Provenance Framework for the Cloud Infrastructure: Why and How?

Muhammad Imran and Helmut Hlavacs
Research Group Entertainment Computing, University of Vienna, Austria
email: {muhammad.imran,helmut.hlavacs}@univie.ac.at

Abstract—**Provenance is an important aspect in the verification, audit trails, reproducibility, privacy and security, trust, and reliability in distributed computing, in-silico experiment and generally in e-science. On the other hand, Cloud computing is the business model of distributed computing and is considered the next generation of computing and storage platforms. Cloud computing requires an extension of the architecture of distributed and parallel systems by using virtualization techniques. Key to this extensible architecture is to support properties such as compute on-demand and pay-as-you-go model. Many research domains have already adopted Cloud paradigm into their existing computational and storage platforms and, thus, a shift of technology is in progress. In this paper, we give an overview of Cloud architecture and the importance of provenance in Cloud computing. We provide the mechanism for the collection of the provenance data while addressing the challenges offered by this new paradigm. These challenges include mainly the abstraction, high scalability and the inability to modify or extend the Cloud services. We provide a framework that requires minimal knowledge and understanding of underlying services and architecture of a Cloud. We assure trust by augmenting a Cloud infrastructure with provenance collection in a structured way. Then, we present the architectural overview of the provenance framework and the performance results of the extended architecture. The experimental results show that our provenance framework has a very low computation overhead (less than milliseconds), which makes it a good fit for the Cloud infrastructure.**

Keywords*—*provenance, middleware, cloud.

I. Introduction

This paper is the extension of our previous article [1], where we proposed a framework that addressed the challenges posed by Cloud paradigm. Here we extend that work and present the more detailed framework and results by adding some contents from our article [2].

Oxford dictionary [3] defines provenance as the place of origin or earliest known history of something. In distributed computing, provenance is defined by a set of different properties about the process, time, and input and manipulated data. Provenance is considered an important ingredient for tracing an object to its origin. Provenance is used to answer a few basic questions such as when the object was created, the purpose of creation, and where the object originated from (e.g., the creator of the object). In computing science, a provenance system is used to collect, parse, and store related metadata. Such data is used for verification and tracking, assurance of reproducibility, trust, and security, fault detection, and audit trials. These metadata include functional data required to trace back the creation process of objects and results, but also nonfunctional data such as the performance of each step including, e.g., energy consumption.

Cloud is an evolving paradigm which is based on virtualization and offers on-demand computing and pay-as-you-go model. Furthermore the architecture of a Cloud enforces high scalability and abstraction. The vision of this new paradigm is to address large scale computation and distributed storage management, e.g., a complex engineering, medical or social problem. Cloud enables the end user to run complex applications and satisfy his needs for mass computational power via resource virtualization. Many experiments are performed on Cloud on a large scale and shift towards Cloud is already in progress [4], [5]. Infrastructure as a Service (IaaS) is one of the service models of Cloud that is utilized by researchers to deploy complex applications [6]. This is different than grid [7] 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.

The execution of complex applications in Cloud means, to request various resources and updating them accordingly. In this case, provenance can be broadly categorized into categories of: user data (applications installed on a virtual machine), instance type (memory, disk size), number of instances, resource type (image ID, location) and information about the users and Cloud provider. 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 [8] 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 [9].

Cloud infrastructure (IaaS) is composed of various services and components that cannot be modify and therefore, existing techniques are not suitable for Cloud environment and to address Cloud specific challenges. A possible approach is to follow an independent and modular provenance scheme as described in [10]. Such a scheme is possible by extending the middleware of Cloud infrastructure where various components and services are deployed (extension of third party tools and libraries). This scheme is loosely coupled (domain and application independent) and hence works independently of Cloud infrastructure, client tools and is of high importance to support future e-science. There is a strong need to propose a provenance scheme for this dynamic, abstract and distributed environment. In addition to challenges for distributed computing, the abstraction and highly flexible usage pose new demands, i.e., a provenance framework for Clouds has to support these issues.

In this paper, we provide a discussion of provenance with a particular focus on open or research Clouds infrastructure. We present underlying architecture of open Cloud, discuss the possible schemes to incorporate provenance, and propose a framework for provenance data collection in the Cloud. Hereby, we address the most important properties of the proposed framework, that is, independence of the Cloud architecture, low storage and computational overhead of provenance data, and usability. Following are the major contribution of this article:

- giving reasons of the importance of provenance data and highlighting the challenges for provenance collection in Clouds;
- a brief overview of research Clouds IaaS and a detailed discussion of possible schemes to incorporate provenance into Cloud environment.
- a framework that can be deployed to the Cloud environment while addressing different vendors and architectures;
- a use case of provenance usage and example metadata from IaaS Cloud.
- the 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 provides the related work in field of distributed computing and gives an overview of the Cloud architecture. Section III highlights the importance and the implication of a provenance enabled Cloud. Section IV discusses the possible schemes to incorporate provenance in Cloud, an overview of the related provenance data while addressing the challenges offered by Cloud infrastructure. Section V gives the details of the provenance framework by extending the underlying architecture (middleware) used by the research Clouds in a seamless and modular fashion. Section VI presents the results for the collection and storage of provenance data in Eucalyptus Cloud. Section VII gives a brief overview to use the provenance data and utilize Cloud resources, where Section VIII concludes our work and presents the directions for the future work.

II. Related Work and Background

Provenance has been addressed in distributed and workflow computing, e.g., Rajendra Bose et al. [11] present a detailed survey of computational models and provenance systems in these environments. However, none of the approaches support provenance in the Cloud environment. These existing schemes rely on the support of native services from distributed or workflow computing, e.g., process schedulers. Generally, provenance systems in grid, workflow, and distributed computing are either strongly part of the enactment engine or they use Application Programming Interfaces (APIs), which are enactment engine specific [12].

Numerous techniques and projects have been proposed during the last few years for provenance in computational sciences for validation, reproduction, trust, audit trials and fault tolerance. These techniques range from tightly coupled provenance system to loosely coupled systems [13]–[16]. Provenance Aware Service Oriented Architecture (PASOA) [17], [18] uses Service Oriented Architecture (SOA) [19] for provenance collection and its usage in distributed computing for workflow management systems. myGrid [20] and Kepler [21] are examples of projects for executing in-silico experiments developed as workflows and they use Taverna [22] and Chimera [23] schemes respectively for provenance data management in these computational systems. However, none of these approaches were designed specifically for Cloud computing architecture.

Recently, Muniswamy-Reddy et al. [24] discussed the importance of provenance for Cloud computing services offered by AMAZON EC2 [25] using Provenance-Aware Storage Systems (PASS) [26]. PASS collects the provenance data on file system level and records the various calls made to different objects. These calls are recorded on kernel level and therefore, virtual images with only PASS installed will collect and produce provenance data. A similar approach to PASS is proposed in [27] which gathers data provenance by recording system calls but without modifying the kernel. This scheme [27] uses a plug-in interface called dtrace for file system and browser, etc., and requires the knowledge of all valid entry points where provenance should be recorded in the corresponding application or domain. Since Clouds are categorized into different categories, e.g., application, infrastructure, platforms and storage, etc. Therefore, mostly research work is focus on one or more particular component. For example, there are research works that consider provenance at the layers like a web browser [28] and virtual machine [29]. Similarly, a short survey about various techniques from grid and distributed computing is provided that discuss the tracking of data in Cloud by following the layered architecture of Cloud and the provenance data for various layers [30]. In the e-science domain, experiments are performed in dry labs (in-silico), which requires hiring various resources from Cloud infrastructure and updating them accordingly. In this case, a provenance system has to address the data collection and availability in the Cloud environment and, therefore, our focus in this paper is to collect the provenance data for Cloud infrastructure.

A. *Cloud Architecture*

Cloud computing is the ability to increase capacity or add capability such as storage, computation and/or networking on the fly. It relies on sharing computational and storage resources over the network. Cloud computing is not a single entity and the architecture is divided into various components. These components depends on the deployment model of services, the deployment model of infrastructure and the characteristics provided by a Cloud environment [31]. Clouds are generally categorized as business Cloud, research or private Cloud and hybrid Cloud. Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) are the terms heavily used in a Cloud computing paradigm and is mostly broken into these three segments.

- IaaS: a service provided for the infrastructure (hardware and software) over the internet. Such an architecture provides servers, virtualized operating systems and data storage units. Elastic Cloud is a commonly used term for IaaS and users pay for required resources as they go. Amazon Elastic Compute Cloud (Amazon EC2), Nimbus[1], OpenNebula[2] and Eucalyptus[3] are the leading examples of IaaS.
- PaaS: a platform which is build on top of IaaS. PaaS provides an interface for software developers to build new or extend existing applications, e.g., Google App Engine and Microsoft Azure.
- SaaS: is an application service provided to the end user by a vendor, e.g., google mail. This application is executed on the Cloud infrastructure and the data is stored in Cloud database but, this is not visible to the end user.

Private Cloud IaaS schemes are mostly used in a research environment and small businesses by using open source technologies. They are rapidly growing in the size and magnitude and expanding in different domains. With the new technologies and advancements, a private Cloud can be part of other public or private Clouds thus, providing the functionality of a hybrid Cloud.

B. *Eucalyptus*

Eucalyptus is an open source implementation of Cloud computing IaaS scheme using JAVA and C/C++ for various components. Users can control an entire Virtual Machine (VM) instance deployed on a physical or virtual resource [32]. It supports modularized approach and is compatible with industry standard in Cloud, i.e., Amazon EC2 and its storage service S3. It is one of the most used platforms to create scientific and hybrid Clouds. Eucalyptus gives researchers the opportunity to modify and instrument the software which is been lacking in the business offerings, e.g., Amazon EC2. Figure 1 presents the extended architecture of Eucalyptus Cloud. 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. This works as a client side application to communicate with Cloud infrastructure.
- Cloud, Cluster and Node Controller (CLC, CC and NC): CLC (middleware), CC and NC communicates with each other and outside applications using Mule [33] and Apache Axis2/C framework. CLC is the entry point to the Cloud when requests are made by users via application tools. CLC interacts and routes the incoming request to a particular CC. CC is the part of Cloud used to manage clusters in the network. CC interacts and controls different NCs by associating and differentiating them using unique addresses and also balancing load in the cluster. NC assign a VM for the job execution submitted by a user.
- Storage: Cloud offers a distributed storage unit (object based storage) to save user data and raw disk images. These raw images (virtual machines) are later run as resources. Communication with a storage unit is controlled by a service, e.g., Walrus in Eucalyptus. Walrus can be used directly by users with REST protocol to stream data in/out of Cloud, e.g., files. Walrus can also be used indirectly by using SOAP protocol while communicating with CLC services to upload, modify and delete virtual images.

[1]http://www.nimbusproject.org/
[2]http://opennebula.org/
[3]http://open.eucalyptus.com/

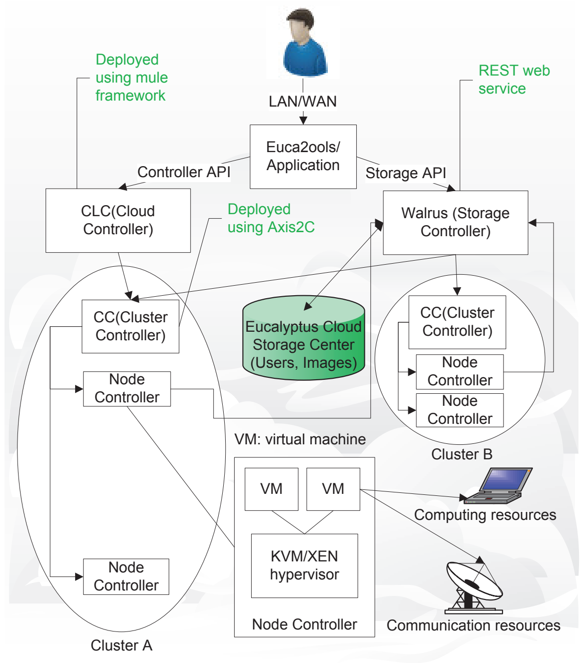

Fig. 1: Extended architecture of Eucalyptus Cloud.

All the communications between different components of Eucalyptus Cloud are achieved by using SOAP, XML, WSDL, and HTTP communication protocols via Axis2/C and Mule framework.

III. Provenance in Cloud: *Why*

There are various alternative terms of Cloud computing, e.g., utility computing, autonomic computing and virtualization along with various definitions [34] because it is used as per the understanding, knowledge and requirements by different organizations, research communities and users. Indeed, there are some differences from previous computing paradigms, for instance virtualization, on-demand computing and storage, pay-as-you-go model, extremely flexible and more abstract architecture. Ian Foster et al. [35] present an overview of the major differences between Cloud and grid and mentions the most important feature of Cloud technology is the total dependence on services (SOA architecture). There is underlying architecture for networking of software and hardware but, to the end user it is completely abstract and hidden. The abstraction allows the end user to send data to Cloud and get data back, without bothering about the underlying details. This behavior is fine for a normal user but, in research environment, scientists are more interested in the overall process of execution and a step by step information to keep a log of sub-data and sub-processes to make their experiments believable, trust able, reproducible and to get inside knowledge. With improvements of in-silico experiments, most of the computation and processing is done by using computing resources and not in a real lab.

The execution and deployment of application on Cloud requires various resources from the Cloud infrastructure and updating them accordingly as per application requirements. This process involves the communication between the client and Cloud infrastructure. Cloud infrastructure is divided into various components as described in previous section that handles the incoming request and routes them accordingly. Each component in the process contributes some specific metadata (provenance) from Cloud, Cluster, Node, Storage, Provider and User perspective. This provenance data is not only important for the verification of the application execution but it plays an important role in the behavior analysis of Cloud. This behavior is further utilized for the efficient allocation of the resources and predicting future request.

Users of Cloud environment may not be interested in the physical resources, e.g., brand of computer but, surely they are interested in the invoked service, input and output parameters, time stamps of invocation and completion, overall time used by a process, methods invoked inside a service and the overall process from start to finish. For the Cloud infrastructure this also includes the details about the provider, users, provided resources and the details of each particular resource hired by a user. This metadata that provides the user an ability to see a process from start to finish or simply track back to find the origin of a final result and the details information about the processes and resources taking part in the final output is called provenance. Generally, provenance is used in different domains by scientists and researchers to trust, track back, verify individual input and output parameters to services, sub process information, reproducibility, compare results and change preferences (parameters) for another simulation run. Provenance is still missing in Cloud environment and needs to be explored in detail as mentioned in [24], [36].

A. *Implication of a Provenance Enabled Cloud*

Introducing the provenance data into Cloud infrastructure would result in following advantages:

- Patterns: The use of provenance data to find patterns in the Cloud resources usage. These patterns can be further utilized to forecast a future request.
- Trust, reliability, and data quality: The final data output can be verified based on the source data and transformation applied.
- Resources utilization: In Cloud, provenance data can be used to utilize the existing running resources by allocating a copy of a running resource. This will be achieved by comparing a new request to the already running resources and this information is available in provenance data.
- Reduced cost and energy consumption: Provenance data results in a cost and energy efficiency by using patterns to forecast a future request and by utilizing existing running resources.
- Fault detection: Provenance data can pinpoint the exact time, service, method and related data in case of a fault.

IV. Provenance Scheme and Discussion

In this section, we focus on the challenges offered by Cloud technology for the collection of provenance data. Thereby, we present the important provenance data and how the Cloud architecture can be extended to incorporate provenance collection.

A. *Provenance Challenges in Cloud*

Usual provenance challenges include: collecting provenance data in a seamless way with a modularized design and approach, with minimal overhead to object identification, provenance confidentiality and reliability, storing provenance data in a way so it can be used more efficiently (energy consumption) and presenting such information to the end user (query, visualization). Cloud brings more challenging to these existing challenges because we have to address the scalable, abstract and on-demand model and architecture of a Cloud. A provenance framework in Cloud should address the following challenges:

- Domain, Platform, and Application independence: How the provenance system works with different domain (scientific, business, database), platforms (windows, linux) and applications.
- Computation overhead: How much extra computation overhead is required for a provenance system in a particular domain.
- Storage overhead: How and where is the provenance data stored. It depends on the type, i.e., copy of original data or a link reference to original data, granularity (coarse-grained or fine-grained) and storage unit (SQLServer, MySQL, file system) of provenance data.
- Usability: It determines the ease of use of a provenance framework from a user and Cloud resources provider perspective. How to activate, deactivate and embed a provenance framework into existing Cloud infrastructure and services, e.g., is it completely independent or modification is required on Cloud services layer.
- Object identification: Identify an object in the Cloud and link the provenance data to source by keeping a reference or by making a copy of the source object.

- Automaticity: With the huge amount of data and process computation within Cloud, collecting and storing provenance data should be automatic and consistent.
- Cloud architecture: Addressing the on-demand, abstract and scalable structure of Cloud environment with availability and extensibility of different components.
- Interaction with Cloud services: Cloud services cannot be modified or extended. Business Clouds are propriety of organizations and open source Clouds needs understanding of every service if change is required. The better approach is to provide an independent provenance scheme, which requires no change in the existing services architecture.

B. Provenance Data

A provenance framework should address two different perspectives in collecting metadata for Cloud architecture. Applications running on Cloud as SaaS or PaaS and provenance of Cloud infrastructure (IaaS). Users of Cloud are more interested in their application provenance where, providers are interested in IaaS services provenance to observe resource usage and find patterns in applications submitted by users to provide with a more sophisticated model for resources usage. Following is the list of mandatory metadata in a Cloud environment:

1) Cloud process data: Cloud code execution and control flow between different processes (web services), e.g., in EUCALYPTUS are CLC, CC, Walrus and NC services. Web service and method name in particular.
2) Cloud data provenance: Data flow, input and output datasets which are consumed and produced and parameters passing between different services.
3) System provenance: System information or physical resources details, e.g., compiler version, operating system and the location of virtualized resources.
4) Timestamps: Invocation and completion time of Cloud services and methods.
5) Provider and user: Details about Cloud users and services provider, e.g., location of clusters and nodes. Different providers have different trust level and there could be laws against usage of resources for a particular geographical area.
6) Instance data: Instance is a running resource and the provenance data includes information about disk size, memory, resource type, number of instances for a particular operation, and number of cores (CPU), etc.
7) Cloud user data: This data is part of the Cloud infrastructure. When users hires various resources, they populate them according to the application requirement before the resource is in running state. This usually includes the initial script and the basic architecture required to deploy the application such as Java Development Kit (JDK), Java Runtime Environment (JRE), database system and web services engine for example.

C. 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 as presented in Figure 2. The following list the advantages of provenance inside the Cloud IaaS.

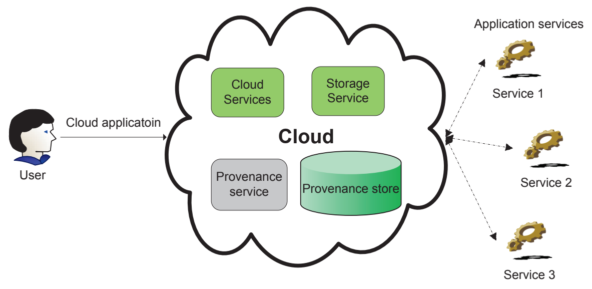

Fig. 2: Provenance service as part of Cloud services.

- 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 list 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 framework as a part of Cloud infrastructure.
- Such a scheme can only work with a particular version of Cloud IaaS. Any change in Cloud model or services signature needs an appropriate change in the provenance framework.

In distributed, grid and workflow computing, there are many examples of provenance data management and schemes [12]–[15]. Each of these schemes is designed for a particular environment and they rely on the underlying services model. Therefore, these existing techniques cannot be applied to Cloud environment as Cloud services are not extensible to third party applications.

D. Provenance is Independent of Cloud Services

A provenance scheme that 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:

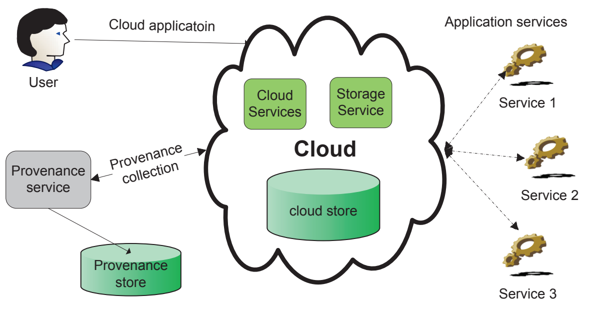

Fig. 3: Provenance as an independent module.

- Independent of Cloud services and various applications domain.
- Failure of Cloud will not affect provenance scheme as it is not a part of a Cloud.
- 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 a user has a 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 [37] 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.

E. *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 that is independent of Cloud IaaS and requires minimal or no changes in the Cloud architecture and services model.

V. PROVENANCE FRAMEWORK: *How*

A Cloud infrastructure is deployed and it relies on the open source third party tools, libraries and applications. Eucalyptus Cloud in particular depends on the Apache Axis, Axis2/C, and Mule framework. These third party libraries are used for the communication mechanism between various components of Cloud infrastructure. Cloud infrastructure is the orchestration of different services and the third party libraries works as a middleware to connect these services. The purpose of Cloud computing is to bring more abstraction than previous technologies like grid and workflow computing and therefore, Cloud services cannot be modified.

One method to implement provenance into the Cloud infrastructure is by changing the source code. This could be very cumbersome as deep understanding of the code is required. This will also restrict the change to the particular version of the Cloud. This method is not feasible to address the provenance challenge for various Cloud providers, domains and applications. The second method is to capture the provenance data on the middleware of a Cloud. This is possible by extending the third party libraries used by a Cloud infrastructure and add custom methods to collect provenance data at various different levels. Such a scheme will lead to the minimum efforts and can be deployed across any Cloud scheme. Further, there will be no change required in Cloud services architecture or signature. To understand this techniques and hence the proposed provenance framework, we give a brief overview of the most important Mule and Axis2/C architecture.

A. *Mule Enterprise Service Bus*

Mule is a lightweight Enterprise Service Bus (ESB) written in JAVA and is based on Service Oriented Architecture (SOA). Mule enables the integration of different application regardless of the communication protocol used by those applications. Eucalyptus CLC services are deployed using Mule framework. CLC services are divided into different components including core, cloud, cluster manager, msgs, etc. These different components are built and deployed as *.jar* files and they use Mule framework messaging protocols (HTTP, SOAP, XML, etc.) to communicate with each other and with other Eucalyptus services (NC and CC).

Extending Mule: Mule framework is based on layered architecture and modular design. Mule offers different kind of interceptors (EnvelopeInterceptor, TimeInterceptor and Interceptor) to intercept and edit the message flow. Since, provenance is metadata information flowing between different components (services) and we do not need to edit the message structure; therefore, we use EnvelopeInterceptor. Envelop interceptors carry the message and are executed before and after a service is invoked.

Configuring Mule Interceptor: There are two steps involved for configuring a Mule interceptors to Cloud services. First step is to built a provenance package (JAVA class files) and copying to the Cloud services directory. Second step requires editing Mule configuration files used by different CLC components. Interceptors can be configured globally to a particular service or locally to a particular method of a service. Listing 1 is a sample *"eucalyptus-userdata.xml"* Mule configuration file used to verify user credentials and groups.

Listing 1: Configuration of provenance into Mule.

```
<?xml version="1.0" encoding="UTF-8"?>
<mule xmlns="http://www.mulesource.org
    /..." >
  <interceptor-stack name="CLCProvenance">
    <custom-interceptor class="eucalyptus.
        CLCprovenance"/>
!.. indicating path of the package and
    class name for CLC services provenance
     data
  </interceptor-stack>
  <model name="eucalyptus-userdata">
    <service name="KeyPair">
      <inbound>
        <inbound-endpoint ref="KeyPairWS
            "/>
      </inbound>
      <component>
        <interceptor-stack ref="
            CLCProvenance"/>
!.. configuring "keypair service" to
    provenance module
        <class="com.eucalyptus.keys.
            KeyPairManager"/>
      </component>
      <outbound>
        <outbound-pass-through-router>
          <outbound-endpoint ref="
              ReplyQueueEndpoint"/>
        </outbound-pass-through-router>
      </outbound>
    </service>
  </model>
</mule>
```

B. Axis2/C Architecture

Eucalyptus NC and CC services are exposed to other components by using Apache Axis2/C framework. Axis2/C is extensible by using handlers and modules [38]. Handlers are the smallest execution unit in Apache engine and are used for different purposes, e.g., web services addressing [39] and security [40]. A message flow between different components of CC and NC go through Axis2/C engine and we deploy custom handlers for provenance data collection inside Axis2/C. Similar concept is used in [41] for workflow services deployed in a tomcat container. This framework is not extensible to Cloud services and architecture. We differ from that work in many factors including interceptors for Mule, Apache Axis and Apache Axis2/C. There is no tomcat container available for Cloud services to deploy the provenance framework and Cloud services use HTTP, XML, SOAP and REST based protocols. Further, the proposed framework is developed for Cloud services provenance data collection and therefore, parsing, storing, and accessing provenance data is different than their architecture.

Configuration: Axis2/C modules and handlers can be configured globally to all services by editing *axis2.xml* file, or to a particular service and method by modifying *services.xml* file. Listing 2 describes the configuration of provenance module to Eucalyptus NC service.

Listing 2: Configuration of provenance into Axis2/C.

```
<?xml version="1.0" encoding="UTF-8"?>
<service name = "EucalyptusNC">
  <module ref="NCprovenance"/>
 !..this will configure provenance to all
     methods in NC
  <Operation name="ncRunInstance">
    <Parameter name = "wsmapping">
      EucalyptusNCncRunInstance
    </Parameter>
  </Operation>
  <Operation name="ncAttachVolume">
    <module ref="NCprovenance"/>
 !..this will configure provenance to this
     particular method
    <Parameter name = "wsmapping">
      EucalyptusNCncAttachVolume
    </Parameter>
  </Operation>
</Service>
```

C. Framework Components

Proposed framework is divided into the following components to address the modularity and layered architecture of Cloud:

1) 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 4 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.

2) 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

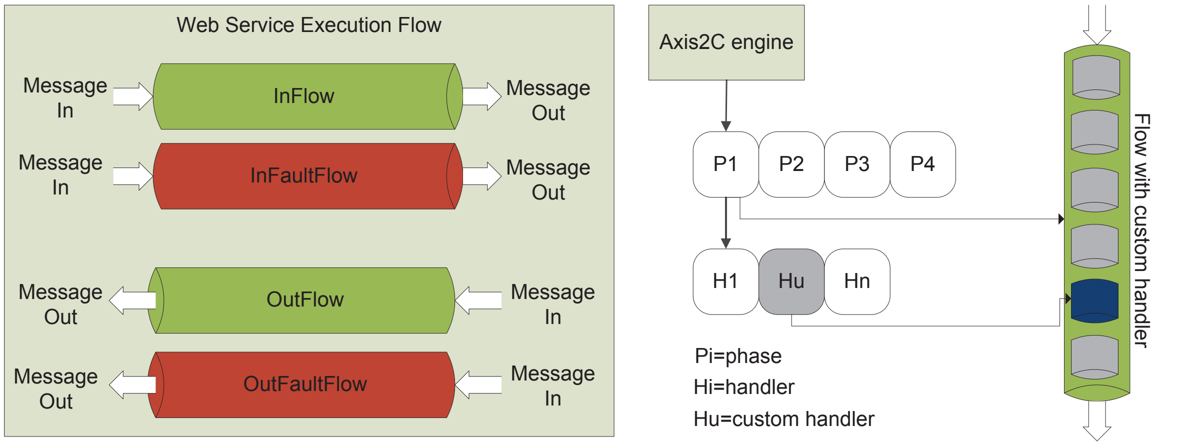

Fig. 4: Apache Axis2/C architecture.

provide a standard schema and hence the usage according to individual preferences.

Table I presents a sample of collected, parsed and stored provenance data by our provenance framework. This data represents a 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 of a web service method.

3) 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
```

4) Provenance Visualization: The visualization component takes the query as input parameter. This query is further analyzed to find the various components and their relationship. For example, a sample query:
Visualize the instances types for user1 from last 48 hours.
This query is analyzed to find the relationship between the user1, various instances requested by user1 over the last 48 hours. The result is visualized in chart form that can be changed on run time with different types of chart, e.g., line, graphs and pie, etc.

Figure 5 presents the framework components and the interaction between Cloud services using various communication protocols.

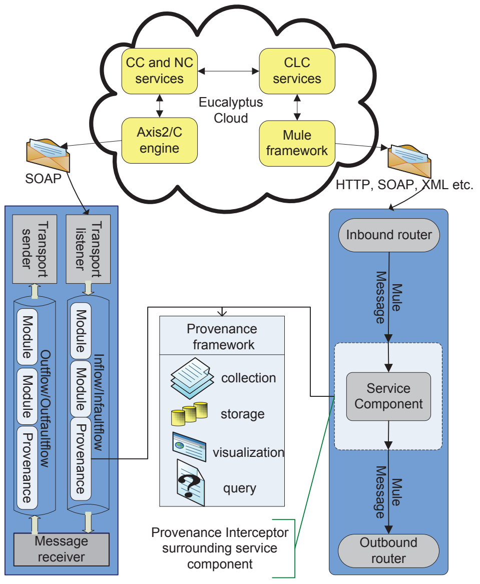

Fig. 5: Framework components.

VI. Evaluation

Different approaches are proposed in literature for collecting and storing provenance data to reduce the computation and storage overhead [42]. 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

<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 of Cloud IaaS.

huge storage overheard. This method is not feasible for Cloud infrastructure because the size of objects (virtual machines and raw resources) is in gigabytes. Similarly, persistent data stored in Walrus can vary in size but storing a copy of this huge amount of data is not practical. The second method proposes to store links of the parent object. This method is faster and storage overhead is very low. The disadvantage of this method is to handle the consistency problem 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, provider 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 very low. To get the evidence, we performed two kind of test cases.

A. Independent Testing of Middleware

First, we evaluate the Mule and Axis2/C framework independently. In this case, we calculated the increase in time for provenance collection, parsing and logging to text file. Echo service that takes an input parameter (string) and log the message to the output container was invoked 100 times in row. Five multiple runs are performed for the calculation of best time, worst time and average time of execution. The process is executed by considering overall (Inflow and Outflow), only Inflow and only Outflow provenance. The underlying architecture and system details for this test case are following:

Operating system: Ubuntu 10.04, Processor: Intel Core 2 (2 GHz), RAM: 2 GB, Axis2/C version: 1.6.0, Web service: Echo

Figure 6 presents the performance of these different execution runs on Axis2/C engine. Left side of the Figure 6 details multiple runs of Echo service without provenance, with provenance (Inflow and Outflow), only Inflow and only Outflow provenance. Y-axis represents the time required for execution. Right side of the Figure 6 shows the increase in time by comparison to without provenance. This increase in time is calculated for overall provenance, only Inflow provenance and only outflow provenance. The comparison is done for average values by using formula 1, where T_2 is time including provenance and T_1 is time excluding provenance.

$$Time\,increase = T_2 - T_1 \tag{1}$$

The average increase in time for 100 simulation runs of Echo service for collecting and logging overall provenance data is only 0.017 ms when compared to the execution without provenance. The average increase in time for only Inflow provenance is 0.009 ms and for only Outflow provenance is 0.013 ms when compared to without provenance. The individual Inflow and Outflow provenance were collected for experimental purposes to observe the respective overhead. In a real lab experiment, the overall provenance of process is essential. The increase in time for provenance collection and logging is too less and negligible when considering the advantages like fault tracking, resource utilization, patterns finding and energy consumption of a provenance enabled Cloud. Furthermore, the successful deployment of provenance collection to Axis2/C and Mule frameworks support our theory of a generalized and independent provenance framework.

B. Testing of Cloud Services

In this test case, we evaluated the cluster and node controller services of the Eucalyptus Cloud. The results were surprising for collection and storage module of the provenance framework. 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 very low (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 and node 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 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.

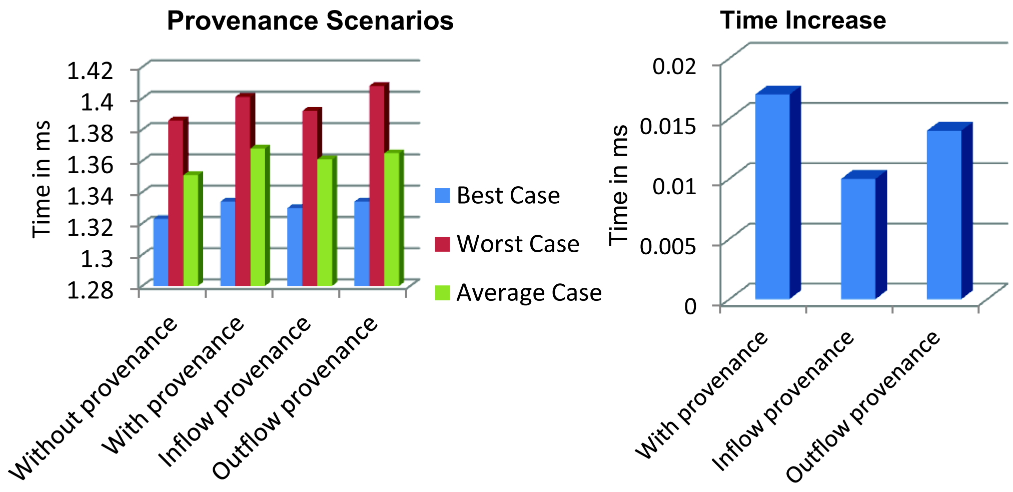

Fig. 6: Test results of Echo service.

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 and Node controller

Table III presents the performance overhead of provenance for CC and NC components. The maximum times are the exceptional cases and, therefore, we calculated the average time from multiple runs (50). The average time presents the overhead for collection, parsing and storing of the provenance data into properly defined XML files. Formula 2 is used to calculate the overall overhead by summation of individual overhead of Cloud, Cluster, Storage and Node components for the Cloud infrastructure.

$$Total\,Overhead = \sum_{i=0}^{n}(\mathbf{CLC})i + \sum_{i=0}^{n}(\mathbf{SC})i + \sum_{i=0}^{n}(\mathbf{NC})i + \sum_{i=0}^{n}(\mathbf{CC})i \quad (2)$$

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 collect and store the provenance data by using a link based approach. This approach saves the space and time required to make a copy of the original object. Furthermore, these objects already exists in Cloud database and therefore we do not need to make a copy of the existing items and objects.

C. Framework Experience

The extension of middleware (Apache, Axis2/C, Mule) by exploiting the handler and module features facilitated in the provenance collection that is independent of any Cloud provider and various IaaS schemes. We followed a modular approach and divided our framework into different components. We believe that the future of provenance in Cloud lies in a lightweight and independent provenance scheme to address cross platform, different Clouds IaaS and application domains. The proposed framework can be deployed without making any changes to the Cloud services or architecture. Following are the **advantages** of such a scheme:

- 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, encapsulation and abstraction, extremely flexible and the inability to modify Cloud services and architecture by nature. The proposed framework address these challenges in automatic fashion as being part of the Cloud middleware.

Major **disadvantage** of the proposed framework is to rely completely on the extension of the middleware and cannot work on any other Cloud IaaS where middleware is not extensible.

TABLE III: Calculation overhead (in time) for provenance.

Cloud Component	Max time(ms)	Min time(ms)	Avg time(ms)
Infrastructure (NC)	15	2	4
Infrastructure (CC)	20	7	12
Combined			16 ms

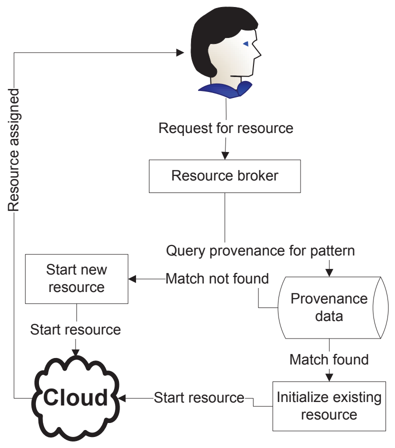

Fig. 7: Resource initialization by using provenance.

VII. Application Scenario

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. If the information regarding the previously populated resources is not available, that would require to build the new resource from scratch. The utilization of Cloud resources can be maximized if one is able to provide automatic discovery of already running instances, saved volumes and snapshots. The volume and snapshot contains information of the user activity such as installing a particular software. 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/or snapshot is deployed.
Actors: End-user and Cloud provider. A user benefits from 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:

- the faster initialization of a resource in case a match is found
- the utilization of existing deployed resource (volumes, snapshots) to save energy, cost and time
- the increase in performance for the overall Cloud architecture

Figure 7 describes the process of using provenance data and making Clouds more efficient and proactive.

VIII. Conclusion

In this paper, we explored the Cloud architecture (IaaS), its dependencies and reasoned about the importance of provenance for this evolving paradigm. With the technology shift and changes, open Clouds such as Eucalyptus, Nimbus and OpenNebula are becoming the target domain for large scale computation and storage, e.g., deploying complex applications. We proposed a framework which collects and stores the important provenance data for these environments. This framework can be deployed with minimum knowledge of the underlying architecture and without modifying the basic architecture or source code of the services. Further, we present the major challenges for the collection of provenance data in a Cloud and the proposed framework address those challenges in a structured way. While addressing the challenges, proposed framework has the properties, e.g., independent of basic architecture, simple to use, easy to deploy and works with open Cloud providers. The framework is modular and divided into different components which address various parts of Cloud infrastructure. The calculated overhead for the collection and storage of the provenance data is very low and hence does not affect the performance of the Cloud architecture and its services.

References

[1] M. Imran and H. Hlavacs, "Provenance in the cloud: Why and how?" in *Third International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2012)*, USA, July 2012.

[2] M. Imran and H. Hlavacs, "Provenance framework for the cloud environment (iaas)," in *Cloud Computing 2012*. IARIA, 2012, pp. 152–158.

[3] "Oxford dictionary," Website http://oxforddictionaries.com/ definition/english/provenance, [retrieved: May, 2013].

[4] C. N. Hoefer and G. Karagiannis, "Taxonomy of cloud computing services." USA: IEEE Communications Society, December 2010, pp. 1345–1350.

[5] 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*. IEEE Computer Society, 2008, pp. 640–645.

[6] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm, "What's inside the cloud? an architectural map of the cloud landscape," in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, ser. CLOUD '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 23–31. [Online]. Available: http://dx.doi.org/10.1109/CLOUD.2009.5071529

[7] 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.

[8] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," ser. CLOUDCOM '10. IEEE Computer Society, pp. 456–463.

[9] M. Imran and H. Hlavacs, "Applications of provenance data for cloud infrastructure," in *The 8th International Conference on Semantics, Knowledge & Grids (SKG2012)*, Beijing, China, October 2012. [Online]. Available: http://eprints.cs.univie.ac.at/3555/

[10] A. Marinho, L. Murta, C. Werner, V. Braganholo, S. M. S. da Cruz, E. S. Ogasawara, and M. Mattoso, "Provmanager: a provenance management system for scientific workflows," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1513–1530, 2012.

[11] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: a survey," *ACM Comput. Surv.*, vol. 37, no. 1, pp. 1–28, Mar. 2005.

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

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

[14] Y. Cui and J. Widom, "Lineage tracing for general data warehouse transformations," in *Proceedings of the 27th International Conference on Very Large Data Bases*, San Francisco, USA, 2001, pp. 471–480.

[15] 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*, 2001, pp. 316–330.

[16] M. Imran and K. A. Hummel, "On using provenance data to increase the reliability of ubiquitous computing environments," in *iiWAS*, 2008.

[17] S. Miles, P. Groth, M. Branco, and L. Moreau, "The requirements of recording and using provenance in e-Science experiments," University of Southampton, Tech. Rep., 2005.

[18] pasoa. [retrieved: May, 2013]. [Online]. Available: http://www.pasoa.org/

[19] oasis. [retrieved: May, 2013]. [Online]. Available: http://www.oasis-open.org/

[20] mygrid project. [retrieved: May, 2013]. [Online]. Available: http://www.mygrid.org.uk/

[21] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, and S. Mock, "Kepler: An extensible system for design and execution of scientific workflows," in *IN SSDBM*, 2004, pp. 21–23.

[22] Taverna workflow management system. [retrieved: May, 2013]. [Online]. Available: http://www.taverna.org.uk/

[23] W. System, I. Altintas, O. Barney, and E. Jaeger-frank, "Provenance collection support in the kepler scientific workflow system," in *IPAW*. Springer-Verlag, 2006, pp. 118–132.

[24] K.-K. Muniswamy-Reddy, P. Macko, and M. Seltzer, "Making a cloud provenance-aware," in *1st Workshop on the Theory and Practice of Provenance*, February 2009 2009.

[25] Amazon elastic compute cloud. [retrieved: May, 2013]. [Online]. Available: http://aws.amazon.com/ec2/

[26] K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. I. Seltzer, "Provenance-aware storage systems." in *USENIX Annual Technical Conference, General Track*. USENIX, 2006, pp. 43–56.

[27] E. Gessiou, V. Pappas, E. Athanasopoulos, A. D. Keromytis, and S. Ioannidis, "Towards a universal data provenance framework using dynamic instrumentation." in *SEC*, ser. IFIP Advances in Information and Communication Technology, D. Gritzalis, S. Furnell, and M. Theoharidou, Eds., vol. 376. Springer, 2012, pp. 103–114.

[28] D. W. Margo and M. I. Seltzer, "The case for browser provenance," in *Workshop on the Theory and Practice of Provenance*, 2009.

[29] P. Macko, M. Chiarini, and M. Seltzer, "Collecting provenance via the xen hypervisor," in *Workshop on the Theory and Practice of Provenance*, 2011.

[30] O. Q. Zhang, M. Kirchberg, R. K. L. Ko, and B.-S. Lee, "How to track your data: The case for cloud computing provenance," in *CloudCom'11*, 2011, pp. 446–453.

[31] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," Tech. Rep., Jul. 2009. [Online]. Available: http://www.csrc.nist.gov/groups/SNS/cloud-computing/

[32] S. Wardley, E. Goyer, and N. Barcet, "Ubuntu Enterprise Cloud Architecture," *Technical White Paper*, Aug. 2009.

[33] Mule esb. [retrieved: May, 2013]. [Online]. Available: http://www.mulesoft.org/what-mule-esb

[34] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1496091.1496100

[35] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," in *2008 Grid Computing Environments Workshop*. IEEE, 2008, pp. 1–10.

[36] M. A. Sakka, B. Defude, and J. Tellez, "Document provenance in the cloud: constraints and challenges," in *Proceedings of the 16th EUNICE/IFIP*. Springer-Verlag, 2010.

[37] Y. L. Simmhan, B. Plale, D. Gannon, and S. Marru, "Performance evaluation of the karma provenance framework for scientific workflows," in *IPAW*. Springer, 2006, pp. 222–236.

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

[39] Axis2- ws-addressing implementation. [retrieved: May, 2013]. [Online]. Available: http://axis.apache.org/axis2/java/core /modules/addressing/index.html

[40] Apache axis2/c manual. [retrieved: May, 2013]. [Online]. Available: http://axis.apache.org/axis2/c/rampart/docs/rampartc_manual.html

[41] F. A. Khan, S. Hussain, I. Janciak, and P. Brezany, "Enactment engine independent provenance recording for e-science infrastructures." in *Proceedings of the Fourth IEEE International Conference on Research Challenges in Information Science RCIS'10*, 2010, pp. 619–630.

[42] D. Koop, E. Santos, B. Bauer, M. Troyer, J. Freire, and C. T. Silva, "Bridging workflow and data provenance using strong links," ser. SSDBM'10. Berlin, Heidelberg: Springer-Verlag, pp. 397–415.