Provenance Framework for the Cloud Environment (IaaS)

Muhammad Imran
Research Group Entertainment Computing
University of Vienna, Austria
Email: imran.mm7@gmail.com

Helmut Hlavacs
Research Group Entertainment Computing
University of Vienna, Austria
helmut.hlavacs@univie.ac.at

***Abstract*—Cloud providers can optimize resource utilization and energy consumption by finding patterns in their usage. One way of finding such patterns is to study the history of Cloud resources activity. This approach is known as Cloud provenance. Provenance can also be used to track errors and faults in Cloud services. We have developed a provenance framework for research Clouds in order to find the history of the resources usage. Our framework collects provenance data in response to the request of users for IaaS scheme. In this paper, we discuss a provenance framework in the Clouds and present different possible approaches of the provenance collection process. To the best of our knowledge, provenance is yet to be addressed in the Cloud environment. Hereby, we provide details of our proposed framework and present its performance evaluation. The experimental results show that our provenance framework has a very low overhead (less than milliseconds), which makes it ideal for the Cloud infrastructure.**

Keywords-provenance framework; cloud IaaS.

I. Introduction

The vision of Cloud is to address a complex engineering, medical or social problem. Cloud enables the end user to process huge amount of data and/or satisfy his needs for mass computational power via resource virtualization. The experiments are performed on Cloud on a large scale and shift of technology is already in progress [1], [2]. Infrastructure as a Service (IaaS) is the new paradigm for researchers to deploy complex applications into Cloud. This is different than Grid [3] and distributed environments where a user had to adopt their application to the grid infrastructure and policies. IaaS scheme provides a raw resource which is hired and updated according to the requirement of the application by a user without knowing the complexity and details of the underlying architecture. A resource is hired when a match is found based on a user and application requirements such as memory, disk space, resource type and/or Cloud provider. This is called on-demand computing and in the process of resource allocation, a user is charged with some price. Once a resource is updated and used, the user may take a snapshot of the resource if the same resource is to be used later on.

Workflow [4] is designed to execute activities in order for a complex application in e-Science. Provenance of a workflow activities [5] is the information about intermediate data and processes to verify the execution of an application. Provenance in general means; "the origin or source of an object". In Clouds, provenance can be broadly categorized into user data (applications installed on a virtual machine), instance type (memory, disk size, number of instances) and resource type (image ID, location). Such information is of high importance to utilize the cloud resources, e.g., a resource already built and updated by one user can be used by others with minimum or no change of the installed applications and components. Furthermore, mining provenance data can be used to forecast a future request, e.g., Eddy Caron [6] used string matching algorithm on recent history data to forecast a next request. Similarly, networks in general and Clouds in particular are prone to errors, and the history data can be utilized in Clouds to resolve the errors with minimum effort.

Clouds are still in the process of evolution and provenance is yet to be implemented (addressed) in Clouds. Contributions of this paper are the following:

- A brief overview of research Clouds IaaS and a detailed discussion of possible schemes to incorporate provenance into Cloud environment.
- A use case of provenance usage and example metadata from IaaS Cloud.
- Detailed architecture of our provenance framework for Cloud IaaS and the evaluation of collecting and storing provenance data.

The rest of the paper is organized as follows. Section II summarizes the research Cloud architecture and discusses the possible provenance schemes. Section III gives the details of the underlying architecture (middleware) used by the research Clouds and the extension of this architecture to collect provenance data. Section IV gives a brief overview to use the provenance data and utilize Cloud resources. In Section V, we present the test results of the collection and storage module. Section VI concludes our work and presents the directions for the future implementations.

II. Research Cloud IaaS Architecture and Discussion of Provenance Schemes

A. Research Cloud IaaS

Cloud computing is generally categorized into three types which are business, Research or private and hybrid Clouds.

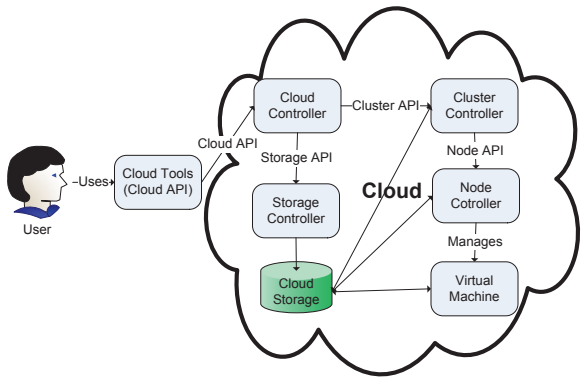

Figure 1. Generalized Cloud Architecture.

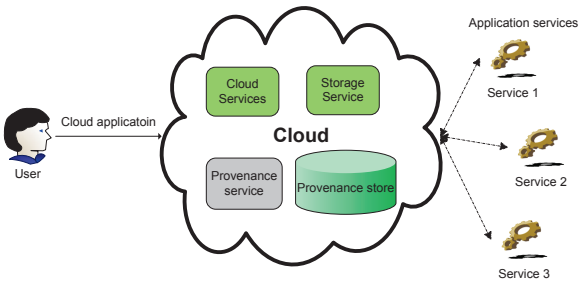

Figure 2. Provenance Service as Part of Cloud Services.

They are further subdivided into three schemes which are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Three different Clouds, i.e., EUCALYPTUS [7], Nimbus [8] and OpenNebula [9] were explored to understand their internal architecture and communication mechanism between various components. This helped us to discuss the possible provenance schemes for Cloud IaaS and implication of our proposed framework. The main components of IaaS Cloud are summarized below:

- Application tools: Application Programming Interface (API) available to communicate with Cloud services, e.g., resource hiring, starting, stopping, saving and/or describing the state of a particular resource.
- Cloud, Cluster and Node: The request of users is handled by the Cloud and routed to Clusters and Nodes respectively. The Node communicates with Virtual Machine (VM) and the job of Cluster is to manage different nodes in the network.
- Storage: Cloud offers storage unit (file system) to save user data and raw disk images to be run as resources. Communication with a storage unit is controlled by a service, e.g., Walrus in Eucalyptus Cloud and a user can save the updated state of a running machine. The process of saving the updated machine into Cloud is called snapshot.

Figure 1 presents a generalized architectural overview and control flow from user to VM in Cloud IaaS.

B. Provenance as a Part of Cloud Services

In this scheme, the Cloud provider needs to provide a service which will communicate with other Cloud services including cluster, node and storage to collect provenance data. This scheme proposes the application of provenance as a part of overall Cloud Infrastructure. The following list the advantages of provenance inside the Cloud IaaS.

- Easy to use as provenance is already a part of Cloud infrastructure and a user can decide to turn it on/off just like other Cloud services.
- Users will prefer this scheme as they do not need to understand the structure of provenance framework and is the responsibility of the Cloud provider to embed such a framework.

The following lists the disadvantages of such a provenance scheme.

- Cloud providers cannot charge users for such a scheme unless it has some benefits of resource utilization and initialization for users.
- In case of Cloud services failure, provenance system will also fail and there is no way to trace the reason for the failure.
- There will be extra burden on the Cloud provider because the usage of Cloud resources must increase due to incorporating the provenance system as a part of Cloud framework.
- Such scheme can only work with a particular version of Cloud IaaS. Any change in Cloud model or services signature needs an appropriate change in provenance application.

In distributed, grid and workflow computing, there are many examples of provenance data management and schemes [10]–[13]. Each of these schemes is designed for a particular environment and they rely on the underlying services model. Therefore, the existing techniques cannot be applied to Cloud environment and further, Cloud services are not extensible to third party applications. Figure 2 presents a provenance system as a part of Cloud services.

C. Provenance is Independent of Cloud Services

A provenance scheme which adopts a modular and an agent like approach to address cross platform, applications and different Cloud providers is independent of Cloud infrastructure. Such a scheme must address on-demand, pay as you go and extremely flexible Cloud architecture. Advantages of an independent provenance scheme are:

- Independent of Cloud services and various applications domain.
- Failure of Cloud will not affect provenance scheme as it is not a part of Cloud.

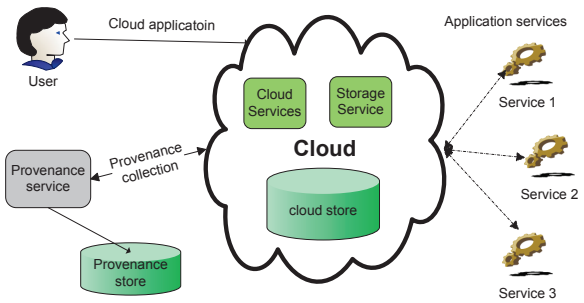

Figure 3. Provenance as an Independent Module

- The users and Cloud providers will be able to track faults and errors if some Cloud services failed to work properly.
- Usability and simplicity of such a scheme is very high because the user has complete control of the provenance system.

Disadvantages of such a scheme are as follows:

- Complete understanding of Cloud services is required to make any changes and communicate with the Cloud infrastructure.
- Trust is required on behalf of the Cloud provider because of request, permission and response from the Cloud services to the provenance module.
- Any change in Cloud services, their signature, or communication mechanism will need an appropriate change in provenance scheme.

In workflow computing, Karma [14] is using a notification broker where all the activities are published to and stored in a provenance store. The technique proposed by the Karma service is not part of a workflow enactment engine and it works as a bridge between the provenance store and the enactment engine. Figure 3 gives a brief overview of an independent provenance scheme in Cloud.

D. Discussion

Both of these approaches have their pros and cons. While considering provenance for Cloud IaaS, the major challenge is to address the Cloud extensibility. Clouds are not extensible by nature and in case of open Clouds, a developer needs a deep understanding of the source code in order to make any changes. Keeping this point in view, we propose a provenance framework which is independent of Cloud IaaS provider and with minimal or no changes required in Cloud services.

III. Provenance Framework

Research Clouds rely on the open source technologies to provide an infrastructure (IaaS). These open source technologies includes JAVA and C/C++ languages, and Apache, Axis and Mule [15] communication frameworks. A general consensus is that Cloud would not be possible without these open source technologies. Research Cloud, e.g., Eucalyptus use Apache, Axis2/C and Mule engines to deploy Cloud services as IaaS and built a communication mechanism between different components. Apache is widely used for its speed, lightweight engine and its support of SOAP, WSDL and REST interfaces. Similarly, Mule is an integration platform used to connect various applications and/or services. These technologies are used to connect various Cloud components and are called middleware.

This middleware is extensible and a developer can add custom methods to the already deployed applications and service. The proposed framework is based on this feature of extensibility from Apache, Mule and other third party tools and consists of the following components: **Provenance Collection**, **Provenance Parsing**, **Provenance Storage**, **Provenance Query** and **Provenance Visualization**. First, the explanation of Apache architecture and its extension for the development of the provenance framework is given.

A. Apache (Axis2/C)

To develop a provenance framework, the following components of the Apache architecture are utilized.

Handler: or interceptor is the smallest execution unit in the message passing system of the Apache Engine. The idea is to intercept the flow of a message and perform the additional task submitted by a user. Handler can read and write to the message context (apache messaging system). A handler has two parts: header and body. The header specifies the name and body the operation. There are predefined handlers in the Apache Axis execution chain and also the ability to provide custom handlers developed by the developers [16]. A group of handlers that is orchestrated and deployed within the Apache engine is called a module.

Phase: is the concept in Apache Axis to support the dynamic ordering of the handlers. It acts like a bucket in which where the handler is put. A phase can have one or more handlers. Apache provides different kinds of phases spanning from global (for overall axis communication) to operational (for a particular operation or web method).

Flow: is a collection of phases. Phase is more like a logical collection where flow is a real execution chain. There are four types of flows in Apache engine.

- InFlow
- OutFlow
- InFaultFlow
- OutFaultFlow

Similarly, other third party libraries and frameworks used by Clouds are also extensible. Examples of such frameworks are the use of Mule in Eucalyptus, Axis in Nimbus and Apache xml-rpc in OpenNebula. The basic architecture of these libraries is different but the main idea of an interceptor or handler is the same. For example, in Mule, the message context is referred to as Mule Message. Interceptors can be deployed before and after a component is invoked in the

Mule framework. The Mule message before and after resides in flows which are called inbound-router and outbound-router respectively.

B. Provenance Collection

When a message enters Apache engine, it goes through InFlow and invokes all the handlers inside. InFaultFlow is similar and handles a faulty incoming request, e.g., sending wrong arguments to the web service method or any other unexpected condition that prevents the request to succeed. OutFlow is invoked when a message is moving out of Apache engine (invoking all handlers in OutFlow) and the OutFaultFlow is invoked when something goes wrong in the out path, e.g., a host is shut down unexpectedly. Various Flows within Apache engine and the execution of a service with input and output messages is described in figure 4. The left side of figure details the different flows and the right side gives an overview of one single flow with phases and handlers concepts (both built in and user defined). Custom handlers, using C/C++ for provenance collection are deployed in four different Flows of the Apache execution chain. When a component inside Cloud IaaS is invoked, provenance collection module intercepts the flow, collects and parses the message for provenance data in the corresponding execution flow.

C. Provenance Parsing and Storing into XML File

SOAP message inside Apache engine is intercepted by the collector module which passes this message to the parser. The parser reads the SOAP message, parse it accordingly and store the data in a well defined XML file. We used XML schema for the collected provenance data because it is widely used model for data representation. The XML can be used to maximize the advantages of custom algorithms and third party applications. To query the provenance data, it is better to provide a standard schema and hence the usage according to individual preferences.

TableI presents a sample of collected, parsed and stored provenance data by our provenance framework. This data represent user activity for methods of Eucalyptus cluster service and detail the timestamps, resource type and instance specific information. <UserData> is the list of applications specified by user to populate the resource and <TimeStamp> are corresponding start and finish time for a web service method.

D. Provenance Query

Custom applications can query provenance data based on the user requirements. We find the activity pattern in Cloud IaaS based on a resource type, instance type, time used or user ID in our example query. This information can be used to monitor Cloud IaaS and the frequently used resources can be moved to a faster CPU/disk unit for better performance. Algorithm 1 is used to find activity patterns based on the the resource-ID.

Algorithm 1 Solve Query Q: Q = Return Resource Types (emi-IDs) in XML Store

Require: XMLStore, ClusterName
Ensure: XMLStore is not Empty
  Begin
  Array ResouceType[] T
  OpenXMLFile(XMLStoreLocation)
  FindCluster(ClusterName)
  while ParentNode<MethodName> == RunInstance) **do**
    $T \leftarrow$ ChildNode(<ImageID>)
  end while
  End

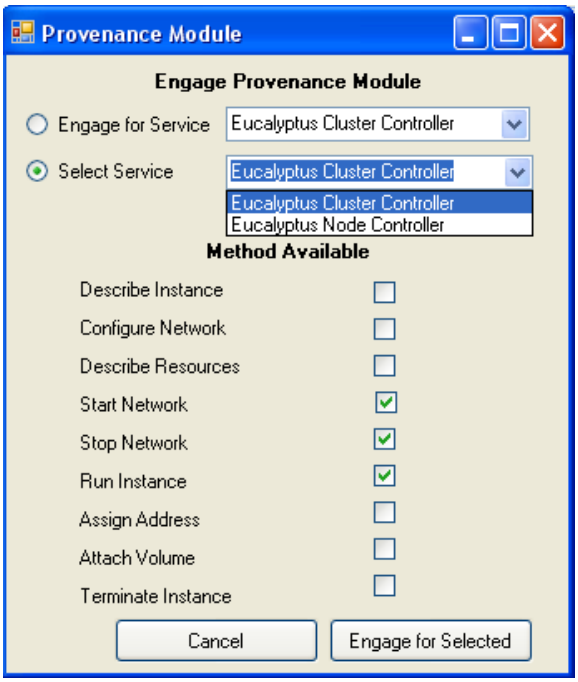

Figure 5. User Interface for Engaging Provenance Module into Cloud

E. User Interface

Usability of the proposed provenance framework is very high. Cloud providers can enable/disable the provenance module according to their choice. Different options are available to enable/disable the provenance module based on the requirements of the Cloud provider. Some of the options are: to enable/disable provenance module for all clusters, a particular cluster, all nodes, a particular node, or selected methods from a particular cluster or node. Figure 5 presents a prototype of user interface available to Cloud IaaS provider.

F. Framework Experience

By extending the middleware (Apache and Mule) using handlers, we are independent of Cloud provider and different IaaS schemes. We followed a modular approach and divided

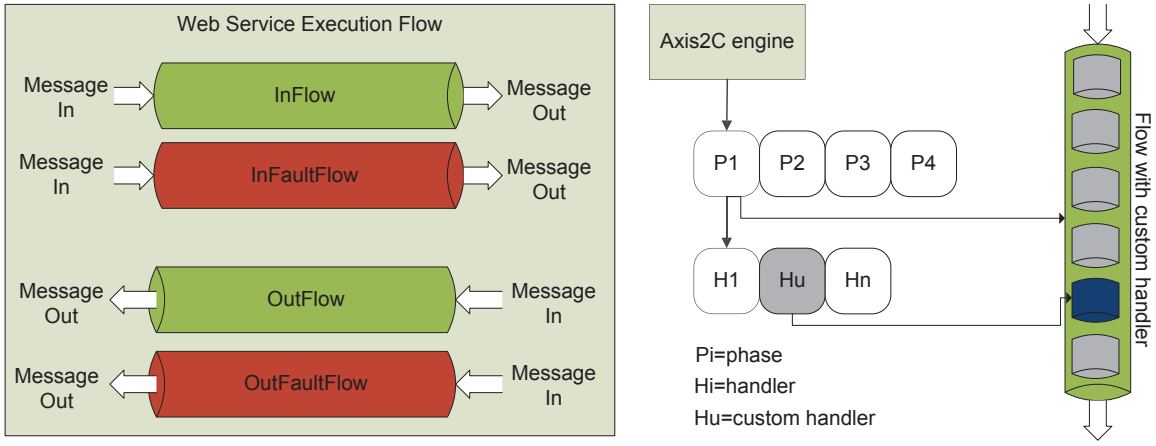

Figure 4. Apache Axis2C Architecture

<EucalyptusServiceName> ClusterController</EucalyptusServiceName>		
<MethodName>StartNetwork</MethodName>	<TimeStamp> Start and End Time of Method</TimeStamp> <ClusterAddress> 131.130.32.12</ClusterAddress> <UserID>admin</UserID>	
<MethodName>RunInstance</MethodName>	<ImageID>emi-392B15F8</ImageID> <KernelID>eki-AE1D17D7</KernelID> <RamdiskID>eri-16981920</RamdiskID> <ImageURL>emi-URL</ImageURL> <RamDiskURL>eri-URL</RamDiskURL> <KernelURL>eki-URL</KernelURL>	Instance Type <Name>m1.small</Name> <Memory>512</Memory> <Cores>1</Cores> <Disk>6</Disk> <UserData>DataFile</UserData>
<MethodName>StopNetwork</MethodName>	<TimeStamp> Start and End Time of Method</TimeStamp> <UserID>admin</UserID>	

Table I
SAMPLE METADATA FOR CLOUD IAAS

our framework into different components. The future of provenance in Cloud lies in a lightweight and independent provenance scheme to address cross platform, Clouds IaaS and application domains. The proposed framework can be deployed without making any changes to the Cloud services or architecture. **Advantages** of the scheme are:

- It is independent of Cloud services and platform and it works with any Cloud IaaS which use the Apache, Mule or similar frameworks.
- The proposed framework follows a soft deployment approach and therefore, no installation is required.
- Some of the challenges offered by Cloud infrastructure are virtualization, "on demand" computing, "pay as you go" model, more abstract, extremely flexible and the services are not extensible by nature. The proposed framework address these challenges in automatic fashion as being part of Cloud middleware.

Major **Disadvantage** of proposed framework is:

- Rely completely on the extension of the middleware and cannot work on any other Cloud IaaS where middleware is not extensible.

IV. CLOUD PROVENANCE: A USE CASE FOR EFFICIENT RESOURCE INITIALIZATION AND ENERGY CONSUMPTION

Description: Resource utilization is critically important both from the resource provider and Cloud performance perspective. In the Cloud resource allocation process, a user may request a resource with the input file of required applications that is the same as a previously initialized resource but will still need to build the resource from scratch. The Cloud resource utilization can be maximized if one is able to provide automatic discovery of already running instances, saved volumes and snapshots. The automatic discovery will not only help in resource utilization but will also provide means to reduce the time and energy consumed. Our proposed framework collects the metadata information regarding time, user, cluster and location of newly created volumes or snapshots and stores it in a provenance database. To make the process of resource allocation efficient and automatic, the broker (which takes input from user) compares the user input file with existing provenance data. If the comparison of input file results in an exact match then instead of starting a new resource from scratch, the existing resource volume and snapshot are deployed.

Actors: End-user and Cloud provider. A user benefits from

Table II
UNDERLYING ARCHITECTURAL COMPONENTS

Cloud provider	Operating system	Cloud services engine	Languages	Storage unit	Virtualization	Service tested
Eucalyptus 1.6.2	Linux Ubuntu 10.04 Server	Axis2/C 1.6.0	C,C++	File system (XML)	KVM/XEN	Cluster controller

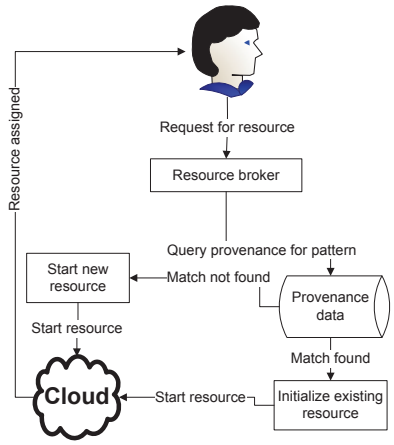

Figure 6. Resource Initialization Using Provenance

this scheme by saving his time and effort to build a resource from scratch. On the other hand, the Cloud provider utilizes existing deployed resources and saves energy.

Advantages:

- Faster resource initialization in case a match is found.
- Utilization of existing deployed resource (volumes, snapshots) to save energy, cost and time.
- Overall Cloud performance will increase.

Figure 6 describes the process of using provenance data and making Clouds more efficient and proactive.

V. EVALUATION

Different approaches are proposed in literature for collecting and storing provenance data to reduce the computation and storage overhead [17]. Mainly, there are two methods. The first method proposes to collect provenance data and store a copy of the parent object. The disadvantage of this method is a huge storage overheard. The second method proposes to store links of the parent object. This method is faster and storage overhead is very low. Disadvantage of this method is consistency in case a parent object is deleted or moved.

To store provenance data we followed the second approach and the proposed framework stores only the link information about the activity of users and Cloud components. The provenance data consists of information like: Cloud images, snapshots, volumes, instance types and user data etc. Real data is already stored in the Cloud storage unit and we do not make a copy of this data. Since links are lightweight, therefore computation and storage overhead for the provenance data is negligible.

We evaluated the cluster controller service and results were surprising for collection and storage module. To get physical evidence, timestamps were calculated at the beginning of provenance module invocation and later on when the data is parsed and saved into XML file. Time overhead including the provenance module for Inflow and Outflow phases of Apache were less than milliseconds. To find the storage overhead we calculated file size of provenance data for individual methods. We chose a worst case scenario where all the incoming and outgoing data was stored. This process was performed for every method in Eucalyptus cluster service and the average file size of stored provenance data is about 5 KB for each method. Evaluation was performed by using the underlying architecture detailed in table II. Physical machine details for running IaaS Cloud are the following:

Number of PCs: 2 (PC1 with Cloud, Cluster and Storage Service, PC2 with Node service), Processor: Intel Core (TM) 2: CPU 2.13 GHz, Memory: 2GB, Disk Space: 250 GB

It is essential to note that the low computation and storage overhead of the provenance frameworks is because of two reasons. First, we used an approach where the extension of the middleware is achieved by built in features. This approach does not add any extra burden except the collection of provenance data. Second, we store the provenance data by using a link based approach. This approach saves on duplicating the storage of huge amounts which already exists in Cloud database.

VI. CONCLUSION AND FUTURE WORK

With the evolution of technology and IaaS, complex applications are target environment for Clouds. Clouds offers "on demand" computing and "pay as you go" model, where applications discover resources at run time. The focus of this paper is provenance data for Cloud IaaS scheme. A client application hires resources from IaaS and populates them according to the requirements. These populated resources are saved in Cloud storage unit and can be used by other applications having the same requirements. This process requires storage of users or application activity. First, we discussed general approaches to collect activity information performed on Cloud IaaS with their pros and cons. By using those approaches as the basis of our study, we developed a framework which is not dependent on Cloud services or underlying architecture. We divided our framework into different components and proposed a use case scenario where the collected provenance data can be used to utilize Cloud resources and to save cost, energy and time. Collecting and

storage overhead of the proposed framework is very low. In the future we will extend the framework for resource utilization in order to save cost, time and energy using provenance.

REFERENCES

[1] C. N. Hoefer and G. Karagiannis, "Taxonomy of cloud computing services," in *Proceedings of the 4th IEEE Workshop on Enabling the Future Service-Oriented Internet (EFSOI'10), Workshop of IEEE GLOBECOM 2010, Miami, USA*, ser. 2010 IEEE GLOBECOM Workshops. USA: IEEE Communications Society, December 2010, pp. 1345–1350.

[2] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ser. ESCIENCE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 640–645.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Int. J. High Perform. Comput. Appl.*, vol. 15, no. 3, pp. 200–222, Aug. 2001.

[4] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[5] R. S. Barga, Y. L. Simmhan, E. Chinthaka, S. S. Sahoo, J. Jackson, and N. Araujo, "Provenance for scientific workflows towards reproducible research." *IEEE Data Eng. Bull.*, vol. 33, no. 3, pp. 50–58, 2010.

[6] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, ser. CLOUDCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 456–463.

[7] Eucalyptus. [retrieved: may, 2012]. [Online]. Available: http://open.eucalyptus.com/

[8] Nimbus. [retrieved: may, 2012]. [Online]. Available: http://www.nimbusproject.org/

[9] Opennebula. [retrieved: may, 2012]. [Online]. Available: http://opennebula.org/

[10] M. Szomszor and L. Moreau, "Recording and reasoning over data provenance in web and grid services." ser. Lecture Notes in Computer Science, R. Meersman, Z. Tari, and D. C. Schmidt, Eds., vol. 2888. Springer, 2003, pp. 603–620.

[11] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," in *Proceedings of the 27th International Conference on Very Large Data Bases*, ser. VLDB '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 471–480.

[12] P. Buneman, S. Khanna, and W. chiew Tan, "Why and where: A characterization of data provenance," in *ICDT '01: Proceedings of the 8th International Conference on Database Theory*. Springer, 2001, pp. 316–330.

[13] Y. L. Simmhan, B. Plale, and D. Gannon, "A Survey of Data Provenance Techniques," Computer Science Department, Indiana University, Bloomington IN, Tech. Rep., 2005.

[14] Y. L. Simmhan, B. Plale, D. Gannon, and S. Marru, "Performance evaluation of the karma provenance framework for scientific workflows," in *in: International Provenance and Annotation Workshop (IPAW)*. Springer, 2006, pp. 222–236.

[15] Mule esb. [retrieved: may, 2012]. [Online]. Available: http://www.mulesoft.org/what-mule-esb

[16] A. S. Foundation, "Apache axis2/java - next generation web services," Website http://ws.apache.org/axis2/, Jul. 2009.

[17] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva, "Bridging workflow and data provenance using strong links," in *Proceedings of the 22nd international conference on Scientific and statistical database management*, ser. SSDBM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 397–415.