

Komadu has its roots in the Karma [2] provenance capture system, an earlier version that complied with the OPM community standard [3] both for defining the type of provenance notifications that the system accepted, and for defining the format of the results. Komadu, on the other hand, supports the W3C PROV specification [1] which provides far richer types of relationships and has a more formal model for handling time than does OPM. Karma was additionally limited by assuming that every notification belonging to the same external activity shared a common global identifier that is shared across all components (services, methods etc.) of the external environment. This limitation was found to be severe in applications where provenance is not only captured at the application level, but also at in the larger environment where the application runs. Take for instance a distributed application running in PlanetLab [7] and running under Twister [8]; it is highly limiting to expect provenance events generated from the application, from PlanetLab, and from Twister to all have shared knowledge about any single global identifier. This limitation derives from Karma's early days where it tracked provenance for applications running within a single workflow system. Additionally, a researcher may be interested in tracking lineage starting from some data product or agent. Such scenarios are not supported by Karma.

In this paper, we introduce Komadu [9] provenance capture and visualization system. Komadu is a complete redesign and reimplementation of Karma that supports new features while addressing the above mentioned limitations of Karma. The main contributions of Komadu are as follows.

1. **PROV Support:** Komadu is completely W3C PROV specification compliant.
2. **Beyond Workflow:** Komadu client API is simple and designed according to standards defined in the

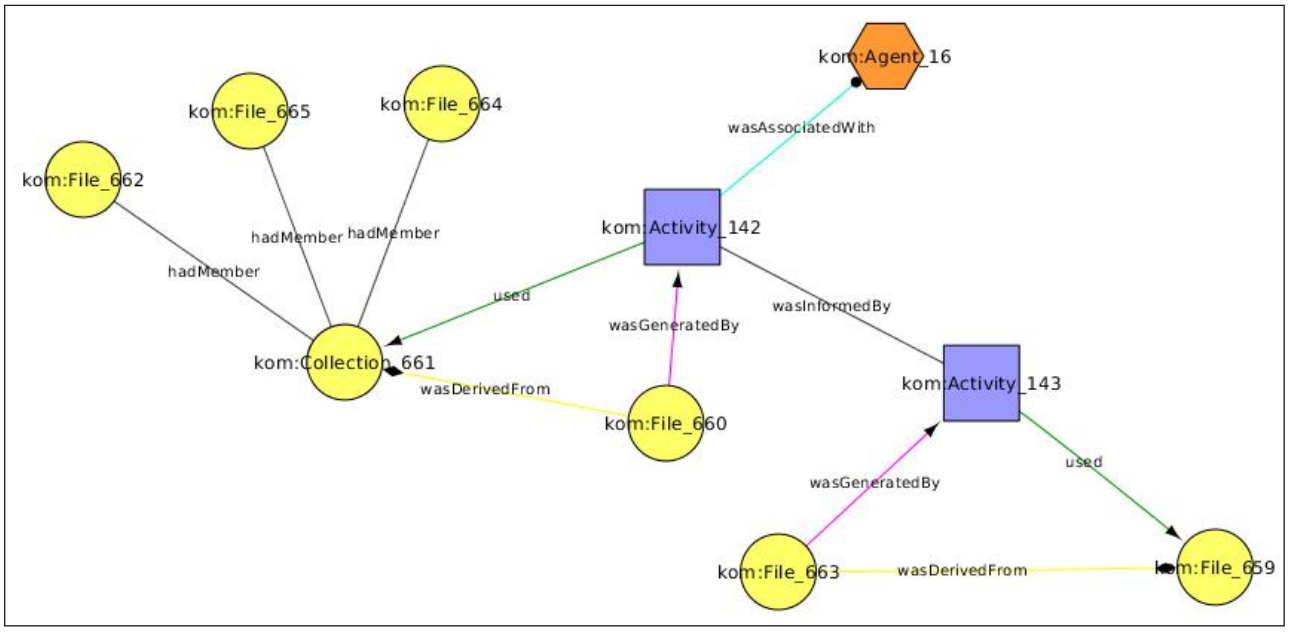


**Figure 1:** Sample Komadu provenance graph.

specification. This makes Komadu usable for any kind of provenance capture.

3. **Context-free:** Unlike Karma, the graph generation algorithm used in Komadu does not depend on any global context identifier. This makes it possible to collect provenance from disparate and unrelated pieces of infrastructure and application. It of course introduces the challenge to be handled within Komadu of stitching together graphs based on events that are not easily identifiable as being causally related. Komadu is backward compatible with Karma through graph generation that uses global context identifiers. A new user is advised to use the more convenient context-less mechanism.
4. **Multiple Perspectives:** In addition to generating provenance from the perspective of an activity, Komadu is capable of generating provenance graphs starting from data products and agents as well.

Provenance is generated through first instrumenting an application, a tool, or system middleware directly; or by processing log files after executing the application. Komadu has support for both generation mechanisms. Queries can be executed any time once the notifications are ingested. Komadu comes with an Ingest API and a Query API, both of which are exposed as a Web Service and a Messaging service.

A tool to enable the visualization of provenance graphs is included in the package as this aids the researcher in making sense of what can be gigabytes and terabytes of provenance. When the provenance is voluminous, which it can easily be, it is hard to extract important information through any other means. Komadu comes with a simple command line tool that converts the generated XML graph into a CSV file. This CSV file can be imported into most of the visualization tools. We use Cytoscape [10] to visualize provenance generated from Komadu (and Karma before it) and have written a provenance specific plugin for Cytoscape. **Figure 1** shows a sample provenance graph generated by Komadu and visualized using Cytoscape. This graph includes dummy Activities, Entities and Agents generated by one of the Komadu integration test cases. An Agent (Agent_16) invokes a service (Activity_142). Activity_142 uses a collection of files (Collection_661) in the generation of an output file (File_660). Service (Activity_142) additionally invokes service (Activity_143); the latter reads input file, File_663, and generates output file, File_659. All elements and relationships contain additional attributes and those are displayed in a separate window when a specific part of the graph is selected.

We recently released Komadu version 1.0, available on GitHub [6] and including source code and documentation.

## Motivation

An example of Komadu use in a scientific setting is in the Sustainable Environment Actionable Data (SEAD) [11] DataNet project. SEAD is developing tools for data management in the long tail of science. The scientific community in the US is recognizing the need for increased availability of the data products of their research, but the mechanisms for submitting data sets to public repositories still involve considerable manual effort. SEAD tools are attempting to reduce the manual barrier to submission of data from scientists in the long tail, those that generate small but highly voluminous data sets. SEAD had adopted the notion of the Research Object (RO) [12] as the unit of preservation, and uses Komadu to track the lifecycle of the RO through derivation, revision, and reuse.

The use of Komadu in SEAD is captured in **Figure 2**. A sustainability science project sets up a shared Project Space for its work. When a collection is ready for publication to a public repository, it is “published”, whereupon the SEAD Virtual Archive (VA) picks it up, and curates it.

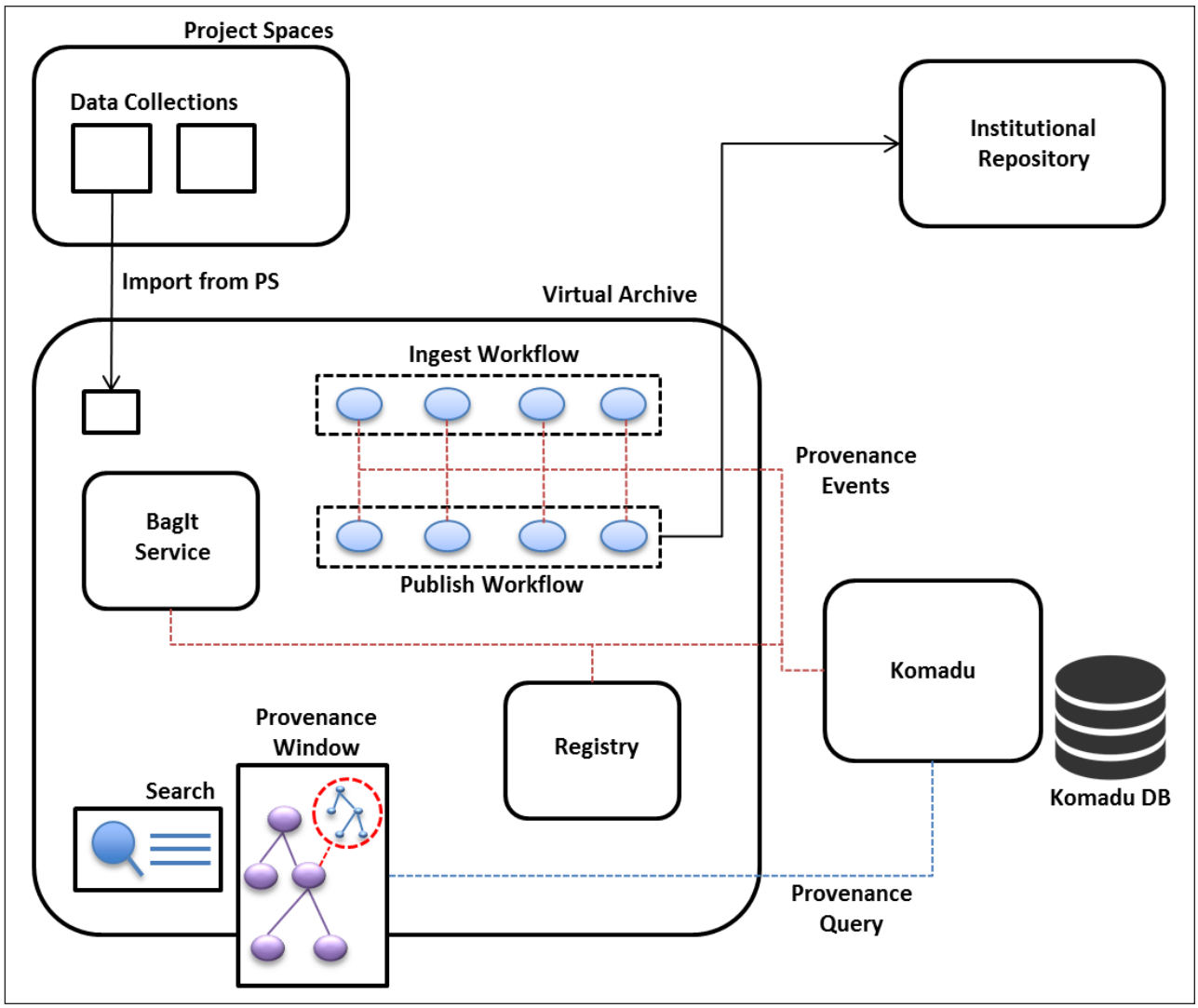


**Figure 2:** Provenance Capture in SEAD VA using Komadu.

A BagIt service inside VA pulls collection metadata and data from Project Spaces and generates additional metadata. An ingest workflow is then invoked, carrying out functions on the ingest package. The provenance of the activity is published into Komadu. Once the data collection is ingested, the data curator who often works in an academic library, can edit related metadata using the VA user interface. Provenance events of the curation actions are also sent to Komadu. Finally, when the curation is complete, the data curator publishes the collection into an appropriate Institutional Repository. Provenance events related to publish workflow are also captured. VA components like BagIt, Workflows and Registry are distributed components that are connected using web service interfaces. All those components push provenance events into Komadu using the web service interface exposed by Komadu. The SEAD VA allows scientists to search for published data collections. When a particular collection is selected, a provenance graph related to that collection is visualized in a separate provenance window. This provenance graph contains the relationships between collections and also provenance information inside each collection.

### Implementation and architecture

**Figure 3** shows the high level architecture of Komadu. Komadu can be run as a Web Service hosted on Apache Axis2 [4] or as a standalone server that listens to a RabbitMQ message queue. In both cases, a client has to be created to send messages into Komadu. A research programmer instruments a researcher's application, services, and tools to ingest provenance notifications into Komadu on the fly or can use a log processing script that parses execution logs for provenance information after the execution of the application. Queries can be issued to Komadu to generate provenance graphs and retrieve provenance information related to Activities, Entities or Agents. Both ingest and query APIs are exposed through Web Services and RabbitMQ channels, the RabbitMQ channels are set up and configured by the research programmer. More details on how to set up Komadu as an Axis2 Web Service or as a RabbitMQ server can be found in the user guide [13].

**Ingest API:** The Ingest API is used to send provenance notifications into Komadu during provenance collection time. XML notifications must be compliant with the Komadu XML Schema. For example, if a service A invokes

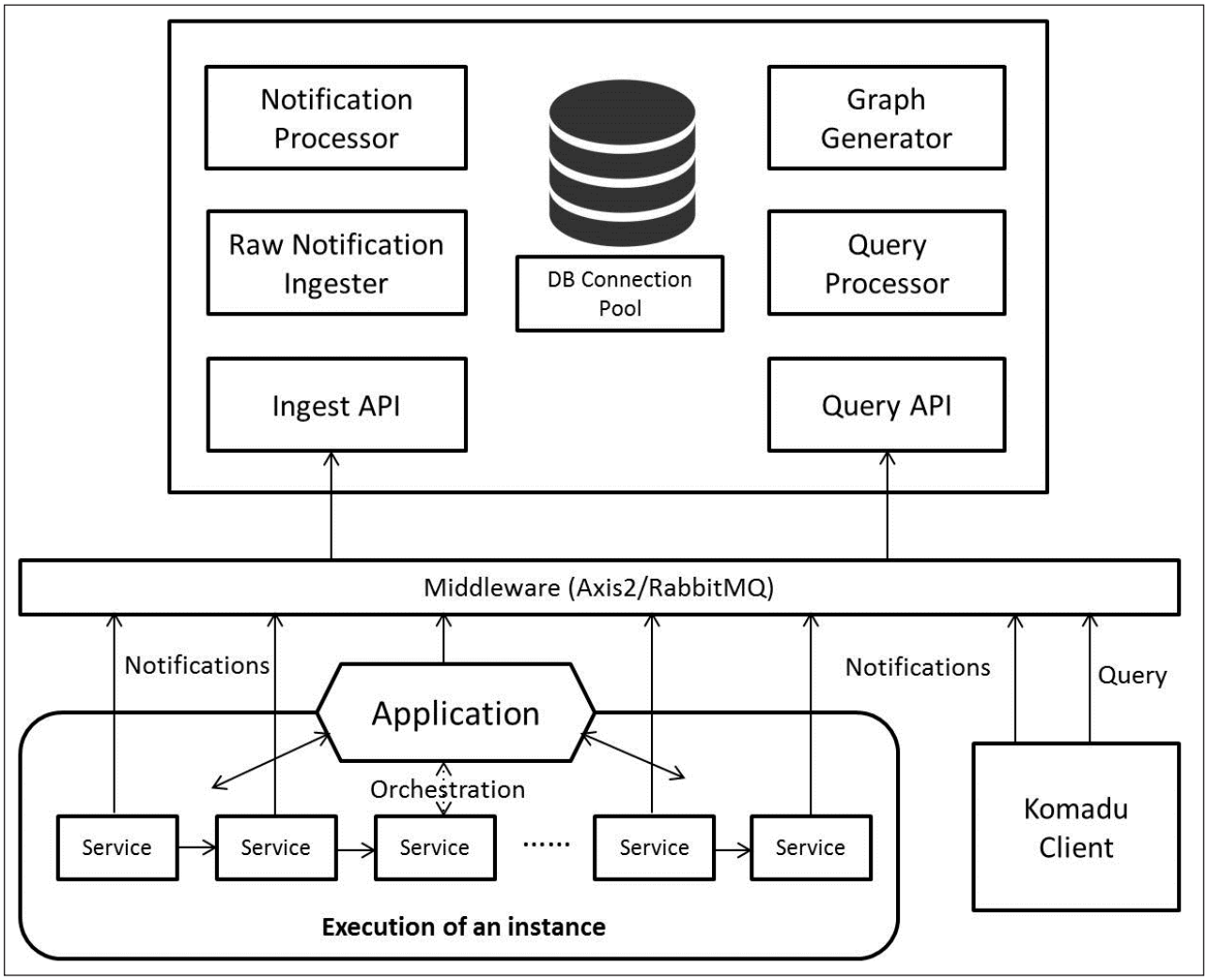


**Figure 3:** Komadu Architecture.

a service B using some parameters, the notification will contain the identifiers of Service A and B and the required parameters.

**Query API:** The Query API is used to issue queries to the Komadu server anytime after the provenance is captured. Queries can be issued to retrieve information about certain Activities, Entities etc. or to get the generated provenance graphs. Queries also must be compliant with Komadu XML Schema.

**Database:** Komadu uses a MySQL database to store all incoming notifications, processed components, their relationships and generated provenance graphs. A connection pool is used to create and efficiently manage database connections under high data rates.

**Raw Notification Ingester:** This component is responsible for ingesting incoming XML raw notifications into the database as quickly as possible. Once the message is ingested, the incoming thread is immediately returned to make the server more responsive under high loads. Raw notifications are processed by a separate component running asynchronously.

**Asynchronous Raw Notification Processor:** This component is responsible for processing raw notifications in the database asynchronously. It consists of a pool of threads that run periodically and check whether there are unprocessed notifications left in the database. If such notifications are found, those are processed and split into Activities, Entities and Agents and stored back into the Komadu database. Once processed, raw notifications are marked as processed in the database. Each notification comes as a relationship between two elements (Ex: Activity-Activity, Activity-Entity etc.) and contains element identifiers. Raw notification processor creates elements in the database if those do not already exist and adds the relationship between those elements.

**Query Processor:** query processor is responsible for handling all incoming queries. There are two types of queries: queries that request specific information about a certain Activity, Entity or Agent or graph queries. The former are easy to handle and the Query Processor directly accesses the database to respond to such queries. For the latter, the Graph Generator is invoked with the relevant start node identifier sent by the client.

**Graph Generator:** The graph generator generates provenance graphs for incoming node identifiers. It starts by creating the start node with the incoming Activity, Entity or Agent identifier, then continues to add connected nodes into the graph until there are no connected nodes left. The graph generation process uses a depth first approach and a stack of unexpanded nodes. Graph Generator uses

caching to improve performance where the cache interval is configurable by the system administrator. When a new graph is created, it is cached in the database and if the same graph is requested again within the cache interval, cached graph is returned. If the cached version is expired, a new graph is created and the cache is updated.

**RabbitMQ Messaging Channel:** RabbitMQ is an eventing system supporting asynchronous communication. A publisher publishes messages to a middleware, often called a broker; one or more subscribers subscribe to channels on which the publishers place notifications. Komadu uses RabbitMQ as a messaging broker to receive provenance notifications and send responses to the incoming queries. RabbitMQ provides a persistent and reliable store for messages.

**Axis2 Web Service Channel:** When Komadu is deployed as a Web Service on top of Axis2, a service client can be used to communicate with the server using the SOAP messaging protocol. A service client can be easily generated using the Komadu WSDL file.

Komadu is implemented completely in Java programming language and it uses MySQL as the backend database. Komadu uses the Prov Toolbox [14] library in the process of generating provenance graphs. Komadu API is clearly defined as an XML schema and it is exposed on both Web Services and Messaging channels.

### Quality control

Komadu comes with a comprehensive set of functional tests that cover ingesting all types of notifications into Komadu, executing graph generation queries and executing detail requesting queries. Maven [15] is used as the build tool for Komadu and all tests are executable using the Maven scripts. Any user who checks out the code can execute these tests to understand the functionality and get familiarized with the client API. These test cases use the Komadu Web Service API. First the code should be built without tests to generate binaries. Once the binary files are generated, those should be deployed as an Axis2 Web Service on an application server like Apache Tomcat [16]. The MySQL database has to be created using Komadu database schema and connected with the Komadu service. Once this is done, the Maven build can be executed with test cases. Komadu client that is integrated with the Maven build send ingest and query requests to the Komadu Web Service hosted on Tomcat.

In addition, Komadu has been load tested under SEAD [11] project with more than 20 concurrent users performing data curation activities using SEAD VA user interface. For each curation activity, provenance information is stored in Komadu and retrieved when the user tries to see the lineage graph. Komadu was able to handle the concurrent ingest and query requests during this test without any failures.

## 2. Availability

### Operating system

Komadu is tested on Ubuntu 12.04, Red Hat Linux 6.4, Windows 7 and Mac OS X 10.8. As it is implemented in Java, Komadu runs on any operating system that runs Java.

### Programming language

Komadu is implemented completely in Java. It runs on JDK 1.6 or higher versions.

### Additional system requirements

We recommend at least 4GB of memory to run Komadu. However, this depends on the expected load and size of notifications and generated provenance graphs.

### Dependencies

MySQL 5.1 or higher
MySQL connector/JDBC 5.1 or higher
Prov ToolBox 4.0
Apache Axis2 1.6.2
Apache Tomcat 6.0.x or higher
RabbitMQ Server 3.3.1 or higher
Apache Maven 3.0

### List of contributors

1. Suriarachchi, Isuru (Indiana University)
2. Zhou, Quan (Indiana University)
3. Ghoshal, Devarshi (Indiana University)
4. Chen, Peng (Indiana University)
5. Chandrasekar, Kavitha (Indiana University)
6. Plale, Beth (Indiana University)

### Software location

***Archive***

***Name***
Komadu: A Provenance Collection and Visualization System

***Persistent identifier***
DOI: http://dx.doi.org/10.5281/zenodo.12698

***Licence***
Apache License, 2.0

***Publisher***
Zenodo

***Date published***
13/11/2014

### Code repository

***Name***
GitHub

***Identifier***
https://github.com/Data-to-Insight-Center/komadu

***Licence***
Apache License, 2.0

***Date published***
24/07/2014

### Language

English

## 3. Reuse potential

The main use of Komadu is for tracking lineage of data generated and used in scientific research. As a standalone system, provenance can be aggregated from tools, services, applications, and middleware. The generated provenance traces have been shown as useful to reproduce the workflow execution and to trace failures.

As mentioned previously, Komadu is the successor of the Karma provenance capture system. One of the drawbacks of Karma is that its APIs are tightly coupled with workflows. Almost all operations in Karma client API are using workflow related terms. Therefore, it is hard to use Karma to collect provenance data in other settings. One of the main goals of Komadu was to overcome this limitation. Komadu APIs are designed using the generic definitions used in the W3C Prov [1] specification. Therefore, Komadu can be easily used by any kind of application where provenance data has to be collected. Komadu comes with a comprehensive documentation [13] explaining how to set up the server, how to create a client, how to execute queries and how to visualize provenance graphs.

Another area where Komadu is useful is research data preservation repositories like SEAD [11]. Actions are taken on research/data objects that reside in long term repositories. These actions could affect the object and thus should be part of the provenance record. For example, once a dataset is submitted into a repository, number of data curators can edit metadata or transform the dataset into different formats. Provenance can be used to distinguish changes to a research object that can be constituted a revision (e.g., by same author, to correct error) from those that should be viewed as a derivation (e.g. subset of data object used for another purpose). This can be accomplished by integrating Komadu with the preservation repository.

A single standalone provenance tool like Komadu can serve as an aggregator of provenance from multiple sources. There is nothing preventing Komadu from representing other data manipulation processes that occur in industry or government. Finally, it can be useful for big data processing frameworks like Apache Hadoop [17] and Apache Storm [18] in certain applications to track lineage of produced data products. Users can integrate Komadu in their application specific code (Ex: In Hadoop mapper or reducer) to collect provenance data.

For support related to Komadu, any of the authors of this paper can be contacted. In addition to that, Komadu user mailing list can be used for free support. Directions for subscribing to the mailing list can be found on the Komadu project page [9].

### Competing Interests

This work funded in part by the National Science Foundation under grant OCI-0940824. No other competing interests exist.

### References

1. **PROV-DM:** *The PROV Data Model*. [online] Available at: http://www.w3.org/TR/prov-dm/ [Accessed 14 Nov. 2014].
2. **Simmhan, Y., Plale, B.** and **Gannon, D.** 2006 A Framework for Collecting Provenance in Data-Centric Scientific Workflows. In: *Proceedings of the IEEE International Conference on Web Services*. Washington: IEEE Computer Society, pp. 427–436.
3. **Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P.** and **Myers, J.** 2011 The open provenance model core specification (v1. 1). *Future Generation Computer Systems*, 27(6), pp. 743–756.
4. **Apache Axis2.** [online] Available at: http://axis.apache.org/axis2/java/core/ [Accessed 14 Nov. 2014].
5. **RabbitMQ.** [online] Available at: http://www.rabbitmq.com/ [Accessed 14 Nov. 2014].
6. **Komadu Git Repository.** [online] Available at: https://github.com/Data-to-Insight-Center/komadu [Accessed 14 Nov. 2014].
7. **PlanetLab.** [online] Available at: https://www.planet-lab.org [Accessed 14 Nov. 2014].
8. **Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S., Qiu, J.** and **Fox, G.** 2010 Twister: A Runtime for Iterative MapReduce. *The First International Workshop on MapReduce and its Applications (MAPREDUCE'10) - HPDC2010*
9. **Komadu Provenance Collection Tool.** [online] Available at: http://d2i.indiana.edu/provenance_komadu [Accessed 14 Nov. 2014].
10. **Cytoscape.** [online] Available at: http://www.cytoscape.org/ [Accessed 14 Nov. 2014].
11. **Sustainable Environment Actionable Data.** [online] Available at: http://sead-data.net/ [Accessed 14 Nov. 2014].
12. **Bechhofer, S., De Roure, D., Gamble, M., Goble, C.** and **Buchan, I.** 2010 Research Objects: Towards Exchange and Reuse of Digital Knowledge. *Nature Precedings*, (ERIM Project Document erim1rep091103ab12). Retrieved from http://eprints.soton.ac.uk/268555/
13. **Komadu Userguide.** [online] Available at: http://d2i.indiana.edu/sites/default/files/komaduuserguide.pdf [Accessed 14 Nov. 2014].
14. **ProvToolbox.** [online] Available at: http://lucmoreau.github.io/ProvToolbox/ [Accessed 14 Nov. 2014].
15. **Apache Maven.** [online] Available at: http://maven.apache.org/ [Accessed 14 Nov. 2014].
16. **Apache Tomcat.** [online] Available at: http://tomcat.apache.org/ [Accessed 14 Nov. 2014].
17. **Apache Hadoop.** [online] Available at: http://hadoop.apache.org/ [Accessed 14 Nov. 2014].
18. **Apache Storm.** [online] Available at: https://storm.apache.org/ [Accessed 14 Nov. 2014].

**How to cite this article:** Suriarachchi, I, Zhou, Q and Plale, B 2015 Komadu: A Capture and Visualization System for Scientific Data Provenance. *Journal of Open Research Software,* 3: e4, DOI: http://dx.doi.org/10.5334/jors.bq

**Published:** 30 March 2015



]u[ *Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS